Logic and Proof

University of Cambridge

Ashwin Ahuja

Computer Science Tripos Part IB
Paper 6

April 2019

Contents

1 Propositional Logic

Logic concerns relationships statement (which can be true, false or meaningless)in some language, whether that be natural language or formal. Logical proofs model human reasoning.

Statements are declarative assertions.

Interpretation maps variables onto real objects. A statement is valid if all interpretations satisfy the statement.

Interpretation I satisfies a formula A if it evaluates to 1 (true), written as $\vDash_I A$. Tautology if every interpretation satisfies A, written as $\vDash A$. Satisfiable if some interpretation satisfies every formula in S

$$A \to B \; means \; \neg A \vee B$$

$$A \vDash B \; means \;\; if \vDash_I A \; then \;\; \vDash_I B \; \forall \; interpretations \; I$$

$$A \vDash B \; iff \;\; \vDash A \to B$$

A set of statements is consistent (**satisfiable**) if some interpretation satisfies all elements of the set at the same time - otherwise the set is inconsistent.

A set of statements entails A if every interpretation that satisfies all elements of S also satisfies A - written as $S \vDash A$

$S \vDash A iff \{\neg A\} \sup S$ is inconsistent. If S is inconsistent then $S \vDash A \forall A$. $\vDash A$ iff A is valid, iff $\{\neg A\}$

Inference: Proving a statement that says A is valid, but we can't test infinitely many cases.

$\{A_1, \ldots, A_n\} \vDash B$. If $A_1, \ldots, A_n$ are true then B must be true - this is written as:

$$\frac{A_1 ... A_n}{B} \tag{1}$$

Schematic Inference Rules

$$\frac{X \; part \; of \; Y \quad Y \; part \; of \; Z}{X \; part \; of \; Z} \tag{2}$$

1.1 Formal Language

Why: Formal language prevents ambiguity

Formal Logics

1. **Propositional Logic**: Traditional boolean algebra
2. **First-Order Logic**: Can say $\forall$ and $\exists$
3. **Higher-Order Logic**: Reasons about sets and functions
4. **Modal / Temporal Logic**: Reason about what must, or may, happen
5. **Type Theories**: Support constructive mathematics

1.2 Syntax of Propositional Logic

- P, Q, R: propositional letter
- **t**: true
- **f**: false
- $\neg$**A**: not A
- $A \wedge B$: A and B
- $A \vee B$: A or B
- $A \to B$: if A then B
- $A \leftrightarrow B$ A iff B

A	B	$\neg$ A	A $\wedge$ B	A $\vee$ B	A $\rightarrow$ B	A $\leftrightarrow$ B
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

1.3 Equivalences

1. $A \simeq B$ means $A \vDash B$ and $B \vDash A$
2. A $\simeq$ B iff A $\leftrightarrow$ B
3. $A \wedge A \simeq A$
4. $A \wedge B \simeq B \wedge A$
5. $(A \wedge B) \wedge C \simeq A \wedge (B \wedge C)$
6. $A \vee (B \wedge C) \simeq (A \vee B) \wedge (A \vee C)$
7. $A \wedge \mathbf{f} \simeq \mathbf{f}$
8. $A \wedge \mathbf{t} \simeq A$
9. $A \wedge \neg A \simeq \mathbf{f}$

1.4 Normal Forms

Negation Normal Form

1. Negation Normal Form
   (a) Get rid of $\leftrightarrow$ and $\rightarrow$, leaving just $\wedge, \vee, \neg$

$$A \leftrightarrow B \simeq (A \rightarrow B) \wedge (B \rightarrow A)$$
$$A \rightarrow B \simeq \neg A \vee B$$

   (b) Push negations in, using de Morgan's laws

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

2. Getting it into Conjunctive Normal Form
   (a) Push disjunctions in, using distributive laws

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

   (b) Simplify
   - Delete any disjunction containing P and $\neg$P
   - Delete any disjunction that includes another: in (P $\vee$ Q) $\wedge$ P, delete P $\vee$ Q
   - Replace $(P \vee A) \wedge (\neg P \vee A)$ by A

