- Akshat Sood

ELEMENTARY LOGIC WITH APPLICATIONS

PROPOSITIONAL LOGIC

→ <u>Proposition</u>: Sentence that is either true or false.

⇒ <u>Boolean Connectives</u>: A symbol that is used to modify a statement or to combine two statements to make more complex statements ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$)

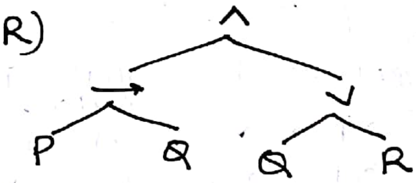
⇒ <u>Propositional Variables</u>: Symbols that is used to represent propositions.

* All propositional symbols are <u>atomic formulae</u>.
* Hierarchy of Logical Operators: $\boxed{\leftrightarrow, \rightarrow, \vee, \wedge, \neg}$ → ∀ and ∃ have the same precedence as ¬

⇒ <u>SYNTACTICAL TREE</u> → computer would prefer this

* Each node of the tree is a subformulae of the formula at its root

<u>Eg</u>: $(P \rightarrow Q) \wedge (Q \vee R)$



⇒ <u>Truth Tables</u>: Each row is an <u>interpretation</u> or a <u>scenario</u> or a <u>valuation</u> of the formula

* If there are k variables in a proposition, then the truth table will have 2^k rows in it.
* <u>Other ways to denote Implication</u> ($A \rightarrow B$)
  - If A then B
  - A only if B
  - B if A
  - B when A
  - B unless ¬A
  - A implies B
  - a necessary condition for A is B.
  - a suficient condition for B is A.
  - B whenever A
  - B follows A
  - A is sufficient for B.
  - B is necessary for A

⇒ <u>Tautology</u> → when a formula is true under all interpretations ($\top$)

⇒ <u>Contradiction</u> → Formula which is false under all interpretations ($\bot$)

SYNTACTICAL TRANSFORMATIONS

⇒ <u>FUNDAMENTAL LOGICAL EQUIVALENCES</u>

- $P \vee P \equiv P$ [Idempotency]
- $P \wedge P \equiv P$ [Idempotency]
- $P \vee Q \equiv Q \vee P$ [Commutativity]
- $P \wedge Q \equiv Q \wedge P$ [Commutativity]
- $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$ [Associativity]
- $P \vee (Q \vee R) \equiv P \vee (Q \vee R)$ [Associativity]
- $\boxed{P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)}$ [Distributivity]
- $\boxed{P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)}$ [Distributivity]
- $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ [De Morgan's Law]
- $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$ [De Morgan's Law]
- $P \vee \neg P \equiv 1$
- $P \wedge \neg P \equiv 0$
- $\neg\neg P = P$
- $P \vee 1 \equiv 1$
- $P \wedge 1 \equiv P$
- $P \wedge 0 \equiv 0$
- $P \vee 0 \equiv P$
- $P \rightarrow Q \equiv \neg P \vee Q$
- $\boxed{P \rightarrow Q \equiv \neg Q \rightarrow \neg P}$ [Contraposition]
- $1 \leftrightarrow P \equiv P$
- $0 \leftrightarrow P \equiv \neg P$
- $1 \rightarrow P \equiv P$
- $P \rightarrow 1 \equiv 1$
- $0 \rightarrow P \equiv 1$
- $P \rightarrow 0 \equiv \neg P$
- $P \leftrightarrow P \equiv 1$
- $P \leftrightarrow \neg Q \equiv \neg(P \leftrightarrow Q)$
- $\boxed{P \vee (P \wedge Q) \equiv P}$
- $\boxed{P \wedge (P \vee Q) \equiv P}$

⇒ NORMAL FORMS.

→ Literal: A propositional symbol or the negation of a propositional symbol

① DISJUNCTIVE NORMAL FORM (DNF)

⇒ Disjunction of 1 or more formulae each of which is a conjunction of 1 or more literals (atoms).

* In TRUTH TABLES or Quinne's Trees → Check interpretations which result in $\boxed{1}$
  → 1 = Variable → 0 = Compliment of Variable

② CONJUNCTIVE NORMAL FORM (CNF)

⇒ A formula is in CNF if it is a conjunction of one or more formulae each of which is a disjunction of 1 or more literals.

