Introduction to Resolution and Logic Programming

Thomas Piecha

University of Belgrade
October/November 2018

Preface

These are the lecture notes of an Erasmus+ mini-course given at the Philosophy Department of the University of Belgrade in October/November 2018. The objectives of this mini-course were to give an introduction to resolution methods, to approach computability on the basis of logic programming, and to provide some useful tools along the way, such as Skolemization and unification.

I would like to thank all students, coming from philosophy as well as from mathematics, for their participation. I am very grateful to Miloš Adžić for his kind invitation, organization and generous hospitality.

Belgrade, October/November 2018 T. P.

Contents

1 Propositional resolution

1.1 The resolution calculus

Logical consequence or universal validity of formulas can be shown by using resolution. We first introduce the resolution calculus, which is sound with respect to classical logic. We then consider resolution refutations. The basic idea is to establish the universal validity of a formula A by using resolution to show that $\neg A$ is unsatisfiable. For resolution refutations the resolution calculus consists of only one rule. However, resolution presupposes sets of clauses; to obtain them, formulas have to be transformed into conjunctive normal form first.

Definition 1.1 (i) A *clause* is a sequent, that is, an expression $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$, where $A_1, \ldots, A_n, B_1, \ldots, B_m$ are atomic formulas. *clause*

(ii) To the left of the *sequent sign* $\vdash$ stands the *antecedent*, to the right the *succedent*.

Definition 1.2 (i) A *literal* is an atom A (*positive literal*) or its negation $\neg A$ (*negative literal*). *literal*

(ii) Two literals are called *complementary*, if one is the negation of the other. *complementary*

Definition 1.3 A formula is in *conjunctive normal form* (CNF for short), if it has the form of a conjunction of disjunctions of literals. *conjunctive normal form*

Remarks. (i) The sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ stands for the disjunction $\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$ of literals.

(ii) Antecedent and succedent of a clause are considered as sets; that is, multiplicity and order of list elements are irrelevant.

(iii) A clause is also considered as a set of literals.

(iv) Metalinguistic variables for finite sets of atoms are X, Y, Z, possibly with indices; for clauses: $S, S_1, S_2, \ldots$; and for sets of clauses: $\Gamma, \Delta, \ldots$.

(v) Since sequents correspond to formulas, we can use the notation of logical consequence for formulas also for sequents: $\Gamma \models S$.

(vi) $X, A \vdash Y, B$ stands for $X \cup \{A\} \vdash Y \cup \{B\}$, where A and B could here also be an element of X or Y, respectively.

(vii) The *empty clause* contains no atoms. Notation: $\vdash$ or $\square$.

The empty clause is unsatisfiable.

Definition 1.4 For atoms A and possibly empty sets of atoms $X, Y, \ldots$ the *propositional resolution calculus* is given by the axiom (or axiom schema) *resolution calculus*

$$(\mathrm{Ax})\ \frac{}{X, A \vdash A, Y}$$

and the *propositional resolution rule*: *resolution rule*

$$\frac{X_1 \vdash Y_1, A \qquad A, X_2 \vdash Y_2}{X_1, X_2 \vdash Y_1, Y_2}\ (\mathcal{R})$$

The conclusion of $(\mathcal{R})$ is called *resolvent* (w.r.t. A) of the premisses. *resolvent*

Definition 1.5 We inductively define *derivations* in the resolution calculus as follows (where we write $\begin{matrix}\mathscr{D}\\ S\end{matrix}$ to express that derivation $\mathscr{D}$ ends with clause S): *derivation*

(i) $X \vdash Y$ is a derivation (of $X \vdash Y$ from $X \vdash Y$).

(ii) $(\mathrm{Ax})\ \dfrac{}{X, A \vdash A, Y}$ is a derivation.

(iii) For derivations $\begin{matrix}\mathscr{D}_1\\ X_1 \vdash Y_1, A\end{matrix}$ and $\begin{matrix}\mathscr{D}_2\\ A, X_2 \vdash Y_2\end{matrix}$ also

$$\frac{\begin{matrix}\mathscr{D}_1\\ X_1 \vdash Y_1, A\end{matrix} \qquad \begin{matrix}\mathscr{D}_2\\ A, X_2 \vdash Y_2\end{matrix}}{X_1, X_2 \vdash Y_1, Y_2}\,(\mathcal{R})$$

is a derivation.

Definition 1.6 The set of *assumptions* (or *hypotheses*) of a derivation is recursively defined as follows: *assumptions*

(i) $Hyp(X \vdash Y) := \{X \vdash Y\}$.

(ii) $Hyp\Big((\mathrm{Ax})\ \dfrac{}{X, A \vdash A, Y}\Big) := \emptyset$.

(iii) $Hyp\left(\dfrac{\begin{matrix}\mathscr{D}_1\\ X_1 \vdash Y_1, A\end{matrix} \qquad \begin{matrix}\mathscr{D}_2\\ A, X_2 \vdash Y_2\end{matrix}}{X_1, X_2 \vdash Y_1, Y_2}\,(\mathcal{R})\right) := Hyp\left(\begin{matrix}\mathscr{D}_1\\ X_1 \vdash Y_1, A\end{matrix}\right) \cup Hyp\left(\begin{matrix}\mathscr{D}_2\\ A, X_2 \vdash Y_2\end{matrix}\right)$.

A derivation is thus a binary tree (branching upward), whose leaves are assumptions. In the case of the axiom (Ax) its empty premiss is an empty assumption and a leave.

Definition 1.7 The *derivability relation* $\Gamma \vdash_{\mathrm{Res}} S$ holds iff there is a derivation $\begin{matrix}\mathscr{D}\\ S\end{matrix}$ from assumptions $Hyp\left(\begin{matrix}\mathscr{D}\\ S\end{matrix}\right) \subseteq \Gamma$ that ends with clause S. *derivability relation*

Thus $\Gamma \vdash_{\mathrm{Res}} S$ means that the clause S is *derivable* from the set of clauses Γ in the propositional resolution calculus. (The derivability relation $\vdash_{\mathrm{Res}}$ is not to be confused with the sequent sign $\vdash$.) *derivable*

Remarks. (i) The axiom creates *tautological clauses*. A clause *tautological clause*

$$\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$$

is tautological iff it contains two complementary literals (i.e., if $A_i = B_j$ for at least one pair i, j). In this case the clause contains the subformula $\neg A \vee A$, which is the case iff the clause is an axiom.

(ii) The axiom can be used to weaken clauses. Using the axiom $X_1, A \vdash A, Y_1$ the clause $A, X_2 \vdash Y_2$ can be weakened by

$$\frac{(\mathrm{Ax})\ \dfrac{}{X_1, A \vdash A, Y_1} \qquad A, X_2 \vdash Y_2}{X_1, A, X_2 \vdash Y_1, Y_2}\,(\mathcal{R})$$

to $X_1, A, X_2 \vdash Y_1, Y_2$. Correspondingly, clauses of the form $X_2 \vdash A, Y_2$ can be weakened to $X_1, X_2 \vdash A, Y_1, Y_2$.

Theorem 1.8 (Soundness)
The resolution calculus is sound, that is, it holds: If $\Gamma \vdash_{\text{Res}} S$*, then* $\Gamma \models S$.

Proof. Exercise. QED

1.2 Resolution refutation

In the following we consider the resolution calculus as a *refutation calculus*. The axiom is no longer needed in this case, although sets of clauses to be refuted may contain clauses of the form of the axiom. The resolution calculus then consists only of the resolution rule

$$\frac{X_1 \vdash Y_1, A \qquad A, X_2 \vdash Y_2}{X_1, X_2 \vdash Y_1, Y_2}\,(\mathcal{R})$$

We will assume that $A \notin Y_1$ and $A \notin X_2$.

Definition 1.9 A *resolution refutation* of a set Γ of clauses is a derivation of the empty clause $\vdash$ (or $\square$, respectively) from clauses in Γ. *resolution refutation*

Since the resolution rule is sound there must for each interpretation be at least one false clause in Γ, if there is a refutation refutation of Γ. In other words: A resolution refutation of Γ establishes $\Gamma \vdash_{\text{Res}} \square$, and it is by the soundness of the resolution rule a proof of $\Gamma \models \square$, that is, of the unsatisfiability of Γ.
If one considers a formula A, one has to translate it into a *set of clauses* $Cl(A)$ first, in order to be able to apply resolution.

Definition 1.10 A *resolution proof* of a formula A is a resolution refutation of $Cl(\neg A)$. *resolution proof*

From a resolution proof of A follows $\models A$, since in general

$$\begin{aligned}\Gamma \models A &\iff A \text{ is true in all models of } \Gamma\\ &\iff \Gamma \cup \{\neg A\} \text{ has no model}\\ &\iff \Gamma \cup \{\neg A\} \text{ is unsatisfiable}\end{aligned}$$

and for $\Gamma = \emptyset$ thus in particular

$$\models A \iff \neg A \text{ is unsatisfiable.}$$

Due to the soundness of the resolution rule, a derivation of the unsatisfiable empty clause $\square$ implies the unsatisfiability of $Cl(\neg A)$, and thus also the unsatisfiability of $\neg A$.
However, before one can attempt a resolution proof of a formula A one must first construct the conjunctive normal form of $\neg A$ in order to obtain the set of clauses $Cl(\neg A)$. The CNF is not constructed by using a semantic procedure (e.g. truth tables); this would decide the formula and would thus render resolution redundant. Instead, one uses a syntactical construction like the one given in Appendix A.1.
To each conjunct in a CNF corresponds a clause. Hence, to the CNF

$$(\neg A_{11} \vee \ldots \vee \neg A_{1k_1} \vee B_{11} \vee \ldots \vee B_{1l_1}) \wedge \ldots \wedge (\neg A_{j1} \vee \ldots \vee \neg A_{jk_j} \vee B_{j1} \vee \ldots \vee B_{jl_j})$$

of a formula A there corresponds a *set of clauses* *set of clauses*

$$Cl(A) = \{A_{11}, \ldots, A_{1k_1} \vdash B_{11}, \ldots, B_{1l_1} \;;\; \ldots \;;\; A_{j1}, \ldots, A_{jk_j} \vdash B_{j1}, \ldots, B_{jl_j}\}$$

to which resolution can be applied. (We use the semicolon to separate the elements of sets of clauses.)

Remark. In disjunctions of literals $A_1 \vee \ldots \vee A_n$ there may be several occurrences of the same literal. If A_i and A_j (for $1 \leq i < j \leq n$) are two occurrences of the same literal, then the disjunction

$$A_1 \vee \ldots \vee A_i \vee A_{i+1} \vee \ldots \vee A_{j-1} \vee A_{j+1} \vee \ldots \vee A_n$$

in which the occurrence A_j is eliminated, is called a *factor* of the initial disjunction. If there are no multiple occurrences of a literal, then the disjunction of literals is called *factor-free*.

factor

factor-free

Since antecedent and succedent of a clause are considered as sets, a formula like $\neg A \vee \neg A \vee B \vee B$ (for atoms A, B) has the corresponding clause $A \vdash B$, to which the factor-free formula $\neg A \vee B$ corresponds.

Theorem 1.11 (Import-Export)
It holds $\Gamma \models A \to B$ *iff* $\Gamma \cup \{A\} \models B$.

Proof. Exercise. QED

Corollary 1.12 *Since*

$$A_1 \to (A_2 \to (\cdots (A_n \to B) \cdots)) \dashv\models (A_1 \wedge A_2 \wedge \ldots \wedge A_n) \to B$$

holds, we have $A_1, \ldots, A_n \models B$ *iff* $\models A_1 \wedge \ldots \wedge A_n \to B$.

Remarks. (i) Instead of $Cl(\neg(A \to B))$ one can also consider $Cl(A) \cup Cl(\neg B)$.

(ii) In the case of logical consequences $A_1, \ldots, A_n \models B$ one can apply resolution to

$$Cl(\neg(A_1 \wedge \ldots \wedge A_n \to B)) \qquad \text{or} \qquad \bigcup_{1 \leq i \leq n} Cl(A_i) \cup Cl(\neg B).$$

Examples. Let A, B, C be atoms. (We omit some intermediate steps in the respective constructions of the CNF.)

(i) $\models A \to A \vee B$

CNF of $\neg(A \to A \vee B)$:

$$\begin{aligned} \neg(A \to A \vee B) &\rightsquigarrow \neg(\neg A \vee A \vee B) \\ &\rightsquigarrow \neg\neg A \wedge \neg A \wedge \neg B \\ &\rightsquigarrow A \wedge \neg A \wedge \neg B \\ &\rightsquigarrow \{ \vdash A \,;\, A \vdash \,;\, B \vdash \} = Cl(\neg(A \to A \vee B)) \end{aligned}$$

Resolution refutation: $\dfrac{\vdash A \qquad A \vdash}{\vdash}\,(\mathcal{R})$

We have shown

$$Cl(\neg(A \to A \vee B)) \vdash_{\text{Res}} \square$$

and by soundness follows

$$Cl(\neg(A \to A \vee B)) \models \square$$

Since $\square$ is unsatisfiable, $Cl(\neg(A \to A \vee B))$ and hence $\neg(A \to A \vee B)$ must be unsatisfiable, that is, $\models A \to A \vee B$.

(ii) $\models A \vee (B \vee C) \to (A \vee B) \vee C$

CNF of $\neg(A \vee (B \vee C) \to (A \vee B) \vee C)$:

$$\begin{aligned}\neg(A \vee (B \vee C) \to (A \vee B) \vee C) &\rightsquigarrow \neg(\neg(A \vee (B \vee C)) \vee ((A \vee B) \vee C))\\ &\rightsquigarrow (A \vee (B \vee C)) \wedge \neg((A \vee B) \vee C)\\ &\rightsquigarrow (A \vee B \vee C) \wedge \neg A \wedge \neg B \wedge \neg C\\ &\rightsquigarrow \{\vdash A, B, C\ ;\ A \vdash\ ;\ B \vdash\ ;\ C \vdash\}\end{aligned}$$

Resolution refutation:
$$\dfrac{\dfrac{\dfrac{\vdash A, B, C \qquad A \vdash}{\vdash B, C}(\mathcal{R}) \qquad B \vdash}{\vdash C}(\mathcal{R}) \qquad C \vdash}{\vdash}(\mathcal{R})$$

Due to the Import-Export Theorem we can alternatively consider $A \vee (B \vee C) \models (A \vee B) \vee C$:

CNF of $A \vee (B \vee C)$: $A \vee B \vee C \rightsquigarrow \{\vdash A, B, C\}$.

