- **Proposition** → a statement that can be evaluated as true or false.
- **Inductive Definition of a Set**:
  1) a universe of all elements denoted by X (eg $X = \mathbb{R}$)
  2) a core set denoted by A (for "atoms"), with $A \subseteq X$. (eg $A = \{0\}$)
  3) a set of operations (functions) $X \to X$, denoted by F.
     - the elements of F are functions, f, each having some arity, $k \geq 1$.

     ie, k is the number of arguments that f takes.

eg, $F = \{s(x) = x + 1\}$, ie, s is the successor function. Different functions have different arities - there is not a single arity k which applies to all the fs.

- **Closed Subsets**: given any subset $Y \subseteq X$, and any set F of operations, Y is closed under F if, for every $f \in F$ (say f is a k-ary function) and every $y_1, \ldots, y_k \in Y$, $f(y_1, \ldots, y_k) \in Y$.

  ↳ **Examples**:
  1) Letting $Y = \emptyset$, the above definition is vaccuously satisfied.
  2) Let Y be the set of even natural numbers. Let F be the set of addition and multiplication. Then we know Y is closed under F.

- **Minimal Sets**: Y is a minimal set with respect to property R if:
  1) Y satisfies R
  2) for every set Z that satisfies R, $Y \subseteq Z$.

  ↳ **Formal definition**: $I(X, A, F)$ = the minimal subset of X that
  1) contains A
  2) is closed under the operations in F.

- **Structural Induction** to prove a desirable property R holds for every element of an inductively defined set, $I(X, A, F)$.
  1) prove that $R(a)$ holds for every a in the core set A (the base case)
  2) prove that, for every k-ary $f \in F$ (for any $k \geq 1$), and any

$y_1, \ldots, y_k \in X$ such that $R(y_1), \ldots, R(y_k)$ all hold, we also have that $R(f(y_1, \ldots, y_k))$ holds (the inductive case).

- $\mathcal{L}^P$ → the propositional language.
- $\text{Form}(\mathcal{L}^P)$ → the set of formulas in $\mathcal{L}^P$.

↳ if an expression in $\mathcal{L}^P$ can be produced by the formation rules, then it is a formula in $\text{Form}(\mathcal{L}^P)$

- Expressions
  - ε is the empty expression.
  - concatenating two expressions, U and V, is denoted by UV.
    ↳ note that $\varepsilon U = U = U\varepsilon$ for any expression U.
  - if $U = W_1 V W_2$, where U, W_1, V, and W_2 are expressions, then:
    - V is a segment of U
    - if $V \neq U$, then V is a proper segment of U!
  - every expression is a segment of itself.
  - the empty expression ε is a segment of every expression.
  - if $U = VW$, then V is an initial segment (prefix) and W is a terminal segment (suffix) of U.
    ↳ if $W \neq \varepsilon$, then V must be a proper initial segment (proper prefix) of U
    ↳ if $V \neq \varepsilon$, then W must be a proper terminal segment (proper suffix) of U.

Review of Mathematical Induction

- a statement "every natural number has property P" corresponds to a sequence of statements $P(0)$, $P(1)$, $P(2)$, $P(3)$, ...
- Principle of Mathematical Induction:

↳ if we establish only these two things:

1) 0 has property P

2) whenever a natural number has property P, then the next natural number also has property P

then we may conclude that every natural number has property P!

- example: show that $\sum_{x=0}^{n} x = \frac{n(n+1)}{2}$ for every natural number n.

↳ let P be the property, ie, let P(n) be "$\sum_{x=0}^{n} x = \frac{n(n+1)}{2}$"

- base case (P(0)): $\sum_{x=0}^{0} x = \frac{0(0+1)}{2} \Rightarrow 0 = \frac{0}{2}$.

since we got $0 = 0$, the base case holds.

- inductive step: hypothesize that an arbitrary natural number k, has property P, ie, $\sum_{x=0}^{k} x = \frac{k(k+1)}{2}$

we now need to demonstrate that k+1 also has property P. ie, we must prove that $\sum_{x=0}^{k+1} x = \frac{(k+1)(k+1+1)}{2} = \frac{(k+1)(k+2)}{2}$.