* In TRUTH TABLES or Quinne's Trees → Check for interpretations resulting in $\boxed{0}$
  → 1 = Compliment of Variable → 0 = Variable

⇒ RULES TO OBTAIN CNF and DNF

- $F \rightarrow G \equiv \neg F \vee G$
- $F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$
- $\neg(F \vee G) \equiv \neg F \wedge \neg G$
- $\neg(F \wedge G) \equiv \neg F \vee \neg G$
- $\neg\neg F = F$
- $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$
- $(F \vee G) \wedge H \equiv (F \wedge H) \vee (G \wedge H)$] for DNF
- $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
- $(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$] for CNF

⇒ COMPLETE SETS OF CONNECTIVES.

→ A set of connectives is called complete (or adequate) if every formula of propositional logic is equivalent to a formula using only connectives from this set.

⇒ Complete Sets: $\{\neg, \wedge, \vee\}$, $\{\neg, \vee\}$, $\{\neg, \wedge\}$, $\{\rightarrow, \neg\}$, $\{\rightarrow, 0\}$, $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$

⇒ QUINNE'S METHOD

→ To construct Quine's Tree (for W):-
- Start with W as the root of the tree.
- Take the first level of the tree with a propositional symbol, say P, • Let the left child of the node be n(P/1) and its right child be n(P/2) [where n is the formula at the node].
- Repeat till no more propositional symbols are left in the leaves.

→ W is a tautology if all the leaves are 1s, a contradiction if they're 0s.

→ Otherwise W is a contingency (satisfiable).

* To get a DNF from Quine's Tree, read all the 1s, and to get a CNF, 0s

---

SATISFIABILITY

→ A formula is satisfiable if there is an interpretation that makes the formula true.

- Models: The interpretations that make the formula satisfiable.
  ↳ It is a set. If interpretations v_1 and v_3 makes S satisfiable → $\boxed{mod(S) = \{v_1, v_3\}}$

* For a contradiction $\boxed{mod(S) = \emptyset}$ ie, the system is inconsistent.

⇒ LOGICAL CONSEQUENCE $\xrightarrow[\text{same thing}]{\text{mean the}}$

- B is a logical consequence of $A_1 \dots A_n$
- The argument $A_1 \dots A_n \vDash B$ is valid
- $A_1 \dots A_n \vDash B$
- B is semantically entailed by $A_1 \dots A_n$
- B is a valid consequence of $A_1 \dots A_n$

* $A_1 \dots A_n \vDash B$ only if:-
  - $A_1 \wedge \dots \wedge A_n \rightarrow B$ is a Tautology
  - $A_1 \wedge \dots \wedge A_n \wedge \neg B$ is a contradiction
  - Set $\{A_1, A_2 \dots A_n, \neg B\}$ is inconsistent.

* Condition for argument to be invalid : $\boxed{v(A_i) = 1 \text{, for all } 1 \le i \le n \text{, but } v(B) = 0}$ then $A_1 \ldots\ldots A_n \nvDash B$

⇒ <u>Inference Systems</u> : Can achieve reasoning at a syntactical level.

⇒ <u>Soundness</u> : An inference system is sound if all ~~the~~ of its rules are sound ie whenever $A_1 \ldots\ldots A_n \vdash B$, then $A_1 \ldots\ldots A_n \vDash B$

⇒ <u>Completeness</u> : An inference system is complete whenever $A_1 \ldots\ldots A_n \vDash B$, B can be derived from $A_1 \ldots A_n$ using its inference rules.

⇒ <u>NATURAL DEDUCTION</u> ⟶ Rules used to manipulate assumptions towards conclusions.

⇒ 2 ~~to~~ ways to manipulate:-

1) <u>Elimination Rules</u> : Obtaining new formoulae by breaking down existing ones.

2) <u>Introduction Rule</u> : Combine some formolae with connectives to generate new ones.

* Rules can be applied only if their premises are in the main formula
* The list of Rules is at the end of the notes.
* Subcomputation boxes will most probably not be in the test

PREDICATE LOGIC ⟶ Logic which uses variables to depict more complicated logics and propositions

* ~~For~~ $\boxed{\forall \rightarrow \text{Universal}}$, $\boxed{\exists \rightarrow \text{Existentral}}$ quantifier.

⇒ <u>Term</u> : Either a variable or a constant, or a function symbol applied to ~~an~~ arguments that are terms.

⇒ <u>Individual Variables</u> : Placeholders for arbitrary objects from the domain.