CNF of $\neg((A \vee B) \vee C)$: $\neg((A \vee B) \vee C) \rightsquigarrow \neg A \wedge \neg B \wedge \neg C \rightsquigarrow \{A \vdash\ ;\ B \vdash\ ;\ C \vdash\}$.

One thus obtains the same set of clauses $\{\vdash A, B, C\ ;\ A \vdash\ ;\ A \vdash\ ;\ B \vdash\ ;\ C \vdash\}$ and the shown resolution refutation.

(iii) $\models (\neg A \to \neg B) \to (B \to A)$

CNF of $\neg((\neg A \to \neg B) \to (B \to A))$:

$$\begin{aligned}\neg((\neg A \to \neg B) \to (B \to A)) &\rightsquigarrow \neg((A \vee \neg B) \to (\neg B \vee A))\\ &\rightsquigarrow \neg(\neg(A \vee \neg B) \vee \neg B \vee A)\\ &\rightsquigarrow \neg((\neg A \wedge B) \vee \neg B \vee A)\\ &\rightsquigarrow \neg(\neg A \wedge B) \wedge B \wedge \neg A\\ &\rightsquigarrow (A \vee \neg B) \wedge B \wedge \neg A\\ &\rightsquigarrow \{B \vdash A\ ;\ \vdash B\ ;\ A \vdash\}\end{aligned}$$

Resolution refutation:
$$\dfrac{\vdash B \qquad \dfrac{B \vdash A \qquad A \vdash}{B \vdash}(\mathcal{R})}{\vdash}(\mathcal{R})$$

Alternatively for $\neg A \to \neg B \models B \to A$:

CNF of $\neg A \to \neg B$: $A \vee \neg B \rightsquigarrow \{B \vdash A\}$,

CNF of $\neg(B \to A)$: $B \wedge \neg A \rightsquigarrow \{\vdash B\ ;\ A \vdash\}$,

Set of clauses (as above): $\{B \vdash A\ ;\ \vdash B\ ;\ A \vdash\}$; with the above resolution refutation.

Remarks. (i) A large part of the effort goes into the construction of the CNF.

(ii) The CNF need not be logically equivalent to the initial formula, however. For the refutation calculus it is sufficient to have an *equi-satisfiable* CNF, that is, we only need that the CNF is satisfiable iff the initial formula is satisfiable. *equi-satisfiable*

(iii) Equi-satisfiability is a much weaker property than logical validity:

Let A be a contingent formula. Then A is satisfiable iff $\neg A$ is satisfiable. But A and $\neg A$ cannot be logically equivalent, of course.

(iv) The number of steps in the construction of a *logically equivalent* CNF or of a disjunctive normal form (DNF) is exponential in the number of propositional variables.

(v) However, there exist polynomial-time algorithms for the construction of *equisatisfiable* CNF (see e.g. Leitsch, 1997, pages 19–21). This is an advantage for resolution. (There is no polynomial-time algorithm for the construction of *equitautological* CNF.)

We now present an algorithm for finding a resolution refutation (Theorem 1.14), which decides for any given finite set of clauses Γ whether there exists a resolution refutation for Γ or not.

Definition 1.13 (i) It is $\mathcal{R}_A(S_1, S_2)$ the *resolvent of clauses S_1 and S_2 with respect to A*, if such a resolvent exists; otherwise let $\mathcal{R}_A(S_1, S_2)$ be undefined. $\mathcal{R}_A(S_1, S_2)$

(ii) It is $\mathcal{R}_A(\Gamma) := \{\mathcal{R}_A(S_1, S_2) \mid S_1, S_2 \in \Gamma\}$ the *set of all possible resolution results with respect to A*. $\mathcal{R}_A(\Gamma)$

(iii) Let Γ_A be the set of all clauses in Γ that contain A.
Then $Res_A(\Gamma) := (\Gamma \setminus \Gamma_A) \cup \mathcal{R}_A(\Gamma_A)$. $Res_A(\Gamma)$

Theorem 1.14 *Let the following procedure be given, which we formulate for sets of clauses Γ, in which exactly the atoms $A_1, \ldots, A_n$ occur.*
For i from 1 to n:

(1) *Eliminate tautological clauses from Γ. Let the resulting set of clauses be Γ'.*

(2) *Construct $Res_{A_i}(\Gamma')$.*

(3) *Set $\Gamma := Res_{A_i}(\Gamma')$.*

Then the following holds: The procedure produces the set of clauses $\{\Box\}$ iff there exists a refutation refutation for Γ.

Proof. Exercise. (Maybe using the completeness results given in the next section.) QED

Example. We consider the set of clauses $\Gamma = \{A_2 \vdash A_1\ ;\ \vdash A_2\ ;\ A_1 \vdash\ ;\ A_1, A_2 \vdash A_2\}$.

(1) Eliminate the tautological clause $A_1, A_2 \vdash A_2$: $\Gamma' := \Gamma \setminus \{A_1, A_2 \vdash A_2\}$.

(2) $Res_{A_1}(\Gamma') = (\Gamma' \setminus \Gamma'_{A_1}) \cup \mathcal{R}_{A_1}(\Gamma'_{A_1}) = \{\vdash A_2\} \cup \{A_2 \vdash\}$

(3) $\Gamma := \{\vdash A_2\ ;\ A_2 \vdash\}$

(1) Γ contains no tautological clauses. $\Gamma' := \Gamma$.

(2) $Res_{A_2}(\Gamma') = (\Gamma' \setminus \Gamma'_{A_2}) \cup \mathcal{R}_{A_2}(\Gamma'_{A_2}) = \emptyset \cup \{\Box\}$

(3) $\Gamma := \{\Box\}$.

1.3 Completeness

Soundness of the resolution calculus (Theorem 1.8) guarantees that each clause S which is derivable from a set of clauses Γ is also a logical consequence of Γ, that is, $\Gamma \vdash_{\text{Res}} S \implies \Gamma \models S$. In the refutation calculus we could in particular use soundness to infer from a resolution refutation of Γ that Γ is unsatisfiable. In addition, by Theorem 1.14

there is a procedure, which decides for any given (propositional) set of clauses whether a resolution refutation exists or not.
One can also show that any clause S which is a logical consequence of a set of clauses Γ can be derived from Γ in the resolution calculus. In other words, the resolution calculus is complete, that is, $\Gamma \models S \implies \Gamma \vdash_{\text{Res}} S$. This can be shown by first proving that the refutation calculus is complete (Theorem 1.16); the completeness of the resolution calculus (Theorem 1.17) can then be obtained in the basis of this result.

Definition 1.15 (i) $Res(S_1, S_2, S_3)$ means that S_3 is a resolvent of S_1 and S_2.

(ii) $Res(\Gamma) := \{S \mid Res(S_1, S_2, S) \text{ for } S_1, S_2 \in \Gamma\}$ is the set of all resolvents S of clauses $S_1, S_2 \in \Gamma$.

(iii) The *resolution step* $Rs^n(\Gamma)$ of a set of clauses Γ is defined as follows: *resolution step*

$$Rs^0(\Gamma) := \Gamma$$
$$Rs^{n+1}(\Gamma) := Rs^n(\Gamma) \cup Res(Rs^n(\Gamma))$$

(iv) It is

$$R(\Gamma) := \bigcup_{n \in \mathbb{N}} Rs^n(\Gamma)$$

the *resolution closure* of Γ. *resolution closure*

Example. Let $\Gamma = \{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \}$. Then

$$\begin{aligned}
Rs^0(\Gamma) &= \Gamma = \{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \} \\
Rs^1(\Gamma) &= Rs^0(\Gamma) \cup Res(Rs^0(\Gamma)) = \Gamma \cup Res(\Gamma) = \Gamma \cup Res(\{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \}) \\
&= \{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \} \cup \{ \vdash B \,;\, B \vdash \,;\, A \vdash \} \\
&= \{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \,;\, \vdash B \,;\, B \vdash \,;\, A \vdash \} \\
Rs^2(\Gamma) &= Rs^1(\Gamma) \cup Res(Rs^1(\Gamma)) \\
&= Rs^1(\Gamma) \cup Res(\{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \,;\, \vdash B \,;\, B \vdash \,;\, A \vdash \}) \\
&= Rs^1(\Gamma) \cup \{ \vdash B \,;\, A \vdash \,;\, B \vdash \,;\, \square \} \\
&= \{ \vdash A \,;\, A \vdash B \,;\, A, B \vdash \,;\, \vdash B \,;\, B \vdash \,;\, A \vdash \,;\, \square \} \\
Rs^3(\Gamma) &= Rs^2(\Gamma) \quad \text{(since no new resolvents can be generated with } \square\text{)}.
\end{aligned}$$

Hence, the resolution closure is $R(\Gamma) = Rs^2(\Gamma)$.

Due to compactness, we can restrict ourselves to finite sets of clauses in the following.

Theorem 1.16 (Completeness of the refutation calculus)
If the set of clauses Γ is unsatisfiable, then there is an $n \in \mathbb{N}$ with $\square \in Rs^n(\Gamma)$, and thus also $\square \in R(\Gamma)$.

Proof. Cp. Goltz & Herre, 1990, p. 45f. QED

Theorem 1.17 (Completeness of the resolution calculus)
Let $\square \notin \Gamma$. Then the propositional resolution calculus is complete, that is: If $\Gamma \models S$, then $\Gamma \vdash_{\text{Res}} S$.

Proof. Exercise. Provide a procedure that transforms any given resolution refutation into a derivation of S from Γ in the resolution calculus, if $\Gamma \models S$. QED

Together with soundness (Theorem 1.8) we thus have for $\square \notin \Gamma$: $\Gamma \models S$ *iff* $\Gamma \vdash_{\mathrm{Res}} S$.

The restriction $\square \notin \Gamma$ in Theorem 1.17 is necessary, since the resolution calculus does not allow for a weakening of the empty clause $\square$. For $\square \in \Gamma$ we have $\Gamma \models S$ for any clause S, since Γ is unsatisfiable. In case $\Gamma \setminus \{\square\}$ is satisfiable, we then have $\Gamma \nvdash_{\mathrm{Res}} S$ for all non-tautological clauses $S \notin \Gamma$. (All tautological clauses and all clauses in Γ are trivially derivable.) But then the propositional resolution calculus cannot be complete. Alternatively one can choose one of the following two options:

(i) In case $\square \in \Gamma$ we replace the empty clause $\square$ by two clauses $\vdash A$ and $A \vdash$. Let the new set of clauses be Γ'.

Since Γ' is unsatisfiable, just like Γ, we also have $\Gamma' \models S$ for all S. For the calculus we have $\Gamma' \vdash_{\mathrm{Res}} S$ for all S, since now any clause $S = (X \vdash Y)$ can be derived from Γ':

$$\dfrac{\dfrac{\vdash A \qquad \dfrac{}{X, A \vdash A, Y}\,(\mathrm{Ax})}{X \vdash A, Y}\,(\mathcal{R}) \qquad A \vdash}{X \vdash Y}\,(\mathcal{R})$$

(Instead of replacing $\square$ in Γ by the clauses $\vdash A$ and $A \vdash$, one can also simply extend Γ by these two clauses to a set Γ_e, because $\Gamma_e \setminus \{\square\}$ is unsatisfiable independently of whether $\Gamma \setminus \{\square\}$ is satisfiable or not. One could thus also weaken the restriction $\square \notin \Gamma$ in Theorem 1.17 by demanding $\square \notin \Gamma$ only in case $\Gamma \setminus \{\square\}$ is satisfiable. Since if $\Gamma \setminus \{\square\}$ is already unsatisfiable, then any clause can be derived from this set.)

(ii) We allow for $\square \in \Gamma$ but extend the resolution calculus by a weakening rule

$$\dfrac{X_1 \vdash Y_1}{X_1, X_2 \vdash Y_1, Y_2}\,(\mathcal{V})$$

where X_1 and X_2 may in particular be empty. Then any clause can be derived from the empty clause as well.

For each of the two options the resolution calculus (extended by $(\mathcal{V})$ in the second option) is sound and complete.

If the resolution calculus is used as a refutation calculus (i.e., without axiom an weakening rule), then one is only interested in finding derivations of the empty clause from given sets of clauses. The empty clause is then always the end point in such derivations.

2 First-order resolution

We consider first-order languages $\mathscr{L}$, in which the non-logical symbols are *relation symbols* $P, Q, R, \ldots$, *individual constants* (*constants* for short) $a, b, c, \ldots$ and *function symbols* $f, g, h, \ldots$. *Terms* $t, s, \ldots$ are constructed from variables $x, y, z, \ldots$, constants and function symbols, as usual.

2.1 Substitution

Definition 2.1 (i) A *substitution* is a function that maps variables to terms and is non-identical for only finitely many variables. *substitution*

(ii) We write substitutions as finite sets (using square brackets [,]) of the form

$$[x_1/t_1, \ldots, x_n/t_n] \qquad (\text{for } 0 \leq i \leq n),$$

for pairwise distinct variables x_i and terms t_i such that $t_i \neq x_i$.

In the notation $[x_1/t_1, \ldots, x_n/t_n]$ we thus only specify the (always finitely many) non-identical assignments of terms to variables of a substitution.

(iii) The expression x_i/t_i is called a *binding* for x_i. *binding*

We also say "t_i is substituted for x_i" or "x_i is replaced by t_i".

(iv) We use $\sigma, \rho, \tau, \vartheta, \ldots$ as names for substitutions.

(v) A substitution $\sigma = [x_1/t_1, \ldots, x_n/t_n]$ is called *ground substitution*, if each term t_i is a ground term. *ground substitution*

(vi) If $\sigma = \emptyset$, then σ is called the *empty substitution* and is denoted by ε. *empty substitution*

Examples. (i) $[x_1/f(x), y/g(a, z), z/x]$ is a substitution.

(ii) $[x/h(a, b)]$ is a ground substitution.

Definition 2.2 Let $\sigma = [x_1/t_1, \ldots, x_n/t_n]$ be a substitution and E an expression, that is, a formula or a term.

