Predicate Logic

When we introduced propositional logic we argued in treating the electrical circuit example **??**, that a string like $high(inv1, o)$ can be read intuitively as ´´the output of inverter 1 is high´´, but then we abstracted from this internal structure of the proposition and further on we represented the proposition by a simple string like ´´the_output_of_inverter_1_is_high´´. In the formal parts of the calculus we even assumed more abstract just a given countable set of propositions $\{P_1, \cdots, P_i, \cdots\}$. Until now, we investigated a logical language, which was able to deal with connecting these propositions,
In this section we will have a closer look into the structure of propositions. We will be able to explicitly refer to *objects* like the inverter $inv1$ or its output o in the sentences of our language,. Besides this *domain*, we will introduce the new concept of a *variable*, which stands for an arbitrary object of our domain. Hence, we will be able to write e.g. $high(x, o)$ to denote the fact that the output of a x is high. As a consequence of this concept we will have *quantifiers*, which allow for sentences like $\exists x\ high(x, o)$, with an intended meaning that there exists an object x, whose output o is high, or $\forall x\ high(x, o)$ with an intended meaning that for all objects x output o is high,
Together with these new concepts it will be possible to express rather abstract properties of elements from the domain. Assume, e.g. that we want to introduce the concept of elements in a circuit to be connected. We could say $connected(inv1, inv2)$ and $connected(inv2, inv3)$. Now, with the intended meaning of being connected, we can assume, that $inv1$ and $inv3$ are connected as well. Of course, we could introduce the new sentence $connected(inv1, inv3)$ to express this; but the disadvantage is obvious: this have to be done for all objects from our domain. In predicate logic this property of transitivity of connectedness can be expressed without referring to explicit objects by the following sentence:

$$\forall x \forall y \forall z((connected(x, y) \wedge connected(y, z)) \rightarrow connected(x, z))$$

Syntax

Definition 1 (Syntax of predicate logic – Terms) *Assume a*

- *countable set of* function symbols $\{{f_i}^k \mid i, k = 1, 2, 3, \cdots\}$
- *a countable set of* variables $\{x_i \mid i = 1, 2, 3, \cdots\}$

The set of terms *is defined by the following induction:*

- *Variables x_i are terms.*
- *If $t_1, \cdots, t_k$ are terms and f_i^k is a function symbol, then $f_i^k(t_1, \cdots, t_k)$ is a term.*

Terms of type $f_i^0()$ are special ones, they are called constants. *In this case we omit the braces and denote them as f_i^0.*

Terms are the syntactic counterpart of the above mentioned objects. Constants will denote the elements of the domain and function symbols will denote a way to refer to such objects.
The following definition introduces the formulae.

Definition 2 (Syntax of predicate logic – Formulae) *Assume a countable set of* predicate symbols$\{p_i^k \mid i = 1, 2, 3, \cdots\}$.
The set of (well-formed) formulae is defined by the following induction:

- *If $t_1, \cdots, t_k$ are terms and P_i^k is a predicate symbol, then $P_i^k(t_1, \cdots, t_k)$ is a formula.*
- *If F and G are formulae, then $(F \wedge G)$ and $(F \vee G)$ are formulae.*
- *If F is a formula, then $\neg F$ is a formula.*
- *If x is a variable and F a formula, then $\forall x F$ and $\exists x F$ are formulae.*

Formulae of type $P_i^k(t_1, \cdots, t_k)$ are called atoms *or atomic formulae.*

Note that the concept of subformulae applies exactly like in the propositional case **??**.
We introduce the following abbreviations, which will be used with indices as well:

$u, v, w, x, z,$	for variables
$a, b, c, \cdots$	for constants
$f, g, h, \cdots$	for function symbols
$p, q, r, \cdots$	for predicate symbols

Note the the arity of function and predicate symbols is ommited in these abbreviations; we assume that it will be obvious from the context.
Example: Assume we want to represent the following equation, which holds for arbitrary elements in a field:

$$x * (y + z) = x * y + x * z$$

The two operators $*$ and $+$ are represented in a predicate logic formula as binary function symbols f_1^2 and f_2^2, the three variables are x_1, x_2 and x_3, and the equality relation $=$ is the binary predicate symbol P_1^2. Altogether we have the following formula in predicate logic:

$$\forall x_1 \forall x_2 \forall x_3 (P_1^2(f_2^2(x_1, f_1^2(x_2, x_3)), f_1^2(f_2^2(x_1, x_2), f_2^2(x_1, x_3))))$$

In the following we will use the obvious and more liberal notation as in the propositional case.

Definition 3 *An occurrence of a variable x in a formula F is called* bound*, if it occurs in a subformula of F which is of the form $\exists x\ G$ or $\forall x\ G$. Otherwise we call the occurrence* free*.*
A formula, which do not contain a free occurrence of a variable is called closed*.*

Example: The following formula contains for x and y free and bound occurences.

$$\forall z (Q(z) \wedge \forall x (P(x, y)) \vee \exists y (P(x, y)))$$

Semantics

Definition 4 (Semantics of predicate logic – Interpretation) *An* interpretation *is a pair* $\mathcal{I} = (U_\mathcal{I}, A_\mathcal{I})$*, where*

- $U_\mathcal{I}$ *is an arbitrary nonempty set, called* domain*, or* universe*.*
- $A_\mathcal{I}$ *is a mapping which associates to*
  - *a* k*-ary predicate symbol, a* k*-ary predicate over* $U_\mathcal{I}$*,*
  - *a* k*-ary function symbol, a* k*-ary function over* $U_\mathcal{I}$*, and*
  - *a variable an element from the domain.*

Let F *be a formula and* $\mathcal{I} = (U_\mathcal{I}, A_\mathcal{I})$ *be an interpretation. We call* $\mathcal{I}$ *an interpretation for* F*, if* $\mathcal{A}_\mathcal{I}$ *is defined for every predicate and function symbol, and for every variable, that occurs free in* F *.*

Example: Let $F = \forall x p(x, f(x)) \wedge q(g(a, z))$ and assume the arities of the symbols as written down. In the following we give two interpretations for F:

- $\mathcal{I}_1 = (N_0, A_1)$, such that
  - $A_1(p) = \{(m, n) \mid m, n \in N_0 \text{ and } m < n\}$
  - $A_1(q) = \{n \in N_0 \mid n \text{ is prime }\}$
  - $A_1(f)(n) = n + 1 \; \forall n \in N_0$
  - $A_1(g)(m, n) = m + n \; \forall n, m \in N_0$
  - $A_1(a) = 2$
  - $A_1(z) = 3$

  Under this interpretation the formula F can be read as " Every natural number is smaller than its successor and the sum of 2 and 3 is a prime number."

- $\mathcal{I}_2 = (U_2, A_2)$, such that
  - $U_2 = \{a, f(a), g(a, a), f(g(a, a)), g(f(a), f(a)), \cdots\}$
  - $A_2(f)(t) = f(t)$ for $t \in U_2$
  - $A_2(g)(t_1, t_2) = g(t_1, t_2)$, if $t_1, t_2 \in U_2$
  - $A_2(a) = a$
  - $A_2(z) = f(f(a))$
  - $A_2(p) = \{p(a, a), p(f(a), f(a)), p(f(f(a)), f(f(a)))\}$
  - $A_2(q) = \{g(t_1, t_2) \mid t_1, t_2 \in U_2\}$

For a given interpretation $\mathcal{I} = (U, A)$ we write in the following $p^\mathcal{I}$ instead of $A(p)$; the same abbreviation will be used for the assignments for function symbols and variables.

Definition 5 (Semantics of predicate logic – Evaluation of Formulae) *Let* F *be a formula and* $\mathcal{I}$ *an interpretation for* F*. For terms* t *which can be composed with symbols from* F *the value* $\mathcal{I}(t)$ *is given by*

- $\mathcal{I}(x) = x^{\mathcal{I}}$
- $\mathcal{I}(f(t_1, \cdots, t_k)) = f^{\mathcal{I}}(\mathcal{I}(t_1), \cdots, \mathcal{I}(t_k))$, *if* $t_1, \cdots, t_k$ *are terms and* f *a* k*-ary function symbol. (This holds for the case* $k = 0$ *as well.)*

The value $\mathcal{I}(F)$ *of a formula* F *is given by*

- $\mathcal{I}(p(t_1, \cdots, t_k)) = \begin{cases} true & \text{if } (\mathcal{I}(t_1), \cdots, \mathcal{I}(t_k)) \in p^{\mathcal{I}} \\ false & \text{otherwise} \end{cases}$
- $\mathcal{I}(F \wedge G)) = \begin{cases} true & \text{if } \mathcal{I}(F) = true \text{ and } \mathcal{I}(G) = true \\ false & \text{otherwise} \end{cases}$
- $\mathcal{I}((F \vee G)) = \begin{cases} true & \text{if } \mathcal{I}(F) = true \text{ or } \mathcal{I}(G) = true \\ false & \text{otherwise} \end{cases}$
- $\mathcal{I}(\neg F) = \begin{cases} true & \text{if } \mathcal{I}(F) = false \\ false & \text{otherwise} \end{cases}$
- $\mathcal{I}(\forall G) = \begin{cases} true & \text{if for every } d \in U : \mathcal{I}_{[x/d]}(G) = true \\ false & \text{otherwise} \end{cases}$
- $\mathcal{I}(\exists G) = \begin{cases} true & \text{if there is a } d \in U : \mathcal{I}_{[x/d]}(G) = true \\ false & \text{otherwise} \end{cases}$

where, $f_{[x/d]}(y) = \begin{cases} f(y) & \text{if } x \neq y \\ d & \text{otherwise} \end{cases}$

