2.9 OBDDs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

Method: Minimized graph representation of decision trees, based on a fixed ordering on propositional variables.

⇒ Canonical representation of formulas.

⇒ Satisfiability checking as a side effect.

BDDs

BDD (Binary decision diagram):

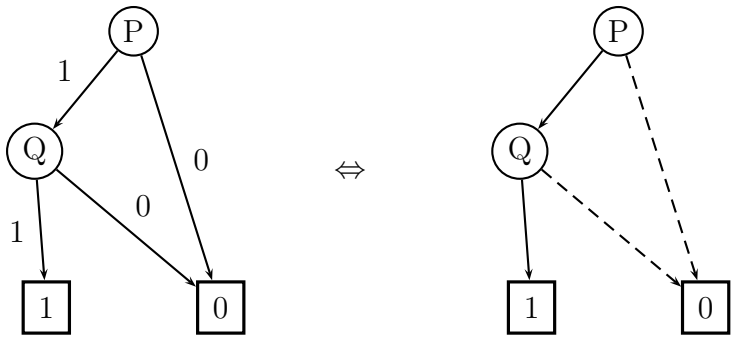
Labelled DAG (directed acyclic graph).

Leaf nodes:

labelled with a truth value (0 or 1).

Non-leaf nodes (interior nodes):

labelled with a propositional variable,
exactly two outgoing edges, labelled with 0 (- - →) and 1 (⟶)



Every BDD node can be interpreted as a mapping from valuations to truth values: Traverse the BDD from the given node to a leaf node; for any node labelled with P take the 0-edge or 1-edge depending on whether $\mathcal{A}(P)$ is 0 or 1.

⇒ Compact representation of truth tables.

OBDDs

OBDD (Ordered BDD):

Let $<$ be a total ordering of the propositional variables.

An OBDD w. r. t. $<$ is a BDD where every edge from a non-leaf node leads either to a leaf node or to a non-leaf node with a strictly larger label w. r. t. $<$.

OBDDs and formulas:

A leaf node $\boxed{0}$ represents $\bot$ (or any unsatisfiable formula).

A leaf node $\boxed{1}$ represents $\top$ (or any valid formula).

If a non-leaf node v has the label P, and its 0-edge leads to a node representing the formula F_0, and its 1-edge leads to a node representing the formula F_1, then v represents the formula

$$\begin{aligned} F &\models\!\models \text{ if } P \text{ then } F_1 \text{ else } F_0 \\ &\models\!\models (P \wedge F_1) \vee (\neg P \wedge F_0) \\ &\models\!\models (P \to F_1) \wedge (\neg P \to F_0) \end{aligned}$$

Conversely:

Define $F\{P \mapsto H\}$ as the formula obtained from F by replacing every occurrence of P in F by H.

For every formula F and propositional variable P:

$$F \models\!\models (P \wedge F\{P \mapsto \top\}) \vee (\neg P \wedge F\{P \mapsto \bot\})$$

(*Shannon expansion* of F, originally due to Boole).

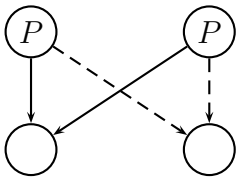
Consequence: Every formula F can be represented by an OBDD.

Reduced OBDDs

An OBDD is called *reduced*, if it has

- no duplicated leaf nodes

- no duplicated interior nodes



- no redundant tests

Theorem 2.20 (Bryant 1986) *Every OBDD can be converted into an equivalent reduced OBDD.*

Assumptions from now on:

One fixed ordering $>$.

We consider only reduced OBDDs.

All OBDDs are sub-OBDDs of a single OBDD.

Implementation:

Bottom-up construction of reduced OBDDs is possible using a hash table.

Keys and values are triples $(PropVar, Ptr_0, Ptr_1)$,

where Ptr_0 and Ptr_1 are pointers to the 0-successor and 1-successor hash table entry.

Theorem 2.21 (Bryant 1986) *If v and v' are two different nodes in a reduced OBDD, then they represent non-equivalent formulas.*

Proof. We use induction over the maximum of the numbers of nodes reachable from v and v', respectively. Let F and F' be the formulas represented by v and v'.

Case 1: v and v' are non-leaf nodes labelled by different propositional variables P and P'. Without loss of generality, $P < P'$.

Let v_0 and v_1 be the 0-successor and the 1-successor of v, and let F_0 and F_1 be formulas represented by v_0 and v_1. We may assume without loss of generality that all propositional variables occurring in F', F_0, and F_1 are larger than P. By reducedness, $v_0 \neq v_1$, so by induction, $F_0 \not\models\!\models F_1$. Hence there must be a valuation $\mathcal{A}$ such that $\mathcal{A}(F_0) \neq \mathcal{A}(F_1)$. Define valuations $\mathcal{A}_0$ and $\mathcal{A}_1$ by

$$\begin{array}{lll} \mathcal{A}_0(P) = 0 & \mathcal{A}_1(P) = 1 & \\ \mathcal{A}_0(Q) = \mathcal{A}(Q) & \mathcal{A}_1(Q) = \mathcal{A}(Q) & \text{for all } Q \neq P \end{array}$$

We know that the node v represents $F \models\!\models (P \wedge F_1) \vee (\neg P \wedge F_0)$, so $\mathcal{A}_0(F) = \mathcal{A}_0(F_0) = \mathcal{A}(F_0)$ and $\mathcal{A}_1(F) = \mathcal{A}_1(F_1) = \mathcal{A}(F_1)$, and therefore $\mathcal{A}_0(F) \neq \mathcal{A}_1(F)$. On the other hand, P does not occur in F', therefore $\mathcal{A}_0(F') = \mathcal{A}_1(F')$. So we must have $\mathcal{A}_0(F) \neq \mathcal{A}_0(F')$ or $\mathcal{A}_1(F) \neq \mathcal{A}_1(F')$, which implies $F \not\models\!\models F'$.