(i) The simultaneous replacement of each free occurrence of x_i in E by t_i for $0 \leq i \leq n$ is called *application* of σ to E. *application*

Notation: $E\sigma$. Substitutions thus operate to the left.

In case E is a formula, t_i in E has to be *freely substitutable* for x_i; that is, x_i may only be replaced by t_i, if variables occurring in t_i will not get bound by a quantifier in E. *freely substitutable*

(ii) The resulting expression $E\sigma$ is called *(substitution-) instance* of E for σ. *instance*

(iii) If $X = \{E_1, \ldots, E_n\}$ is a finite set of expressions, then $X\sigma$ stands for the set $\{E_1\sigma, \ldots, E_n\sigma\}$.

Remark. Since substitutions are carried out simultaneously, it is

$$f(x, y)[x/y, y/b] = f(y, b)$$

and *not* $f(b, b)$. The latter would be obtained by subsequent replacements:

$$(f(x, y)[x/y])[y/b] = f(y, y)[y/b] = f(b, b)$$

Definition 2.3 The *composition* $\sigma\tau$ of two substitutions *composition*

$$\sigma = [x_1/s_1, \ldots, x_n/s_n] \quad \text{and} \quad \tau = [y_1/t_1, \ldots, y_m/t_m]$$

with $0 \leq i \leq n$ and $0 \leq j \leq m$ is given as follows: We form the series of bindings

$$x_1/(s_1\tau), \ldots, x_n/(s_n\tau), y_1/t_1, \ldots, y_m/t_m.$$

In this series we eliminate

– all bindings $x_i/(s_i\tau)$, for which $x_i = (s_i\tau)$,

– and all bindings y_j/t_j, for which $y_j \in \{x_1, \ldots, x_n\}$.

The substitution consisting of the bindings in the resulting series is the *composition* $\sigma\tau$ of σ and τ.

Example. Let $\sigma = [x/f(y), y/z]$ and $\tau = [x/a, y/b, z/y]$. Then the series of bindings is

$$\underbrace{x/(f(y)[x/a, y/b, z/y])}_{x/f(b)}, \underbrace{y/(z[x/a, y/b, z/y])}_{y/y}, x/a, y/b, z/y$$

in which we have to eliminate the bindings y/y, x/a and y/b. One obtains the composition $\sigma\tau = [x/f(b), z/y]$.

Lemma 2.4 *Let ρ, σ, ϑ be substitutions and ε the empty substitution. Then the following holds:*

(i) $\rho\varepsilon = \varepsilon\rho = \rho$.

(ii) $(\rho\sigma)\vartheta = \rho(\sigma\vartheta)$.

(iii) $(E\rho)\sigma = E(\rho\sigma)$*, for arbitrary formulas and terms E.*

Proof. Exercise. QED

Lemma 2.5 *The composition of substitutions is not commutative.*

Proof. Exercise. QED

Definition 2.6 Let S be a clause with $\mathrm{FV}(S) = \{x_1, \ldots, x_n\}$ for $n \geq 0$, and let $\sigma = [x_1/t_1, \ldots, x_n/t_n]$ be a ground substitution. Then $S\sigma$ is a *ground instance* of S. *ground instance*

Theorem 2.7 (Skolem–Herbrand–Gödel)
A set of clauses Γ is unsatisfiable iff there is a finite unsatisfiable set Γ' of ground instances of clauses from Γ.

Proof. See Gallier, 2015, § 7.6. QED

The Skolem–Herbrand–Gödel Theorem is formulated for the semantic notion of unsatisfiability. There is also *Herbrand's Theorem*, which holds for the syntactic notion of provability (see e.g. Troelstra & Schwichtenberg, 2000): A formula A in prenex normal form with kernel $B(x_1, \ldots, x_n)$ is provable (in a proof system like the sequent calculus LK, for example) iff there is a disjunction D of substitution instances of $B(x_1, \ldots, x_n)$, which is derivable by propositional rules only, and where A is derivable from D by using only first-order (and maybe structural) rules.

2.2 Skolemization

In order to generate a set of clauses for a given formula in prenex normal form in a language $\mathscr{L}$ one first has to eliminate quantifiers in an adequate way. This can be achieved by *Skolemization* (named after Thoralf Skolem, 1887–1963), which yields a quantifier-free formula in an extended language $\mathscr{L}_S \supseteq \mathscr{L}$. This formula is in general no longer logically equivalent to the initial formula, but it is in any case equi-satisfiable with it.
To explain the idea of Skolemization we consider the following statement, which is true in the standard interpretation:

$$\text{For all } n \in \mathbb{N} \text{ there is an } m \in \mathbb{N} \text{ such that } n < m.$$

For the function $f(n) = n+1$ we have $n < f(n)$ for all n. The variable m, which is bound by the existential quantifier, can thus be replaced by a function $f(n)$. This renders the existential quantifier useless, and it can thus be removed. If, in addition, one understands free variables universally, then the universal quantifier can be removed as well. If we also allow for interpretations that deviate from the standard interpretation, then the resulting statement is no longer logically equivalent to the initial statement; however, the two statements are equi-satisfiable.
The above statement has the form

$$\forall x \exists y P(x, y)$$

where we here want to presuppose that this is a formula in a language $\mathscr{L}$ without function symbols. One now replaces in $\forall x \exists y P(x, y)$ the variable y, which is bound by an existential quantifier, by a function term $f(x)$. We have thus extended $\mathscr{L}$ by the unary function symbol f. If we now also remove the universal quantifier, then we obtain as Skolem normal form the formula $P(x, f(x))$. This formula is satisfiable iff the initial formula is satisfiable.

Definition 2.8 The *universal closure* $\forall A$ of a formula A is the formula *universal closure*

$$\forall x_1 \ldots \forall x_n A[y_1/x_1, \ldots, y_n/x_n]$$

where $y_1, \ldots, y_n$ are *all* free variables in A, and $x_1, \ldots, x_n$ are variables not occurring in A. (This is a non-deterministic definition, since the pairwise distinct variables x_i can otherwise be chosen freely.)

Definition 2.9 Let $A \in \mathscr{L}$ be a formula of the form $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n B$ in prenex normal form with kernel B. Then a *Skolem normal form* A_S of A in a suitable language $\mathscr{L}_S$ extending $\mathscr{L}$ is defined by the following procedure: *Skolem normal form*

(1) Set $A_S = \forall A$.

(2) If the prefix of A_S contains only universal quantifiers, eliminate them and halt. If the prefix is empty, then halt as well. A_S is a Skolem normal form of A.

(3) Let $\mathsf{Q}_i x_i$ be the first existential quantifier (from the left) in A_S. Let $x_{i_1}, \ldots, x_{i_j}$ be the variables to the left of x_i; they are the variables in $\{x_1, \ldots, x_{i-1}\}$ that have not been eliminated yet.

(4) If in A_S there are no universal quantifiers to the left of x_i, then extend $\mathscr{L}$ by adding a new constant a_i and replace each occurrence of x_i in the kernel of A_S by a_i.

(5) Otherwise extend $\mathscr{L}$ by adding a new j-ary function sign f_i. Replace each occurrence of x_i in the kernel of A_S by the term $f_i(x_{i_1}, \ldots, x_{i_j})$.

(6) Eliminate $\exists x_i$ from the prefix of A_S. Go to step (2).

Examples. We use indexed variables in the following. The choice of individual constants and function signs is then fixed by the procedure. In formulas with non-indexed variables one has to take care that in steps (4) and (5) new constants and new function signs are introduced, respectively (cp. Example (v)).

(i) $\exists x_1 P(x_1) \rightsquigarrow P(a_1)$

(ii) $\exists x_1 \forall x_2 P(x_1, x_2) \rightsquigarrow \forall x_2 P(a_1, x_2) \rightsquigarrow P(a_1, x_2)$

(iii) $\forall x_1 \exists x_2 P(x_1, x_2) \rightsquigarrow \forall x_1 P(x_1, f_2(x_1)) \rightsquigarrow P(x_1, f_2(x_1))$

(iv) $\begin{aligned} \exists x_1 \forall x_2 \exists x_3 P(x_1, x_2, x_3) &\rightsquigarrow \forall x_2 \exists x_3 P(a_1, x_2, x_3) \\ &\rightsquigarrow \forall x_2 P(a_1, x_2, f_3(x_2)) \\ &\rightsquigarrow P(a_1, x_2, f_3(x_2)) \end{aligned}$

(v) $\begin{aligned} \forall x_1 \exists x_2 \forall x_3 \exists x_4 P(x_1, x_2, x_3, x_4) &\rightsquigarrow \forall x_1 \forall x_3 \exists x_4 P(x_1, f_2(x_1), x_3, x_4) \\ &\rightsquigarrow \forall x_1 \forall x_3 P(x_1, f_2(x_1), x_3, f_3(x_1, x_3)) \\ &\rightsquigarrow P(x_1, f_2(x_1), x_3, f_4(x_1, x_3)) \end{aligned}$

For non-indexed variables:

$$\begin{aligned} \forall x \exists y \forall z \exists u P(x, y, z, u) &\rightsquigarrow \forall x \forall z \exists u P(x, f(x), z, u) \\ &\rightsquigarrow \forall x \forall z P(x, f(x), z, g(x, z)) \\ &\rightsquigarrow P(x, f(x), z, g(x, z)) \end{aligned}$$

Alternatively, one can skolemize according to the following definition, where universal quantifiers are already eliminated in intermediate steps.

Definition 2.10 Let $A \in \mathscr{L}$ be a formula of the form $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n B$ (for $n \geq 0$) in prenex normal form with kernel B. The following procedure yields a *Skolem normal form* A_S of A in a suitable language $\mathscr{L}_S \supseteq \mathscr{L}$. *Skolem normal form*

For i from 1 to n:

(1) If $n = 0$, set $A_S \coloneqq A$.

(2) If Q_i is a universal quantifier, set $A \coloneqq \mathsf{Q}_{i+1} x_{i+1} \ldots \mathsf{Q}_n x_n B$.

(3) If Q_i is an existential quantifier, and $y_1, \ldots, y_m$ are the variables occurring free in A, set

$$A \coloneqq \mathsf{Q}_{i+1} x_{i+1} \ldots \mathsf{Q}_n x_n B[x_i / f(y_1, \ldots, y_m)]$$

where f is a new m-ary function sign (i.e., f does not yet occur in A), or a new constant, in case $m = 0$.

Remarks. (i) The procedure is non-deterministic, if one does not fix an order of signs to be introduced.

(ii) In contradistinction to the procedure given in Definition 2.9, where in the first step one has to form the universal closure, one can apply the latter procedure directly also to open formulas. In step (3) all variables occurring free in A are taken care of.

Examples. (i) $\exists x P(x) \rightsquigarrow P(a)$

(ii) $\exists x \forall y P(x, y) \rightsquigarrow \forall y P(a, y) \rightsquigarrow P(a, y)$

(iii) $\forall x \exists y P(x, y) \rightsquigarrow \exists y P(x, y) \rightsquigarrow P(x, f(x))$

(iv) $\exists x \forall y \exists z P(x, y, z) \rightsquigarrow \forall y \exists z P(a, y, z) \rightsquigarrow \exists z P(a, y, z) \rightsquigarrow P(a, y, f(y))$

(v) $\begin{aligned} \forall x \exists y \forall z \exists u P(x, y, z, u) &\rightsquigarrow \exists y \forall z \exists u P(x, y, z, u) \\ &\rightsquigarrow \forall z \exists u P(x, f(x), z, u) \\ &\rightsquigarrow \exists u P(x, f(x), z, u) \\ &\rightsquigarrow P(x, f(x), z, g(x, z)) \end{aligned}$

In general, the Skolem normal form A_S of a formula A is *not* logically equivalent to A: We always have $\forall A_S \models A$, but $A \not\models A_S$ for certain formulas A. For example, consider the formula $\exists x P(x)$ with Skolem normal form $P(a)$. The structure $\langle \{0, 1\}, \mathcal{I} \rangle$ with $\mathcal{I}(P) = \{0\}$ and $\mathcal{I}(a) = 1$ is a model of $\exists x P(x)$, but it is not a model of $P(a)$.
Since not every formula has a logically equivalent Skolem normal form, one sometimes speaks of a Skolem *standard* form.

However, equi-satisfiability holds:

Theorem 2.11 *It is $\forall A$ satisfiable under an interpretation iff $\forall A_S$ is satisfiable in a suitably extended interpretation, in which the constants and function signs introduced by Skolemization are interpreted.*

Proof. See Nienhuys-Cheng & de Wolf, 1997 (ch. 3) or Doets, 1994 (ch. 3), where only closed formulas A are considered. QED

Corollary 2.12 (i) *In order to show that $\forall A$ is unsatisfiable, that is, to refute $\forall A$, it is sufficient to refute $\forall A_S$.*

(ii) *Let $C_1 \wedge \ldots \wedge C_m$ be a CNF of A_S, that is, the conjunctive normal form of the Skolem normal form (*conjunctive Skolem normal form *for short) of the initial formula A. In order to show that $\forall A$ is unsatisfiable it is sufficient to refute $\forall C_1 \wedge \ldots \wedge \forall C_m$.*

The conjunctive Skolem normal form $C_1 \wedge \ldots \wedge C_m$ of A can be written as a *set of clauses* (just as in propositional logic); we denote this set of clauses by A_{SC} (*conjunctive Skolem normal form in clause form*). *set of clauses*

Clauses are again defined as sequents $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$, where the atomic formulas $A_1, \ldots, A_n, B_1, \ldots, B_m$ are now first-order atoms. *clause*

If one defines a *refutation procedure* for clauses in such a way that free variables are understood universally (see Section 2.3), then in order to show that $\forall A$ is unsatisfiable it is sufficient to refute A_{SC} with this procedure; to show the universal validity of a closed formula A it is sufficient to refute $(\neg A)_{SC}$ with this procedure. *refutation procedure*

For an arbitrary first-order formula A the following three steps have thus to be carried out:

(1) Construct a Skolem normal form for $\neg A$.

(2) Transform it into a CNF.

(3) Apply the refutation procedure.

or

(1) Construct a prenex normal form for $\neg A$ with kernel in CNF.

(2) Skolemize.

(3) Apply the refutation procedure.

For claims of logical consequences $A_1, \ldots, A_n \vDash A$ this means:

(1) Construct $(A_1)_{\mathrm{SC}}, \ldots, (A_n)_{\mathrm{SC}}$ and $(\neg A)_{\mathrm{SC}}$.