The notions of satifiable, valid, and $\models$ are defined according to the propositional case **??**. Note that, predicate calculus is an extension of propositional calculus: Assume only 0-ary predicate symbols and a formula which contains no variable, i.e. there can be no terms and no quantifier in a well-formed formula.
On the other hand, predicate calculus can be extended: If one allows for quantifications over predicate and function symbols, we arrive at a second order predicate calculus. E.g.

$$\forall p \exists f \; p(f(x))$$

Another example for a second order formula of is the induction principle from **??**.
Problem 1
The interpretation $\mathcal{I} = isgiven(U_{\mathcal{I}}, A_{\mathcal{I}})$ as follows:

$$\begin{aligned} U_{\mathcal{I}} &= \mathbb{N} \\ p^{\mathcal{I}} &= \{(m, n) \mid m < n\} \\ f^{\mathcal{I}}(m, n) &= m + n \\ x^{\mathcal{I}} = 5 \quad &; \quad y^{\mathcal{I}} = 7 \end{aligned}$$

Determine the value of following terms and formulae:

1. $\mathcal{I}(f(f(x, x), y))$

2. $\mathcal{I}(\forall x \forall y (p(x,y) \vee p(y,x))$

3. $\mathcal{I}(p(x,x) \rightarrow p(y,x))$

4. $\mathcal{I}(\exists x\ p(y,x))$

Problem 2
The interpretation $\mathcal{I} = is\ given(U_\mathcal{I}, A_\mathcal{I})$ as follows:

$$\begin{array}{rcl} U_\mathcal{I} & = & \mathbb{R} \\ P^\mathcal{I} & = & \{z \mid z \geq 0\} \\ f^\mathcal{I}(z) & = & z^2 \\ x^\mathcal{I} & = & \sqrt{2} \\ E^\mathcal{I} & = & \{(x,y) \mid x = y\} \\ g^\mathcal{I}(x,y) & = & x + y \\ y^\mathcal{I} & = & -1 \end{array}$$

Determine the value of following terms and formulae:

1. $\mathcal{I}(g(f(x), f(y)))$
2. $\mathcal{I}(\forall x\, P(f(x)))$
3. $\mathcal{I}(\exists z \forall x \forall y\, E(g(x,y), z))$
4. $\mathcal{I}(\forall y (E(f(x), y) \rightarrow P(g(x,y))))$

Problem 3
The following formula is given:

$$F = \forall x \forall y \forall z\, R(h(h(x,y), z), h(x, h(y,z))) \ \wedge\ \exists x \exists y\, \neg R(h(x,y), h(y,x))$$

Indicate a structure $\mathcal{A}$, which is a model for F and a structure $\mathcal{B}$ which is no model for F!

Equivalence and Normal Forms

Equivalence of formulae is defined as in the propositional case:

Definition 6 *The formulae F and G are called (semantically)* equivalent*, iff for all interpretations $\mathcal{I}$ for F and G, $\mathcal{I}(F) = \mathcal{I}(G)$. We write $F \equiv G$.*

The equivalences from the propositional case in theorem **??** hold and in addition we have the following cases for quantifiers.

Theorem 1 *The following equivalences hold:*

$$\begin{array}{lcl} \neg \forall x F & \equiv & \exists x \neg F \\ \neg \exists x F & \equiv & \forall x \neg F \end{array}$$

If x does not occur free in G:

$$\begin{array}{lcl} \forall x F \wedge G & \equiv & \forall x (F \wedge G) \\ \forall x F \vee G & \equiv & \forall x (F \vee G) \\ \exists x F \wedge G & \equiv & \exists x (F \wedge G) \\ \exists x F \vee G & \equiv & \exists x (F \vee G) \end{array}$$

$$\begin{array}{lcl} \forall x F \wedge \forall x G & \equiv & \forall x (F \wedge G) \\ \exists x F \vee \exists x G & \equiv & \exists x (F \vee G) \\ \forall x \forall y\ F & \equiv & \forall y \forall x F \\ \exists x \exists y\ F & \equiv & \exists y \exists x F \end{array}$$

Proof: We will prove only the equivalence

$$\forall x F \wedge G \equiv \forall x (F \wedge G)$$

with x has no free occurence in G, as an example.
Assume an interpretation $\mathcal{I} = (U, \mathcal{A})$ such that

$$\begin{aligned}
& \mathcal{I}(\forall x F \wedge G) = \mathit{true} \\
\text{iff} \quad & \mathcal{I}(\forall x F) = \mathit{true} \text{ and } \mathcal{I}(G) = \mathit{true} \\
\text{iff} \quad & \text{for all } d \in U : \mathcal{I}_{[x/d]}(F) = \mathit{true} \text{ and } \mathcal{I}(G) = \mathit{true} \\
\text{iff} \quad & \text{for all } d \in U : \mathcal{I}_{[x/d]}(F) = \mathit{true} \text{ and } \mathcal{I}_{[x/d]}(G) = \mathit{true} \ (x \text{ does not occur free in } G) \\
\text{iff} \quad & \text{for all } d \in U : \mathcal{I}_{[x/d]}((F \wedge G)) = \mathit{true} \\
\text{iff} \quad & \mathcal{I}(\forall x (F \wedge G)) = \mathit{true}.
\end{aligned}$$

Note that the following symmetric cases do not hold:

$\forall x F \vee \forall x G$	is not equivalent to	$\forall x (F \vee G)$
$\exists x F \wedge \exists x G$	is not equivalent to	$\exists x (F \wedge G)$

The theorem for substituitivity holds as in the propositional case **??**.
Example: Let us transform the following formulae by means of subsituitivity and the equivalences from theorem 1:

$$\begin{aligned}
& \equiv (\neg(\exists x\, P(x,y) \vee \forall z\, Q(z)) \wedge \exists w\, P(f(a,w))) \\
& \equiv ((\neg\exists x\, P(x,y) \wedge \neg\forall z\, Q(z)) \wedge \exists w\, P(f(a,w))) \\
& \equiv ((\forall x\, \neg P(x,y) \wedge \exists z\, \neg Q(z)) \wedge \exists w\, P(f(a,w))) \\
& \equiv (\exists w\, P(f(a,w)) \wedge (\forall x\, \neg P(x,y) \wedge \exists z\, \neg Q(z))) \\
& \equiv \exists w\, (P(f(a,w)) \wedge \forall x\, (\neg P(x,y) \wedge \exists z\, \neg Q(z))) \\
& \equiv \exists w\, (\forall x\, (\exists z\, \neg Q(z) \wedge \neg P(x,y)) \wedge P(f(a,w))) \\
& \equiv \exists w\, (\forall x\, \exists z\, (\neg Q(z) \wedge \neg P(x,y)) \wedge P(f(a,w))) \\
& \equiv \exists w\, \forall x\, \exists z\, ((\neg Q(z) \wedge \neg P(x,y)) \wedge P(f(a,w)))
\end{aligned}$$

Definition 7 *Let F be a formula, x a variable and t a term. $F[x/t]$ is obtained from F by substituting every free occurrence of x by t.*

Note, that this notion can be iterated: $F[x/t_1][y/t_2]$ and that t_1 may contain free occurrences of y.

Lemma 1 *Let F be a formula, x a variable and t a term.*

$$\mathcal{I}(F[x/t]) = \mathcal{I}_{[x/\mathcal{I}(t)]}(F)$$

Lemma 2 (Bounded Renaming) *Let $F = QxG$ be a formula, where $Q \in \{\forall, \exists\}$ and y a variable without an occurrence in G, then $F \equiv QyG[x/y]$*

Definition 8 *A formula is called* proper *if there is no variable which occurrs bound and free and after every quantifier there is a distinct variable.*

Lemma 3 *For every formula F there is a formula G which is proper and equivalent to F.*

Proof: Follows immediately by bounded renaming.
Example:

$$F = \forall x \exists x\ p(x, f(y)) \wedge \forall x (q(x, y) \vee r(x))$$

has the eqivalent and proper formula

$$G = \forall x \exists x\ p(x, f(y)) \wedge \forall z (q(u, y) \vee r(u)$$

Definition 9 *A formula is called in* prenex form *if it has the form $Q_1 \cdots Q_n F$, where $Q_i \in \{\forall, \exists\}$ with no occurrences of a quantifier in F*

Theorem 2 *For every formula there is a proper formula in prenex form, which is equivalent.*

Example:

$$\forall x \exists y\ p(x, g(y, f(x)) \vee \neg q(z)) \vee \neg \forall x\ r(x, z)$$

$$\forall x \exists y\ p(x, g(y, f(x)) \vee \neg q(z)) \vee \exists x\ \neg r(x, z)$$

$$\forall x \exists y p(x, g(y, f(x)) \vee \neg q(z)) \vee \exists v \neg r(v, z)$$

$$\forall x \exists y \exists v p(x, g(y, f(x)) \vee \neg q(z)) \vee \neg r(v, z)$$

Proof: Induction over the the structure of the formula gives us the theorem for an atomic formula immediately.