Case 2: v and v' are non-leaf nodes labelled by the same propositional variable.
Case 3: v is a non-leaf node, v' is a non-leaf node, or vice versa.
Case 4: v and v' are different leaf nodes.

Analogously. □

Corollary 2.22 *F is valid, if and only if it is represented by $\boxed{1}$. F is unsatisfiable, if and only if it is represented by $\boxed{0}$.*

Operations on OBDDs

Example:

Let $\circ$ be a binary connective.

Let P be the smallest propositional variable that occurs in F or G or both.

$$\begin{aligned} F \circ G &\models\!\models (P \wedge (F \circ G)\{P \mapsto \top\}) \vee (\neg P \wedge (F \circ G)\{P \mapsto \bot\}) \\ &\models\!\models (P \wedge (F\{P \mapsto \top\} \circ G\{P \mapsto \top\}) \\ &\qquad \vee (\neg P \wedge (F\{P \mapsto \bot\} \circ G\{P \mapsto \bot\}))) \end{aligned}$$

Note: $F\{P \mapsto \top\}$ is either represented by the same node as F (if P does not occur in F), or by its 1-successor (otherwise).

$\Rightarrow$ Obvious recursive function on OBDD nodes
(needs memoizing for efficient implementation).

OBDD operations are not restricted to the connectives of propositional logic.

We can also compute operations of *quantified boolean formulas*

$$\forall P.\, F \models\!\models (F\{P \mapsto \top\}) \wedge (F\{P \mapsto \bot\})$$

$$\exists P.\, F \models\!\models (F\{P \mapsto \top\}) \vee (F\{P \mapsto \bot\})$$

and images or preimages of propositional formulas w. r. t. boolean relations (needed for typical verification tasks).

The size of the OBDD for $F \circ G$ is bounded by mn, where F has size m and G has size n. (Size = number of nodes)

With memoization, the time for computing $F \circ G$ is also at most $O(mn)$.

The size of an OBDD for a given formula depends crucially on the chosen ordering of the propositional variables:

Let $F = (P_1 \wedge P_2) \vee (P_3 \wedge P_4) \vee \cdots \vee (P_{2n-1} \wedge P_{2n})$.

$P_1 < P_2 < P_3 < P_4 < \cdots < P_{2n-1} < P_{2n}$: $2n + 2$ nodes.

$P_1 < P_3 < \cdots < P_{2n-1} < P_2 < P_4 < \cdots < P_{2n}$: 2^{n+1} nodes.

Even worse: There are (practically relevant!) formulas for which the OBDD has exponential size *for every ordering* of the propositional variables.

Example: middle bit of binary multiplication.

Literature

Randal E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation; IEEE Transactions on Computers, 35(8):677-691, 1986.

Randal E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams; ACM Computing Surveys, 24(3), September 1992, pp. 293-318.

Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems*, Chapter 6.1/6.2; Cambridge Univ. Press, 2000.

2.10 FRAIGs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

Smaller representation than OBDDs.

Method: Minimized graph representation of boolean circuits.

FRAIG (Functionally Reduced And-Inverter Graph):

Labelled DAG (directed acyclic graph).

Leaf nodes:

labelled with propositional variables.

Non-leaf nodes (interior nodes):

labelled with $\wedge$ (two outgoing edges) or $\neg$ (one outgoing edge).

Reducedness (i. e., no two different nodes represent equivalent formulas) must be established explicitly, using

structural hashing,
simulation vectors,
CDCL,
OBDDs.

$\Rightarrow$ Semi-canonical representation of formulas.

Literature

A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton: FRAIGs: A unifying representation for logic synthesis and verification; ERL Technical Report, EECS Dept., UC Berkeley, March 2005.

2.11 Other Calculi

Ordered resolution
Tableau calculus
Hilbert calculus
Sequent calculus
Natural deduction

see next chapter

3 First-Order Logic

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- is not too expressive (e. g. not axiomatizable: natural numbers, uncountable sets)
- has a rich structure of decidable fragments
- has a rich model and proof theory

First-order logic is also called (first-order) *predicate logic*.

3.1 Syntax

Syntax:

- non-logical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- logical connectives (domain-independent)
  $\Rightarrow$ Boolean combinations, quantifiers

Signatures

A signature $\Sigma = (\Omega, \Pi)$ fixes an alphabet of non-logical symbols, where

- Ω is a set of *function symbols* f with *arity* $n \geq 0$, written $\text{arity}(f) = n$,
- Π is a set of *predicate symbols* P with arity $m \geq 0$, written $\text{arity}(P) = m$.