   (c) If you get to a specific result, you can find out it if the statement is a tautology

Deducibility: A is deducible from the set S if there is a finite proof of A starting from elements of S, can be written as $S \vdash A$

1. **Soundness Theorem**: If $S \vdash A$, then $S \vDash A$
2. **Completeness Theorem**: If $S \vDash A$ then $S \vdash A$
3. **Deduction Theorem**: If $S \sup\{A\} \vdash B$ then $S \vdash A \rightarrow B$

1.5 Gentzen's Natural Deduction Systems

Introduction Rule for $\wedge$:

$$\frac{A \quad B}{A \wedge B}$$

Elimination Rule for $\wedge$:

$$\frac{A \wedge B}{A}, \frac{A \wedge B}{B}$$

1.6 Sequent Calculus

$A_1, \ldots, A_m \Rightarrow B_1, \ldots, B_n$ means that if $A_1 \wedge \ldots \wedge A_m$ then $B_1 \vee \ldots \vee B_n$

- **Assumptions**: $A_1, \ldots, A_m$
- **Goals**: $\mathrm{B}_1, \ldots, \mathrm{B}_\mathrm{n}$
- Γ and Δ are sets in $\Gamma \Rightarrow \Delta$
- **Basic Sequent**: $A, \Gamma \Rightarrow A, \Delta$ is trivially true

1.6.1 Rules

1. **Cut**

$$\frac{\Gamma \Rightarrow \Delta, A \quad A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$$

2. $\neg l$

$$\frac{\Gamma \Rightarrow \Delta, A}{\neg A, \Gamma \Rightarrow \Delta}$$

3. $\neg r$

$$\frac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A}$$

4. $\wedge l$

$$\frac{A, B, \Gamma \Rightarrow \Delta}{A \wedge B, \Gamma \Rightarrow \Delta}$$

5. $\wedge r$

$$\frac{\Gamma \Rightarrow \Delta, A \quad \Gamma \Rightarrow \Delta, \mathrm{B}}{\Gamma \Rightarrow \Delta, A \wedge \mathrm{B}}$$

6. $\vee l$

$$\frac{A, \Gamma \Rightarrow \Delta \quad \mathrm{B}, \Gamma \Rightarrow \Delta}{A \vee B, \Gamma \Rightarrow \Delta}$$

7. $\vee r$

$$\frac{\Gamma \Rightarrow \Delta, A, \mathrm{B}}{\Gamma \Rightarrow \Delta, A \vee \mathrm{B}}$$

8. $\rightarrow l$

$$\frac{\Gamma \Rightarrow \Delta, A \quad \mathrm{B}, \Gamma \Rightarrow \Delta}{A \rightarrow B, \Gamma \Rightarrow \Delta}$$

9. $\rightarrow r$

$$\frac{A, \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \rightarrow B}$$

2 First-Order Logic

Function Symbol stands for an n-place function, **Constant Symbol** is a 0-place function symbol, a **variable** ranges over all individuals, **term** is a variable, constant or a function application. Considered as $f(t_1, ..., t_n)$ where f is an n-place function symbol and $t_1, ..., t_n$ are terms

Relation Symbol stands for an n-place relation, **Equality** is the 2-place relation symbol '=', **atomic formula** has the form $R(t_1, ..., t_n)$ where R is an n-place relation symbol and the ts are terms. A **formula** is built from atomic formulae using negation, OR, AND, quantifiers, etc.