- $F = \neg F_0$: There is a $G_0 = Q_1 y_1 \cdots Q_n y_n G'$ with $Q_i \in \{\forall, \exists\}$, which is equivalent to F_0. Hence we have

  $$F \equiv \overline{Q_1} y_1 \cdots \overline{Q_n} y_n \neg G'$$

  where $\overline{Q_i} = \begin{cases} \exists & if Q_i = \forall \\ \forall & if Q_i = \exists \end{cases}$

- $F = F_1 \circ F_2$ with $\circ \in \{\wedge, \vee\}$. There exists G_1, G_2 which are proper and in prenex form and $G_1 \equiv F_1$ and $G_2 \equiv F_2$. With bounded renaming we can construct

  $$\begin{aligned} G_1 &= Q_1 y_1 \cdots Q_k y_k G'_1 \\ G_2 &= Q'_1 z_1 \cdots Q'_l z_l G'_2 \end{aligned}$$

  where $\{y_1, \cdots, y_n\} \cap \{z_1, \cdots, z_l\} = \emptyset$ and hence

  $$F \equiv Q_1 y_1 \cdots Q_k y_k Q'_1 z_1 \cdots Q'_l z_l (G'_1 \circ G'_2)$$

In the following we call proper formulae in prenex form PP-formulae or PPF's.

Definition 10 *Let F be a PPF. While F contains a $\exists$-Quantifier, do the following transformation:*
F has the form

$$\forall y_1, \cdots \forall y_n \exists z G$$

where G is a PPF and f is a n-ary function symbol, which does not occur in G.
Let F be

$$\forall y_1, \cdots \forall y_n G[z/f(y_1, \cdots, y_n)]$$

If there exists no more $\exists$-quantifier, F is in Skolem form.

Theorem 3 *Let F be a PPF. F is satisfiable iff the Skolem form of F is satisfiable.*

Proof: Let $F = \forall y_1 \cdots \forall y_n \exists z G$; after one transformation according to the while-loop we have

$$F' = \forall y_1 \cdots \forall y_n \exists z G[z/f(y_1, \cdots, y_n)$$

where f is a new function symbol.
We have to prove that this transformation is satisfiability preserving:
Assume F' is satisfiable. than there exists a model $\mathcal{I}$ for F' $\mathcal{I}$ is an interpretation for F.
From the model property we have for all $u_1, \cdots, u_n \in U_{\mathcal{I}}$

$$\mathcal{I}_{[y_1/u_1]\cdots[y_n/u_n]}(G[z/f(y_1, \cdots, y_n)]) = true$$

From Lemma 1 we conclude

$$\mathcal{I}_{[y_1/u_1]\cdots[y_n/u_n][z/v]}(G) = true$$

where $v = \mathcal{I}(f(u_1, \cdots, u_n)$. Hence we have, that for all $u_1, \cdots, u_n \in U_{\mathcal{I}}$ there is a $v \in U_{\mathcal{I}}$, where

$$\mathcal{I}_{[y_1/u_1]\cdots[y_n/u_n][z/v]}(G) = true$$

and hence we have, that $\mathcal{I}(\forall y_1 \cdots \forall y_n \exists z G) = true$, which means, that $\mathcal{I}$ is a model for F.
For the opposite direction of the theorem, asume that F has a model $\mathcal{I}$. Then we have, that for all $u_1, \cdots, u_n \in U_{\mathcal{I}}$, there is a $v \in U_{\mathcal{I}}$, where

$$\mathcal{I}_{[y_1/u_1]\cdots[y_n/u_n][z/v]}(G) = true$$

Let $\mathcal{I}$' be an interpretation, which deviates from $\mathcal{I}$ only, by the fact that it is defined for the function symbol f, where $\mathcal{I}$ is not defined. We assume that $f^{\mathcal{I}'}(u_1, \cdots, u_n) = v$, where v is choosen according to the above equation.
Hence we have that for all $u_1, \cdots, u_n \in U'_{\mathcal{I}}$

$$\mathcal{I}'_{[y_1/u_1]\cdots[y_n/u_n][z/f^{\mathcal{I}'}(u_1,\cdots,u_n)]}(G) = true$$

and from Lemma 1 we conclude that for all $u_1, \cdots, u_n \in U_{\mathcal{I}}$

$$\mathcal{I}'_{[y_1/u_1]\cdots[y_n/u_n]}(G[z/f^(y_1, \cdots, y_n)]) = true$$

which means, that $\mathcal{I}'(\forall y_1 \cdots \forall y_n G[z/f(y_1, \cdots, y_n]) = true$, and hence $\mathcal{I}'$ is a model for F'.
The above results can be used to transfrom a Formula into a set of clauses, its clause normal form:

Transformation into Clause Normal Form

Given a first order formula F.

- Transform F into an equivalent proper F_1 by bounded renaming.
- Let $y_1, \cdots, y_k$ the free variables from F_1. Transform F_1 into $F_2 = \exists y_1 \cdots \exists y_k\ F_1$
- Transform F_2 into an equivalent prenex form F_3.
- Transform F_3 into its Skolemform $F_4 = \forall x_1 \cdots \forall x_l\ G$
- Transform G into its CNF $G\prime = (\bigwedge_{i=1}^{n}\ (\bigvee_{j=1}^{m}\ L_{i,j}))$ where L_{ij} is a literal. This results in $F_5 = \forall x_1 \cdots \forall x_l\ G\prime$
- Write F_5 as a set of clauses:

  $$F_6 = \{C_1, \cdots, C_n\}$$

  where $C_i = \{L_{i,1}, \cdots, L_{i,m}\}$

Interaction: CNF Transformation for Predicate Logic

This is our current set of predicate logic formulas:

Currently empty.

Current formulas manipulation:

Enter some formulas: Help:

Add to / **Overwrite** current formulas

Delete selected / **Delete all** current formulas

Save/Reload current formulas:

Save to database (enter name):

Reload from database (select name):

Reload from file (enter file name):

Convert selected / **Convert all** formulas to CNF, replacing the current clauses.
And this the resulting clause set (possibly from a previous run):

Currently empty.

Problem 1
Let F be a formula, x a variable and t a term. Then $F[x/t]$ denotes the formula which results from F by replacing every free occurrence of x by t. Give a formal definition of this three argument function $F[x/t]$, by induction over the structure of the formula F.

Problem 2
Show the following semantic equivalences:

1. $\forall x(p(x) \to (q(x) \wedge r(x))) \equiv \forall x(p(x) \to q(x)) \ \wedge \ \forall x(p(x) \to r(x))$
2. $\forall x(p(x) \to (q(x) \vee r(x))) \not\equiv \forall x(p(x) \to q(x)) \ \vee \ \forall x(p(x) \to r(x))$

Problem 3
Show the following semantic equivalences:

1. $(\forall x\ p(x)) \to q(b) \equiv \exists x\ (p(x) \to q(b))$
2. $(\forall x\ r(x)) \vee (\exists y\ \neg r(y)) \equiv (\forall x\ r(x)) \to (\exists y\ r(y))$

Problem 4
Show that for arbitrary formulae F and G, the following holds:

1. $\forall x(F \vee G) \not\equiv \forall x\ F \vee \forall x\ G$
2. If $G = F[x/t]$, than $G \models \exists x\ F$.

Problem 5
Transform the following formulae in Skolem form !

1. $\forall x \forall y \exists z \big((p(x,y) \wedge p(y,z)) \to \neg p(x,z)\big)$
2. $\forall x \big(\forall y \exists z\ p(x,y,z) \wedge \exists z \forall y\ \neg p(x,y,z)\big)$
3. $\big(\exists x\ p(x,y)\big) \to \big(\exists x\ q(x,x)\big)$

Herbrand Theories

Until now we considered arbitrary interpretations of formulae in predicate logics. In particular we sometimes used numbers as interpretation domain and functions, like addition or successor. In the following, we will concentrate on a special case, the Herbrand interpretation and we will discuss their relation to the general case.

Definition 11 *Let S be a set of clauses. The* Herbrand universe *U_H for S is given by:*

- *All constants which occur in S are in U_H (if no constant appears in S, we assume a single constant, say a to be in U_H).*
- *For every n-ary function symbol f in S and every $t_1, \cdots, t_n \in U_H$, $f(t_1, \cdots, t_n) \in U_H$.*

Examples: Given the clause set $S_1 = \{P(a), \neg P(x) \vee P(f(x))\}$, we construct the Herbrand universe

$$U_H = \{a, f(a), f(f(a)), f(f(f(a))), \cdots\}$$

For the clause set $S_2 = \{p(x) \vee q(x), r(z)\}$ we get the Herbrand universe $U_H = \{a\}$.

Definition 12 *Let S be a set of clauses. An interpretation $\mathcal{I} = (U_\mathcal{I}, A_\mathcal{I})$ is an* Herbrand interpretation *iff*

- $U_\mathcal{I} = U_H$
- *For every n-ary function symbol ($n \leq 0$) f and $t_1, \cdots, t_n \in U_H$*

$$f^\mathcal{I}(t_1, \cdots, t_n) = f(t_1, \cdots, t_n)$$

Note, that there is no restriction on the assignements of relations to predicate symbols (except, that, of course, they have to be realtions over the Herbrand universe U_H).

In order to discuss the interpretation of predicate symbols, we need the notion of Herbrand basis.