Function symbols are also called *operator symbols*.
If $n = 0$ then f is also called a *constant (symbol)*.
If $m = 0$ then P is also called a *propositional variable*.

We will usually use

b, c, d for constant symbols,

f, g, h for non-constant function symbols,

P, Q, R, S for predicate symbols.

Convention: We will usually write $f/n \in \Omega$ instead of $f \in \Omega$, arity$(f) = n$ (analogously for predicate symbols).

Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages); no big change from a logical point of view.

Variables

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that X is a given countably infinite set of symbols which we use to denote *variables*.

Terms

Terms over Σ and X (Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{lcll} s,t,u,v & ::= & x & , x \in X \quad \text{(variable)} \\ & | & f(s_1,...,s_n) & , f/n \in \Omega \quad \text{(functional term)} \end{array}$$

By $\mathrm{T}_\Sigma(X)$ we denote the set of Σ-terms (over X). A term not containing any variable is called a *ground term*. By T_Σ we denote the set of Σ-ground terms.

In other words, terms are formal expressions with well-balanced parentheses which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the *subterms* of the term. A node v that is marked with a function symbol f of arity n has exactly n subtrees representing the n immediate subterms of v.

Atoms

Atoms (also called atomic formulas) over Σ are formed according to this syntax:

$$\begin{array}{lcll} A,B & ::= & P(s_1,\ldots,s_m) & , P/m \in \Pi \quad \text{(non-equational atom)} \\ & \left[| & (s \approx t) & \quad \text{(equation)} \right] \end{array}$$

Whenever we admit equations as atomic formulas we are in the realm of *first-order logic with equality*. Admitting equality does not really increase the expressiveness of first-order logic (see next chapter). But deductive systems where equality is treated specifically are much more efficient.

Literals

$$\begin{array}{rcll} L & ::= & A & \text{(positive literal)} \\ & | & \neg A & \text{(negative literal)} \end{array}$$

Clauses

$$\begin{array}{rcll} C, D & ::= & \bot & \text{(empty clause)} \\ & | & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)} \end{array}$$

General First-Order Formulas

$\mathrm{F}_\Sigma(X)$ is the set of *first-order formulas* over Σ defined as follows:

$$\begin{array}{rcll} F, G, H & ::= & \bot & \text{(falsum)} \\ & | & \top & \text{(verum)} \\ & | & A & \text{(atomic formula)} \\ & | & \neg F & \text{(negation)} \\ & | & (F \wedge G) & \text{(conjunction)} \\ & | & (F \vee G) & \text{(disjunction)} \\ & | & (F \rightarrow G) & \text{(implication)} \\ & | & (F \leftrightarrow G) & \text{(equivalence)} \\ & | & \forall x\, F & \text{(universal quantification)} \\ & | & \exists x\, F & \text{(existential quantification)} \end{array}$$

Notational Conventions

We omit parentheses according to the conventions for propositional logic.

$\forall x_1, \ldots, x_n\, F$ and $\exists x_1, \ldots, x_n\, F$ abbreviate $\forall x_1 \ldots \forall x_n\, F$ and $\exists x_1 \ldots \exists x_n\, F$.

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

$$\begin{array}{rcl} s + t * u & \text{for} & +(s, *(t, u)) \\ s * u \leq t + v & \text{for} & \leq (*(s, u), +(t, v)) \\ -s & \text{for} & -(s) \\ s! & \text{for} & !(s) \\ |s| & \text{for} & |\_|(s) \\ 0 & \text{for} & 0() \end{array}$$

Example: Peano Arithmetic

$\Sigma_{\mathrm{PA}} = (\Omega_{\mathrm{PA}},\ \Pi_{\mathrm{PA}})$
$\Omega_{\mathrm{PA}} = \{0/0,\ +/2,\ */2,\ s/1\}$
$\Pi_{\mathrm{PA}} = \{</2\}$

Examples of formulas over this signature are:

$\forall x, y\,((x < y \vee x \approx y) \leftrightarrow \exists z\,(x + z \approx y))$
$\exists x \forall y\,(x + y \approx y)$
$\forall x, y\,(x * s(y) \approx x * y + x)$
$\forall x, y\,(s(x) \approx s(y) \to x \approx y)$
$\forall x \exists y\,(x < y \wedge \neg\exists z(x < z \wedge z < y))$

Positions in Terms and Formulas

The set of positions is extended from propositional logic to first-order logic:

The *positions* of a term s (formula F):

$\mathrm{pos}(x) = \{\varepsilon\}$,
$\mathrm{pos}(f(s_1, \ldots, s_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{\, ip \mid p \in \mathrm{pos}(s_i)\,\}$,

$\mathrm{pos}(P(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{\, ip \mid p \in \mathrm{pos}(t_i)\,\}$,

$\mathrm{pos}(\forall x\, F) = \{\varepsilon\} \cup \{\, 1p \mid p \in \mathrm{pos}(F)\,\}$,
$\mathrm{pos}(\exists x\, F) = \{\varepsilon\} \cup \{\, 1p \mid p \in \mathrm{pos}(F)\,\}$.