2.1 Semantics

Allows different interpretations of symbols depending on the circumstances. An interpretation $\mathcal{I} = (D, I)$ defines the semantics of a first-order language where:

- D is domain - non-empty set
- I maps symbols to real elements, functions and relations

  c is constant: I[c] $\in D$

  f is an n-place function symbol: I[f] $\in D^n \to D$

  P is an n-place relation symbol: I[P] $\in D^n \to \{1, 0\}$

Valuation: V : Var $\to$ D supplies values of free variables. V and $\mathcal{I}$ determines the value of any term t, by recursion. This value is written as $I_V[t]$ with the following recursion rules:

1. $I_V[x] \stackrel{\text{def}}{=} V(x)$ if x is a variable
2. $I_V[c] \stackrel{\text{def}}{=} I[c]$
3. $I_V[f(t_1, ..., t_n)] \stackrel{\text{def}}{=} I[f](I_V[t_1], ..., I_V[t_n])$

2.2 Truth

Tarski Truth-Definition: Interpretation $\mathcal{I}$ and valuation function V specify the truth value (1 or 0) of any formula A. The only issue with this is quantifiers as they bind variables. V[a/x] is the valuation that maps x to a and is otherwise like V.

Define $\vDash_{\mathcal{I},V}$ by recursion:

1. $\vDash_{\mathcal{I},V} P(t)$ - if I[P]($\mathcal{I}_V[t]$) equals 1 (is true)
2. $\vDash_{\mathcal{I},V} t = u$ - if $\mathcal{I}_V[t] = \mathcal{I}_V[u]$
3. $\vDash_{\mathcal{I},V} A \wedge B$ - if $\vDash_{\mathcal{I},V} A$ and $\vDash_{\mathcal{I},V} B$
4. $\vDash_{\mathcal{I},V} \exists x A$ - if $\vDash_{\mathcal{I},V\{m/x\}}$ A holds for some m $\in D$
5. $\vDash_{\mathcal{I}} A$ - if $\vDash_{\mathcal{I},V} A$ holds for all V

Closed formula A is satisfiable if $\vDash_{\mathcal{I}} A$ for some $\mathcal{I}$

2.2.1 Free vs Bound Variables

- Occurrences of x in $\forall x A$ *and* $\exists x A$ are bound - can rename bound variables without affecting the meaning
- x is free if it not bound
- A[t/x] means substitute t for x in A. When substituting A[t/x], no variable of t may be bound in A

2.2.2 Equivalences involving quantifiers

1. $\neg(\forall x A) \simeq \exists x \neg A$
2. $\forall x A \simeq \forall x A \wedge A[t/x]$
3. $(\forall x A) \wedge (\forall x B) \simeq \forall x (A \wedge B)$

If x is not free in B

4. $(\forall x A) \wedge \mathrm{B} \simeq \forall \chi (A \wedge \mathrm{B})$
5. $(\forall x A) \vee \mathrm{B} \simeq \forall x (A \vee \mathrm{B})$
6. $(\forall x A) \rightarrow \mathrm{B} \simeq \exists x (A \rightarrow \mathrm{B})$

2.2.3 Sequent Rules for Quantifiers

1. $\forall l$: can create many instances of $\forall x A$

$$\frac{\mathrm{A[t/x]}, \Gamma \Rightarrow \Delta}{\forall x A, \Gamma \Rightarrow \Delta}$$

2. $\forall r$: holds, provided x is not free in the conclusion

$$\frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \forall x A}$$

3. $\exists l$: holds provided x is not free in the conclusion

$$\frac{A, \Gamma \Rightarrow \Delta}{\exists x A, \Gamma \Rightarrow \Delta}$$

4. $\exists r$: can create many instances of $\exists x A$

$$\frac{\Gamma \Rightarrow \Delta, A[\mathrm{t}/x]}{\Gamma \Rightarrow \Delta, \exists x A}$$

3 Clause Methods for Propositional Logic

A clause is a disjunction of literals: $\neg \mathrm{K}_1 \vee \cdots \vee \neg \mathrm{K}_\mathrm{m} \vee \mathrm{L}_1 \vee \cdots \vee \mathrm{L}_\mathrm{n}$