Definition 13 *A* ground atom *or a* ground term *is an atom or a term without an occurrence of a variable.*
The Herbrand basis *for a set of clauses S is the set of ground atoms $p(t_1, \ldots, t_n)$, where p is a n-ary predicate symbol from S and $t_1, \ldots, t_n \in U_H$*

We will notate the assignments of relations to predicate symbols by simply giving a set $I = \{m_1, m_2, \cdots, m_n, \cdots\}$, where each element is a literal with its atom is from the Herbrand basis.
Examples:

- $S = \{p(x) \vee q(x), r(f(y))\}$
- $U_H = \{a, f(a), f(f(a)), \ldots\}$
- $I = \{p(a), q(a), \neg r(a), p(f(a)), q(f(a)), \neg r(f(a)), \cdots\}$

Definition 14 *Let $\mathcal{I} = (U_\mathcal{I}, A_\mathcal{I})$ be an interpretation for a set of clauses S; the Herbrand interpretation $\mathcal{I}^*$ corresponding to $\mathcal{I}$ is a Herbrand interpretation satisfying the following condition:*
Let $t_1, \ldots, t_n$ be Elements from the Herbrand universe U_H for S. By the interpretation $\mathcal{I}$ every t_i is mapped to a $d_i \in U_\mathcal{I}$. If $p^\mathcal{I}(d_1, \ldots, d_n) = true(false)$, then $p(t_1, \ldots, t_n)$ have to be assigned $true(false)$ in $\mathcal{I}^*$.

Let us know state a simple, very obvious lemma, which will help us, to focus on Herbrand interpretation in the following.

Lemma 4 *If $\mathcal{I}$ is a model for a set of clauses, then every corresponding Herbrand interpretation $\mathcal{I}^*$ is a model for S*

Theorem 4 *A set of clauses S is unsatisfiable iff there is no Herbrand model for S.*

Proof: If S is unsatisfiable, there is obviously no Herbrand model for S.
Asume that there is no Herbrand model for S and that S is satisfiable. Then, there is a model $\mathcal{I}$ for S and according to lemma 4 the corresponding Herbrand interpretation $\mathcal{I}^*$ is a model for S, which is a contradiction.

Semantic Trees

In this section we introduce the concept of semantic trees, which, then can be used to proof completeness of resolution. For this we asume familiarity with the concepts of trees, binary trees and paths in trees.

Definition 15 *A* semantic tree *for a set of clauses S is a binary tree T with root N_0, such that the edges are label with literals, build from elements from the Herbrand basis of S, such that:*

- *If N is an inner node, its two outgoing endges are label with complementary literals A and $\neg A$.*
- *Every path to a node N in T does not contain complementary literals in $I(N)$, where $I(N)$ is the set of literals which are labels along the edges of the path.*

Definition 16 *A semantic tree is called complete, if every path contains every atom from the Herbrand basis either positiv or negativ.*

Example: $S = \{p(x), q(f(x))\}$ with Herbrand basis

$$\{p(a), q(a), p(f(a)), q(f(a)), p(f(f(a))), \ldots\}$$

Note, that for a semantic tree T and a node N the set $I(N)$ can be seen as an assignement of truth-values to ground atoms, as it is done in an Herbrand interpretation. Hence we call $I(N)$ a partial interpretation. A complete semantic tree corresponds to an exhaustive "enumeration" of interpretations.

Definition 17 • *A node N of a semantic tree T is a* failure node *if $I(N)$ falsifies some ground instance of a clause in S, but $I(N')$ does not falsifiy any ground instance of a clause in S for every ancestor node N' of N.*

- *A semantic tree T is called* closed *if every path contains a faliure node.*
- *A node N of a closed semantic tree is called an* inference node*, if both immediate descendant nodes are failure nodes.*

Example: $S = \{p(x), \neg p(x) \vee q(f(x)), \neg q(f(a))\}$ with Herbrand basis

$$\{p(a), q(a), p(f(a)), q(f(a)), p(f(f(a))), \ldots\}$$

Let T be a complete semantic tree, than we call T' the correspondig closed tree, if it is obtainable from T by cutting all branches at a failure node.

Theorem 5 (Herbrand's Theorem – Version 1) *A set S of clauses is unsatisfiable, iff for every complete semantic tree for S there is a correspondign finite closed semantic tree.*

Proof: Asume S to be unsatisfiable and T a complete semantic tree for S. For every path P we have the set of labels I_B, which is an interpretation, because the tree is complete. Hence, I_B falsifies a gorund instance C' of a clause $C \in S$, because S is unsatisfiable. Since there are only finitely many literals in C', there must be a failure node N_B in a finite distance from the root. Since every path has such a failure node, there is a corresponding closed semantic tree T', which is finite.
For the opposite direction, assume that for every complete semantic tree T there is a finite closed corresponding tree T'. Then, every path contains a failure node, and hence, every interpretation falifies S; hence S is unsatisfiable.

Theorem 6 (Herbrand's Theorem – Version 2) *A set S of clauses is unsatisfiable, iff there is a finite unsatisfiable set S' of ground instances of clauses in S.*

Proof: Asume S to be unsatisfiable and T a complete semantic tree for S. By Herbrand's theorem, version 1, there is a finite closed semantic tree T' for S. Let S' be the set of ground instances of clauses, which are falsified at all failure nodes of T'. S' is finite and is falisfied by every interpretion and hence unsatisfiable.
The opposite direction we show by contraposition:
Note, that this version of Herbrand's theorem can be turned directly into a proof procedure:
Given a set of clauses S, for which we want to proof unsatisfiability.

- Generate $S'_1, \ldots, S'_n, \ldots$ sets of ground instances of clauses of S. Perform a propositional test for unsatisfiability on each of them.
- According to Herbrand's theorem, there is a finite S'_N which is unsatisfiable, if S is unsatisfiable.

1. **Problem 1**
   Assume the following set of clauses:

$$M_0 = \{\{p(x)\}, \{q(x, f(x)), \neg p(x)\}, \{\neg q(g(y), z)\}\}$$

(a) Indicate (a) the Herbrand-universe and (b) the Herbrand-basis for M_0!

(b) Give a closed semantic tree for M_0!

2. **Problem 2**
The following formulae F and G are given.

$$\begin{aligned} F &= \forall y\ p(y) \wedge \exists z\ \neg p(z) \\ G &= r(a) \wedge \forall x(r(x) \rightarrow r(f(x))) \end{aligned}$$

Give for F and G (a) a finate and (b) an infinite Herbrand model, or argument if this is not possible.

3. **Problem 3**
The following sets of clauses are given:

(a) $M_1 = \big\{\{p(a,x)\}, \{\neg p(y,b), q(y)\}\big\}$

(b) $M_2 = \big\{\{r(x), r(f(x))\}\big\}$

Give for each of them (a) a Herbrand universe, (b) Herbrand model and (c) a non-Herbrand model.

Resolution

In the propositional case we defined the resolution inference rule by "cutting away" a pair of complementary literals in two clauses which are resolved upon. In the first order case however this is not always sufficient:

$$\begin{aligned} C_1 :&\quad p(x) \vee q(x) \\ C_2 :&\quad \neg p(f(x)) \end{aligned}$$

In these two clauses there are no complementary literals, however, after substituting the term $f(a)$ for the variable x in C_1 and a for x in C_2 we arrive at:

$$\begin{aligned} C_1' :&\quad p(f(a)) \vee q(f(((a)) \\ C_2' :&\quad \neg p(f(a)) \end{aligned}$$

Now we can apply the inference rule from propositional logic and arrive at the resolvent $q(f(a))$.
Another possibility is to substitute $f(x')$ for x in C_1 to get

$$C_1'' :\quad p(f(x')) \vee q(f(x'))$$

and then we can have the resolvent $q(f(x'))$ from C_1'' and C_2, which is in a certain sense more general then the resolvent derived previously.

Definition 18 *A* substitution σ *is a function, which maps variables to terms and which is the identical mapping almost everywhere. Hence it can be represented as*

$$\sigma = \{x_1/t_1, \cdots, x_n/t_n\}$$

If $t_1, \cdots, t_n$ *are groundterms, we call* σ *a* ground substitution. *The* empty subsitution *is notated by* ϵ *.*

Definition 19 *Let* $\theta = \{x_1/t_1, \cdots, x_n/t_n\}$ *be a subsitution and* E *an expression (i.e. a literal or a term), then* $E\theta$ *is the expression, obtained from* E *by replacing simultaneously each occurence of* $X_i, 1 \leq i \leq n$ *in* E *by the term* t_i*.*

Example:
With $\theta = \{x/a, y/f(b), z/e\}$ and $E = p(x, y, z)$, we get $E\theta = p(a, f(b), c)$

Definition 20 *Let* $\sigma = \{x_1/t_1, \cdots, x_n/t_n\}$ *and* $\lambda = \{y_1/s_1, \cdots, y_m/s_m\}$ *be substitutions. Then the* composition of substitutions, *denoted by* $\sigma \circ \lambda$*, is the substitution, which is obtained from* $\{x_1/t_1\lambda, \cdots, x_n/t_n\lambda, y_1/s_1, \cdots, y_m/s_m\}$ *by deleting any element* $x_j/t_j\lambda$ *for which* $t_j\lambda = x_j$ *and any element* y_i/s_i *such that* $y_i \in \{x_1, \cdots, x_n\}$ *.*

Example:

Definition 21 *Let* $\{E_1, \cdots, E_n\}$ *be a set of expressions and* θ *a substitution,* θ *is* unifier *for* $\{E_1, \cdots, E_n\}$ *iff*

$$E_1\theta = E_2\theta = \cdots E_n\theta$$

.
A unifier θ *is called* most general unifier *iff for every unifier* σ *there is a substitution* λ *such that* $\sigma = \theta \circ \lambda$*.*

In the following we discuss an algorithm for computing most general unifiers. For this we assume a set of terms $\{t_1, \cdots, t_n\}$ to be unified. First we transform this into a set of equations by introducing a new variable not yet occurring in this set, say y and by defining the set of equations

$$N = \{y = t_1, \cdots, y = t_n\}$$

We will now transform this set such that its unifiers stay invariant, where a σ is a unifier of a set of $\{s_1 = t_1, \cdots, s_n = t_n\}$ if $\{s_1\sigma = t_1\sigma, \cdots, s_n\sigma = t_n\sigma\}$ holds.