The prefix order $\leq$, the subformula (subterm) operator, the formula (term) replacement operator and the size operator are extended accordingly. See the definitions in Sect. 2.

Variables

The *set of variables* occurring in a term t is denoted by $\mathrm{var}(t)$ (and analogously for atoms, literals, clauses, and formulas).

Bound and Free Variables

In $\mathsf{Q}x\,F$, $\mathsf{Q} \in \{\exists, \forall\}$, we call F the *scope* of the quantifier $\mathsf{Q}x$. An *occurrence* of a variable x is called *bound*, if it is inside the scope of a quantifier $\mathsf{Q}x$. Any other occurrence of a variable is called *free*.

Formulas without free variables are also called *closed formulas* or *sentential forms*.

Formulas without variables are called *ground*.

Example:

$$\overbrace{\forall y \quad ((\forall x \overbrace{P(x)}^{\text{scope of } x}) \quad \rightarrow \quad R(x,y))}^{\text{scope of } y}$$

The occurrence of y is bound, as is the first occurrence of x. The second occurrence of x is a free occurrence.

Substitutions

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic.

Substitutions are mappings

$$\sigma : X \to \mathrm{T}_\Sigma(X)$$

such that the *domain* of σ, that is, the set

$$\mathrm{dom}(\sigma) = \{\, x \in X \mid \sigma(x) \neq x \,\},$$

is finite. The set of variables *introduced* by σ, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in \mathrm{dom}(\sigma)$, is denoted by $\mathrm{codom}(\sigma)$.

Substitutions are often written as $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$, with x_i pairwise distinct, and then denote the mapping

$$\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}(y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

The *modification* of a substitution σ at x is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$

Why Substitution is Complicated

We define the application of a substitution σ to a term t or formula F by structural induction over the syntactic structure of t or F by the equations depicted on the next page.

In the presence of quantification it is surprisingly complex: We need to make sure that the (free) variables in the codomain of σ are not *captured* upon placing them into the scope of a quantifier $\mathsf{Q}y$, hence the bound variable must be renamed into a "fresh", that is, previously unused, variable z.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$
\begin{aligned}
f(s_1, \ldots, s_n)\sigma &= f(s_1\sigma, \ldots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
P(s_1, \ldots, s_n)\sigma &= P(s_1\sigma, \ldots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg F\sigma &= \neg(F\sigma) \\
(F \circ G)\sigma &= (F\sigma \circ G\sigma) \quad \text{for each binary connective } \circ \\
(\mathsf{Q}x\, F)\sigma &= \mathsf{Q}z\, (F\, \sigma[x \mapsto z]) \quad \text{with } z \text{ a fresh variable}
\end{aligned}
$$

If $s = t\sigma$ for some subsitution σ, we call the term s an *instance* of the term t, and we call t a *generalization* of s (analogously for formulas).

3.2 Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

As in the propositional case, we use a two-valued logic with truth values "true" and "false" denoted by 1 and 0, respectively.

Algebras

A Σ-*algebra* (also called Σ-interpretation or Σ-structure) is a triple

$$\mathcal{A} = (U_{\mathcal{A}},\ (f_{\mathcal{A}} : U_{\mathcal{A}}^n \to U_{\mathcal{A}})_{f/n\in\Omega},\ (P_{\mathcal{A}} \subseteq U_{\mathcal{A}}^m)_{P/m\in\Pi})$$

where $U_{\mathcal{A}} \neq \emptyset$ is a set, called the *universe* of $\mathcal{A}$.

By Σ-*Alg* we denote the class of all Σ-algebras.

Σ-algebras generalize the valuations from propositional logic.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A *(variable) assignment* (over a given Σ-algebra $\mathcal{A}$), is a map $\beta : X \to U_{\mathcal{A}}$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $\mathcal{A}$ with Respect to β

By structural induction we define

$$\mathcal{A}(\beta) : \mathrm{T}_\Sigma(X) \to U_{\mathcal{A}}$$

as follows:

$$\begin{aligned} \mathcal{A}(\beta)(x) &= \beta(x), && x \in X \\ \mathcal{A}(\beta)(f(s_1,\ldots,s_n)) &= f_{\mathcal{A}}(\mathcal{A}(\beta)(s_1),\ldots,\mathcal{A}(\beta)(s_n)), && f/n \in \Omega \end{aligned}$$

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to U_{\mathcal{A}}$, for $x \in X$ and $a \in U_{\mathcal{A}}$, denote the assignment

$$\beta[x \mapsto a](y) = \begin{cases} a & \text{if } x = y \\ \beta(y) & \text{otherwise} \end{cases}$$