Here the terms introduced by the respective Skolemizations have to be chosen in such a way that the sets of these terms for $(A_1)_{\mathrm{SC}}, \ldots, (A_n)_{\mathrm{SC}}$ and $(\neg A)_{\mathrm{SC}}$ are pairwise disjoint.

(2) Apply the refutation procedure.

Examples. (The Skolem normal form is constructed according to Definition 2.10 in each case.)

(i) $\vDash \exists x \forall y P(x, y) \to \forall y \exists x P(x, y)$

$$
\begin{array}{ll}
\neg(\exists x \forall y P(x, y) \to \forall y \exists x P(x, y)) & \text{(negation of the initial formula)} \\
\rightsquigarrow \neg(\exists x \forall y P(x, y) \to \forall z \exists u P(u, z)) & \text{(renaming of bound variables)} \\
\rightsquigarrow \exists x \forall y P(x, y) \wedge \exists z \forall u \neg P(u, z) & \text{(move negation inward)} \\
\rightsquigarrow \exists x \forall y \exists z \forall u (P(x, y) \wedge \neg P(u, z)) & \text{(PNF with kernel in CNF)} \\
\rightsquigarrow \forall y \exists z \forall u (P(a, y) \wedge \neg P(u, z)) & \text{(skolemize ...)} \\
\rightsquigarrow \exists z \forall u (P(a, y) \wedge \neg P(u, z)) & \\
\rightsquigarrow \forall u (P(a, y) \wedge \neg P(u, f(y))) & \\
\rightsquigarrow P(a, y) \wedge \neg P(u, f(y)) & \text{(Skolem normal form in CNF)} \\
\rightsquigarrow \{ \vdash P(a, y) \,;\, P(u, f(y)) \vdash \} & \text{(set of clauses)}
\end{array}
$$

(ii) $\exists x \forall y P(x, y) \vDash \forall y \exists x P(x, y)$

$$\exists x \forall y P(x, y) \rightsquigarrow \forall y P(a, y) \rightsquigarrow P(a, y)$$

$$\neg \forall y \exists x P(x, y) \rightsquigarrow \exists y \forall x \neg P(x, y) \rightsquigarrow \forall x \neg P(x, b) \rightsquigarrow \neg P(x, b)$$

One obtains the set of clauses $\{ \vdash P(a, y) \,;\, P(x, b) \vdash \}$.

The Import-Export Theorem 1.11 holds for first-order logic, too; thus the claim of universal validity in (i) holds iff the claimed logical consequence in (ii) holds. Nonetheless, the resulting sets of clauses differ: In (ii) the constant b occurs instead of the function sign f. This is not a problem, however, since the free variables in each clause are understood universally.

Although both sets of clauses are unsatisfiable, a refutation using only the propositional resolution rule is not possible, since $P(a, y)$ and $P(u, f(y))$, respectively $P(a, y)$ and $P(x, b)$, are in both cases two different formulas. (The fact that these are not propositional but first-order formulas is secondary here.)

2.3 Unification and first-order resolution

In order to be able to obtain a resolution refutation also for unsatisfiable sets of clauses like those in the last example, one has to identify atoms A and B, for which $\{A, \neg B\}$ is unsatisfiable, in a suitable way by using *unification*. This is achieved by giving an instance A' of A and an instance B' of B such that A' and B' are syntactically identical.

Definition 2.13 (i) If the expression (i.e., the formula or the term) E' is an instance of the expression E (i.e., $E' = E\sigma$, for a substitution σ), then E is *more general* than E'. *more general*

(ii) For two substitutions σ, τ, we call σ *more general* than τ, if there is a substitution ϑ such that $\sigma\vartheta = \tau$.

(iii) It is $\mathrm{dom}(\vartheta) := \{x \mid x\vartheta \neq x\}$ and $\mathrm{ran}(\vartheta) := \{y_i \mid y_i \text{ occurs in } x_i\vartheta, \text{ where } x_i \in \mathrm{dom}(\vartheta)\}$.

(iv) A substitution ϑ is a *variable substitution*, if all $x_i\vartheta$ for $x_i \in \mathrm{dom}(\vartheta)$ are variables. *variable substitution*

(v) A variable substitution ϑ is a *renaming*, if ϑ is a bijective mapping of the variables. (A renaming is thus a permutation of variables.) *renaming*

(vi) If ϑ is a renaming, then $E\vartheta$ is called a *variant* of E. *variant*

Examples. (i) It is x more general than $g(a, h(c))$, since for $[x/g(a, h(c))]$ the term $g(a, h(c))$ is an instance of x.

(ii) It is $f(x, y)$ more general than $f(x, x)$, since $f(x, y)[y/x] = f(x, x)$.

(iii) $[x/y]$ is more general than $[x/a, y/a]$, since $[x/y][y/a] = [x/a, y/a]$.

(iv) It is σ more general than σ for any substitution σ, since $\sigma\varepsilon = \sigma$ for the empty substitution ε.

(v) $[x/y]$ is *not* more general than $[x/a]$.

Suppose there were a substitution ϑ such that the binding x/a is contained in $[x/y]\vartheta$. Then y/a must be contained in ϑ, and thus $y \in \mathrm{dom}([x/y]\vartheta)$. Consequently, there can be no substitution ϑ with $[x/y]\vartheta = [x/a]$.

(vi) $[x/z, y/z]$ is a variable substitution.

(vii) $[x/z, z/y, y/x]$ is a renaming.

The empty substitution ε is a renaming, too.

(viii) It is $f(x, y)$ a variant of $f(y, x)$, since $f(y, x)[y/x, x/y] = f(x, y)$.

(ix) It is $f(x, z)$ a variant of $f(x, y)$, since $f(x, y)[y/z, z/y] = f(x, z)$.

The binding z/y is needed; otherwise the given substitution would not be a renaming according to our definition.

(x) The term $f(x, x)$ is *not* a variant of $f(x, y)$.

If it were a variant, then $f(x, y)\vartheta = f(x, x)$ would have to hold for a renaming ϑ with binding y/x. Since ϑ is a renaming, ϑ must also contain a binding x/y with $x \neq y$. But then $f(x, y)\vartheta = f(y, x) \neq f(x, x)$.

Definition 2.14 (i) A substitution σ is a *unifier* of two atoms A and B, if the following holds: $A\sigma = B\sigma$ (i.e., if the two expressions $A\sigma$ and $B\sigma$ are syntactically identical). *unifier*

(ii) If there exists a unifier σ of A and B, then the atoms A and B are called *unifiable*. *unifiable*

If $\Gamma = \{A, B\}$ for two atoms A and B with unifier σ, then we also say that σ unifies the set Γ.

Example. The two atoms $P(a, x)$ and $P(y, b)$ are unifiable: the substitution $\sigma = [x/b, y/a]$ is a unifier, since $P(a, x)\sigma = P(y, b)\sigma$.

Definition 2.15 The *first-order resolution rule* is

first-order resolution rule

$$\frac{X_1 \vdash Y_1, A \qquad B, X_2 \vdash Y_2}{(X_1, X_2 \vdash Y_1, Y_2)\sigma}\,(\mathcal{R})$$

where σ is a unifier for A and B, and where the sets of free variables of the two premisses have to be disjoint.

Example. The two clauses $Q(x) \vdash P(a,x)$ and $P(y,b) \vdash R(y)$ have no variables in common, and the substitution $[x/b, y/a]$ unifies $P(a,x)$ and $P(y,b)$. Hence

$$[x/b, y/a]\,\frac{Q(x) \vdash P(a,x) \qquad P(y,b) \vdash R(y)}{Q(b) \vdash R(a)}\,(\mathcal{R})$$

is a correct application of the first-order resolution rule. (We noted the used unifier besides the rule bar.)

The free variables in the premisses are understood universally; arbitrary terms may thus be substituted for them. If σ is a substitution which unifies A and B, then the application of the unifier σ to the formulas in the first-order resolution rule leads to the propositional resolution rule (by form):

$$\begin{array}{ccc} X_1 \vdash Y_1, A & & B, X_2 \vdash Y_2 \\ \Big\downarrow \sigma & & \Big\downarrow \sigma \\ \dfrac{X_1\sigma \vdash Y_1\sigma, A\sigma(=B\sigma) \qquad B\sigma(=A\sigma), X_2\sigma \vdash Y_2\sigma}{X_1\sigma, X_2\sigma \vdash Y_1\sigma, Y_2\sigma} & & \end{array}$$

(Since the formulas are in a first-order language, one does not obtain the propositional resolution rule in the strict sense, as this rule was only defined for propositional formulas. But this is not essential here.)

Example. For $\sigma = [x/b, y/a]$ being a unifier for $P(a,x)$ and $P(y,b)$ one obtains:

$$\begin{array}{cc} Q(x) \vdash P(a,x) & P(y,b) \vdash R(y) \\ \Big\downarrow \sigma & \Big\downarrow \sigma \\ \dfrac{Q(b) \vdash P(a,b) \qquad P(a,b) \vdash R(a)}{Q(b) \vdash R(a)}\,(\mathcal{R}) & \end{array}$$

Since free variables are understood universally, the restriction to disjoint sets of free variables of the two premisses in $(\mathcal{R})$ is not a restriction in generality. Disjoint sets can always be obtained by suitable substitutions for the free variables that *separate* the free variables in different clauses. (Compare also Corollary 2.12 (ii).)

separation of free variables

For example, if one considers the two clauses

$$\vdash P(a), Q(x) \quad \text{and} \quad P(a) \vdash R(x)$$

then one obtains by (incorrect) resolution

$$\frac{\vdash P(a), Q(x) \qquad P(a) \vdash R(x)}{\vdash Q(x), R(x)}\;\lightning$$

the clause $\vdash Q(x), R(x)$ as resolvent; the resolution step is incorrect, since x occurs in both premisses as a free variable. If one first separates the free variables instead, for example by the renaming

$$P(a) \vdash R(x)[x/y] = P(a) \vdash R(y)$$

then one obtains the clause $\vdash Q(x), R(y)$ as resolvent. The two resolvents are not logically equivalent, since

$$Q(x) \vee R(y) \models Q(x) \vee R(x) \quad \text{but} \quad Q(x) \vee R(x) \not\models Q(x) \vee R(y).$$

The problem consists in the fact that the variable x occurs in both initial clauses, although the two clauses are independent. Hence, we are not really dealing with two occurrences of the same variable x, but rather with two different variables that have the same name. This distinction of two variables is, however, lost in the above (incorrect) resolution step. In order to obtain the most general resolvent in each case, one always has to first separate the variables occurring in the premisses by a renaming.

Examples. (i) We can now continue the example (i) resp. (ii) from p. 18:

$$\{ \vdash P(a, y) \,;\, P(u, f(y)) \vdash \} \overset{\text{separation of variables}}{\rightsquigarrow} \{ \vdash P(a, y) \,;\, P(u, f(v)) \vdash \}$$

Resolution refutation:

$$[y/f(v), u/a] \dfrac{\vdash P(a, y) \qquad P(u, f(v)) \vdash}{\vdash} (\mathcal{R})$$

Respectively for $\{ \vdash P(a, y) \,;\, P(x, b)) \vdash \}$:

$$[x/a, y/b] \dfrac{\vdash P(a, y) \qquad P(x, b) \vdash}{\vdash} (\mathcal{R})$$

(ii) $\exists x(P(x) \wedge \neg Q(x)), \forall x(P(x) \to R(x)) \models \exists x(R(x) \wedge \neg Q(x))$

$$\begin{array}{lll} \exists x(P(x) \wedge \neg Q(x)) & \forall x(P(x) \to R(x)) & \neg\exists x(R(x) \wedge \neg Q(x)) \\ \rightsquigarrow \{Q(a) \vdash \,;\, \vdash P(a)\} & \rightsquigarrow \forall x(\neg P(x) \vee R(x)) & \rightsquigarrow \forall x(\neg R(x) \vee Q(x)) \\ & \rightsquigarrow \{P(x) \vdash R(x)\} & \rightsquigarrow \{R(y) \vdash Q(y)\} \end{array}$$

One can skolemize separately, because the variables can be separated.

One obtains the set of clauses $\{Q(a) \vdash \,;\, \vdash P(a) \,;\, P(x) \vdash R(x) \,;\, R(y) \vdash Q(y)\}$.

Resolution refutation:

$$\dfrac{[y/a] \dfrac{\vdash P(a) \qquad [x/y] \dfrac{P(x) \vdash R(x) \qquad R(y) \vdash Q(y)}{P(y) \vdash Q(y)} (\mathcal{R})}{\varepsilon \dfrac{\vdash Q(a) \qquad\qquad Q(a) \vdash}{\vdash} (\mathcal{R})}$$

The first-order resolution rule $(\mathcal{R})$ alone is not sufficient to, for example, derive from the two clauses

$$\vdash P(x), P(y) \quad \text{and} \quad P(z), P(u) \vdash$$

the empty clause, although the corresponding set

$$\{P(x) \vee P(y), \neg P(z) \vee \neg P(u)\}$$

is unsatisfiable. Each resolution step results in a clause with two atoms, for example

$$[y/z]\,\frac{\vdash P(x), P(y) \qquad P(z), P(u) \vdash}{P(u) \vdash P(x)}\,(\mathcal{R})$$

Since in each resolution step the sets of free variables of the two premisses have to be disjoint, one could only continue with suitable variants; for example as follows:

$$[y/u]\,\frac{\vdash P(v), P(y) \qquad [y/z]\,\dfrac{\vdash P(x), P(y) \qquad P(z), P(u) \vdash}{P(u) \vdash P(x)}\,(\mathcal{R})}{\vdash P(v), P(u)}\,(\mathcal{R})$$

where $\vdash P(v), P(u)$ is a variant of the first clause $\vdash P(x), P(y)$. But this also only leads to a clause with two atoms, which in this case has the form of the first initial clause. Likewise, one obtains only clauses with two atoms in all other cases.
In order to obtain a sound resolution procedure, one needs to be able to form so-called *factors* of clauses in addition.

Definition 2.16 A *factor* S' of a clause S is the result of an application of a substitution to S such that several atoms in S get unified. *factor*

We enable the derivation of factors by adding the following rules.