1. **Set Notation**: $\{\neg \mathrm{K}_1, \ldots, \neg \mathrm{K}_\mathrm{m}, \mathrm{L}_1, \ldots, \mathrm{L}_\mathrm{n}\}$
2. **Kowalski Notation**:
   $\mathrm{K}_1, \cdots, \mathrm{K}_\mathrm{m} \rightarrow \mathrm{L}_1, \cdots, \mathrm{L}_\mathrm{n}$
   $\mathrm{L}_1, \cdots, \mathrm{L}_\mathrm{n} \leftarrow \mathrm{K}_1, \cdots, \mathrm{K}_\mathrm{m}$
3. Empty Clause: $\{\}$ or $\square$

Proving A: Obtain contradiction from $\neg$ A

1. Translate $\neg$ A into CNF as $A_1 \wedge \cdots \wedge A_m$ = clauses $A_1, \ldots, A_m$
2. Transform clause set, preserving consistency
3. An empty clause set means $\neg$ A is satisfiable - otherwise, obtain contradiction

3.1 DPLL Method

: Decision procedure, either finds a contradiction or a model

1. Delete tautological clauses: P, $\neg$ P
2. For each unit clause: (1) delete all clauses containing L, (2) delete $\neg$ L from all clauses
3. Delete all clauses containing pure literals
4. Perform case split on some literal and stop if a model is found

3.2 Resolution Rule

Allows us to say $B \vee A \wedge \neg B \vee C \implies A \vee C$

1. $$\frac{\{\mathrm{B}, \mathrm{A}_1, \ldots, \mathrm{A}_\mathrm{m}\} \quad \{\neg \mathrm{B}, \mathrm{C}_1, \ldots, \mathrm{C}_\mathrm{n}\}}{\{A_1, \ldots, A_m, C_1, \ldots, C_n\}}$$
2. $$\frac{\{\mathrm{B}\} \quad \{\neg \mathrm{B}, \mathrm{C}_1, \ldots, \mathrm{C}_\mathrm{n}\}}{\{\mathrm{C}_1, \ldots, \mathrm{C}_\mathrm{n}\}}$$
3. $$\frac{\{\mathrm{B}\} \quad \{\neg \mathrm{B}\}}{\square}$$

3.3 Saturation Algorithm

- At the start, all clauses are passive
  1. Transfer a clause (current) from passive to active
  2. Form all resolvents between current and active clause
  3. Use new clauses to simplify both passive and active
  4. Put the new clauses into passive
- Repeat until a contradiction is found or if the passive becomes empty

Heuristics

1. **Orderings** to focus search on specific literals
2. **Subsumption**: deleting redundant clauses
3. **Indexing**: elaborate data structures for speed
4. **Preprocessing**: removing tautologies and symmetries
5. **Weighting**: Giving priority to good clauses over those containing unwanted constants

4 Skolem Functions, Herbrand Theorem and Unification

4.1 FOL to Propositional Logic

NNF: Eliminate all connectives except $\vee$, $\wedge$ and $\neg$
Skolemize: Remove quantifiers, preserving consistency
Herbrand Models: Reduce the class of interpretations
Herbrand's THM: Contradictions have finite, ground proofs
Unification: Automatically find the correct instantiations

4.2 Skolemisation

1. Start with formula in NNF, with quantifiers nested
2. Choose fresh k-place function symbol, say f
3. Delete $\exists$y and replace y with $\mathrm{f}(x_1, x_2, ..., x_k)$
4. Repeat until no $\exists$ quantifiers remain

Correctness:

- Formula $\forall x \exists y A$ is consistent
- Holds in some interpretation: $\mathcal{I} = (\mathrm{D}, \mathrm{I})$
- $\forall x \in D \exists y \in D$ *st A holds*
- Some function $\hat{f}$ in $\mathrm{D} \to \mathrm{D}$ yields suitable values of y
- A[f(x)/y] holds in some $\mathcal{I}'$ extending $\mathcal{I}$ so that f denotes $\hat{\mathrm{f}}$
- Formula $\forall x A[f(x)/y]$ is consistent