⇒ <u>Individual Constants</u> : Particular objects from the domain (eg: a, b, c etc).

⇒ <u>Function Symbols</u> : Particular functions over the domain.

⇒ <u>Atomic Formulae</u> : Made of predicate symbols applied ~~over~~ to arguments that are terms (they have no connectives).

* <u>Unary Predicate Symbols</u> : denote a subset of specified domain (Eg: Car(x))
* <u>Binary Predicate Symbols</u> : denote binary relations (Eg $>(x, y)$)

⇒ <u>WELL FORMED FORMULAE</u> ⟶ A complex formula constructed by combining atomic formulae using connectives.

* an <u>atom</u> is a wff.
* If F and G are wffs, then $(\neg F)$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$ are wffs.
* If F is a wff and x is a variable, $\forall x F(x)$ and $\exists x F(x)$ are wffs.

⇒ In $\forall x F(x)$, Formula F is the <u>scope</u> of x.

⇒ Iff x occurs in the scope of the quantifier ⟶ x is <u>bound</u>, otherwise x is free.

⇒ <u>Closed wff</u> : one with no free variables. <u>Open wff</u> : one with at least 1 free.
⟶ Also called a '<u>sentence</u>'

⇒ <u>Interpretation</u> gives a wff a <u>truth value</u>.

* $I(\neg F) = 1$ iff $I(F) = 0$
* $I(F \vee G) = 1$ iff $I(F) = 1$ or $I(G) = 1$
* $I(F \wedge G) = 1$ iff $I(F) = 1$ and $I(G) = 1$
* $I(F \leftrightarrow G) = 1$ iff $I(F \wedge G) = 1$ or $I(\neg F \wedge \neg G) = 1$
* $I(F \rightarrow G) = 1$ iff $I(\neg F \vee G) = 1$

* ⇒ <u>RELATIONSHIP BETWEEN QUANIFIERS</u> $\boxed{\neg \forall x F \equiv \exists x \neg F}$ $\boxed{\neg \exists x F \equiv \forall x \neg F}$

⇒ $\boxed{\forall x F \equiv \neg\neg \forall x F \equiv \neg \exists x \neg F}$ $\boxed{\exists x F \equiv \neg\neg \exists x F \equiv \neg \forall x \neg F}$

REASONING WITH QUANIFIERS

* Order of appearance of <u>same type</u> of quantifier is irrelevant. $\boxed{\forall x \forall y F \equiv \forall y \forall x F}$ and $\boxed{\exists x \exists y F \equiv \exists y \exists x F}$
* When <u>different types</u> of quantifiers are used:- $\boxed{\forall x \exists y F \not\equiv \exists x \forall y F}$ $\boxed{\exists y \forall x F \models \forall x \exists y F}$ $\boxed{\forall x \exists y F \not\models \exists y \forall x F}$

=> <u>Distribution of Quantifiers</u>

$$\begin{array}{|l|}\hline \forall x(P(x) \land Q(x)) \equiv \forall x(P(x)) \land \forall x(Q(x)) \\ \exists x(P(x) \lor Q(x)) \equiv \exists x(P(x)) \lor \exists x(Q(x)) \\ \exists x(P(x) \land Q(x)) \models \exists x(P(x)) \land \exists x(Q(x)) \\ \forall x P(x) \lor \forall x Q(x) \models \forall x(P(x) \lor Q(x)) \\ \hline\end{array}$$

→ These 2 are true converely. (first two)

→ These 2 are not true converely (last two)

γ is independant of x

* $\forall x[P(x) \lor \gamma] \equiv \forall x P(x) \lor \gamma$
* $\forall x[P(x) \land \gamma] \equiv \forall x P(x) \land \gamma$
* $\exists x[P(x) \lor \gamma] \equiv \exists x P(x) \lor \gamma$
* $\exists x[P(x) \land \gamma] = \exists x P(x) \land \gamma$
* $\forall x[P(x) \rightarrow \gamma] \equiv [\exists x P(x)] \rightarrow \gamma$
* $\exists x[P(x) \rightarrow \gamma] \equiv [\forall x P(x)] \rightarrow \gamma$
* $\forall x[\gamma \rightarrow P(x)] \equiv \gamma \rightarrow [\forall x P(x)]$
* $\exists x[\gamma \rightarrow P(x)] \equiv \gamma \rightarrow \exists x P(x)$

⇒ QUANTIFIER RULES

① <u>Universal Instantiation</u> : $\boxed{\dfrac{\forall x F}{F(x/d)}}$

② <u>Universal Generalisation</u> : $\boxed{\dfrac{F(x/d)}{\forall x d}}$

③ <u>Existential Instantiation</u> : $\boxed{\dfrac{\exists x F}{F(x/e)}}$ [e is specific]

④ <u>Existential Generalisation</u>: $\boxed{\dfrac{F(x/d)}{\exists x F}}$ [d is arbitrary]

DEFINITE CLAUSES

* <u>Clause</u> : finite disjunction of 1 or more literals.