Definition 2.17 The *factoring rule* $(\mathcal{F})$ is given as a pair of rules for factorizable atoms A, B in the antecedent or succedent, respectively, as follows: *factoring rule*

$$\frac{X, A, B \vdash Y}{(X, A \vdash Y)\sigma}\,(\mathcal{F}) \qquad\qquad \frac{X \vdash A, B, Y}{(X \vdash A, Y)\sigma}\,(\mathcal{F})$$

where σ is a unifier for A and B.

The factoring rule is a substitution rule, in whose application A and B are identified by unification. This is justified by the fact that variables in clauses are understood universally.

Example. By using factoring we can obtain a resolution refutation of

$$\{\vdash P(x), P(y)\,;\; P(z), P(u) \vdash\}$$

as follows:

$$[x/z]\,\frac{[y/x]\,\dfrac{\vdash P(x), P(y)}{\vdash P(x)}\,(\mathcal{F}) \qquad [u/z]\,\dfrac{P(z), P(u) \vdash}{P(z) \vdash}\,(\mathcal{F})}{\vdash}\,(\mathcal{R})$$

It is also possible to combine $(\mathcal{R})$ and $(\mathcal{F})$ in a single rule by allowing to unify and resolve more than one pair of formulas in one step.

Definition 2.18 The *generalized resolution rule* $(\mathcal{R}+\mathcal{F})$ is: *generalized resolution rule*

$$\frac{X_1 \vdash Y_1, A, B \qquad C, D, X_2 \vdash Y_2}{(X_1, X_2 \vdash Y_1, Y_2)\sigma}\,(\mathcal{R}+\mathcal{F})$$

if σ unifies the set $\{A, B, C, D\}$.

Example. For the set of clauses given in the last example one can derive the empty clause by just one application of the generalized resolution rule:

$$[y/x, u/z][z/x]\,\frac{\vdash P(x), P(y) \qquad P(z), P(u) \vdash}{\vdash}\,(\mathcal{R}+\mathcal{F})$$

2.4 Most general unifiers and the unification algorithm

In order to obtain completeness of resolution, one has to use *most general unifiers*. These can be computed by the *unification algorithm*.

Definition 2.19 A substitution σ is a *most general unifier* (mgu for short) of A and B, if for all unifiers τ of A and B we have $\tau = \sigma\rho$, for a substitution ρ. *most general unifier*

Examples. (i) For $P(f(x, g(a, y)))$ and $P(f(b, z))$ the substitution $[x/b, z/g(a, y)]$ is a most general unifier.

For the given atoms, $[x/b, y/c, z/g(a, c)]$ is a unifier, but it is *not* a most general unifier, since $[x/b, y/c, z/g(a, c)] = [x/b, z/g(a, y)][y/c]$.

(ii) For the set $\{P(x), P(y)\}$ both $[x/y]$ and $[y/x]$ are most general unifiers.

Hence, most general unifiers are not determined uniquely.

(iii) The substitution $\sigma = [x/z, y/z]$ is *not* an mgu for $\{P(x), P(y)\}$, since there is no substitution ρ such that $[x/y] = \sigma\rho$.

In particular, $[z/y]$ is no such substitution ρ, since $\sigma[z/y] = [x/y, z/y] \neq [x/y]$.

Theorem 2.20 *Multiple most general unifiers of A and B differ only by renaming.*

Proof. See Lassez, Maher & Marriot, 1987, p. 605. QED

Definition 2.21 Let Γ be a finite set of terms or atoms. Then the *difference set* U of Γ is defined as follows: *difference set*

Find the leftmost position at which not all expressions in Γ have the same symbol, and select from each expression in Γ the subexpression which begins at this position.

The set of all these subexpressions is the difference set.

Examples. (i) Let $\Gamma = \{P(a, \underline{f(x, y)}, g(z)), P(a, \underline{z}, h(u)), P(a, \underline{x}, y)\}$.

(The relevant subexpressions are underlined.)

Then the difference set is $U = \{f(x, y), z, x\}$.

(ii) The difference set for $\Gamma = \{\underline{Q(x)}, \underline{R(x, y)}\}$ is $U = \{Q(x), R(x, y)\}$.

We can restrict ourselves to sets of two elements $\Gamma = \{A, B\}$ for atoms A, B. For these, the following *unification algorithm* creates an mgu, if A and B are unifiable. *unification algorithm*

Unification algorithm

Input: A set $\Gamma = \{A, B\}$ of two atoms A and B.

Output: An mgu for Γ, if A and B are unifiable.

(1) Set $n = 0$ and $\sigma_0 = \varepsilon$, where ε is the empty substitution.

(2) If $\Gamma\sigma_n$ is a singleton, then halt; σ_n is an mgu for Γ.

Otherwise form the difference set U_n of $\Gamma\sigma_n$.

(3) If there is a variable x and a term t in U_n such that the variable x does not occur in t, then set $\sigma_{n+1} = \sigma_n[x/t]$ (that is, form the composition $\sigma_n[x/t]$ with the preceding substitution σ_n), increment n by 1, and go to (2).

Otherwise stop with the output that Γ is not unifiable.

For applications of the unification algorithm in resolution the input $\Gamma = \{A, B\}$ has the property $\mathrm{FV}(A) \cap \mathrm{FV}(B) = \emptyset$; this is the case, since in a resolution step we demand that the sets of free variables of the two premisses are disjoint.
For inputs $\Gamma = \{A, B\}$ with $\mathrm{FV}(A) \cap \mathrm{FV}(B) \neq \emptyset$ one can first separate the variables occurring in the atoms A and B. Such a separation is justified (as for clauses) by the fact that A and B can always be understood as being independent.

Remark. The check in step (3) whether x occurs in t is called *occur check*. Without it the algorithm would not terminate in each case. For example, *without* the occur check the algorithm would yield the following for the non-unifiable set $\Gamma = \{P(\underline{x}, x), P(\underline{y}, f(x))\}$: *occur check*

(1) $\sigma_0 = \varepsilon$

(2) $U_0 = \{x, y\}$

(3) $\sigma_1 = [x/y]$, $\Gamma\sigma_1 = \{P(y, \underline{y}), P(y, \underline{f(y)})\}$

(2) $U_1 = \{y, f(y)\}$

(3) The variable y occurs in the term $f(y)$! Without occur check one obtains:

$\sigma_2 = [x/y][y/f(y)] = [x/f(y), y/f(y)]$,

$\Gamma\sigma_2 = \{P(f(y), f(\underline{y})), P(f(y), f(\underline{f(y)}))\}$

(2) $U_2 = \{y, f(y)\}$

(3) The variable y occurs in the term $f(y)$! Without occur check one obtains:

$\sigma_3 = [x/f(y), y/f(y)][y/f(y)] = [x/f(f(y)), y/f(f(y))]$,

$\Gamma\sigma_3 = \{P(f(f(y)), f(f(\underline{y}))), P(f(f(y)), f(f(\underline{f(y)})))\}$

$\vdots$

Hence, the algorithm would not terminate; instead it would always set $\sigma_{n+1} = \sigma_n[y/f(y)]$ in step (3).

Examples. (i) $\Gamma = \{P(\underline{f(x)}, g(y)), P(\underline{z}, z)\}$

(1) $\sigma_0 = \varepsilon$

(2) $U_0 = \{f(x), z\}$

(3) $\sigma_1 = [z/f(x)]$, $\Gamma\sigma_1 = \{P(f(x), \underline{g(y)}), P(f(x), \underline{f(x)})\}$

(2) $U_1 = \{g(y), f(x)\}$

(3) The difference set U_1 contains no variable as an element; thus $P(f(x), g(y))$ and $P(z, z)$ are not unifiable.

(ii) $\Gamma = \{P(\underline{f(x)}, y), P(\underline{z}, a)\}$

(1) $\sigma_0 = \varepsilon$

(2) $U_0 = \{f(x), z\}$

(3) $\sigma_1 = [z/f(x)]$, $\Gamma\sigma_1 = \{P(f(x), \underline{y}), P(f(x), \underline{a})\}$

(2) $U_1 = \{y, a\}$

(3) $\sigma_2 = [z/f(x)][y/a]$, $\Gamma\sigma_2 = \{P(f(x), a), P(f(x), a)\} = \{P(f(x), a)\}$

(2) $\Gamma\sigma_2$ is a singleton; thus Γ is unifiable with mgu $\sigma_2 = [z/f(x)][y/a]$.

(iii) $\Gamma = \{P(\underline{x}, x), P(\underline{y}, f(y))\}$

(1) $\sigma_0 = \varepsilon$

(2) $U_0 = \{x, y\}$

(3) $\sigma_1 = [x/y]$, $\Gamma\sigma_1 = \{P(y, \underline{y}), P(y, \underline{f(y)})\}$

(2) $U_1 = \{y, f(y)\}$

(3) The variable y occurs in $f(y)$ (occur check); thus $P(x, x)$ and $P(y, f(y))$ are not unifiable.

(iv) $\Gamma = \{P(\underline{a}, x, h(g(z))), P(\underline{u}, h(y), h(y))\}$

(1) $\sigma_0 = \varepsilon$

(2) $U_0 = \{a, u\}$

(3) $\sigma_1 = [u/a]$, $\Gamma\sigma_1 = \{P(a, \underline{x}, h(g(z))), P(a, \underline{h(y)}, h(y))\}$

(2) $U_1 = \{x, h(y)\}$

(3) $\sigma_2 = [u/a][x/h(y)]$, $\Gamma\sigma_2 = \{P(a, h(y), h(\underline{g(z)})), P(a, h(y), h(\underline{y}))\}$

(2) $U_2 = \{g(z), y\}$

(3) $\sigma_3 = [u/a][x/h(y)][y/g(z)]$, $\Gamma\sigma_3 = \{P(a, h(g(z)), h(g(z)))\}$

(2) $\Gamma\sigma_3$ is a singleton; thus Γ is unifiable with mgu $\sigma_3 = [u/a][x/h(y)][y/g(z)] = [u/a, x/h(g(z)), y/g(z)]$.

Definition 2.23 A substitution ϑ is called *idempotent*, if $\vartheta\vartheta = \vartheta$. *idempotent*

Lemma 2.24 *Let ϑ be a unifier for A and B. Then the following statements are equivalent:*

(i) *It is ϑ an idempotent most general unifier of A and B.*

(ii) *For each unifier σ of A and B we have $\vartheta\sigma = \sigma$.*

Proof. (i) $\Longrightarrow$ (ii). Let ϑ be an idempotent most general unifier for A and B, and σ a unifier for A and B. Then there is a substitution ρ such that $\sigma = \vartheta\rho$. Therefore $\vartheta\sigma = \vartheta(\vartheta\rho) = (\vartheta\vartheta)\rho = \vartheta\rho = \sigma$.

(ii) $\Longrightarrow$ (i). Suppose $\vartheta\sigma = \sigma$ holds for any unifier σ of A and B. Then it holds in particular for the most general unifier ϑ that $\vartheta\vartheta = \vartheta$, that is, ϑ is idempotent. QED

Not every most general unifier is also idempotent. For the identical atoms $P(a)$ and $P(a)$ the substitution $[x/y, y/x]$ is trivially an mgu, but $[x/y, y/x][x/y, y/x] = \varepsilon \neq [x/y, y/x]$.

Lemma 2.25 *A substitution ϑ is idempotent iff* $\mathrm{dom}(\vartheta) \cap \mathrm{ran}(\vartheta) = \emptyset$.

Proof. Exercise. QED

Remark. By checking that $\mathrm{dom}(\vartheta) \cap \mathrm{ran}(\vartheta) = \emptyset$ one easily finds that $[x/u, z/f(u,v), y/v]$ is idempotent, while $[x/u, z/f(u,v), v/y]$ is not.

Theorem 2.26 (Correctness of the unification algorithm)
Let A and B be two atoms. Then the following holds:

(i) *The unification algorithm always terminates.*

(ii) *If A and B are unifiable, then the unification algorithm computes a most general unifier for A and B.*

(iii) *If A and B are not unifiable, then the unification algorithm terminates with the output that A and B are not unifiable.*

Proof. See Appendix A.3. QED

Definition 2.27 A unifier ϑ of two atoms A and B is called *relevant*, if *relevant*

$$\mathrm{FV}(\vartheta) \subseteq \mathrm{FV}(A) \cup \mathrm{FV}(B)$$

(i.e., if ϑ does not introduce any new variables that neither occur in A nor in B).

The unification algorithm obviously yields a relevant unifier for A and B, if A and B are unifiable. Note that in step (3) only such substitutions are added that contain variables occurring in A or B.

Lemma 2.28 *Let ϑ be an idempotent most general unifier for A and B. Then ϑ is relevant.*

Proof. See Apt, 1997, p. 38 (Theorem 2.22). QED

The reverse direction does not hold, that is, not every relevant mgu is also idempotent. The substitution $\vartheta = [x/f(z,y), y/z, z/y]$ is a relevant mgu of $\{x, f(y,z)\}$; however, it is not idempotent, since $\vartheta\vartheta = [x/f(y,z)] \neq \vartheta$.

Due to the unification algorithm the following holds:

(i) Unifiability of two atoms is decidable.

(ii) If two atoms have a unifier, then they also have a most general unifier.

(iii) Most general unifiers can be computed.

(iv) If two atoms are unifiable, then they have an idempotent most general unifier.

(v) The idempotent most general unifier is also relevant.

3 SLD-resolution and logic programming

Logic programming in the strict sense is based on a restricted form of resolution, namely *SLD-resolution*. In SLD-resolution the language is further restricted to so-called *Horn clauses* (named after Alfred Horn, 1918–2001) that may contain any number of negative literals but *at most one* positive literal. In addition, each resolution step may contain as premisses only one Horn clause with no positive literal and one Horn clause with exactly one positive literal. The latter has to be chosen as a variant from a given set of clauses, the so-called *logic program*. These restrictions lead to more simple resolution derivations. Nevertheless, the Horn clause fragment of first-order logic is undecidable (like first-order logic itself). Moreover, unifiers computed in a derivation become more important; they represent answers to queries that are addressed to the respective logic program. Logic programming in the sense of SLD-resolution is computationally adequate, that is, any partial recursive function can be computed by a logic program, and any function definable by a logic program is partial recursive.

