Propositional and Predicate Logic - V

Petr Gregor

KTIML MFF UK

WS 2025/2026

Properties of theories

We introduce syntactic variants of previous semantically defined notions.

Let T be a theory over $\mathbb{P}$. If φ is provable from T, we say that φ is a *theorem* of T. The set of theorems of T is denoted by

$$\mathrm{Thm}^{\mathbb{P}}(T) = \{\varphi \in \mathrm{PF}_{\mathbb{P}} \mid T \vdash \varphi\}.$$

We say that a theory T is

- *inconsistent* if $T \vdash \bot$, otherwise T is *consistent*,
- *complete* if it is consistent and every proposition is provable or refutable from T, i.e. $T \vdash \varphi$ or $T \vdash \neg\varphi$ for every $\varphi \in \mathrm{PF}_{\mathbb{P}}$,
- *extension* of a theory T' over $\mathbb{P}'$ if $\mathbb{P}' \subseteq \mathbb{P}$ and $\mathrm{Thm}^{\mathbb{P}'}(T') \subseteq \mathrm{Thm}^{\mathbb{P}}(T)$; we say that an extension T of a theory T' is *simple* if $\mathbb{P} = \mathbb{P}'$; and *conservative* if $\mathrm{Thm}^{\mathbb{P}'}(T') = \mathrm{Thm}^{\mathbb{P}}(T) \cap \mathrm{PF}_{\mathbb{P}'}$,
- *equivalent* with a theory T' if T is an extension of T' and vice-versa.

Corollaries

From the soundness and completeness of the tableau method it follows that these syntactic definitions agree with their semantic variants.

Corollary *For every theory T and propositions φ, ψ over $\mathbb{P}$,*

- $T \vdash \varphi$ *if and only if* $T \models \varphi$,
- $\mathrm{Thm}^{\mathbb{P}}(T) = \theta^{\mathbb{P}}(T)$,
- T *is inconsistent if and only if* T *is unsatisfiable, i.e. it has no model,*
- T *is complete if and only if* T *is semantically complete, i.e. it has a single model,*

Theorem on compactness

Theorem *A theory T has a model iff every finite subset of T has a model.*

Proof 1 The implication from left to right is obvious. If T has no model, then it is inconsistent, i.e. $\bot$ is provable by a systematic tableau τ from T. Since τ is finite, $\bot$ is provable from some finite $T' \subseteq T$, i.e. T' has no model. □

Remark *This proof is based on finiteness of proofs, soundness and completeness. We present an alternative proof (applying König's lemma).*

Proof 2 Let $T = \{\varphi_i \mid i \in \mathbb{N}\}$. Consider a tree S on (certain) finite binary strings σ ordered by being a prefix. We put $\sigma \in S$ if and only if there exists an assignment ν with prefix σ such that $\nu \models \varphi_i$ for every $i \leq \mathrm{lth}(\sigma)$.

Observation *S has an infinite branch if and only if T has a model.*

Since $\{\varphi_i \mid i \leq n\} \subseteq T$ has a model for every $n \in \mathbb{N}$, every level in S is nonempty. Thus S is infinite and moreover binary, hence by König's lemma, S contains an infinite branch. □

Application of compactness

A graph (V, E) is *k-colorable* if there exists $c\colon V \to \{1, \ldots, k\}$ such that $c(u) \neq c(v)$ for every edge $\{u, v\} \in E$.

Theorem *A countably infinite graph $G = (V, E)$ is k-colorable if and only if every finite subgraph of G is k-colorable.*

Proof The implication $\Rightarrow$ is obvious. Assume that every finite subgraph of G is k-colorable. Consider $\mathbb{P} = \{p_{u,i} \mid u \in V, 1 \leq i \leq k\}$ and a theory T with axioms

$$p_{u,1} \vee \cdots \vee p_{u,k} \qquad \text{for every } u \in V,$$
$$\neg(p_{u,i} \wedge p_{u,j}) \qquad \text{for every } u \in V, i < j \leq k,$$
$$\neg(p_{u,i} \wedge p_{v,i}) \qquad \text{for every } \{u, v\} \in E, i \leq k.$$