Unification

Given a set of expression. Transform it into a set of equations N as defined above. Apply the following transformation rules as long as possible:

1. $\dfrac{R \uplus \{t = x\}}{R \uplus \{x = t\}}$ *Orient*
   where x is a variable and t a non-variable term

2. $\dfrac{R \uplus \{s = s\}}{R}$ *Delete*

3. $\dfrac{R \uplus \{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\}}{R \uplus \{s_1 = t_1, \ldots, s_n = t_n\}}$ *Decompose* (Termreduction)

4. $\dfrac{R \uplus \{x = t\}}{R[x/t] \uplus \{x = t\}}$ *Eliminate* (Elimination of variable I)
   if x not in t, but in R

5. $\dfrac{R \uplus \{x = y\}}{R[x/y] \uplus \{x = y\}}$ *Coalesce* (Elimination of variable II)
   if $x \neq y$ in R

6. $\dfrac{R \uplus \{f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)\}}{\text{FAIL}}$ *Conflict*
   if $f \neq g$ or $m \neq n$

7. $\dfrac{R \uplus \{x = t\}}{\text{FAIL}}$ *Occur Check*
   if x in t

Interaction: Unification

This is your set of current terms:

Currently empty.

Current terms manipulation:

Enter some terms: Help:

Add to / **Overwrite** current terms

Delete selected / **Delete all** current formulas

Save/Reload current term:

Save to database (enter name):

Reload from database (select name):

Reload from file (enter file name):

Unify selected / **Unify all** terms.

This is the result, (possibly from a previous run): Currently none.

Theorem 7 *Let N be a set of expressions. The above unification algorithm teminates. If it returns $FAIL$, there is no unifier fo otherwise N is transformed into a set of equation $\{y_1 = u_1, \cdots, y_m = u_m\}$, which represents the most general unifier for N.*

Definition 22 *Let two or more literals of a clause C have a unifier σ, then $C\sigma$ is called a* factor *of C.*

Example:
With $C = \{p(x), p(f(y)), \neg q(x)\}$ and $\sigma = \{x/f(y)\}$ we get the factor $C\sigma = \{p(f(y)), \neg q(f(y))\}$

Definition 23 *Let C_1 and C_2 be two clauses with no variables in common, such that $L_1 \in C_1$ and $L_2 \in C_2$ and L_1 and L_2 have a most general unifier σ. A* binary resolvent *of C_1 and C_2 is*

$$(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$$

Example:
Given $C_1 = \{p(x), q(x)\}$ and $C_2 = \{\neg p(a), r(x)\}$. After renaming C_2 into $C_2 = \{\neg p(a), r(y)\}$ we get the resolvent $\{q(a), r(y)\}$ by using the most general unifier $\{x/a\}$.
We often depict resolvent graphically, e.g.

$$\begin{tikzcd}[column sep=small] \{p(x), q(x)\} \arrow[dr, dash] & & \{\neg p(a), r(x)\} \arrow[dl, dash] \\ & \{q(a), r(y)\} & \end{tikzcd}$$

Definition 24 *A* resolvent *of two clauses C_1 and C_2 is one of the following binary resolvents:*

- *a binary resolvent of C_1 and C_2*

- *a binary resolvent of C_1 and a factor of C_2*
- *a binary resolvent of a factor of C_1 and C_2*
- *a binary resolvent of a factor of C_1 and a factor of C_2*

Example:
Given $C_1 = \{p(x), p(f(y)), r(g(y))\}$ and $C_2 = \{\neg p(f(g(a))), q(b)\}$.
A factor of C_1 is $C_1\prime = \{p(f(y)), r(g(y))\}$. A binary resolvent of $C_1\prime$ and C_2 and hence also of C_1 and C_2 is $C_3 = \{r(g(g(a))), q(b)\}$.
Example:
You can use the theorem prover Otter to experiment with resolution:

Interaction: Otter

For informations about Otter look here. First, we need some clauses.
This is our current set of clauses:

Currently empty.

Current clauses manipulation:

Enter some clauses:

Help:

Add to / **Overwrite** current clauses

Delete selected / **Delete all** current clauses

Convert selected / **Convert all** current predicate logic formulas to CNF, replacing the current clauses.

Save/Reload current clauses:

Save to database (enter name):

Reload from database (select name):

Reload from file (enter file name):

Apply Otter to the current clauses **Delete current Otter Result**

This is the result, (possibly from a previous run):
Currently none.
The following lemma is used in the completeness proof of resolution.

Lemma 5 (Lifting lemma) *If $C_1\prime$ and $C_2\prime$ are instances of C_1 and C_2, respectively, and $C\prime$ is a resolvent of $C_1\prime$ and $C_2\prime$, then there is a resolvent C of C_1 and C_2 such that $C\prime$ is an instance of C.*

$$\begin{tikzcd}
C_1' \arrow[dr, dash] & & C_2' \arrow[dl, dash] & & C_1 \arrow[llll, dashed, bend right, "\sigma"'] \arrow[dr, dash] & & C_2 \arrow[llll, dashed, bend right, "\sigma"'] \arrow[dl, dash] \\
& C' & & & & C \arrow[llll, dashed, "\theta"'] &
\end{tikzcd}$$

Theorem 8 *A set S of clauses is unsatisfiable iff the empty clause can be derived from S by resolution.*

Proof:
Assume that S is unsatisfiable. Let $A = \{A_1, A_2, \ldots\}$ be the ground atom set of S, hence the Herbrand basis. Let T be a complete binary tree, as given in Figure **??**. According to Herbrand's theorem (version1) 5 there exists a closed finite semantic tree T'. There are two cases:

- If T' consists only of one node (hence the root), The interpretation to be collected from the empty branch in this tree falsifies only the empty clause. Hence the empty clause must be in S.

- Assume T' consists of more than one node. Then there must be an inference node N in T', hence both its descendants N_1 and N_2 are failure nodes. If such a node would not exist, every node would have at least one non-failure node, which would mean that there is at least an infinite path in T', which would violate, that fact that it is a finite closed semantic tree.

  Let N, N_1, N_2 given as described above; and let

  $$\begin{aligned} I(N) &= \{m_1, m_2, \ldots, m_n\} \\ I(N_1) &= \{m_1, m_2, \ldots, m_n, m_n, m_{n+1}\} \\ I(N_2) &= \{m_1, m_2, \ldots, m_n, m_n, \neg m_{n+1}\} \end{aligned}$$

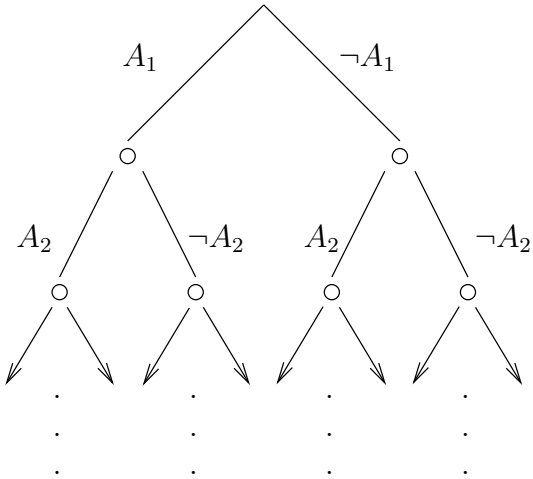
  Now, let C_1' and C_2' be ground instances of clauses C_1 and C_2, such that C_1' is falsified by $I(N_1)$ and C_2' by $I(N_2)$, such that both are not falsified by $I(N)$. Hence we have $\neg m_{n+1} \in C_1'$ and $m_{n+1} \in C_2'$ and we can construct the resolven

  $$C' = (C_1' - \{\neg m_{n+1}\}) \cup (C_2' - \{m_{n+1}\})$$

  C' must be false in $I(N)$, because both $(C_1' - \neg m_{n+1})$ and $(C_2' - m_{n+1})$ are false in $I(N)$. According to the Lifting Lemma 5 there exists a resolvent C of C_1 and C_2, such that C' is a ground instance of C. Let T'' be the closed semantic tree for $S \cup \{C\}$, obtained from T' by deleting all nodes below the first node which falsifies C'. Note, that S is unsatisfiable if and only if $S \cup \{C\}$ is unsatifiable. Clearly, T'' has less nodes than T' and we now can iterate this process until ony the root of the semantic tree is remaining. This, however is only possible if the empty clause $\square$ is derivable.

For the opposite direction, assume that $\square$ is derivable by resolution from S and let $R_1, \ldots, R_k$ the resolvents constructed during thisn process. Assume S is satisfiable and M to be a model for S. From the correctness lemma according to the propositional case we known, that if a model satifies two clauses it also satisfies its resolvent. Therefore M has to satisfy $R1, \ldots, R_k$; this, however, is impossible, because one of this resolvents is $\square$.

T :



Problem 1
Indicate in each case a derivation of the empty clause with predicate-logical resolution!