3.1 Logic programs and SLD-resolution

Definition 3.1 (i) A *Horn clause* is a clause with at most one positive literal, that is, a clause of the form $X \vdash$ or $X \vdash A$ (where A is a positive literal, and where the set X of atoms may also be empty). *Horn clause*

A clause with exactly one positive literal is called a *definite Horn clause*. *definite Horn clause*

Using the *reverse implication* '$\leftarrow$' one also writes $\leftarrow X$ or $A \leftarrow X$, respectively. *reverse implication*

Motivation for the notation: $X \vdash A$ can be read as "A, if X", $X \vdash$ as "X is false" (or as "absurdity, if X").

(ii) A clause of the form $\leftarrow X$ is also called *goal clause* (*goal* for short) or *query*. *goal clause*

The empty goal is denoted by $\leftarrow$.

(iii) A clause of the form $A \leftarrow X$ is also called a *rule*; A is its *head*, and X is its *body*. *rule*

A clause of the form $A \leftarrow$ is also called a *fact*.

(iv) *Program clauses* are rules or facts (i.e., definite Horn clauses). *program clause*

(v) A *logic program* (*program* for short) Π is a finite set of program clauses. *logic program*

Definition 3.2 A (non-restricted) *SLD-resolution step* has the form *SLD-resolution step*

$$\frac{\leftarrow X, A \qquad B \leftarrow Y}{(\leftarrow X, Y)\sigma}\,\sigma$$

where σ is a unifier for A and B, and the sets of free variables of the two premisses have to be disjoint.

Remarks. (i) In this definition, σ just has to be a unifier for A and B, and not necessarily a most general unifier. The SLD-resolution step is in this sense non-restricted. However, completeness can only be obtained, if in each step most general unifiers are used.

We presuppose in the following that we always use the unification algorithm to find σ, which ensures that σ is a most general unifier.

(ii) In view of the *selection function* to be discussed below (see Section 3.3) we will view the bodies of clauses as lists of atoms instead of sets of atoms. That is, the order and multiplicity of atoms becomes relevant.

Definition 3.3

(i) An *SLD-derivation* of a goal G from a program Π consists of — *SLD-derivation*

- a series $G_0, G_1, \ldots, G_n$ of goals,
- a series $S_1, \ldots, S_n$ of variants of clauses in Π,
- a series $\vartheta_1, \ldots, \vartheta_n$ of substitutions,

such that $G_0 = G$, and for all $i < n$ the following holds (where the lists of atoms X_i, Y_i and Z_{i+1} can also be empty):

- G_i is of the form $\leftarrow X_i, A_i, Y_i$; this includes the case $\leftarrow$.
- S_{i+1} is of the form $B_{i+1} \leftarrow Z_{i+1}$.
- S_{i+1} has no variables in common with G_i or $G_0\vartheta_1 \ldots \vartheta_i$.
- ϑ_{i+1} is an mgu of A_i and B_{i+1}.
- $G_{i+1} = (\leftarrow X_i, Z_{i+1}, Y_i)\vartheta_{i+1}$.

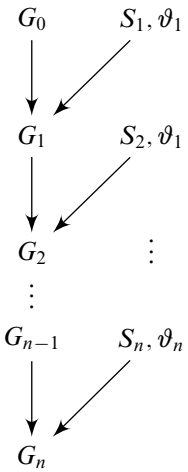
(ii) The number $n + 1$ is called the *length* of the SLD-derivation. — *length*

(iii) The transition from a goal G_i to a new goal G_{i+1} is an *SLD-resolution step*. — *SLD-resolution step*

(iv) The clause S_i is called i-th *input clause*. — *input clause*

(v) The atom A_i is called *selected atom*. — *selected atom*

An SLD-derivation of length $n + 1$ has the following form:



But SLD-derivations can also be infinite.

Remark. In a resolution step from G_i to G_{i+1} the following happens:

(1) In the goal G_i an atom A_i is selected.

(2) A clause is chosen from the program Π.

(3) By a separation of variables (if necessary) one obtains from the chosen program clause the variant S_{i+1}; this is the new input clause $B_{i+1} \leftarrow Z_{i+1}$.

(4) If there exists an mgu ϑ_{i+1} for the head B_{i+1} of the clause S_{i+1} and the selected atom A_i, then A_i is replaced by the body Z_{i+1} of S_{i+1}, and ϑ_{i+1} is applied to the new goal G_{i+1}.

Definition 3.4 Let a logic program Π and a goal $\leftarrow X$ be given. An *SLD-refutation of* $\leftarrow X$ *relative to* Π is an SLD-derivation that starts with $\leftarrow X$ as left uppermost assumption, uses only (variants of) clauses from Π as further assumptions, and ends with the empty clause $\leftarrow$. *SLD-refutation*

Definition 3.5 (i) An SLD-derivation is called *successful*, if it is an SLD-refutation, that is, if G_n is the empty clause $\leftarrow$. *successful*

(ii) The composition $\vartheta_1 \ldots \vartheta_n$ of the most general unifiers in a successful SLD-derivation can be restricted to $\mathrm{FV}(G_0)$. This composition is then called *computed answer substitution* for the goal G_0, and $G_0\vartheta_1 \ldots \vartheta_n$ is called *computed instance* of G_0. *computed answer substitution*

Let V be a set of variables. Then $\vartheta \mid V$ denotes the *restriction* of ϑ to the variables in V. *restriction to variables*

(iii) An SLD-derivation is called *failed*, if G_n is not the empty clause, and no (variant of a) clause from Π is applicable for the atom selected in G_n. *failed*

Definition 3.6 An *SLD-proof* of X from Π is an SLD-refutation of $\leftarrow X$ relative to Π. *SLD-proof*

Remarks. (i) If one writes down the SLD-resolution steps according to Definition 3.2, then an *SLD-proof* of X_1 from Π has the following form:

$$\dfrac{\dfrac{\leftarrow X_1 \qquad A_1 \leftarrow Y_1}{\leftarrow X_2}\,\sigma_1 \qquad\qquad A_2 \leftarrow Y_2}{\leftarrow X_3}\,\sigma_2$$

$$\vdots$$

$$\leftarrow$$

The substitutions $\sigma_1, \sigma_2, \ldots$ are the unifiers computed in the respective resolution steps. (These are most general unifiers, if the unification algorithm is used.)

(ii) The abbreviation SLD stands for the following:

S: A *selection function* is used to select an atom in the current goal.

L: The form of the derivation is *linear*.
This becomes particularly clear, if the SLD-derivation is written down as follows:

$$G_0 \xrightarrow{S_1, \vartheta_1} G_1 \xrightarrow{S_2, \vartheta_2} G_2 \quad \cdots \quad G_{n-1} \xrightarrow{S_n, \vartheta_n} G_n \quad \cdots$$

D: Logic programs consist of *definite* Horn clauses.

In our first-order languages $\mathscr{L}$ we we also use 'speaking' relation symbols like *add*, function signs like *s* (for *successor*) and constants like 0, in order to indicate their

intended interpretation. In doing so, we just extend the alphabet; the new symbols are not introduced with any fixed meaning, however. Their meaning is only given operationally with respect to SLD-resolution.
Note that we cannot implement an n-ary function directly by using only function terms. Instead, we have to use $n+1$-ary relation symbols that are used to define the graph of the n-ary function. For example, in the case of addition we cannot specify $f(x, y) = x + y$; instead, we have to define a relation (symbol) $add(x, y, z)$ such that $x + y = z$ (in case we choose z to be the value of the addition function with arguments x and y).

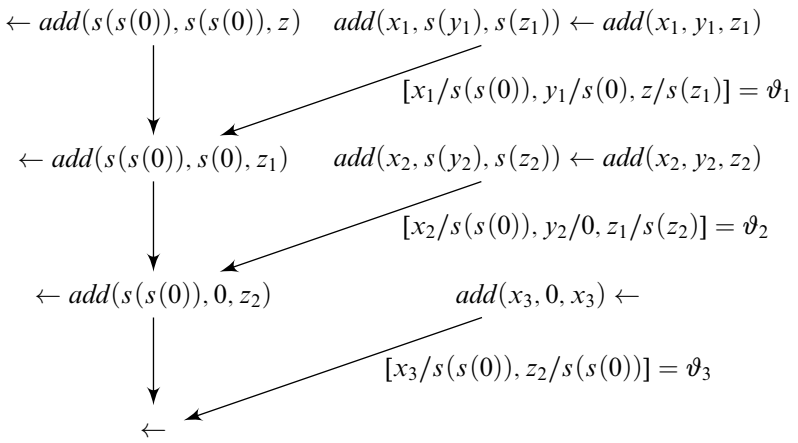
Example. We consider the logic program Π_{add} for addition. It consists of two program clauses, namely a fact S_1 and a rule S_2:

$$S_1. \quad add(x, 0, x) \leftarrow$$
$$S_2. \quad add(x, s(y), s(z)) \leftarrow add(x, y, z)$$

An SLD-refutation of the goal

$$\leftarrow add(s(s(0)), s(s(0)), z)$$

relative to the program Π_{add} is then for example:

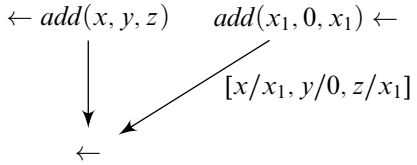


We now restrict the composition $\vartheta_1\vartheta_2\vartheta_3$ of the most general unifiers to the variables occurring in the goal $\leftarrow add(s(s(0)), s(s(0)), z)$ (i.e., to z) to obtain the computed answer substitution:

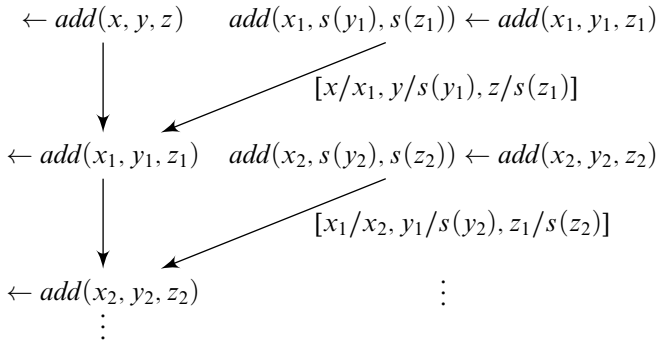
$$\begin{aligned}
&\vartheta_1\vartheta_2\vartheta_3 \mid \mathrm{FV}(\leftarrow add(s(s(0)), s(s(0)), z)) \\
&= \vartheta_1\vartheta_2\vartheta_3 \mid \{z\} \\
&= [x_1/s(s(0)), y_1/s(0), z/s(z_1)][x_2/s(s(0)), y_2/0, z_1/s(z_2)]\vartheta_3 \mid \{z\} \\
&= [x_1/s(s(0)), x_2/s(s(0)), y_1/s(0), y_2/0, z/s(s(z_2))][x_3/s(s(0)), z_2/s(s(0))] \mid \{z\} \\
&= [x_1/s(s(0)), x_2/s(s(0)), x_3/s(s(0)), y_1/s(0), y_2/0, z/s(s(s(s(0))))] \mid \{z\} \\
&= [z/s(s(s(s(0))))]
\end{aligned}$$

The computed answer substitution is thus $[z/s(s(s(s(0))))]$; the computed instance of the goal is $\leftarrow add(s(s(0)), s(s(0)), s(s(s(s(0)))))$. (For the intended interpretation we therefore have $2 + 2 = 4$.)

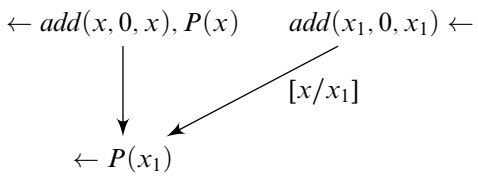
For the query $\leftarrow add(x, y, z)$ there exists a successful derivation relative to Π_{add}:



There are also infinite SLD-derivations like, for example,



For the query $\leftarrow add(x, y, z), P(x)$ there is, for example, the following failed derivation relative to Π_{add}:



SLD-derivations can thus be successful, infinite or failed.

In contradistinction to first-order resolution for arbitrary formulas (which have to be skolemized first), SLD-resolution deals with formulas that already have a certain form. These formulas are program clauses, which describe some problem or subject area, and goal clauses, which describe our questions concerning the problem or the subject area. The corresponding logic is undecidable, but can be automated in the framework of logic programming.

3.2 Non-determinism in SLD-derivations

In SLD-derivations there are different kinds of non-determinism to be considered, since in each resolution step four decisions have to be made:

(A) The selection of an atom in the current goal.

(B) The choice of a program clause for the selected atom.

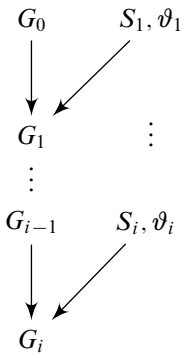
(C) The renaming of variables in the chosen program clause; that is, the choice of a variant.

(D) The choice of the most general unifier.

On the one hand, the non-determinism due to (C) and (D) is unproblematic. The choice of the most general unifier (D) is fixed by our use of the unification algorithm. Moreover, by Theorem 2.20 the concrete variant of a most general unifier does not matter with respect to the computed answer substitution for a goal G. Likewise, the chosen variant (C) does not matter, since different renamings only lead to variants of the computed answer substitution. Hence, the renaming of variables cannot lead to a loss of possible answers.
On the other hand, it is decisive for SLD-resolution how the non-determinism due to (A) and (B) is dealt with. The selection (A) of an atom in the current goal clause is determined by *selection functions*, which we consider next. Afterwards we deal with the choice (B) of the program clause for the selected atom by using *SLD-trees*.

3.3 Selection functions

Definition 3.7 A *selection function* $\mathcal{R}$ maps a given SLD-derivation *selection function*



(short: $\langle G_0, G_1, \ldots, G_i\rangle, \langle S_1, \ldots, S_i\rangle, \langle \vartheta_1, \ldots, \vartheta_i\rangle$) for $i \geq 0$ to an atom A_j in the goal clause $G_i = (\leftarrow A_1, \ldots, A_k)$ for $1 \leq j \leq k$, if G_i is not the empty clause. It is A_j the *selected atom*. *selected atom*

A selection function $\mathcal{R}$ is thus a ternary function from SLD-derivations to natural numbers, which obeys the following condition:

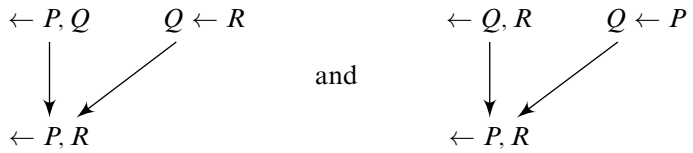
If
$$G_i = (\leftarrow A_1, \ldots, A_k) \quad \text{for } k \geq 1$$
and
$$j = \mathcal{R}(\langle G_0, G_1, \ldots, G_i\rangle, \langle S_1, \ldots, S_i\rangle, \langle \vartheta_1, \ldots, \vartheta_i\rangle),$$
then $1 \leq j \leq k$ must hold.