Then G is k-colorable if and only if T has a model. By compactness, it suffices to show that every finite $T' \subseteq T$ has a model. Let G' be the subgraph of G induced by vertices u such that $p_{u,i}$ appears in T' for some i. Since G' is k-colorable by the assumption, the theory T' has a model. $\square$

Resolution method - introduction

Main features of the resolution method (*informally*)

- is the underlying method of many systems, e.g. Prolog interpreters, SAT solvers, automated deduction / verification systems, . . .
- assumes input formulas in CNF (in general, *"expensive"* transformation),
- works under set representation (clausal form) of formulas,
- has a single rule, so called a resolution rule,
- has no explicit axioms (or atomic tableaux), but certain axioms are incorporated *"inside"* via various formatting rules,
- is a *refutation* procedure, similarly as the tableau method; that is, it tries to show that a given formula (or theory) is unsatisfiable,
- has several refinements e.g. with specific conditions on when the resolution rule may be applied.

Set representation (clausal form) of CNF formulas

- A *literal* l is a prop. letter or its negation. $\overline{l}$ is its *complementary* literal.
- A *clause* C is a finite set of literals (*"forming disjunction"*). The empty clause, denoted by $\square$, is never satisfied (has no satisfied literal).
- A *formula* S is a (possibly infinite) set of clauses (*"forming conjunction"*). An empty formula $\emptyset$ is always satisfied (it has no unsatisfied clause). Infinite formulas represent infinite theories (as conjunction of axioms).
- A (*partial*) *assignment* $\mathcal{V}$ is a consistent set of literals, i.e. not containing any pair of complementary literals. An assignment $\mathcal{V}$ is *total* if it contains a positive or negative literal for each propositional letter.
- $\mathcal{V}$ *satisfies* S, denoted by $\mathcal{V} \models S$, if $C \cap \mathcal{V} \neq \emptyset$ for every $C \in S$.

$((\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg r \vee \neg s) \wedge (\neg t \vee s) \wedge s)$ is represented by

$$S = \{\{\neg p, q\}, \{\neg p, \neg q, r\}, \{\neg r, \neg s\}, \{\neg t, s\}, \{s\}\} \quad \text{and}$$

$$\mathcal{V} \models S \quad \text{for} \quad \mathcal{V} = \{s, \neg r, \neg p\}$$

Resolution rule

Let C_1, C_2 be clauses with $l \in C_1$, $\bar{l} \in C_2$ for some literal l. Then from C_1 and C_2 infer through the literal l the clause C, called a *resolvent*, where

$$C = (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\bar{l}\}).$$

Equivalently, if $\sqcup$ means union of disjoint sets,

$$\frac{C_1' \sqcup \{l\}, C_2' \sqcup \{\bar{l}\}}{C_1' \cup C_2'}$$

For example, from $\{p, q, r\}$ and $\{\neg p, \neg q\}$ we can infer $\{q, \neg q, r\}$ or $\{p, \neg p, r\}$.

Observation *The resolution rule is sound; that is, for every assignment* $\mathcal{V}$

$$\mathcal{V} \models C_1 \text{ and } \mathcal{V} \models C_2 \quad \Rightarrow \quad \mathcal{V} \models C.$$

Remark *The resolution rule is a special case of the (so called) cut rule*

$$\frac{\varphi \vee \psi, \ \neg\varphi \vee \chi}{\psi \vee \chi}$$

where φ, ψ, χ are arbitrary formulas.

Resolution proof

- A *resolution proof* of a clause C from a formula S is a finite sequence $C_0, \ldots, C_n = C$ such that for every $i \leq n$, $C_i \in S$ or C_i is a resolvent of some previous clauses,
- a clause C is (resolution) *provable* from S, denoted by $S \vdash_R C$, if it has a resolution proof from S,
- a (resolution) *refutation* of formula S is a resolution proof of $\square$ from S,
- S is (resolution) *refutable* if $S \vdash_R \square$.