=> <u>Propositional Horn clause</u>: Finite disjunction of literals with <u>no more than one</u> positive literal (ie literal without a negation).

⇒ <u>Propositional Definite Clause</u> : A propositional horn clause with <u>exactly one</u> positive literal.

$\underbrace{(\neg Q \lor \neg S)}_{\text{Horn Clause, but not definite}} \land \underbrace{(\neg Q \lor P)}_{\text{Definite Clause}} \land \underbrace{(\neg Q \lor P \lor R)}_{\text{Neither}}$

=> A definite clause of the form $\neg X_1 \lor \dots \lor \neg X_m \lor X$ can be represented as $\boxed{(X_1 \land X_2 \land \dots \land X_m) \rightarrow X}$ → Rule form of the definite clause.

* Rule form of $(\neg P \lor Q) \land (\neg P \lor R)$: '$P \rightarrow Q$, $P \rightarrow R$'
* Rule form of P : '$\rightarrow P$'
* Rule form of $\neg S \lor \neg R \lor T$: $(S \land R) \rightarrow T$

=> <u>BACKWARD REASONING</u> : Opposite of natural deduction. Used to prove soundness and correctness.

<u>Steps:-</u>
→ Can be applied to a set of formulae which are all definite clauses.
→ First convert all of them into their rule ~~form~~ form (Body → Head).
→ Start with query ?T, where T is what you have to prove.
→ Find a clause where T is the head and replace the query with the body.
→ For each symbol in the query find an → with the symbol as the head and replace it with the body.
→ Repeat till no more queries are left. Then you have SUCCESS

* You can choose any symbol of a query to expand, but by convention, the leftmost one is chosen.

* Backward Reasoning only works for those set of formulae which are all Definite Clauses.
* Most backward reasoning rules are Natural Deduction rules (E and I) applied backwards.
* We can represent a rule of the form $X_1 \wedge \ldots \wedge X_n \rightarrow X$, in the form $X_1, X_2 \ldots\ldots X_n \longrightarrow X$, so that we don't have to seperate them.
* Empty query is denoted by $\square$ (success), $\blacksquare$ incase of failure.
* If there are more than 1 rules that have the query literal as their head, there is a possibility that one of them succeeds and the other doesn't. You will have to show all possible derivation trees in this case.

—— × —— × —— × —— × —— × ——

$\boxed{\text{PRENEX NORMAL FORMS}}$ → formula which starts with 0 or more quantifiers followed by a quantifier free formula.

⇒ It is of the form $\boxed{Q_1x_1 \ldots\ldots Q_nx_nF}$ $Q_1x \ldots\ldots Q_nx_n$ is Prefix, F is matrix

⇒ Algorithm To Convert to Prenex normal Form:-

- Eliminate all occurences of $\longrightarrow$ and $\leftrightarrow$ from the formula in question.
- Move all negations inward such that, in the end, negations only appear in front of atoms
- Standardize the variables apart (when necessary)
- The prenex normal form can now be obtained by moving all ~~the~~ quantifiers to the front of the formula (not changing order of quantifiers).

→ Renaming the formula such that distinct variables (those which are in the scopes of different quantifiers) can be seperated → do this so that once all the quantifiers are pulled out, there is no confusion.

Eg: $\forall x (P(x) \longrightarrow Q(x)) \wedge \exists x R(x)$ becomes $\forall x (P(x) \longrightarrow Q(x)) \wedge \exists y R(y)$

Use these: $\boxed{F \wedge \exists x G \equiv \exists x (F \wedge G)}$ $\boxed{F \vee \exists x G \equiv \exists x (F \vee G)}$ where x does not appear F, but does in G.