4.3 Herbrand Interpretations

4.3.1 Herbrand Universe for a Set of Clauses S

$$H_0 \stackrel{\text{def}}{=} \text{ set of constants in S}$$

$$\mathrm{H_{i+1}} \stackrel{\text{def}}{=} \mathrm{H_i} \cup \{\mathrm{f}(\mathrm{t_1}, \ldots, \mathrm{t_n}) \,|\, \mathrm{t_1}, \ldots, \mathrm{t_n} \in \mathrm{H_i} \text{ and f is an n-place function symbol in S }\}$$

$$HerbrandUniverse(H) \stackrel{\text{def}}{=} \bigcup_{\mathrm{i} \geq 0} \mathrm{H_i}$$

H consists of the terms in S that contains no variables (ground terms). H_i contains the terms with at most i nested function applications

4.3.2 Herbrand Semantics of Predicates

Herbrand interpretation defines an n-place predicate P to denote the truth-valued function in $H^n \to \{1, 0\}$ making $\mathrm{P}(t_1, ..., t_n)$ true iff formula $\mathrm{P}(\mathrm{t_1}, \ldots, \mathrm{t_n})$ holds in desired interpretation of the clauses. Hence, Herbrand interpretation can imitate any other interpretation.

4.3.3 Herbrand's Theorem

S is a set of clauses, S is unsatisfiable $\iff$ there is a finite unsatisfiable set S' of ground instances of clauses of S

- **Finite**: can be computed
- **Instance**: result of substituting for variables
- **Ground**: no variables remain - it is propositional

4.4 Unification

Finding a common instance of two terms - generalisation of pattern-matching, used for:

1. Prolog and other logic programming languages
2. Theorem proving
3. Tools for reasoning with equations or satisfying constraints
4. Polymorphic type-checking

Output of a unification is a substitution, mapping variables to terms - where the other occurrences of the variables must also be updated - in general it yields the most general solution

5 First-Order Resolution and Prolog

5.1 Binary Resolution Rule

Where σ is a most general unifier of B and D st $\mathrm{B}\sigma = \mathrm{D}\sigma$: requires us to first rename variables apart in the clauses

$$\frac{\{B, A_1, ..., A_m\} \quad \{\neg D, C_1, ..., C_n\}}{\{A_1, ..., A_m, C_1, ..., C_n\}\sigma}$$

5.2 Factoring Rule

Inference collapses unifiable literals in one clause. Provided $B_1\sigma = ... = B_k\sigma$

$$\frac{\{B_1, ..., B_k, A_1, ..., A_m\}}{\{B_1, A_1, ..., A_m\}\sigma}$$

5.3 Equality

Equality Axioms: these work in theory

1. Reflexive
2. Symmetric
3. Transitive
4. Substitution laws for each f st $\{x \neq y, f(x) = f(y)\}$
5. Substitution laws for each P st $\{x \neq y, \neg P(x), P(y)\}$

But in practise, need something special - **paramodulation rule**:

$$\frac{\{\mathrm{B}\left[\mathfrak{t}'\right], A_1, \ldots, A_\mathrm{m}\} \quad \{\mathfrak{t} = \mathfrak{u}, \mathrm{C}_1, \ldots, \mathrm{C}_\mathfrak{n}\}}{\{\mathrm{B}[\mathfrak{u}], A_1, \ldots, A_\mathrm{m}, \mathrm{C}_1, \ldots, \mathrm{C}_\mathrm{n}\}\,\sigma}$$

where tσ = t'σ

5.4 Prolog

Clauses

- Prolog clauses have a restricted form, with at most one positive literal
- Definite clauses form the program - ie procedure B with body $A_1, ..., A_m$ is: B $\leftarrow A_1, ..., A_m$
- Single goal clauses is like the execution stack, with tasks to be done: $\leftarrow A_1, ..., A_m$