Theorem (soundness) *If S is resolution refutable, then S is unsatisfiable.*

Proof Let $S \vdash_R \square$. If it was $\mathcal{V} \models S$ for some assignment $\mathcal{V}$, from the soundness of the resolution rule we would have $\mathcal{V} \models \square$, which is impossible. ■

Resolution trees and closures

A *resolution tree* of a clause C from formula S is finite binary tree with nodes labeled by clauses so that

(i) the root is labeled C,

(ii) the leaves are labeled with clauses from S,

(iii) every inner node is labeled with a resolvent of the clauses in his children.

Observation *C has a resolution tree from S if and only if $S \vdash_R C$.*

A *resolution closure* $\mathcal{R}(S)$ of a formula S is the smallest set satisfying

(i) $C \in \mathcal{R}(S)$ for every $C \in S$,

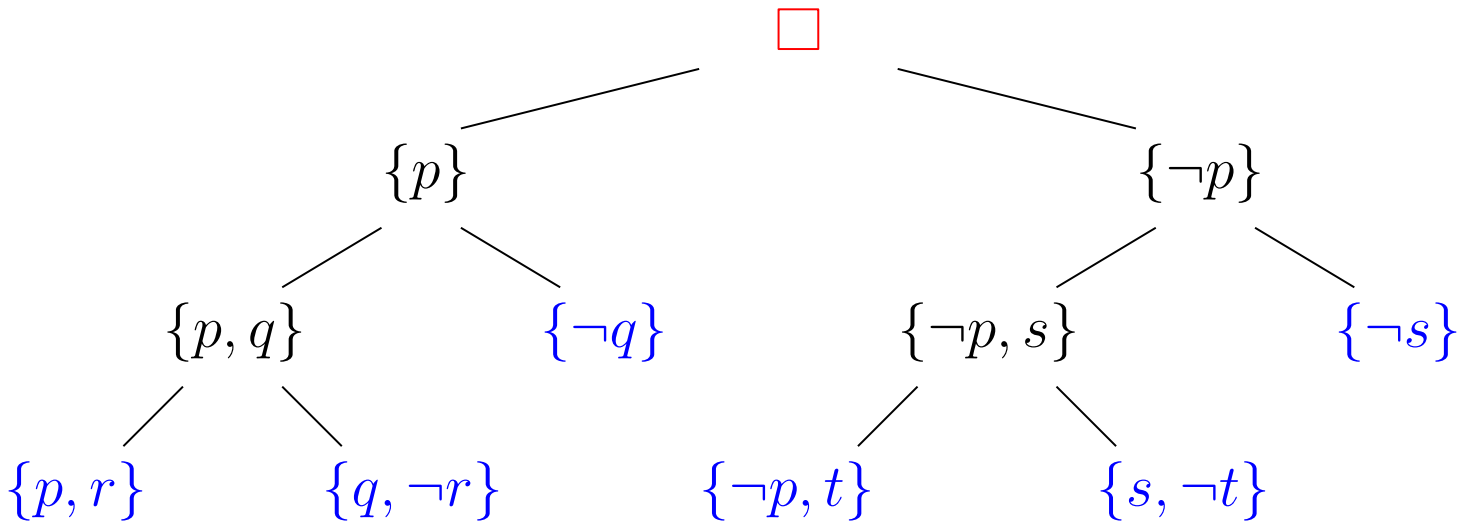
(ii) if $C_1, C_2 \in \mathcal{R}(S)$ and C is a resolvent of C_1, C_2, then $C \in \mathcal{R}(S)$.

Observation *$C \in \mathcal{R}(S)$ if and only if $S \vdash_R C$.*

Remark *All notions on resolution proofs can therefore be equivalently introduced in terms of resolution trees or resolution closures.*

Example

Formula $((p \vee r) \wedge (q \vee \neg r) \wedge (\neg q) \wedge (\neg p \vee t) \wedge (\neg s) \wedge (s \vee \neg t))$ is unsatisfiable since for $S = \{\{p, r\}, \{q, \neg r\}, \{\neg q\}, \{\neg p, t\}, \{\neg s\}, \{s, \neg t\}\}$ we have $S \vdash_R \square$.