Truth Value of a Formula in $\mathcal{A}$ with Respect to β

$\mathcal{A}(\beta) : \mathrm{F}_\Sigma(X) \to \{0,1\}$ is defined inductively as follows:

$$
\begin{aligned}
\mathcal{A}(\beta)(\bot) &= 0 \\
\mathcal{A}(\beta)(\top) &= 1 \\
\mathcal{A}(\beta)(P(s_1,\ldots,s_n)) &= \text{if } (\mathcal{A}(\beta)(s_1),\ldots,\mathcal{A}(\beta)(s_n)) \in P_\mathcal{A} \text{ then 1 else 0} \\
\mathcal{A}(\beta)(s \approx t) &= \text{if } \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \text{ then 1 else 0} \\
\mathcal{A}(\beta)(\neg F) &= 1 - \mathcal{A}(\beta)(F) \\
\mathcal{A}(\beta)(F \wedge G) &= \min(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
\mathcal{A}(\beta)(F \vee G) &= \max(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
\mathcal{A}(\beta)(F \to G) &= \max(1 - \mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
\mathcal{A}(\beta)(F \leftrightarrow G) &= \text{if } \mathcal{A}(\beta)(F) = \mathcal{A}(\beta)(G) \text{ then 1 else 0} \\
\mathcal{A}(\beta)(\forall x\, F) &= \min_{a \in U_\mathcal{A}} \{\mathcal{A}(\beta[x \mapsto a])(F)\} \\
\mathcal{A}(\beta)(\exists x\, F) &= \max_{a \in U_\mathcal{A}} \{\mathcal{A}(\beta[x \mapsto a])(F)\}
\end{aligned}
$$

Example

The "Standard" Interpretation for Peano Arithmetic:

$$
\begin{aligned}
U_\mathbb{N} &= \{0,1,2,\ldots\} \\
0_\mathbb{N} &= 0 \\
s_\mathbb{N} &: n \mapsto n+1 \\
+_\mathbb{N} &: (n,m) \mapsto n+m \\
*_\mathbb{N} &: (n,m) \mapsto n*m \\
<_\mathbb{N} &= \{\, (n,m) \mid n \text{ less than } m \,\}
\end{aligned}
$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Values over $\mathbb{N}$ for sample terms and formulas:

Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$
\begin{aligned}
\mathbb{N}(\beta)(s(x)+s(0)) &= 3 \\
\mathbb{N}(\beta)(x+y \approx s(y)) &= 1 \\
\mathbb{N}(\beta)(\forall x,y\,(x+y \approx y+x)) &= 1 \\
\mathbb{N}(\beta)(\forall z\,(z<y)) &= 0 \\
\mathbb{N}(\beta)(\forall x \exists y\,(x<y)) &= 1
\end{aligned}
$$

Ground Terms and Closed Formulas

If t is a ground term, then $\mathcal{A}(\beta)(t)$ does not depend on β:

$$\mathcal{A}(\beta)(t) = \mathcal{A}(\beta')(t)$$

for every β and β'.

Analogously, if F is a closed formula, then $\mathcal{A}(\beta)(F)$ does not depend on β:

$$\mathcal{A}(\beta)(F) = \mathcal{A}(\beta')(F)$$

for every β and β'.

An element $a \in U_{\mathcal{A}}$ is called *term-generated*, if $a = \mathcal{A}(\beta)(t)$ for some ground term t.

In general, not every element of an algebra is term-generated.

3.3 Models, Validity, and Satisfiability

F is *true* in $\mathcal{A}$ under assignment β:

$$\mathcal{A}, \beta \models F \quad :\Leftrightarrow \quad \mathcal{A}(\beta)(F) = 1$$

F is *true* in $\mathcal{A}$ ($\mathcal{A}$ is a *model* of F; F is *valid* in $\mathcal{A}$):

$$\mathcal{A} \models F \quad :\Leftrightarrow \quad \mathcal{A}, \beta \models F \text{ for all } \beta \in X \to U_{\mathcal{A}}$$

F is *valid* (or is a *tautology*):

$$\models F \quad :\Leftrightarrow \quad \mathcal{A} \models F \text{ for all } \mathcal{A} \in \Sigma\text{-Alg}$$

F is called *satisfiable* iff there exist $\mathcal{A}$ and β such that $\mathcal{A}, \beta \models F$. Otherwise F is called *unsatisfiable*.

Entailment and Equivalence

F entails (implies) G (or *G is a consequence of F*), written $F \models G$, if for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \to U_{\mathcal{A}}$, whenever $\mathcal{A}, \beta \models F$, then $\mathcal{A}, \beta \models G$.

F and G are called *equivalent*, written $F \models\!\models G$, if for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \to U_{\mathcal{A}}$ we have $\mathcal{A}, \beta \models F \quad \Leftrightarrow \quad \mathcal{A}, \beta \models G$.

Proposition 3.1 *F entails G iff $(F \to G)$ is valid*

Proposition 3.2 *F and G are equivalent iff $(F \leftrightarrow G)$ is valid.*

Extension to sets of formulas N in the "natural way", e. g., $N \models F$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-Alg and $\beta \in X \to U_{\mathcal{A}}$: if $\mathcal{A}, \beta \models G$, for all $G \in N$, then $\mathcal{A}, \beta \models F$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 3.3 *Let F and G be formulas, let N be a set of formulas. Then*

(i) F is valid if and only if $\neg F$ is unsatisfiable.

(ii) $F \models G$ if and only if $F \wedge \neg G$ is unsatisfiable.

(iii) $N \models G$ if and only if $N \cup \{\neg G\}$ is unsatisfiable.

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Substitution Lemma

The following propositions, to be proved by structural induction, hold for all Σ-algebras $\mathcal{A}$, assignments β, and substitutions σ.