1. $\{\{p(x,0,x)\},\{p(x,s(y),s(z)),\neg p(x,y,z)\},\{\neg p(s(s(s(0))),s(s(0)),u)\}\}$
2. $\{\{q(x),q(s(x))\},\{\neg q(x),\neg q(s(s(x)))\}\}$ $(\star)$
3. $\{\{\neg r(x,f(x),y),\neg r(x,g(y),z)\},\{r(c,u,i(v)),r(h(u),v,j(v))\}\}$

Problem 2
Show the following *Lifting lemma* by means of induction over the term- and formula construction:
Is F a predicate-logical formula, and $\mathcal{I}$ a fiiting interpretation for F and $F[x/t]$. Then

$$\mathcal{I}(F[x/t]) = \mathcal{I}_{[x/\mathcal{I}(t)]}(F),$$

is valid, if t does not contain any variable that $[x/t]$ is laced by the substitution in F.
Problem 3
Compute - if possible - the most general unifier of following sets of clauses:

1. $\{p(x,a),p(f(c),y)\}$
2. $\{p(f(x),a,x),p(y,z,z)\}$
3. $\{q(x,x),q(g(y),y)\}$
4. $\{r(x,x),r(a,h(y))\}$

Problem 4
Determine all direct resolvents of the following pairs of clauses:

1. $\{\neg p(x),q(x,b)\}$ und $\{p(a),q(a,b)\}$
2. $\{p(x),p(f(x))\}$ und $\{\neg p(x),\neg p(f(f(x)))\}$
3. $\{\neg q(c,g(c))\}$ und $\{\neg p(x),q(x,x)\}$
4. $\{\neg p(x,y,z),\neg p(y,u,v),\neg p(x,v,w),p(z,u,w)\}$ und $\{p(g(x,y),x,y)\}$

Problem 5
Compute - if possible - the most general unifier of following sets of clauses:

1. $\{o(x,x), o(a,f(y))\}$
2. $\{p(x,a), p(f(c),y)\}$
3. $\{q(g(x),a,x), q(y,z,z)\}$
4. $\{r(x,x), r(h(y),y)\}$

Problem 6
Determine all direct resolvents of the following pairs of clauses:

1. $\{\neg p(x), \neg p(b), q(x,b)\}$ and $\{p(a), q(a,b)\}$
2. $\{r(x), r(f(x))\}$ and $\{\neg r(x), \neg r(f(f(x)))\}$
3. $\{\neg s(c,g(c))\}$ and $\{s(x,x), \neg t(x)\}$

Problem 7
Give for the following set of clauses (a) a linear derivation, (b) a derivation with unit resolution, (c) a further (maximally short) derivation of the empty clause by means of predicate-logical resolution!

$$\{\{\neg e(x), o(s(x))\}, \{\neg e(x), \neg o(s(x)), e(s(s(x)))\}, \{e(a)\}, \{\neg o(s(s(s(s(s(a))))))\}\}$$

Problem 8
Indicate in each case a derivation of the empty clause with predicate-logical resolution!

1. $\{\{p(x,0,x)\}, \{p(x,s(y),s(z)), \neg p(x,y,z)\}, \{\neg p(s(s(s(0))), s(s(0)), u)\}\}$
2. $\{\{q(x), q(s(x))\}, \{\neg q(x), \neg q(s(s(x)))\}\}$ $(\star)$
3. $\{\{\neg r(x,f(x),y), \neg r(x,g(y),z)\}, \{r(c,u,i(v)), r(h(u),v,j(v))\}\}$

Strategies for Resolution

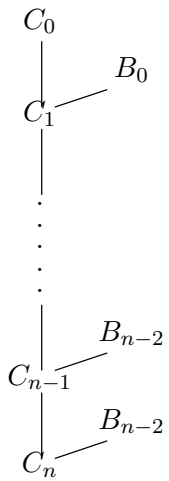
Linear Resolution In contrast to the saturation-based procedure, which we gave for propositional resolution, we will discuss now a stargegy which allows goal directed generation of resolvents. We will see later, nameöly in the case of Horn clauses, that this linear strategy, is the basis for the interpretation of logic programs.

Definition 25 *Given a set of clauses S and a clause C_0 in S. A* linear deduction *of* top clause C_0 *is a sequence $C_0, C_1, \cdots, C_n$, where $\forall 1 \leq i \leq n$ C_i is a resolvent of C_i and B with $B \in S \cup \{C_j \mid j < i\}$.*
If $C_n = \square$ the sequence is called a linear refutation.

The following is an example for a linear deduction. The clause set $S$is given by:

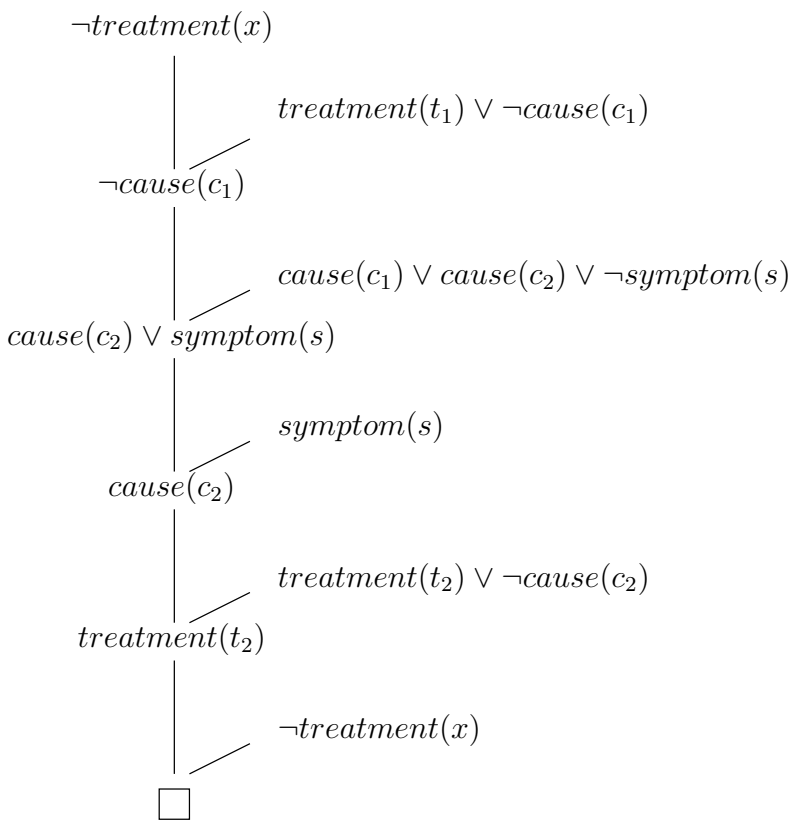
$$
\begin{aligned}
symptom(s) &\quad (1)\\
cause(c_1) \vee cause(c_2) \vee \neg symptom(s) &\quad (2)\\
treatment(t_0) \vee \neg cause(c_1) &\quad (3)\\
treatment(t_1) \vee \neg cause(c_1) &\quad (4)\\
treatment(t_0) \vee \neg cause(c_2) &\quad (5)\\
treatment(t_2) \vee \neg cause(c_2) &\quad (6)
\end{aligned}
$$

and together with the goal $\neg treatment(x)$, we get the following refutation, where clauses frOM S are given by the respective numbers:
$\neg treatment(X)$, (4),$\neg cause(c_1)$, (2), $cause(c_2) \vee \neg symptom(s)$, (1), $cause(c_2)$, (6), $treatment(t_2)$,$\square$
the same refutation can be given more naturally be the following picture:
The following theorem states correctness and completeness of linear resolution. Note that completeness only states that there exists a linear refutation, there is no guaranty that every clause in the sequence really is necessary to derive the empty clause.



Theorem 9 *Linear resolution is complete and correct.*

Example



Input and Unit Resolution

Definition 26 *Given a set of clauses S. The inference rule* input resolution *is resolution, such that one parent clause is a clause from S.*
The inference rule unit resolution *is resolution, such that at least one parent clause is a unit clause or a unit factor of a parent clause.*

In an obvious way the notions of unit (input) derivations and refutations are defined.

Theorem 10 *For a set S of clauses there exists a unit refutation iff there exists an input refutation.*

Unit resolution (and hence input resolution) is not complete for full first order logics! For Horn clauses, however, it is a complete strategy and indeed, it is the basis for the SLD-resolution principle, which is the core of

SLD-Resolution In this section we will introduce a special form of linear resolution for Horn clauses. We will intreprete a clause a program by notationg it in the following form:

A	$\leftarrow$		program clause (fact)
A	$\leftarrow$	$B_1, \ldots, B_n$	program clause
	$\leftarrow$	$B_1, \ldots, B_n$	goal

Definition 27 *Let P be a set of program clauses. Assume a* selection function*, which gives for a given goal $\leftarrow A_1, \ldots, A_n$ one of its subgoals A_i.*
Further assume a goal $G_i = \leftarrow A_1, \ldots, A_m, \ldots, A_n$ and a selection function which selects A_m. Let $C_i = A \leftarrow B_1, \ldots, B_q$ be a variant of a clause in P, such that C_i and G_i have no variable in common. If θ_{i+1} is most general unifier of A_m and A, the goal

$$G_{i+1} = \leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_n)\theta_{i+1}$$

is called SLD-resolvent

Definition 28 *An* SLD-deduction *(-refutation)of $P \cup \{G\}$ for a set of program clauses P and a goal clause G is a linear deduction (refutation) in which only SLD-resolution steps occur and G is the start clause.*

- *An* R-computed answer subsitution *θ for $P \cup \{G\}$ is $\theta_1 \circ \ldots \circ \theta_n|_{Var(G)}$, where $\theta_1, \ldots, \theta_n$ are the mgUs from a SLD-refutation of $P \cup \{G\}$ with selection function R.*
- *A substitution θ for $Var(G)$ is an* answer substitution *for $P \cup \{G\}$.*
- *It is a* correct answer substitution *for $P \cup \{G\}$, if $P \models \forall G\theta$*