The resolution closure of S (*the closure of S under resolution*) is

$$\mathcal{R}(S) = \{\{p, r\}, \{q, \neg r\}, \{\neg q\}, \{\neg p, t\}, \{\neg s\}, \{s, \neg t\}, \{p, q\}, \{\neg r\}, \{r, t\}, \{q, t\}, \{\neg t\}, \{\neg p, s\}, \{r, s\}, \{t\}, \{q\}, \{q, s\}, \square, \{\neg p\}, \{p\}, \{r\}, \{s\}\}.$$

Reduction by substitution

Let S be a formula and l be a literal. Let us define

$$S^l = \{C \setminus \{\bar{l}\} \mid l \notin C \in S\}.$$

Observation

- S^l is equivalent to a formula obtained from S by substituting the constant $\top$ (true, 1) for all literals l and the constant $\bot$ (false, 0) for all literals $\bar{l}$ in S,
- Neither l nor $\bar{l}$ occurs in (the clauses of) S^l.
- If $\{\bar{l}\} \in S$, then $\square \in S^l$.

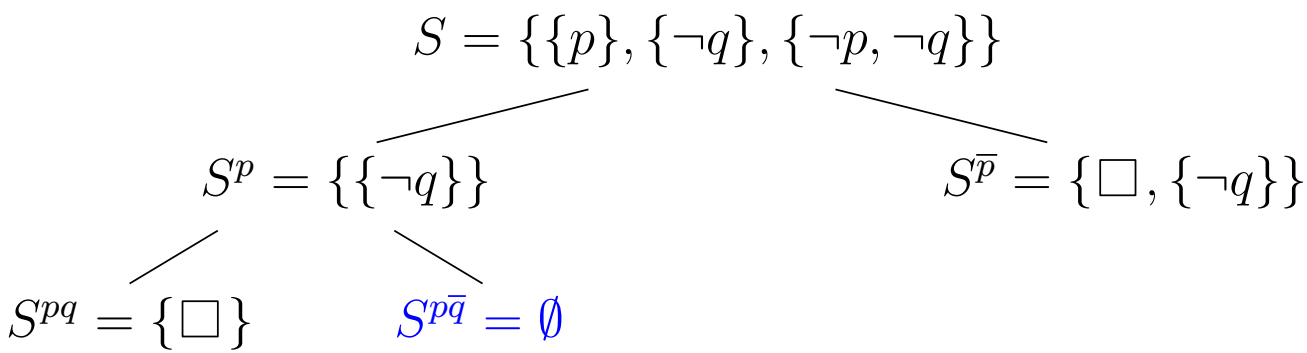
Lemma *S is satisfiable if and only if S^l or $S^{\bar{l}}$ is satisfiable.*

Proof ($\Rightarrow$) Let $\mathcal{V} \models S$ for some $\mathcal{V}$ and assume (w.l.o.g.) that $\bar{l} \notin \mathcal{V}$.

- Then $\mathcal{V} \models S^l$ as for $l \notin C \in S$ we have $\mathcal{V} \setminus \{l, \bar{l}\} \models C$ and thus $\mathcal{V} \models C \setminus \{\bar{l}\}$.
- On the other hand ($\Leftarrow$), assume (w.l.o.g.) that $\mathcal{V} \models S^l$ for some $\mathcal{V}$.
- Since neither l nor $\bar{l}$ occurs in S^l, we have $\mathcal{V}' \models S^l$ for $\mathcal{V}' = (\mathcal{V} \setminus \{\bar{l}\}) \cup \{l\}$.
- Then $\mathcal{V}' \models S$ since for $C \in S$ containing l we have $l \in \mathcal{V}'$ and for $C \in S$ not containing l we have $\mathcal{V}' \models (C \setminus \{\bar{l}\}) \in S^l$. ■

Tree of reductions

Step by step reductions of literals can be represented in a binary tree.



Corollary *S is unsatisfiable if and only if every branch contains $\square$.*

Remarks *Since S can be infinite over a countable language, this tree can be infinite. However, if S is unsatisfiable, by the compactness theorem there is a finite $S' \subseteq S$ that is unsatisfiable. Thus after reduction of all literals occurring in S', there will be $\square$ in every branch after finitely many steps.*

(Refutation) completeness of resolution

Theorem *If a finite S is unsatisfiable, it is resolution refutable, i.e. $S \vdash_R \square$.*

Proof By induction on the number of variables in S we show that $S \vdash_R \square$.