Lemma 3.4 *For any Σ-term t*

$$\mathcal{A}(\beta)(t\sigma) = \mathcal{A}(\beta \circ \sigma)(t),$$

where $\beta \circ \sigma : X \to U_{\mathcal{A}}$ is the assignment $\beta \circ \sigma(x) = \mathcal{A}(\beta)(x\sigma)$.

Proposition 3.5 *For any Σ-formula F, $\mathcal{A}(\beta)(F\sigma) = \mathcal{A}(\beta \circ \sigma)(F)$.*

Corollary 3.6 $\mathcal{A}, \beta \models F\sigma \quad \Leftrightarrow \quad \mathcal{A}, \beta \circ \sigma \models F$

These theorems basically express that the syntactic concept of substitution corresponds to the semantic concept of an assignment.

Two Lemmas

Lemma 3.7 *Let $\mathcal{A}$ be a Σ-algebra and let F be a Σ-formula with free variables $x_1, \ldots, x_n$. Then*

$$\mathcal{A} \models \forall x_1, \ldots, x_n\, F \quad \textit{if and only if} \quad \mathcal{A} \models F$$

Lemma 3.8 *Let F be a Σ-formula with free variables $x_1, \ldots, x_n$, let σ be a substitution, and let $y_1, \ldots, y_m$ be the free variables of $F\sigma$. Then*

$$\mathcal{A} \models \forall x_1, \ldots, x_n\, F \quad \textit{implies} \quad \mathcal{A} \models \forall y_1, \ldots, y_m\, F\sigma$$

3.4 Algorithmic Problems

Validity(F): $\models F$?

Satisfiability(F): F satisfiable?

Entailment(F,G): does F entail G?

Model($\mathcal{A}$,F): $\mathcal{A} \models F$?

Solve($\mathcal{A}$,F): find an assignment β such that $\mathcal{A}, \beta \models F$.

Solve(F): find a substitution σ such that $\models F\sigma$.

Abduce(F): find G with "certain properties" such that $G \models F$.

Theory of an Algebra

Let $\mathcal{A} \in \Sigma$-Alg. The *(first-order) theory* of $\mathcal{A}$ is defined as

$$\mathrm{Th}(\mathcal{A}) = \{\, G \in \mathrm{F}_\Sigma(X) \mid \mathcal{A} \models G \,\}$$

Problem of axiomatizability:

For which algebras $\mathcal{A}$ can one *axiomatize* $\mathrm{Th}(\mathcal{A})$, that is, can one write down a formula F (or a recursively enumerable set F of formulas) such that

$$\mathrm{Th}(\mathcal{A}) = \{\, G \mid F \models G \,\}?$$

(analogously for classes of algebras).

Two Interesting Theories

Let $\Sigma_{\mathrm{Pres}} = (\{0/0, s/1, +/2\},\ \{<\})$ and $\mathbb{N}_+ = (\mathbb{N}, 0, s, +, <)$ its standard interpretation on the natural numbers. $\mathrm{Th}(\mathbb{N}_+)$ is called *Presburger arithmetic* (M. Presburger, 1929). (There is no essential difference when one, instead of $\mathbb{N}$, considers the integer numbers $\mathbb{Z}$ as standard interpretation.)

Presburger arithmetic is decidable in 3EXPTIME (D. Oppen, JCSS, 16(3):323–332, 1978), and in 2EXPSPACE, using automata-theoretic methods (and there is a constant $c \geq 0$ such that $\mathrm{Th}(\mathbb{Z}_+) \notin \mathrm{NTIME}(2^{2^{cn}})$).

However, $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *, <)$, the standard interpretation of $\Sigma_{\mathrm{PA}} = (\{0/0, s/1, +/2, */2\}, \{<\})$, has as theory the so-called *Peano arithmetic* which is undecidable and not even recursively enumerable.

(Non-)Computability Results

1. For most signatures Σ, validity is undecidable for Σ-formulas.
   (One can easily encode Turing machines in most signatures.)
2. Gödel's completeness theorem:
   For each signature Σ, the set of valid Σ-formulas is recursively enumerable.
   (We will prove this by giving complete deduction systems.)
3. Gödel's incompleteness theorem:
   For $\Sigma = \Sigma_{\mathrm{PA}}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *, <)$, the theory $\mathrm{Th}(\mathbb{N}_*)$ is not recursively enumerable.

These complexity results motivate the study of subclasses of formulas (*fragments*) of first-order logic

Some Decidable Fragments

Some decidable fragments:

- *Monadic class*: no function symbols, all predicates unary; validity is NEXPTIME-complete.
- Variable-free formulas without equality: satisfiability is NP-complete. (why?)
- Variable-free Horn clauses (clauses with at most one positive atom): entailment is decidable in linear time.
- Finite model checking is decidable in exponential time and PSPACE-complete.

3.5 Normal Forms and Skolemization

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving.

The main problem in first-order logic is the treatment of quantifiers. The subsequent normal form transformations are intended to eliminate many of them.

Prenex Normal Form (Traditional)

Prenex formulas have the form

$$\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n \; F,$$

where F is quantifier-free and $\mathsf{Q}_i \in \{\forall, \exists\}$; we call $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n$ the *quantifier prefix* and F the *matrix* of the formula.