Theorem 11 *Let P be a set of program clauses, G a goal clause and R a selection function. For every correct answer substitution θ for $P \cup \{G\}$, there is an R-computed answer subsitution σ for $P \cup \{G\}$ and a substitution γ, such that $\theta = \sigma \circ \gamma \mid_{Var(G)}$*

Model Elimination

In the Section on Propositional Logic we already explained propositional tableaux and its variants, like the connection calculus and model elimination. In this section we will give model elimination in the first order case. Note that we need one more inference rule, the reduction rule, in this case

Definition 29 *A* clause (normalform) tableau *for a set of clauses S is a tableau for S, whose nodes are literals from S and which is constructed by a (possibly infinite) sequence of applications of the following rules:*

- *The tree consisting of root true and immediate successors $L_1, \cdots, L_n$, where $C = L_1, \cdots, L_n$ is a new variant of a clause from S is a tableau for S (initialisation rule).*
- *Let T be a tableau for S, B a branch of T, and $C = L_1, \cdots, L_n$ an new variant of a clause from S, such that the link-condition with mgu σ is satisfied. If the tree T' is constructed by extending B by the n subtrees L_i, then $T'\sigma$ is a tableau for S (expansion rule).*
- *Let T be a tableau for S, B a branch of T, L a leaf of B, and $L' \in C$, such that $\overline{L'}$ and L have a mgu σ, than $T\sigma$ is a tableau for S (reduction rule).*

The following are three possible link conditions:

1. No condition.
2. *Weak link condition*: There is a literal $L \in B$ and $L' \in C$, such that $\overline{L'}$ and L have a mgu σ
3. *Strong link condition*: There is a leaf L of B, and $L' \in C$, such that $\overline{L'}$ and L have a mgu σ .

Analog to the propositinal case the different link conditions result in different calculi:

- The empty condition results in a clause normal form tableau calculus.
- The weak condition results in a *connection calculus*.
- The strong link condition results in a *model elimination calculus*.

A Prolog-like Implementation

```
% gprove(G,A) is true if G <- A follows from KB
gprove(true,_).
gprove((G & H),A):-
   gprove(G,A),
   gprove(H,A).
gprove(G,A):-
   member(G,A).
gprove(G,A):-
  (G <- B),
   neg(G,NG),
   gprove(B,[NG|A]).

% neg denotes the negation of atoms

% to prove a theorem do
% gprove(yes,[]).

% (a) Set of contrapositives

 p(a,X) <- ~p(b,Y) & q(X,Y).
 p(b,Y) <- ~p(a,X) & q(X,Y).
~q(X,Y) <- ~p(a,X) & ~p(b,Y).
 p(Z,Z) <- true.
 q(b,a) <- true.
 yes    <- p(U,V).
~p(U,V) <- ~yes.
```

Iterativ deepening Iterative deepening is a complete search strategy, which combines depth-first with breadth-first search.

- $depthbound = 1$
- do while (not found)
  Search by limited depth-first search until *depthbound*.
  Set $depthbound = depthbound + 1$

Number of expansions in a tree with branching factor b until depth d is

$$1 + b + b^2 + \cdots + b^{d-1} + b^d$$

The total number of expansions in iterative deepening search is

$$(d+1) * 1 + d * b + (d-1) * b^2 + \cdots + 3 * b^{d-2} + 2 * b^{d-1} + 1 * b^d$$

Hence time complexity of iterative deepening is still $O(b^d)$.

PTTP- Prolog Technology Theorem Proving

As exemplified by PTTP ("Prolog Technology Theorem Prover") [**?**, **?**], Prolog can be viewed as an "almost complete" theorem prover, which has to be extended by only a few ingredients in order to handle the non-Horn case. By this technique the benefits of optimizing Prolog compilers are accessible to theorem proving. First we will briefly review the standard approach, and then we will describe the necessary modifications to obtain restart model elimination.
The PTTP-approach transforms a given clause set into a Prolog program. The transformed Prolog program must execute the clauses according to some complete proof procedure. *Model elimination* turns out to be particularly useful for this, since it is, like Prolog, an input proof procedure. In particular, the transformation from the input clauses to Prolog works as follows:

- An input clause such as

  $$C \leftarrow A \wedge B$$

  is transformed into a Prolog clause

  ```
  (1)     c :- a, b.
  ```

  Additionally, since in the model elimination calculus every literal in a clause can equally well serve as an entry point into the clause, all contrapositives are needed. In this case these are

  ```
  (2)     not_a :- not_c, b.
  (3)     not_b :- a, not_c.
  ```

  This example also shows how negation is treated, namely by making it part of the predicate name.

- Prolog's unsound unification has to be replaced by a sound unification algorithm. This can either be done by directly building-in sound unification into the Prolog implementation, or by reprogramming sound unification in Prolog and calling this code instead of Prolog's unsound unification.

- A complete search strategy is needed. Usually depth bounded iterative deepening is used. The strategy can be compiled into the prolog program by additional parameters, being used as "current depth" and "limit depth". The cost of an extension step can be uniformly 1 (depth bounded search), or can be proportional to the length of the input clause (inference bounded search).

- The model elimination reduction operation has to be implemented. This can be realized by memorizing the subgoals solved so far (the A-literals) as a list in an additional argument, and by Prolog code that checks a goal for a complementary member of that list. Of course, this check has to be carried out with sound unification.

  The Prolog clause (1) from above then looks like

  ```
  (1')     c(Anc) :- a([-a|Anc]), b([-b|Anc]).
  ```

  where `Anc` is a Prolog list which contains the ancestor literals (called A-literals in [**?**]); code for reduction steps then looks like

  ```
  (Red-C)        c(Anc) :- member(c,Anc).
  (Red-notC)     not_c(Anc) :- member(-c,Anc).
  ```

Model Generation Theorem Proving

SATCHMO

The SATCHMO Theorem Prover was one of the first systems which used model generation, i.e. a bottom-up proof procedure. The prover was given by a small Prolog-program, which implements a tableau proof procedure. One restriction is that it requires range restricted formulae.

Definition 30 *A first order clause $A_1 \vee \cdots \vee A_n \leftarrow B_1 \wedge \cdots \wedge B_m$ is called* range restricted *if every variable which occurs in the head $A_1 \vee \cdots \vee A_n$ occurs in the body $B_1 \wedge \cdots \wedge B_m$ as well.*

1. Convert clauses to range restricted form:

   $$q(x) \vee p(x,y) \leftarrow q(x) \qquad \rightsquigarrow \qquad \texttt{q(X) ; p(X,Y) <- q(X), dom(Y)}$$

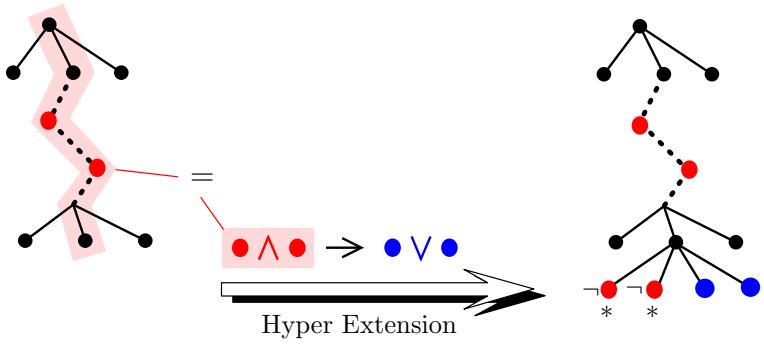
2. `assert` range-restricted clauses and `dom` clauses in Prolog database.

3. Call `satisfiable`:

```
satisfiable :-                      assume(X) :- asserta(X).
     (Head <- Body),                assume(X) :-
     Body, not Head, !,                  retract(X), !, fail.
     component(HLit, Head),         component(E, (E ; _)).
     assume(HLit),                  component(E, (_ ; R)) :-
     not false,                          !, component(E, R).
     satisfiable.                   component(E, E).
satisfiable.
```

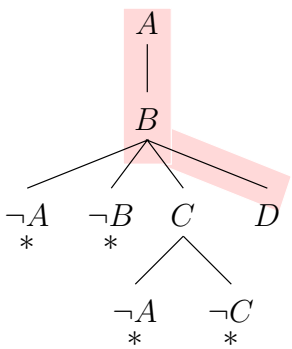
First-Order completeness via Level-Saturation modification.
This proof procedure implements Hyper Tableaux in the ground case.