- If unsatisfiable S has no variable, it is $S = \{\square\}$ and thus $S \vdash_R \square$,
- Let l be a literal occurring in S. By Lemma, S^l and $S^{\bar{l}}$ are unsatisfiable.
- Since S^l and $S^{\bar{l}}$ have less variables than S, by induction there exist resolution trees T^l and $T^{\bar{l}}$ for derivation of $\square$ from S^l resp. $S^{\bar{l}}$.
- If every leaf of T^l is in S, then T^l is a resolution tree of $\square$ from S, $S \vdash_R \square$.
- Otherwise, by appending the literal $\bar{l}$ to every leaf of T^l that is not in S, (and to all predecessors) we obtain a resolution tree of $\{\bar{l}\}$ from S.
- Similarly, we get a resolution tree $\{l\}$ from S by appending l in the tree $T^{\bar{l}}$.
- By resolution of roots $\{\bar{l}\}$ and $\{l\}$ we get a resolution tree of $\square$ from S. ■

Corollary *If S is unsatisfiable, it is resolution refutable, i.e. $S \vdash_R \square$.*

Proof Follows from the previous theorem by compactness.

Predicate logic

Deals with statements about objects, their properties and relations.

"She is intelligent and her father knows the rector." $I(x) \wedge K(f(x), r)$

- x is a variable, representing an object,
- r is a constant symbol, representing a particular object,
- f is a function symbol, representing a function,
- I, K are relation (predicate) symbols, representing relations (the property of *"being intelligent"* and the relation *"to know"*).

"Everybody has a father." $(\forall x)(\exists y)(y = f(x))$

- $(\forall x)$ is the universal quantifier (*for every* x),
- $(\exists y)$ is the existential quantifier (*there exists* y),
- $=$ is a (binary) relation symbol, representing the identity relation.

Language

A first-order language consists of

- variables $x, y, z, \ldots, x_0, x_1, \ldots$ (countable many), the set of all variables is denoted by Var,
- function symbols $f, g, h, \ldots$, including constant symbols $c, d, \ldots$, which are nullary function symbols,
- relation (predicate) symbols $P, Q, R, \ldots$, eventually the symbol $=$ (equality) as a special relation symbol,
- quantifiers $(\forall x)$, $(\exists x)$ for every variable $x \in \mathrm{Var}$,
- logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- parentheses $($, $)$

Every function and relation symbol S has an associated *arity* $\mathrm{ar}(S) \in \mathbb{N}$.

Remark *Compared to propositional logic we have no (explicit) propositional variables, but they can be introduced as nullary relation symbols.*

Signatures

- *Symbols of logic* are variables, quantifiers, connectives and parentheses.
- *Non-logical symbols* are function and relation symbols except the equality symbol. The equality is (usually) considered separately.
- A *signature* is a pair $\langle \mathcal{R}, \mathcal{F} \rangle$ of disjoint sets of relation and function symbols with associated arities, whereas none of them is the equality symbol. A signature lists all non-logical symbols.
- A *language* is determined by a signature $L = \langle \mathcal{R}, \mathcal{F} \rangle$ and by specifying whether it is a language with equality or not. A language must contain at least one relation symbol (non-logical or the equality).

Remark *The meaning of symbols in a language is not assigned, e.g. the symbol $+$ does not have to represent the standard addition.*

Examples of languages

We describe a language by a list of all non-logical symbols with eventual clarification of arity and whether they are relation or function symbols.

The following examples of languages are all with equality.

- $L = \langle\ \rangle$ is the language of pure equality,
- $L = \langle c_i \rangle_{i \in \mathbb{N}}$ is the language of countable many constants,
- $L = \langle \leq \rangle$ is the language of orderings,
- $L = \langle E \rangle$ is the language of the graph theory,
- $L = \langle +, -, 0 \rangle$ is the language of the group theory,
- $L = \langle +, -, \cdot, 0, 1 \rangle$ is the language of the field theory,
- $L = \langle -, \wedge, \vee, 0, 1 \rangle$ is the language of Boolean algebras,
- $L = \langle S, +, \cdot, 0, \leq \rangle$ is the language of arithmetic,

where c_i, 0, 1 are constant symbols, S, $-$ are unary function symbols, $+$, $\cdot$, $\wedge$, $\vee$ are binary function symbols, E, $\leq$ are binary relation symbols.