Execution

- **Linear Resolution**: Always resolves some program clause with the goal clause, and the result becomes the new goal clause
- Program clauses done in left-to-right order
- Left-to-right order used to solve the goal clause's literals
- Use depth-first search (performs backtracking using choice points)
- Does a unification without occurs check

Model Elimination: Prolog-like method to do FOL Proof that runs on fast Prolog architectures using Contrapositives where you treat the clause $A_1, ..., A_m$ like the m clauses:

- $A_1 \leftarrow \neg A_2, ..., \neg A_m$
- $A_2 \leftarrow \neg A_3, ..., \neg A_m, \neg A_1$
- ...
- $A_m \leftarrow \neg A_1, ..., \neg A_{m-1}$

When trying to prove the goal P, we assume $\neg P$

5.5 Automatic Theorem Provers

1. **First-Order Resolution**: E, SPASS, Vampire
2. **Higher-Order Logic**: TPS, LEO and LEO-II, Satallax
3. **Model Elimination**: Prolog, SETHEO
4. **Parallel ME**: PARTHENON, PARTEO
5. **Tableau (Sequent) Based**: LeanTAP, 3TAP

6 Decision Procedures and SMT Solvers

6.1 Decision procedures and Problems

Decision problems are any formally-stated problems:

1. Propositional Formulas are decidable using the DPLL algorithm
2. Linear Arithmetic Formulas are decidable
3. Polynomial Arithmetic is decidable, as is Euclidean geometry

6.2 Fourier-Motzkin Variable Elimination

Decides conjunctions of liner constraints over reals and rationals. It works by eliminating variables one-by-one until there is one to remain, or reaches a contradiction - resembles Gaussian elimination. It has a worst case complexity with $O(m^{2^n})$

$$\bigwedge_{\mathrm{i}=1}^{\mathrm{m}} \sum_{\mathrm{j}=1}^{\mathrm{n}} \mathfrak{a}_{\mathrm{ij}} x_{\mathrm{j}} \leq \mathrm{b}_{\mathrm{i}}$$

Process

1. To eliminate variable x_n, consider constraint i, for i = 1, ..., m
2. Define $\beta_i = b_i$ - $\sum_{\mathrm{j}=1}^{\mathrm{n}-1} \mathfrak{a}_{\mathrm{ij}} \mathfrak{x}_{\mathrm{j}}$ Rewrite the constraint i:
   (a) If $a_{in} > 0$ then $x_n \leq \frac{\beta_i}{a_{in}}$
   (b) If $a_{in} < 0$ then $-x_n \leq -\frac{\beta_i}{a_{in}}$
   (c) Hence, $0 \leq \frac{\beta_i}{a_{in}} - \frac{\beta_{i\prime}}{a_{i\prime n}}$
3. Do this for all combinations with opposite signs
4. Then delete the original constraints (except where $a_{in} = 0$)

6.3 Quantifier Elimination

Transforms a formula to a quantifier-free but equivalent formula - since skolemization eliminates quantifiers but only preserves consistency - can use Quantifier Elimination to do this and allows us to reach true or false - however, this yields a long formula - taking a long amount of time - hyper-exponential time complexity.

6.4 Other Decidable Theories

Linear Integer Arithmetic: use Omega test or Cooper's algorithm, but any decision algorithm has a worst case runtime of at least $2^{2^{cn}}$

QE for quadratic equations:

$$\exists x \left[ax^2 + bx + c = 0\right] \iff \mathrm{b}^2 \geq 4\mathrm{ac} \wedge \left(\mathrm{c} = 0 \vee \mathrm{a} \neq 0 \vee \mathrm{b}^2 > 4\mathrm{ac}\right)$$

You can have have decidable procedures that cooperate to decide combinations of theories - however, these procedures expect existentially quantified conjunctions. Formulas have to be converted to disjunctive normal form - therefore have to eliminate universal quantifier