Hyper Tableau - Ground Case



All open branches consist of positive literals only

Take the following clause set as an example
$\{\rightarrow A, \quad \rightarrow B, \quad A \wedge B \rightarrow C \vee D, \quad A \wedge B \rightarrow E \vee D, \quad A \wedge C \rightarrow\}$



Definition 31 (Literal tree, Clausal Tableau [?]) *A* literal tree *is a pair* (t, λ) *consisting of a finite, ordered tree* t *and a labeling function* λ *that assigns a literal to every non-root node of* t*. The* successor sequence *of a node* N *in an ordered tree* t *is the sequence of nodes with immediate predecessor* N*, in the order given by* t*.*
A (clausal) tableau T *of a set of clauses* $\mathcal{S}$ *is a literal tree* (t, λ) *in which, for every successor sequence* $N_1, \ldots, N_n$ *in* t *labeled with literals* $K_1, \ldots, K_n$*, respectively, there is a substitution* σ *and a clause* $\{L_1, \ldots, L_n\} \in \mathcal{S}$ *with* $K_i = L_i\sigma$ *for every* $1 \leq i \leq n$*.* $\{K_1, \ldots, K_n\}$ *is called a* tableau clause *and the elements of a tableau clause are called* tableau literals*.*

Definition 32 (Branch, Open and Closed Tableau, Selection Function) *A* branch *of a tableau* T *is a sequence* $N_0, \ldots, N_n$ $(n \geq 0)$ *of nodes in* T *such that* N_0 *is the root of* T*,* N_i *is the immediate predecessor of* N_{i+1} *for* $0 \leq i < n$*, and* N_n *is a leaf of* T*. We say branch* $b = N_0, \ldots, N_n$ *is a* prefix *of branch* c*, written as* $b \leq c$ *or* $c \geq b$*, iff* $c = N_0, \ldots, N_n, N_{n+1}, \ldots, N_{n+k}$ *for some nodes* $N_{n+1}, \ldots, N_{n+k}$*,* $k \geq 0$*.*
The branch literals *of branch* $b = N_0, \ldots, N_n$ *are the set* $lit(b) = \{\lambda(N_1), \ldots \lambda(N_n)\}$*. We find it convenient to use a branch in place where a literal set is required, and mean its branch literals. For instance, we will write expressions like* $A \in b$ *instead of* $A \in lit(b)$*.*
In order to memorize the fact that a branch contains a contradiction, we allow to label a branch as either open *or* closed*. A tableau is* closed *if each of its branches is closed, otherwise it is* open*.*

A selection function *is a total function f which maps an open tableau to one of its open branches. If $f(T) = b$ we also say that b* is selected in T by f.

Note that branches are always finite, as tableaux are finite.
Fortunately, there is no restriction on which selection function to use. For instance, one can use a selection function which always selects the "leftmost" branch.

Definition 33 (Hyper Tableau - Ground Case) *Let S be a finite set of clauses and f be a selection function.* Hyper tableaux *for S are inductively defined as follows:*
Initialization step: *A one node literal tree is a hyper tableau for S. Its single branch is marked as "open".*
Hyper extension step: *If*

1. *T is an open hyper tableau for S, $f(T) = b$ (i.e. b is selected in T by f) with open leaf node N, and*
2. *$C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is a clause from S ($m \geq 0$, $n \geq 0$), called* extending clause *in this context, and*
3. *such that $\{B_1, \ldots, B_n\} \subseteq b$ (referred to as* hyper condition*)*

then the literal tree T' is a hyper tableau for S, where T' is obtained from T by attaching $m + n$ child nodes $M_1, \ldots, M_m, N_1, \ldots, N_n$ to b with respective labels

$$A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n$$

and marking every new branch $(b, M_1), \ldots, (b, M_m)$ with positive leaf as "open", and marking every new branch $(b, N_1), \ldots, (b, N_n)$with negative leaf as "closed".

Minimal Model Reasoning

The clause set $M = \{A \vee B \leftarrow, \quad B \leftarrow A\}$ obviously has two different models: $\{A,\ B\}$ and $\{B\}$. Under set inclusion these models can be compared and there are some tasks where it is apropriate to compute the (or in general a) smallest one. This is for example the case with

- Knowledge Representation, Circumscription
- Basis for default negation (GCWA)
- Applications: Deductive database updates, Diagnosis

There are basically two different methods to compute minimal models.

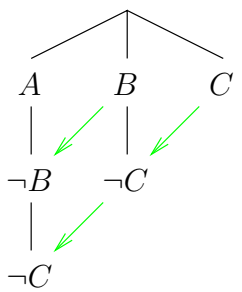
Minimal Model Reasoning – Niemelä's Approach

Given a set of ground clauses M the methods applies a model generating procedure, e.g. hyper tableau, which is able to generate all models.
Lemma 1: For every minimal model p for M there is a branch with literals p.
Asume that Σ is the set of atoms, which occur in the head of a clause from M, than the following Lemma holds.
Lemma 2: p is a minimal model for M iff $M \cup \{\neg A \mid A \in \Sigma \setminus p\} \models p$

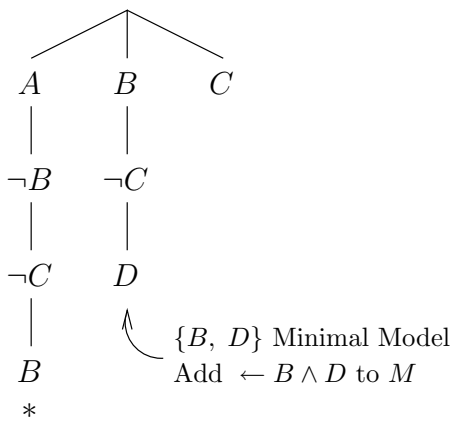
This offers a general method: Generate model candidates, and test with Lemma 2.
$p = \{A,\ B\}$ is not a minimal model in our example from above, because
$M \cup \{\} \models \{A,\ B\}$ iff $M \cup \{\leftarrow A \wedge B\}$ is unsatisfiable, which is not the case, hence p does not correspond to a minimal model and hence the branch is closed.
$p = \{B\}$ is minimimal because
$M \cup \{\leftarrow A\} \models \{B\}$ iff $M \cup \{\ \leftarrow A\} \cup \{\ \leftarrow B\}$ is unsatisfiable. This is the case and hence p is minimal and the branch remains open.
Properties: Soundness (by Lemma 2) Completeness (by Lemma 1), *space efficiency*.

Minimal Model Reasoning – Bry& Yayha's Approach

As an example we have the set $M = \{A \vee B \vee C \leftarrow, \quad B \leftarrow A, \quad D \leftarrow B\}$



Complement Splitting



Lemma: With complement splitting, the leftmost open branch is a minimal model for M.
General method: Repeat: generate minimal model p, add $\leftarrow p$ to M.
Properties: Soundness (by Lemma) Completeness as before,
possibly exponentially many new clauses $\leftarrow p$.
In the Section on Propositional Logic we already explained propositional tableaux and its variants, like the connection calculus and model elimination. In this section we will give model elimination in the first order case. Note that we need one more inference rule, the reduction rule, in this case

Definition 34 *A* clause (normalform) tableau *for a set of clauses S is a tableau for S, whose nodes are literals from S and which is constructed by a (possibly infinite) sequence of applications of the following rules:*

- *The tree consisting of root true and immediate successors $L_1, \cdots, L_n$, where $C = L_1, \cdots, L_n$ is a new variant of a clause from S is a tableau for S (initialisation rule).*
- *Let T be a tableau for S, B a branch of T, and $C = L_1, \cdots, L_n$ an new variant of a clause from S, such that the link-condition with mgu σ is satisfied. If the tree T' is constructed by extending B by the n subtrees L_i, then $T'\sigma$ is a tableau for S (expansion rule).*
- *Let T be a tableau for S, B a branch of T, L a leaf of B, and $L' \in C$, such that $\overline{L'}$ and L have a mgu σ, than $T\sigma$ is a tableau for S (reduction rule).*

The following are three possible link conditions:

1. No condition.
2. *Weak link condition*: There is a literal $L \in B$ and $L' \in C$, such that $\overline{L'}$ and L have a mgu σ
3. *Strong link condition*: There is a leaf L of B, and $L' \in C$, such that $\overline{L'}$ and L have a mgu σ .

Analog to the propositinal case the different link conditions result in different calculi:

- The empty condition results in a clause normal form tableau calculus.
- The weak condition results in a *connection calculus*.
- The strong link condition results in a *model elimination calculus*.

Protein

This is our current set of clauses:

Currently empty.

Current clauses manipulation:

Enter some clauses: Help:

Add to / **Overwrite** current clauses

Delete selected / **Delete all** current clauses

Convert selected / **Convert all** current predicate logic formulas to CNF, replacing the current clauses.

Save/Reload current clauses:

Save to database (enter name):

Reload from database (select name):

Reload from file (enter file name):

Apply Protein to the current clauses **Delete current Protein result**

This is the result, (possibly from a previous run):

Currently none.

A Prolog-like Implementation

```
% gprove(G,A) is true if G <- A follows from KB
gprove(true,_).
gprove((G & H),A):-
   gprove(G,A),
   gprove(H,A).
gprove(G,A):-
   member(G,A).
gprove(G,A):-
  (G <- B),
   neg(G,NG),
   gprove(B,[NG|A]).

% neg denotes the negation of atoms

% to prove a theorem do
% gprove(yes,[]).

% (a) Set of contrapositives

 p(a,X) <- ~p(b,Y) & q(X,Y).
 p(b,Y) <- ~p(a,X) & q(X,Y).
~q(X,Y) <- ~p(a,X) & ~p(b,Y).
 p(Z,Z) <- true.
 q(b,a) <- true.
 yes    <- p(U,V).
~p(U,V) <- ~yes.
```

Iterativ deepening Iterative deepening is a complete search strategy, which combines depth-first with breadth-first search.

- $depthbound = 1$
- do while (not found)
  Search by limited depth-first search until *depthbound*.
  Set $depthbound = depthbound + 1$

Number of expansions in a tree with branching factor b until depth d is

$$1 + b + b^2 + \cdots + b^{d-1} + b^d$$

The total number of expansions in iterative deepening search is

$$(d+1) * 1 + d * b + (d-1) * b^2 + \cdots + 3 * b^{d-2} + 2 * b^{d-1} + 1 * b^d$$

Hence time complexity of iterative deepening is still $O(b^d)$.

Layout

Format document for screen