We then say that $\mathcal{R}$ selects the atom A_j in G_i.

Definition 3.8 Let $\mathcal{R}$ be the selection function used in an SLD-derivation. We then say that it is an SLD-derivation *via* $\mathcal{R}$. *SLD-derivation via* $\mathcal{R}$

The selection of an atom can depend on the whole SLD-derivation developed so far. This allows, for example, for the implementation of a *first in, first out selection strategy*,

in which one always selects an atom in the current goal that occurred in previous goals at least as often as the other atoms. For example, for the two SLD-derivations



which both end with the same goal, this means that in the left goal the atom P is selected, while in the right goal the atom R is selected. If one restricted selection functions in such a way that they map goals (instead of SLD-derivation) to an atom (the selected atom), then the same atom would be selected in both cases (as long as we do not use a random function for the selection).
The programming language *Prolog*, which is based on SLD-resolution, uses a simple selection function that always selects the leftmost atom in a goal.
A consequence of the completeness result for SLD-resolution (see Theorem 3.13 below) is that the successful SLD-derivations are independent of the used selection function. This can also be shown directly by proving that the selection of atoms A_i and A_j in two consecutive goals can always be swapped, that is, by first selecting A_j and then A_i (while swapping the input clauses accordingly); this result is called the *Switching Lemma*.

3.4 SLD-trees

Although the *existence* of a successful SLD-derivation does not depend on the used selection function (i.e., the non-determinism due to (A) is in this sense unproblematic), the choice of the program clause (non-determinism due to (B)) is decisive for *finding* such an SLD-derivation. Non-determinism due to (B) creates a *search space* for SLD-derivations via $\mathcal{R}$, the so-called *SLD-tree*.

Definition 3.9 An *SLD-tree* for a goal G_0 relative to a program Π via a selection function $\mathcal{R}$ is a (downward branching) tree, given as follows: *SLD-tree*

(i) Each node of the tree is a (possibly empty) goal.

(ii) The root node is G_0.

(iii) Each node G_i with selected atom A_i has exactly one successor for each clause S from Π that is applicable to A_i.

The successor is a resolvent of G_i and S (respectively of G_i and the chosen variant of S) with respect to A_i.

(iv) Nodes which are the empty goal have no successors.

Branches in SLD-trees are SLD-derivations for G_0 relative to Π. We write these in the form

$$G_0 \xrightarrow{S_1, \vartheta_1} G_1 \xrightarrow{S_2, \vartheta_2} G_2 \quad \cdots \quad G_{n-1} \xrightarrow{S_n, \vartheta_n} G_n \quad \cdots$$

from the top (i.e., from the root node G_0) downwards, as shown in the following examples.

Definition 3.10 (i) An SLD-tree is called *successful*, if it contains the empty clause. *successful*
(Branches ending with the empty clause are marked 'success'.)

(ii) An SLD-tree is called *finitely failed*, if it is finite but not successful. This is the case if every branch is a failed SLD-derivation. *finitely failed*
(Branches of failed SLD-derivations are marked 'failed'.)

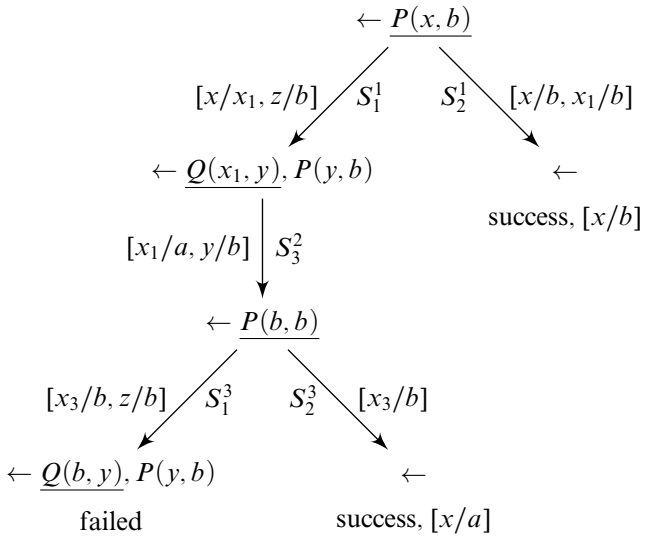
Example. We consider the logic program

S_1. $P(x,z) \leftarrow Q(x,y), P(y,z)$
S_2. $P(x,x) \leftarrow$
S_3. $Q(a,b) \leftarrow$

and the goal clause

G. $\leftarrow P(x,b)$

If the selection function always selects the *leftmost atom*, then we obtain an SLD-tree of the following form. The input clauses used in the i-th resolution step are those variants S_1^i, S_2^i and S_3^i of the corresponding program clauses S_1, S_2 and S_3, respectively, in which the renamed variables get the index i. These variants are given explicitly in Appendix A.4, where one can find the three branches of the following SLD-tree (i.e., the three SLD-derivations) written down separately, together with the substitutions that have been used to produce the most general unifiers.



The SLD-tree is finite and successful. The computed answer substitutions are

$$\begin{aligned}[x/x_1, z/b][x_1/a, y/b][x_3/b] \mid \{x\} &= [x/a, x_1/a, y/b, z/b][x_3/b] \mid \{x\} \\ &= [x/a, x_1/a, y/b, z/b, x_3/b] \mid \{x\} \\ &= [x/a] \quad \text{(middle branch)}\end{aligned}$$

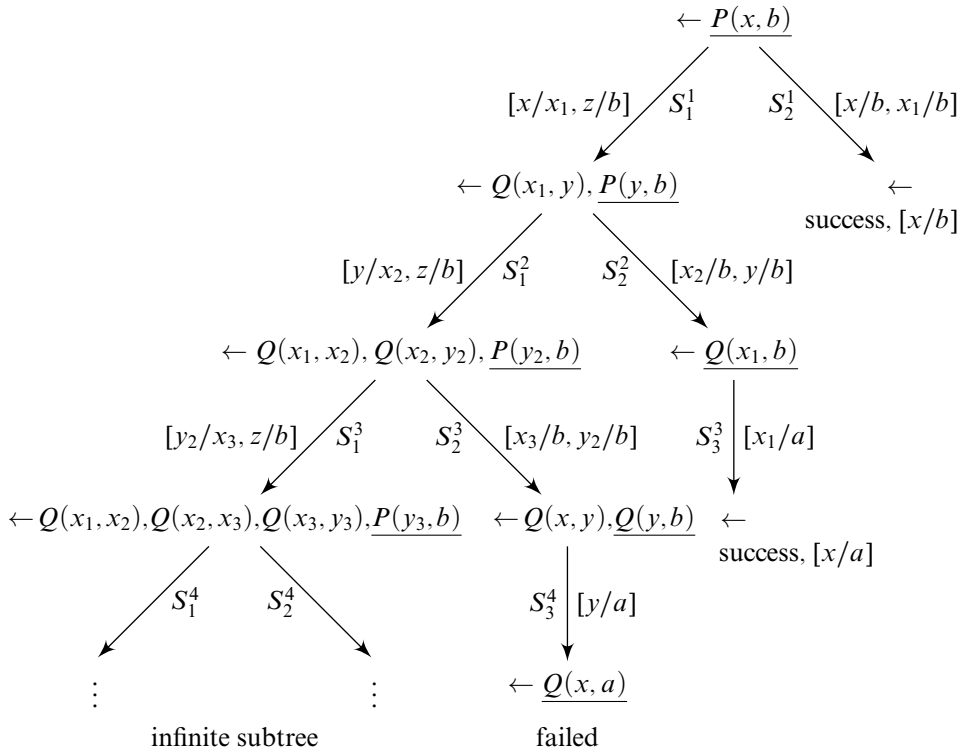
and

$$[x/b, x_1/b] \mid \{x\} = [x/b] \quad \text{(right branch)}.$$

Other SLD-trees differ from the shown SLD-tree only with respect to renamings, but they have the same form.

In the above SLD-tree, a *depth-first search* would descend into the left, failed branch. By *backtracking*, that is, by returning to an earlier branching, where we can choose an alternative, we can nevertheless find the successful middle branch. By backtracking again, we then also find the right branch and another computed answer substitution. *backtracking*

If the selection function always selects the *rightmost atom*, then we obtain an SLD-tree of the following form. (In the i-th resolution step, the variants S_1^i, S_2^i and S_3^i of the corresponding program clauses S_1, S_2 and S_3, respectively, are chosen as explained above.)



This SLD-tree is also successful, with the same computed answer substitutions $[x/a]$ and $[x/b]$ as in the first SLD-tree. In contradistinction to the first, this second tree is infinite, however; in the left branch one can always create new goal clauses.

If a depth-first search starts in the left branch, then the successful SLD-derivations to the right will never be found, since the left branch is infinite, and *backtracking* is of course only possible in finite branches. Whereas in a *breadth-first search* every successful SLD-derivation will always be found.

3.5 Soundness and completeness of SLD-resolution

Definition 3.11 (i) For clauses $S = (A \leftarrow X)$ let $\overline{S}$ be the formula $\forall(\mathbin{\text{⩓}} X \rightarrow A)$, where $\mathbin{\text{⩓}} X$ stands for the conjunction of all atoms in X.

(ii) For facts $S = (A \leftarrow)$ we have the formula $\overline{S} = \forall(\bigwedge\emptyset \rightarrow A) = \forall(\top \rightarrow A) = \forall A$.

(iii) For goal clauses $G = (\leftarrow X)$ we have the formula $\overline{G} = \forall\neg(\bigwedge X)$.

We also write $\bigwedge G$ for the conjunction of all atoms occurring in the body X of a goal clause $G = (\leftarrow X)$.

To the empty clause $\leftarrow$ corresponds the formula $\neg\top$. In this case $\bigwedge G$ $(= \bigwedge X)$ is the empty conjunction, and it is $\bigwedge G = \top$.

(iv) For a logic program Π a set of formulas $\overline{\Pi} := \{\overline{S} \mid S \in \Pi\}$ is given, which contains only closed formulas.

Example. For $S = (P(x,y) \leftarrow Q(x,z), P(z,y))$ we have

$$\overline{S} = (\forall x \forall y \forall z (Q(x,z) \wedge P(z,y) \rightarrow P(x,y))).$$

Due to the logical equivalence

$$\forall x \forall y \forall z (Q(x,z) \wedge P(z,y) \rightarrow P(x,y)) \dashv\vDash \forall x \forall y (\exists z (Q(x,z) \wedge P(z,y)) \rightarrow P(x,y))$$

we understand variables that occur only in the body of a clause as being existentially quantified.

Theorem 3.12 (Soundness of SLD-resolution)
If ϑ is a computed answer substitution for the goal G relative to a logic program Π, then $\overline{\Pi} \vDash \bigwedge G\vartheta$ holds.

Proof. See Lloyd, 1993, § 8 and Stärk (1990). QED

Theorem 3.13 (Completeness of SLD-resolution)
Let $\mathcal{R}$ be a selection function. If $\overline{\Pi} \vDash \bigwedge G\sigma$, then there exists a successful SLD-derivation of G from Π relative to $\mathcal{R}$ with a computed answer substitution ϑ such that $G\vartheta$ is more general than $G\sigma$ (i.e., there is a substitution τ with $G\vartheta\tau = G\sigma$).

Proof. See Lloyd, 1993, § 8 and Stärk (1990). QED

Since the selection function is arbitrary here, completeness holds for any given selection function. Nevertheless, the way one chooses a program clause for a selected atom may yield a loss of completeness in the sense that although there might exist a successful SLD-derivation we do not find it.
For example, we may always choose program clauses in the order of their appearance in the program. For the program Π, given by the two clauses

S_1. $P(a) \leftarrow P(a)$
S_2. $P(a) \leftarrow$

the formula $P(a)$ follows from $\overline{\Pi}$, that is, $\overline{\Pi} \vDash P(a)$. But we cannot generate a successful SLD-derivation for the goal $\leftarrow P(a)$. Only clause S_1 will be chosen, never clause S_2. But only by using S_2 one can derive the empty clause.
This does not contradict the completeness theorem, however, since the completeness theorem is only a statement about the *existence* of a successful SLD-derivation; it says nothing about whether such a derivation is also *found* in each case.

Logic programming based on SLD-resolution establishes a notion of computability that is extensionally equivalent to notions of computability like Turing-computability, λ-definability etc., which exactly characterize the partial recursive functions.

Theorem 3.14 (Computational adequacy of logic programs)
Let f be an n-ary partial recursive function. Then there exists a logic program Π_f and an $n+1$-ary relation symbol p_f such that all computed answer substitutions for the set of clauses $\Pi_f \cup \{ \leftarrow p_f(s^{k_1}(0), \ldots, s^{k_n}(0), x)\}$ have the form $[x/s^k(0)]$, and for all $k_1, \ldots, k_n, k \geq 0$ we have $f(k_1, \ldots, k_n) = k$ iff $[x/s^k(0)]$ is a computed answer substitution for $\Pi_f \cup \{ \leftarrow p_f(s^{k_1}(0), \ldots, s^{k_n}(0), x)\}$.

Proof. See Lloyd, 1993, Theorem 9.6. QED

A Appendix

A.1 Construction of conjunctive normal form

(1) Eliminate logical constants different from $\neg, \wedge, \vee$; that is, eliminate occurrences of $\rightarrow$ by the following transformation:

$$(A \rightarrow B) \rightsquigarrow (\neg A \vee B)$$

(2) Move negations inward (De Morgan) and eliminate double negations:

$$\neg(A \wedge B) \rightsquigarrow (\neg A \vee \neg B)$$
$$\neg(A \vee B) \rightsquigarrow (\neg A \wedge \neg B)$$
$$\neg\neg A \rightsquigarrow A$$

(3) Move $\wedge$ outwards by using distributivity:

$$(A \wedge B) \vee C \rightsquigarrow (A \vee C) \wedge (B \vee C)$$

Associativity of $\wedge$ and $\vee$ are used implicitly.