Computing prenex normal form by the reduction system $\Rightarrow_P$:

$$\begin{array}{rcl}
H[(F \leftrightarrow G)]_p & \Rightarrow_P & H[(F \to G) \wedge (G \to F)]_p \\
H[\neg \mathsf{Q} x\, F]_p & \Rightarrow_P & H[\overline{\mathsf{Q}} x\, \neg F]_p \\
H[((\mathsf{Q} x\, F) \circ G)]_p & \Rightarrow_P & H[\mathsf{Q} y\, (F\{x \mapsto y\} \circ G)]_p, \\
 & & \quad \circ \in \{\wedge, \vee\} \\
H[((\mathsf{Q} x\, F) \to G)]_p & \Rightarrow_P & H[\overline{\mathsf{Q}} y\, (F\{x \mapsto y\} \to G)]_p, \\
H[(F \circ (\mathsf{Q} x\, G))]_p & \Rightarrow_P & H[\mathsf{Q} y\, (F \circ G\{x \mapsto y\})]_p, \\
 & & \quad \circ \in \{\wedge, \vee, \to\}
\end{array}$$

Here y is always assumed to be some fresh variable and $\overline{\mathsf{Q}}$ denotes the quantifier *dual* to Q, i. e., $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.

Skolemization

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.

Transformation $\Rightarrow_S$
(to be applied outermost, *not* in subformulas):

$$\forall x_1, \ldots, x_n \exists y\, F \;\; \Rightarrow_S \;\; \forall x_1, \ldots, x_n\, F\{y \mapsto f(x_1, \ldots, x_n)\}$$

where f/n is a new function symbol (*Skolem function*).

Together: $F \Rightarrow_P^* \underbrace{G}_{\text{prenex}} \Rightarrow_S^* \underbrace{H}_{\text{prenex, no } \exists}$

Theorem 3.9 *Let F, G, and H as defined above and closed. Then*

(i) F and G are equivalent.

(ii) $H \models G$ but the converse is not true in general.

(iii) G satisfiable (w. r. t. Σ-Alg) $\Leftrightarrow$ H satisfiable (w. r. t. Σ'-Alg) where $\Sigma' = (\Omega \cup SKF, \Pi)$ if $\Sigma = (\Omega, \Pi)$.

The Complete Picture

$$
\begin{array}{lll}
F & \Rightarrow_P^* & \mathsf{Q}_1 y_1 \ldots \mathsf{Q}_n y_n \, G \qquad\qquad (G \text{ quantifier-free}) \\
 & \Rightarrow_S^* & \forall x_1, \ldots, x_m \, H \qquad (m \leq n,\ H \text{ quantifier-free}) \\
 & \Rightarrow_{CNF}^* & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \bigwedge_{i=1}^{k} \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'}
\end{array}
$$

$N = \{C_1, \ldots, C_k\}$ is called the *clausal (normal) form* (CNF) of F.
Note: The variables in the clauses are implicitly universally quantified.

Theorem 3.10 *Let F be closed. Then $F' \models F$. (The converse is not true in general.)*

Theorem 3.11 *Let F be closed. Then F is satisfiable iff F' is satisfiable iff N is satisfiable*

Optimization

The normal form algorithm described so far leaves lots of room for optimization. Note that we only can preserve satisfiability anyway due to Skolemization.

- the size of the CNF is exponential when done naively; the transformations we introduced already for propositional logic avoid this exponential growth;
- we want to preserve the original formula structure;
- we want small arity of Skolem functions (see next section).

3.6 Getting Skolem Functions with Small Arity

A clause set that is better suited for automated theorem proving can be obtained using the following steps:

- eliminate trivial subformulas
- replace beneficial subformulas
- produce a negation normal form (NNF)
- apply miniscoping
- rename all variables
- Skolemize
- push quantifiers upward
- apply distributivity

We start with a closed formula.

Elimination of Trivial Subformulas

Eliminate subformulas $\top$ and $\bot$ essentially as in the propositional case modulo associativity/commutativity of $\wedge$, $\vee$:

$$
\begin{aligned}
H[(F \wedge \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \vee \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \leftrightarrow \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[\neg F]_p \\
H[(F \leftrightarrow \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[(F \vee \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(F \wedge \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[\bot]_p \\
H[\neg\top]_p &\Rightarrow_{\mathrm{OCNF}} H[\bot]_p \\
H[\neg\bot]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(F \rightarrow \bot)]_p &\Rightarrow_{\mathrm{OCNF}} H[\neg F]_p \\
H[(F \rightarrow \top)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(\bot \rightarrow F)]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[(\top \rightarrow F)]_p &\Rightarrow_{\mathrm{OCNF}} H[F]_p \\
H[\mathsf{Q}x\,\top]_p &\Rightarrow_{\mathrm{OCNF}} H[\top]_p \\
H[\mathsf{Q}x\,\bot]_p &\Rightarrow_{\mathrm{OCNF}} H[\bot]_p
\end{aligned}
$$

Replacement of Beneficial Subformulas

The functions ν and $\bar{\nu}$ that give us an overapproximation for the number of clauses generated by a formula are extended to quantified formulas by

$$\nu(\forall x\, F) = \nu(\exists x\, F) = \nu(F),$$
$$\bar{\nu}(\forall x\, F) = \bar{\nu}(\exists x\, F) = \bar{\nu}(F).$$

The other cases are defined as for propositional formulas.