6.5 Satisfiability Modulo Theories

The idea is that we can use DPLL for logical reasoning, using a decision procedure for theories. The clauses can have literals which are used as names. If DPLL finds a contradiction, then the clauses are unsatisfiable. Checking asserted literals:

1. Unsatisfiable conjunctions of literals are noted as new clauses
2. Case splitting is interleaved with decision procedure calls

7 Binary Decision Diagrams

A canonical form for boolean expressions - decision trees with sharing consisting of:

1. Ordered propositional symbols (variables)
2. Sharing of identical subtrees
3. Hashing and other optimisations
4. Dashed line is false and non-dashed line is true

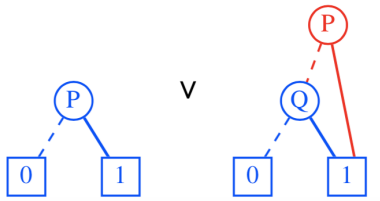
Shows, (1) satisfiability (existence of models) - path to 1, (2) tautologous, (3) inconsistency.

Building BDDs Efficiently

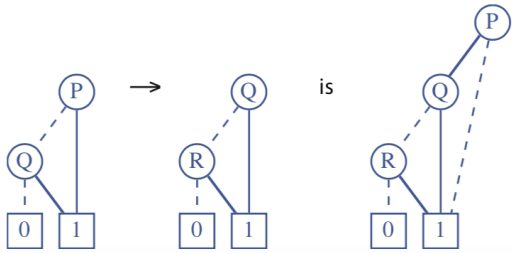
1. Don't expand connectives
2. Recursively convert operands to BDDs
3. Combine operand BDDs respecting ordering and sharing
4. Delete redundant variable tests

7.1 Examples

1. $P \vee Q$



2. $P \vee Q \to Q \vee R$



7.2 Optimisations

1. Never build the same BDD twice, but share pointers, allows us to (1) check if(P, X, Y) is redundant by checking if X = Y and (2) quickly simplify special cases like X $\wedge$ X
2. Never convert X $\wedge$ Y twice, but keep a hash table of known canonical forms - preventing redundant computations

8 Modal Logics

- W: set of possible worlds (machine states, future times, ...)
- R: Accessibility Relation between worlds
- (W, R) is a modal frame
- $\Box A$ means A is necessarily true
$\diamond$A means A is possibly true } in all worlds accessible from here
- $\neg \diamond A \simeq \Box \neg A$

8.1 Semantics

For a particular frame (W, R), an interpretation I maps the propositional letters to subsets of W:

$w \,||- A$ means that A is true in world w:

- $w \,||- P \iff w \in I(P)$
- $w \,||- A \wedge B \iff w \,||- Aandw \,||- B$
- $w \,||- \Box A \iff v \,||- A \forall vstR(w, v)$
- $w \,||- \diamond A \iff \exists v v \,||- AstR(w, v)$

Truth and Validity: For frame (W, R) and interpretation I:

- $w \,||- A$ means A is true in world w
- $\vDash_{W,R,I} A$ means $w \,||- A \; \forall w \in W$
- $\vDash_{W,R} A$ means $w \,||- A \;\; \forall w \, \forall I$
- $\vDash A$ means $\vDash_{W,R} A \;\; \forall frames$ that is to say, A is universally valid but generally constrain R to be transitive

8.2 Hilbert-Style Modal Proof Systems

1. **Dist**: $\Box(A \to \mathrm{B}) \to (\Box A \to \Box \mathrm{B})$
2. **Inference Rule - necessitation**: $\frac{A}{\Box A}$
3. **Definition** - treat $\diamond$ as a definition: $\diamond A \stackrel{\text{def}}{=} \neg\Box\neg A$

We can add axioms to pure modal logic (K) to constrain the accessibility relation

4. **Reflexive**: $\Box A \to A$ *(T)*
5. **Transitive**: $\Box A \to \Box\Box A$ *(S4)*
6. **Symmetric**: $A \to \Box \diamond A$ *(S5)*