Terms

Are expressions representing values of (composed) functions.

Terms of a language L are defined inductively by

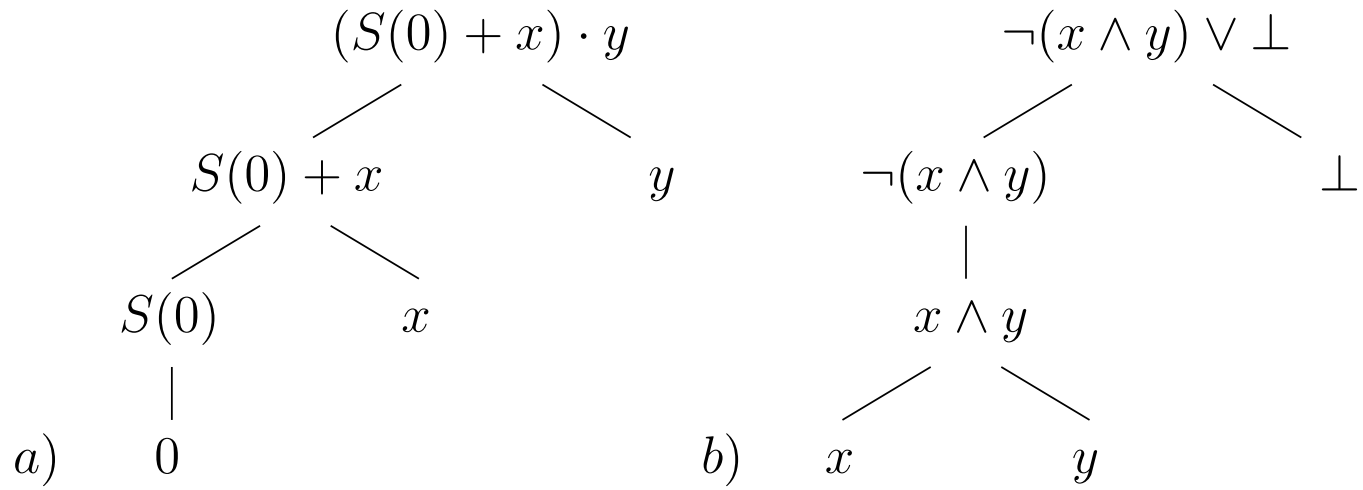
(i) every variable or constant symbol in L is a term,

(ii) if f is a function symbol in L of arity $n > 0$ and $t_1, \ldots, t_n$ are terms, then also the expression $f(t_1, \ldots, t_n)$ is a term,

(iii) every term is formed by a finite number of steps (i), (ii).

- A *ground term* is a term with no variables.
- The set of all terms of a language L is denoted by Term_L.
- A term that is a part of another term t is called a *subterm* of t.
- The structure of terms can be represented by their formation trees.
- For binary function symbols we often use infix notation, e.g. we write $(x + y)$ instead of $+(x, y)$.

Examples of terms



a) The formation tree of the term $(S(0) + x) \cdot y$ of the language of arithmetic.

b) Propositional formulas only with connectives $\neg$, $\wedge$, $\vee$, eventually with constants $\top$, $\bot$ can be viewed as terms of the language of Boolean algebras.

Atomic formulas

Are the simplest formulas.

- An *atomic formula* of a language L is an expression $R(t_1, \ldots, t_n)$ where R is an n-ary relation symbol in L and $t_1, \ldots, t_n$ are terms of L.
- The set of all atomic formulas of a language L is denoted by AFm_L.
- The structure of an atomic formula can be represented by a formation tree from the formation subtrees of its terms.
- For binary relation symbols we often use infix notation, e.g. $t_1 = t_2$ instead of $=(t_1, t_2)$ or $t_1 \leq t_2$ instead of $\leq(t_1, t_2)$.
- *Examples of atomic formulas*

$$K(f(x), r), \quad x \cdot y \leq (S(0) + x) \cdot y, \quad \neg(x \wedge y) \vee \bot = \bot.$$

Formula

Formulas of a language L are defined inductively by

(i) every atomic formula is a formula,

(ii) if φ, ψ are formulas, then also the following expressions are formulas

$$(\neg\varphi)\ ,(\varphi \wedge \psi)\ ,(\varphi \vee \psi)\ ,(\varphi \to \psi)\ ,(\varphi \leftrightarrow \psi),$$

(iii) if φ is a formula and x is a variable, then also the expressions $((\forall x)\varphi)$ and $((\exists x)\varphi)$ are formulas.