$\boxed{F \wedge \forall x G \equiv \forall x (F \wedge G)}$ $\boxed{F \vee \forall x G \equiv \forall x [F \vee G)}$

$\boxed{Q_1 x F \wedge Q_2 y G \equiv Q_1 x Q_2 y (F \wedge G)}$ $\boxed{Q_1 x F \vee Q_2 y G \equiv Q_1 x Q_2 y (F \vee G)}$

* When transforming a formula to its prenex form we want to preserve the variations which make it true so that they also make the pnf true. To do so, it is important to standardize the variables.

• [Q_1 and Q_2 are quantifiers, x does not occur in G, y does not occur in F.]

* A clause in the matrix of a PNF formula is a first order Horn Clause if the prefix consist only of universal quantifiers quantifying over all variables in the clause and the clause ~~contains~~ consists of a finite disjunction of positive or negative atoms, with no more than one positive atom.
* To convert PNF ~~to~~ a FO definite rules, after transformation to PNF, convert the matrix to a CNF and then check whether it can be converted into a definite rule.

Eg $\forall x \forall y \forall z \, ((\neg R(x,y) \vee S(x,y)) \wedge Q(z))$

$\Rightarrow \forall x \forall y \forall z \, (R(x,y) \rightarrow S(x,y))$ and $\forall x \forall y \forall z \, Q(z)$

$\Rightarrow \forall x \forall y \, R(x,y) \rightarrow S(x,y)$ and $\forall z \, Q(z)$

—— × —— × —— × —— × —— × —— × ——

FO DEFINITE CLAUSE PROGRAMMING

→ If P is a ~~Pro~~gram of FO definite rules, and a query to P is a PNF formula $\exists x_1 \ldots \exists x_n F$.

→ ~~Just~~ Same as in the case of backward reasoning with propositional logic, state a query and then match it with the body of a rule which has the query for the head.

→ When matching queries to bodies, substituting variable where possible is fine. Once a substitution is applied, then continue with the substitution in the next query (remainder of the query).

* Only variables can be substituted, but, they can be substituted by variables constants or functions applied on terms.

* Substitutions of F and G if, $\boxed{S \text{ unifies } F \text{ and } G \text{ if } S(F) = S(G)}$

Eg: $Loves(x, becks)$ and $loves(posh, y)$, $S = \{(x/posh), (y/becks)\}$
$P(a, F(y))$ and $P(x, F(g(b)))$, $S = \{(x/a), (y/g(b))\}$

* Substitutions can be only applied on a formula once, not iteratively.

⇒ Most General Unifier (mgu): Is the substitution which does not make any unnecessary substitutions.

⇒ A $S1$ is the mgu of F_1 and F_2 iff for all other unifiers $S2$, there is some substitution $S3$ such that $\boxed{S2(\{F_1, F_2\}) = S3(S1(\{F_1, F_2\}))}$

Eg: $F_1 = m(x, y)$, $F_2 = m(a, z)$
$S1 = \{(x/a), (y/z)\} \Longrightarrow m(a, z) \Longrightarrow$ mgu
$S2 = \{(x/a), (y/b), (z/b)\} \Longrightarrow m(a, b)$
∴ There is a $S3 = \{(z/b)\}$, which applied to $m(a, z)$ gives $m(a, b)$

PREDICATE LOGIC PROGRAMMING

* While substituting in a derivation tree, keep in mind that the variables in the query should be different from the ones in rules, otherwise the mgu could be wrong

Formal Definition of DEFINITE CLAUSE PROGRAMMING

- Input: A program P of definite rules and a query $Q = \exists x_1 \ldots \exists x_m \alpha_1 \wedge \ldots \wedge \alpha_n$
  * We can replace $\wedge$ with comma.
  * We can drop $\exists x_1 \ldots \exists x_m$ as it is implicitly assumed that all variables in $\alpha_1, \ldots, \alpha_n$ are existentially quantified.
- Output: If $Q = \alpha_1 \wedge \ldots \wedge \alpha_n$ is logical consequence of P, then output Yes along with the substitution of variables in query Q, else output no.
- Method: 1. Initialize Progress = true
  2. While $Q \neq$ empty and progress = true, do
     choose atom α_i from $Q = \alpha_1 \ldots \alpha_n$
     2.1. Choose rules $\beta_1 \ldots \beta_m \rightarrow \gamma$ ($m \geq 0$) such that γ and α_i unify
     2.1.1. If necessary rename variables in $\beta_1 \ldots \beta_m \rightarrow \gamma$
     2.1.2. Find mgu, S, for α_i and γ.
     2.1.3. Replace α_i (in Q) by $\beta_1 \ldots \beta_m$
     2.1.4. Apply S to new query $\alpha_1 \ldots \alpha_{i-1}, \beta_1 \ldots \beta_m, \alpha_{i+1} \ldots \alpha_n$

2.2. else progress = false.