A.2 Construction of prenex normal form

Definition A.1 (i) A formula A in a language $\mathscr{L}$ is in *prenex normal form* (PNF for short), if it has the form $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n B$ where B is quantifier-free, $n \geq 0$, Q_i either $\forall$ or $\exists$, and all x_i are pairwise distinct. *prenex normal form*

(ii) It is $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n$ the *prefix* of the formula A, and B is called the *kernel* or the *matrix of* A.

If $n = 0$, then A is quantifier-free and identical with B.

Theorem A.2 *Let A be a formula. Then there is a formula B in prenex normal form such that A and B are logically equivalent.*

Proof. We present a procedure that transforms the formula A into a logically equivalent formula B in prenex normal form. The transformation steps are based on logical equivalences, which are not proved here. Each step is illustrated by using the formula

$$\forall z(\forall x(P(x) \rightarrow P(f(x))) \vee \neg\forall x(Q(x) \vee R(x, a)))$$

as an example.

(1) Eliminate all empty quantifiers in A, that is

(a) $\forall x A_1 \rightsquigarrow A_1$, if x is not free in A_1; (b) $\exists x A_1 \rightsquigarrow A_1$, if x is not free in A_1.

It is $A \dashv\models A_1$.

EXAMPLE. $\forall z(\forall x(P(x) \rightarrow P(f(x))) \vee \neg\forall x(Q(x) \vee R(x, a))) \rightsquigarrow$
$\forall x(P(x) \rightarrow P(f(x))) \vee \neg\forall x(Q(x) \vee R(x, a))$

(2) Rename bound variables in A_1 in such a way that all quantifiers have different variables, no variable occurs free as well as bound, and free variables do not get bound.

Let the resulting formula be A_2. It is $A_1 \dashv\models A_2$.

EXAMPLE. $\forall x(P(x) \to P(f(x))) \vee \neg\forall x(Q(x) \vee R(x,a)) \rightsquigarrow$
$\forall x(P(x) \to P(f(x))) \vee \neg\forall y(Q(y) \vee R(y,a))$

(3) Move negations inwards, so that they occur only in front of atoms, and eliminate double negations:

(a) $\neg\forall xC \rightsquigarrow \exists x\neg C$
(b) $\neg\exists xC \rightsquigarrow \forall x\neg C$
(c) $\neg(C \wedge D) \rightsquigarrow (\neg C \vee \neg D)$
(d) $\neg(C \vee D) \rightsquigarrow (\neg C \wedge \neg D)$
(e) $\neg(C \to D) \rightsquigarrow (C \wedge \neg D)$
(f) $\neg\neg C \rightsquigarrow C$

Let the resulting formula be A_3. It is $A_2 \dashv\models A_3$.

EXAMPLE. $\forall x(P(x) \to P(f(x))) \vee \neg\forall y(Q(y) \vee R(y,a)) \rightsquigarrow$
$\forall x(P(x) \to P(f(x))) \vee \exists y\neg(Q(y) \vee R(y,a)) \rightsquigarrow$
$\forall x(P(x) \to P(f(x))) \vee \exists y(\neg Q(y) \wedge \neg R(y,a))$

(4) Move quantifiers outwards:

(a) $\forall xC \wedge D \rightsquigarrow \forall x(C \wedge D)$
(b) $C \wedge \forall xD \rightsquigarrow \forall x(C \wedge D)$
(c) $\exists xC \wedge D \rightsquigarrow \exists x(C \wedge D)$
(d) $C \wedge \exists xD \rightsquigarrow \exists x(C \wedge D)$
(e) $\forall xC \vee D \rightsquigarrow \forall x(C \vee D)$
(f) $C \vee \forall xD \rightsquigarrow \forall x(C \vee D)$
(g) $\exists xC \vee D \rightsquigarrow \exists x(C \vee D)$
(h) $C \vee \exists xD \rightsquigarrow \exists x(C \vee D)$
(i) $\forall xC \to D \rightsquigarrow \exists x(C \to D)$
(j) $C \to \forall xD \rightsquigarrow \forall x(C \to D)$
(k) $\exists xC \to D \rightsquigarrow \forall x(C \to D)$
(l) $C \to \exists xD \rightsquigarrow \exists x(C \to D)$

By (2) it is guaranteed that no free variables become bound.

The resulting formula is the desired prenex normal form B of the initial formula A. It is $A_3 \dashv\models B$, and, since $A \dashv\models A_1 \dashv\models A_2 \dashv\models A_3 \dashv\models B$, also $A \dashv\models B$. That is, the prenex normal form is logically equivalent to the initial formula.

EXAMPLE. $\forall x(P(x) \to P(f(x))) \vee \exists y(\neg Q(y) \wedge \neg R(y,a)) \rightsquigarrow$
$\forall x((P(x) \to P(f(x))) \vee \exists y(\neg Q(y) \wedge \neg R(y,a))) \rightsquigarrow$
$\forall x\exists y((P(x) \to P(f(x))) \vee (\neg Q(y) \wedge \neg R(y,a)))$

or alternatively $\forall x(P(x) \to P(f(x))) \vee \exists y(\neg Q(y) \wedge \neg R(y,a)) \rightsquigarrow$
$\exists y(\forall x(P(x) \to P(f(x))) \vee (\neg Q(y) \wedge \neg R(y,a))) \rightsquigarrow$
$\exists y\forall x((P(x) \to P(f(x))) \vee (\neg Q(y) \wedge \neg R(y,a)))$

In this procedure only the order of steps (1)-(4) has to be observed. Within these steps the substeps (1a)/(1b), (3a)-(3f) and (4a)-(4l) can be applied in any order.

Examples. (i) $\forall z(\exists x P(x) \to \neg\exists x Q(x,y)) \overset{(1a)}{\rightsquigarrow} \exists x P(x) \to \neg\exists x Q(x,y)$

$\overset{(2)}{\rightsquigarrow} \exists x P(x) \to \neg\exists z Q(z,y)$

$\overset{(3b)}{\rightsquigarrow} \exists x P(x) \to \forall z\neg Q(z,y)$

$\overset{(4j)}{\rightsquigarrow} \forall z(\exists x P(x) \to \neg Q(z,y))$

$\overset{(4k)}{\rightsquigarrow} \forall z\forall x(P(x) \to \neg Q(z,y))$

(ii) Alternatively one can first apply (4k) and afterwards (4j):

$\exists x P(x) \to \forall z\neg Q(z,y) \overset{(4k)}{\rightsquigarrow} \forall x(P(x) \to \forall z\neg Q(z,y)) \overset{(4j)}{\rightsquigarrow} \forall x\forall z(P(x) \to \neg Q(z,y))$

The prenex normal form of a given formula is not uniquely determined, since

(i) bound variables can be renamed differently,

(ii) the order in which quantifiers are moved outwards is not determined,

(iii) and the kernel may in principle be replaced by any logically equivalent quantifier-free formula.

A.3 Correctness of the unification algorithm

Theorem A.3
Let A and B be two atoms. Then the following holds:

(i) *The unification algorithm always terminates.*

(ii) *If A and B are unifiable, then the unification algorithm computes a most general unifier for A and B.*

(iii) *If A and B are not unifiable, then the unification algorithm terminates with the output that A and B are not unifiable.*

Proof. (i) The difference set U_n contains only finitely many variables, and in step (3) a variable is eliminated. Step (3) can thus be called only a finite number of times.

(ii) We first show:

($*$) If ϑ is a unifier for A and B, then the algorithm does not terminate in step (3), and in step (2) $\sigma_n\vartheta = \vartheta$ holds each time.

For $n = 0$ it is $\sigma_0 = \varepsilon$. Thus $\sigma_0\vartheta = \vartheta$.

Suppose we are in step (2) with $\sigma_n\vartheta = \vartheta$.

In case $A\sigma_n = B\sigma_n$ the algorithm terminates in step (2).

In case $A\sigma_n \neq B\sigma_n$ the difference set U_n is formed, and the algorithm goes to step (3).

Let $U_n = \{t, t'\}$. It is $t\sigma_n\vartheta = t\vartheta = t'\vartheta = t'\sigma_n\vartheta$, since ϑ is by the supposition made in ($*$) a unifier for A and B. Consequently, $t\vartheta = t'\vartheta$.

Case 1: t is a variable x.

Since $x\vartheta = t'\vartheta$ and $x \neq t'$, the variable x cannot occur in t'.
Then $\sigma_{n+1} = \sigma_n[x/t']$ holds.
Since $x\vartheta = t'\vartheta$, we have $[x/t']\vartheta = \vartheta$.
This implies $\sigma_{n+1}\vartheta = (\sigma_n[x/t'])\vartheta = \sigma_n([x/t']\vartheta) = \sigma_n\vartheta = \vartheta$.

Case 2: t' is a variable. Analogous to case 1.

Case 3: Neither t nor t' is a variable.

Since $t\vartheta = t'\vartheta$, the terms t and t' must begin with the same symbol. However, by supposition, $U_n = \{t, t'\}$ holds, that is, t and t' are exactly those subterms, for which $A\sigma_n \neq B\sigma_n$. Hence this case is impossible, and the algorithm does not terminate in step (3).

We have thus shown $(*)$.

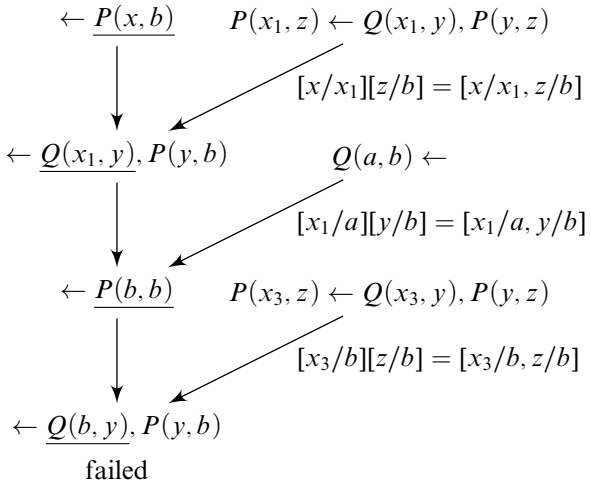
Let now A and B be unifiable. From $(*)$ together with (i) it follows that the algorithm terminates after n steps in step (2). The output σ_n is a unifier for A and B.

Let ϑ be another unifier for A and B. Then it follows with $(*)$ that $\sigma_n\vartheta = \vartheta$. Since ϑ is arbitrary, we can conclude with Lemma 2.24 that σ_n is an idempotent most general unifier.

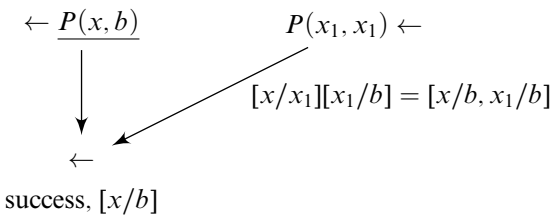
(iii) If A and B are not unifiable, then the unification algorithm cannot terminate in step (2) (A and B would be unifiable in this case). But since the algorithm terminates in any case, as shown in (i), it has in this case to terminate in step (3) with the output that A and B are not unifiable. QED

A.4 Addendum to the example on page 34

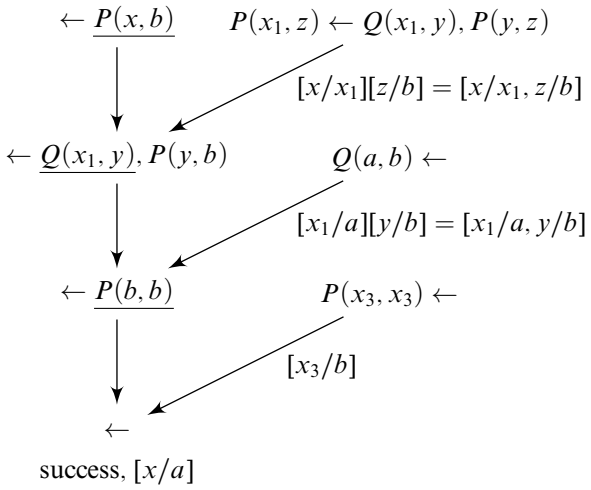
Left branch:



Right branch:



Middle branch:



References

K. R. Apt (1997), *From Logic Programming to Prolog*. London: Prentice Hall.

P. Blackburn, J. Bos & K. Striegnitz (2006), *Learn Prolog Now!*. London: College Publications, freely available online: http://www.learnprolognow.org/.

W. F. Clocksin & C. S. Mellish (2003), *Programming in Prolog*, 5th edition. Berlin: Springer.

K. Doets (1994), *From Logic to Logic Programming*. Cambridge, Massachusetts: The MIT Press.

J. H. Gallier (2015), *Logic for Computer Science. Foundations of Automatic Theorem Proving*, 2nd edition. Mineola: Dover Publications.

H. J. Goltz & H. Herre (1990), *Grundlagen der logischen Programmierung*. Weinheim: Wiley-VCH.

J-L. Lassez, M. J. Maher & K. Marriot (1987), Unification Revisited. In: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Los Altos: Morgan Kaufmann. Pages 587–625.

A. Leitsch (1997), *The Resolution Calculus*. Berlin: Springer.

J. W. Lloyd (1993), *Foundations of Logic Programming*, 2nd edition. Berlin: Springer.

S.-H. Nienhuys-Cheng & R. de Wolf (1997), *Foundations of Inductive Logic Programming*. Lecture Notes in Artificial Intelligence 1228, Berlin: Springer.

R. F. Stärk (1990), A direct proof for the completeness of SLD-resolution. In: E. Börger, H. Kleine Büning & M. M. Richter (eds), *CSL '89, 3rd Workshop on Computer Science Logic, Kaiserslautern, Germany, October 2–6, 1989*. Lecture Notes in Computer Science 440, Berlin: Springer. Pages 382–383.

A. S. Troelstra & H. Schwichtenberg (2000), *Basic Proof Theory*, 2nd edition. Cambridge University Press.

Index