(iv) every formula is formed by a finite number of steps (i), (ii), (iii).

- The set of all formulas of a language L is denoted by Fm_L.
- A formula that is a part of another formula φ is called a *subformula* of φ.
- The structure of formulas can be represented by their formation trees.

Conventions

- After introducing *priorities* for binary function symbols e.g. $+$, $\cdot$ we are in infix notation allowed to omit parentheses that are around a subterm formed by a symbol of higher priority, e.g. $x \cdot y + z$ instead of $(x \cdot y) + z$.
- After introducing *priorities* for connectives and quantifiers we are allowed to omit parentheses that are around subformulas formed by connectives of higher priority.

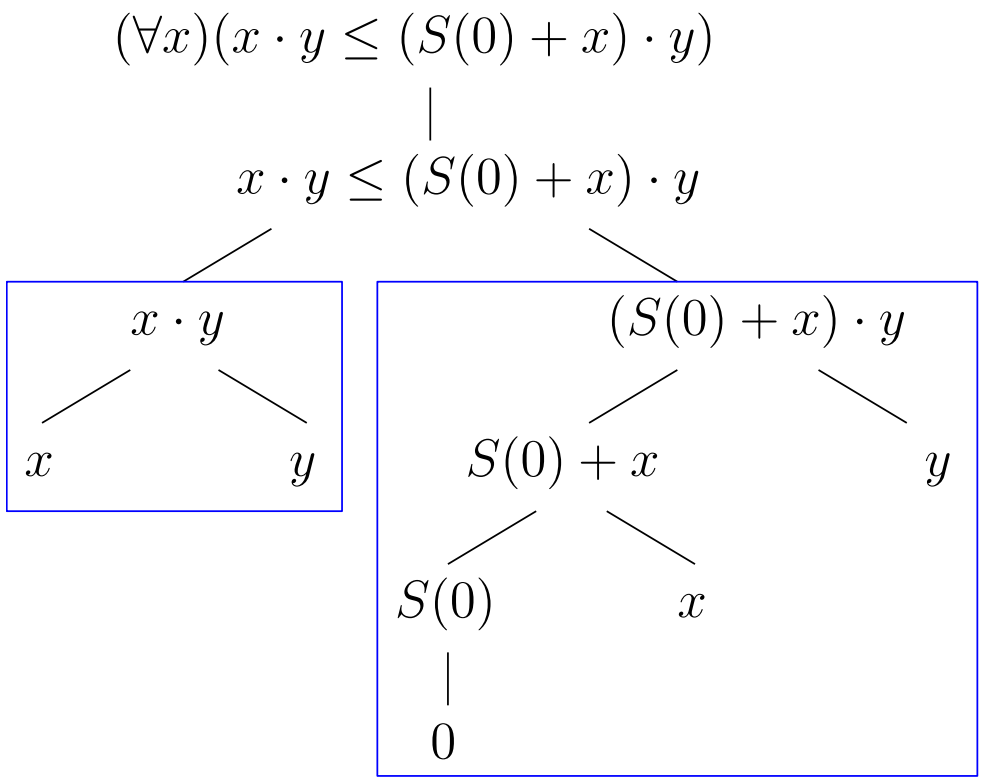
$$(1)\ \neg,\ (\forall x),\ (\exists x) \qquad (2)\ \wedge,\ \vee \qquad (3)\ \rightarrow,\ \leftrightarrow$$

- They can be always omitted around subformulas formed by $\neg$, $(\forall x)$, $(\exists x)$.
- We may also omit parentheses in $(\forall x)$ and $(\exists x)$ for every $x \in \mathrm{Var}$.
- The outer parentheses may be omitted as well.

$$(((\neg((\forall x)R(x))) \wedge ((\exists y)P(y))) \rightarrow (\neg(((\forall x)R(x)) \vee (\neg((\exists y)P(y))))))$$

$$\neg(\forall x)R(x) \wedge (\exists y)P(y) \rightarrow \neg((\forall x)R(x) \vee \neg(\exists y)P(y))$$

An example of a formula



The formation tree of the formula $(\forall x)(x \cdot y \leq (S(0) + x) \cdot y)$.