Introduce top-down fresh predicates for beneficial subformulas:

$$H[F]_p \;\Rightarrow_{\mathrm{OCNF}}\; H[P(x_1, \ldots, x_n)]_p \wedge \mathrm{def}(H, p, P, F)$$

if $\nu(H[F]_p) > \nu(H[P(\ldots)]_p \wedge \mathrm{def}(H, p, P, F))$,

where $\{x_1, \ldots, x_n\}$ are the free variables in F, P/n is a predicate new to $H[F]_p$, and $\mathrm{def}(H, p, P, F)$ is defined by

$$\begin{array}{l} \forall x_1, \ldots, x_n\ (P(x_1, \ldots, x_n) \to F), \text{ if } \mathrm{pol}(H, p) = 1, \\ \forall x_1, \ldots, x_n\ (F \to P(x_1, \ldots, x_n)), \text{ if } \mathrm{pol}(H, p) = -1, \\ \forall x_1, \ldots, x_n\ (P(x_1, \ldots, x_n) \leftrightarrow F), \text{ if } \mathrm{pol}(H, p) = 0. \end{array}$$

As in the propositional case, one can test $\nu(H[F]_p) > \nu(H[P]_p \wedge \mathrm{def}(H, p, P, F))$ in constant time without actually computing ν.

Negation Normal Form (NNF)

Apply the reduction system $\Rightarrow_{\mathrm{NNF}}$:

$$H[F \leftrightarrow G]_p \;\Rightarrow_{\mathrm{NNF}}\; H[(F \to G) \wedge (G \to F)]_p$$

if $\mathrm{pol}(H, p) = 1$ or $\mathrm{pol}(H, p) = 0$.

$$H[F \leftrightarrow G]_p \;\Rightarrow_{\mathrm{NNF}}\; H[(F \wedge G) \vee (\neg G \wedge \neg F)]_p$$

if $\mathrm{pol}(H, p) = -1$.

$$\begin{array}{rcl} H[F \to G]_p & \Rightarrow_{\mathrm{NNF}} & H[\neg F \vee G]_p \\ H[\neg\neg F]_p & \Rightarrow_{\mathrm{NNF}} & H[F]_p \\ H[\neg(F \vee G)]_p & \Rightarrow_{\mathrm{NNF}} & H[\neg F \wedge \neg G]_p \\ H[\neg(F \wedge G)]_p & \Rightarrow_{\mathrm{NNF}} & H[\neg F \vee \neg G]_p \\ H[\neg \mathsf{Q} x\ F]_p & \Rightarrow_{\mathrm{NNF}} & H[\overline{\mathsf{Q}} x\ \neg F]_p \end{array}$$

Miniscoping

Apply the reduction system $\Rightarrow_{\mathrm{MS}}$ modulo associativity and commutativity of $\wedge$, $\vee$. For the rules below we assume that x occurs freely in F, F', but x does not occur freely in G:

$$\begin{array}{lll}
H[\mathsf{Q}x\,(F \wedge G)]_p & \Rightarrow_{\mathrm{MS}} & H[(\mathsf{Q}x\,F) \wedge G]_p \\
H[\mathsf{Q}x\,(F \vee G)]_p & \Rightarrow_{\mathrm{MS}} & H[(\mathsf{Q}x\,F) \vee G]_p \\
H[\forall x\,(F \wedge F')]_p & \Rightarrow_{\mathrm{MS}} & H[(\forall x\,F) \wedge (\forall x\,F')]_p \\
H[\exists x\,(F \vee F')]_p & \Rightarrow_{\mathrm{MS}} & H[(\exists x\,F) \vee (\exists x\,F')]_p \\
H[\mathsf{Q}x\,G]_p & \Rightarrow_{\mathrm{MS}} & H[G]_p
\end{array}$$

Variable Renaming

Rename all variables in H such that there are no two different positions p, q with $H|_p = \mathsf{Q}x\,F$ and $H|_q = \mathsf{Q}'x\,G$.

Standard Skolemization

Apply the reduction system:

$$H[\exists x\,F]_p \Rightarrow_{\mathrm{SK}} H[F\{x \mapsto f(y_1, \ldots, y_n)\}]_p$$

where p has minimal length,
$\{y_1, \ldots, y_n\}$ are the free variables in $\exists x\,F$,
and f/n is a new function symbol to H.

Final Steps

Apply the reduction system modulo commutativity of $\wedge$, $\vee$ to push $\forall$ upward:

$$\begin{array}{lll}
H[(\forall x\,F) \wedge G]_p & \Rightarrow_{\mathrm{OCNF}} & H[\forall x\,(F \wedge G)]_p \\
H[(\forall x\,F) \vee G]_p & \Rightarrow_{\mathrm{OCNF}} & H[\forall x\,(F \vee G)]_p
\end{array}$$

Note that variable renaming ensures that x does not occur in G.

Apply the reduction system modulo commutativity of $\wedge$, $\vee$ to push disjunctions downward:

$$H[(F \wedge F') \vee G]_p \Rightarrow_{\mathrm{CNF}} H[(F \vee G) \wedge (F' \vee G)]_p$$