3. If (Query = empty and progress = true) then output yes and substitution of variables in query, else output no.

* When expressing the following as a definite clause (predicate logic) program:-

P if A and B and..... and Q $\Rightarrow$ A, B, ..., Q $\rightarrow$ P

P if A or B or..... or Q $\Rightarrow$ A $\rightarrow$ P , B $\rightarrow$ P ..... Q $\rightarrow$ P

$\Rightarrow$ Choice Points: In derivation trees are those points which have multiple outcomes. In propositional logic this used to happen when a query was the head of 2 rules and could thus be mapped in 2 ways. For this we had 2 make 2 derivation trees (one for success and other for failure). In propositional logic, if a choicepoint leads to failure, then you can backtrack to the choicepoint to test for a different case.

$\Rightarrow$ For 'not α' to be true (ie for α to be false) we need to show that α is not known to be true $\rightarrow$ every attempt to prove α fails $\rightarrow$ there is no derivation tree for α.

$\Rightarrow$ Closed World Assumption: Presumption that what can currently ~~be~~ not be shown to be true is false.

* When you encounter a 'not' in a derivation tree, then make a side derivation to try and prove that every attempt to prove it fails.

* PREDICATE LOGIC PROGRAMMING = Definite Clause Programming +
  (1) Control (procedural features with selection of leftmost query atom and topmost program rule or fact).
  (2) Backtracking to choice points.
  (3) Negation as failure (Closed World Assumptions (CWA)).

$\Rightarrow$ Predicate Logic programming derivation includes all derivation trees obtained on backtracking (in order to prove query and all trees attempting to prove ~~a~~ given a query 'not α'.

$\Rightarrow$ Recursion: Process of repeating items in a self-similar way. It is a method where the solution to a problem depends on solutions to smaller instances of the same problem. The smallest instance of the solution to the problem is the base case.

I have summarised the rules for everything, but for a better understanding of subcomputation boxes and derivation trees, go through the examples in the lecture slides.

Summary of natural deduction rules

Basic rules

(∧I)	$\dfrac{A, B}{A \wedge B}$	**Conjunction**	$\dfrac{A, B}{B \wedge A}$	(∧I)
(∧E)	$\dfrac{A \wedge B}{A}$		$\dfrac{A \wedge B}{B}$	(∧E)
(∨I)	$\dfrac{A}{A \vee B}$	**Disjunction**	$\dfrac{B}{A \vee B}$	(∨I)
(∨E)	$\dfrac{A \to C, B \to C, A \vee B}{C}$			
(→I)	If $\dfrac{\text{assumptions, } A}{B}$ then $\dfrac{\text{assumptions}}{A \to B}$ (antecedent of the "If . . . then" shown in a subcomputation box)	**Implication**	$\dfrac{A, A \to B}{B}$ (also known as modus ponens)	(→E)
(¬I)	$\dfrac{A \to B, A \to \neg B}{\neg A}$	**Negation**	$\dfrac{\neg A \to B, \neg A \to \neg B}{A}$	(¬E)

Derived rules

(∨E1)	$\dfrac{A \vee B, \neg A}{B}$ (disjunctive syllogism)	**Disjunction**	$\dfrac{A \vee B, \neg B}{A}$ (disjunctive syllogism)	(∨E2)
(CD)	$\dfrac{A \to C, B \to D, A \vee B}{C \vee D}$ (constructive dilemma)		$\dfrac{A \to C, B \to D, \neg C \vee \neg D}{\neg A \vee \neg B}$ (destructive dilemma)	(DD)
(→I1)	$\dfrac{\neg A}{A \to B}$	**Implication**	$\dfrac{B}{A \to B}$	(→I2)
(→E1)	$\dfrac{A \to B}{\neg A \vee B}$			
(¬E1)	$\dfrac{\neg\neg A}{A}$	**Negation**	$\dfrac{\neg A \to B, A \to B}{B}$	(¬E2)
(¬I1)	$\dfrac{A}{\neg\neg A}$		$\dfrac{A \to B, \neg A \to B}{B}$	(¬2)