Occurrences of variables

Let φ be a formula and x be a variable.

- An *occurrence* of x in φ is a leaf labeled by x in the formation tree of φ.
- An occurrence of x in φ is *bound* if it is in some subformula ψ that starts with $(\forall x)$ or $(\exists x)$. An occurrence of x in φ is *free* if it is not bound.
- A variable x is *free* in φ if it has at least one free occurrence in φ.
  It is *bound* in φ if it has at least one bound occurrence in φ.
- A variable x can be both free and bound in φ. For example in
$$(\forall x)(\exists y)(x \leq y) \vee x \leq z.$$
- We write $\varphi(x_1, \ldots, x_n)$ to denote that $x_1, \ldots, x_n$ are all free variables in the formula φ. *(φ states something about these variables.)*

Remark *We will see that the truth value of a formula (in a given interpretation of symbols) depends only on the assignment of free variables.*

Open and closed formulas

- A formula is *open* if it is without quantifiers. For the set OFm_L of all open formulas in a language L it holds that $\mathrm{AFm}_L \subsetneq \mathrm{OFm}_L \subsetneq \mathrm{Fm}_L$.
- A formula is *closed* (a *sentence*) if it has no free variable; that is, all occurrences of variables are bound.
- A formula can be both open and closed. In this case, all its terms are ground terms.

$x + y \leq 0$	*open*, $\varphi(x, y)$
$(\forall x)(\forall y)(x + y \leq 0)$	*a sentence*,
$(\forall x)(x + y \leq 0)$	*neither open nor a sentence*, $\varphi(y)$
$1 + 0 \leq 0$	*open sentence*

Remark *We will see that in a fixed interpretation of symbols a sentence has a fixed truth value; that is, it does not depend on the assignment of variables.*

Instances

After substituting a term t for a free variable x in a formula φ, we would expect that the new formula (newly) says about t "the same" as φ did about x.

$\varphi(x)$	$(\exists y)(x + y = 1)$	*"there is an element* $1 - x$*"*
for $t = 1$ *we can* $\varphi(x/t)$	$(\exists y)(1 + y = 1)$	*"there is an element* $1 - 1$*"*
for $t = y$ *we cannot*	$(\exists y)(y + y = 1)$	*"*1 *is divisible by* 2*"*

- A term t is *substitutable* for a variable x in a formula φ if substituting t for all free occurrences of x in φ does not introduce a new bound occurrence of a variable from t.
- Then we denote the obtained formula $\varphi(x/t)$ and we call it an *instance* of the formula φ after a *substitution* of a term t for a variable x.
- t is not substitutable for x in φ if and only if x has a free occurrence in some subformula that starts with $(\forall y)$ or $(\exists y)$ for some variable y in t.
- Ground terms are always substitutable.

Variants

Quantified variables can be (under certain conditions) renamed so that we obtain an equivalent formula.

Let $(Qx)\psi$ be a subformula of φ where Q means $\forall$ or $\exists$ and y is a variable such that the following conditions hold.

1) y is substitutable for x in ψ, and
2) y does not have a free occurrence in ψ.

Then by replacing the subformula $(Qx)\psi$ with $(Qy)\psi(x/y)$ we obtain a *variant* of φ *in subformula* $(Qx)\psi$. After variation of one or more subformulas in φ we obtain a *variant* of φ. *For example,*

$(\exists x)(\forall y)(x \leq y)$	*is a formula* φ,
$(\exists u)(\forall v)(u \leq v)$	*is a variant of* φ,
$(\exists y)(\forall y)(y \leq y)$	*is not a variant of* φ, $1)$ *does not hold*,
$(\exists x)(\forall x)(x \leq x)$	*is not a variant of* φ, $2)$ *does not hold*.