8.3 S4 Sequent Calculus Rules

1. $\Box l$

$$\frac{A, \Gamma \Rightarrow \Delta}{\Box A, \Gamma \Rightarrow \Delta}$$

2. $\Box r$

$$\frac{\Gamma^* \Rightarrow \Delta^*, A}{\Gamma \Rightarrow \Delta, \Box A}$$

3. $\diamond l$

$$\frac{A, \Gamma^* \Rightarrow \Delta^*}{\diamond A, \Gamma \Rightarrow \Delta}$$

4. $\diamond r$

$$\frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \diamond A}$$

5. $\Gamma^* \stackrel{\text{def}}{=} \{\Box \mathrm{B} | \Box \mathrm{B} \in \Gamma\}$
   Erase all non-$\Box$ assumptions
6. $\Delta^* \stackrel{\text{def}}{=} \{\diamond \mathrm{B} | \diamond \mathrm{B} \in \Delta\}$
   Erase all non-$\diamond$ goals

9 Tableaux-Based Methods

9.1 Simplifying the Sequent Calculus

Can simplify sequent calculus by working in Negation Normal Form which reduces required connectives from: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$, $\Box$, $\diamond$ **to**: $\wedge$, $\vee$, $\forall$, $\exists$, $\Box$, $\diamond$. This means that the sequents need only one side.

9.2 Rules

Left-side only system uses proof by contradiction while right-side only system is an exact dual

1. **Basic**
$$\frac{}{\neg A, A, \Gamma \Rightarrow}$$
2. **Cut**
$$\frac{\neg A, \Gamma \Rightarrow \quad A, \Gamma \Rightarrow}{\Gamma \Rightarrow}$$
3. $\wedge l$
$$\frac{A, B, \Gamma \Rightarrow}{A \wedge B, \Gamma \Rightarrow}$$
4. $\vee l$
$$\frac{A, \Gamma \Rightarrow \quad B, \Gamma \Rightarrow}{A \vee B, \Gamma \Rightarrow}$$
5. $\forall l$
$$\frac{A[t/x], \Gamma \Rightarrow}{\forall x A, \Gamma \Rightarrow}$$
6. $\exists l$: provided x is not free in the conclusion
$$\frac{A, \Gamma \Rightarrow}{\exists x A, \Gamma \Rightarrow}$$
7. $\Box l$
$$\frac{A, \Gamma \Rightarrow}{\Box A, \Gamma \Rightarrow}$$
8. $\diamond l$
$$\frac{A, \Gamma^* \Rightarrow}{\diamond A, \Gamma \Rightarrow}$$
9. $\Gamma^* \stackrel{\text{def}}{=} \{\Box B | \Box B \in \Gamma\}$ - erase non-$\Box$ assumptions

9.3 Free-Variable Tableau Calculus

$\forall l$: inserts a new free variable

$$\frac{A[z/x], \Gamma \Rightarrow}{\forall x A, \Gamma \Rightarrow}$$

- Lets unification instantiate any free variable
- In $\neg$A, B, $\Gamma \implies$, try unifying A with B to make a basic sequent
- Better not to use the rule $\exists$l instead skolemising

Theorem Prover

```
prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,                    and
        prove(A,[B|UnExp],Lits,FreeV,VarLim).
prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,                     or
        prove(A,UnExp,Lits,FreeV,VarLim),
        prove(B,UnExp,Lits,FreeV,VarLim).
prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,              forall
        \+ length(FreeV,VarLim),
        copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
        append(UnExp,[all(X,Fml)],UnExp1),
        prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim).
prove(Lit,_,[L|Lits],_,_) :-                        literals; negation
        (Lit = -Neg; -Lit = Neg) ->
        (unify(Neg,L); prove(Lit,[],Lits,_,_)).
prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-             next formula
        prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).
```