↳ $\sum_{x=0}^{k+1} x = \sum_{x=0}^{k} x + k+1 = \frac{k(k+1)}{2} + k+1 = \frac{k^2+k}{2} + \frac{2k+2}{2} = \frac{k^2+3k+2}{2} = \frac{(k+1)(k+2)}{2}$

(defn. of Σ; inductive hypothesis; algebra; algebra; algebra)

∴ since we have shown that P holds for k=0, and holds for k+1 when it holds for k, this property must hold for all natural numbers.

- inductive principle template: prove base case, hypothesize P(k), and prove P(k+1).

- Simple vs Strong Induction
  - Base Case
    - ↳ simple: show P(0) holds

      strong: show P(0) holds
  - Inductive Hypothesis
    - ↳ simple: assume P(k) holds

strong: assume P(m) holds for every $m \leq k$

- Inductive Step
  - ↳ simple: show P(k+1) holds

    strong: show P(k+1) holds
- Conclusion
  - ↳ simple: P(k) holds for every k

    strong: P(k) holds for every k

Recursively Defined Sets

- made up of:
  1) Base - a statement that certain objects belong to the set
  2) Recursion - a collection of rules indicating how to form new set objects from those already known to be in the set
  3) Restriction - a statement that no objects belong to the set other than those coming from 1 and 2.
- example: the set of natural numbers $\mathbb{N}$ is a recursively defined set:
  1) Base: 0 is a natural number in $\mathbb{N}$
  2) Recursion: if k is a natural number in $\mathbb{N}$, then k+1 is a natural number in $\mathbb{N}$.
  3) Restriction: no other numbers are in $\mathbb{N}$.

Structural Induction for Recursively Defined Sets

- let S be a recursively defined set, and consider a property S that objects in S may or may not have. The proof that every object in S satisfies the property consists of:
  1) Base Case(s): show that each object in the Base for S satisfies the property
  2) (Composite) Inductive Step: show that, for each rule in the Recursion

of S, if the rule is applied to objects in S that satisfy the property, then the new objects defined by the rule also satisfy the property.

- if the set S is the set of natural numbers $\mathbb{N}$, structural induction amounts to classical mathematical induction.

Unique Readability Theorem: every formula is exactly one of: an atom, $\neg B$, $B \wedge C$, $B \vee C$, $B \Rightarrow C$, or $B \Leftrightarrow C$, and, in each case, it is of that form in that form in exactly one way.

Precedence Rules:

↳ $\neg$ has precedence over $\wedge$

$\wedge$ has precedence over $\vee$

$\vee$ has precedence over $\Rightarrow$

$\Rightarrow$ has precedence over $\Leftrightarrow$

Scope

- if $(\neg A)$ is a segment of formula C, then A is called the scope of this negation in the formula C.
- if $(A * B)$ is a segment of a formula C, then A is called the left scope of the connective and B is called the right scope of the connective in the formula C.

Syntax vs Semantics

- Syntax is concerned with the rules used for constructing the formulas in $Form(\mathcal{L}^P)$
- Semantics is concerned with meaning:
  - Atoms (proposition symbols) are intended to express simple propositions ↗ Sentences
  - The connectives take their meanings (ie, $\neg$ expresses "not", etc)

A truth valuation is a function t → $t: Atom(\mathcal{L}^P) \to \{0, 1\}$ with the set of all proposition symbols as domain and $\{0, 1\}$ as range.

↳ corresponds to a row in a truth table!!

- **Satisfiability**: we say that a truth valuation t satisfies a formula A in $Form(\mathcal{L}^p)$ iff $A^t = 1$.
  - we use Σ to denote any set of formulas.

↳ define $\Sigma^t = \begin{cases} 1 & \text{if for each formula } B \in \Sigma,\ B^t = 1 \\ 0 & \text{otherwise} \end{cases}$

↳ note that $\Sigma^t = 1$ means that under truth valuation t, ALL the formulas of Σ are true.
$\Sigma^t = 0$ means that for at least one formula $B \in \Sigma$, we have that $B^t = 0$.

- A set of formulas $\Sigma \subseteq Form(\mathcal{L}^p)$ is satisfiable iff there exists a truth valuation t such that $\Sigma^t = 1$. Otherwise, the set of formulas Σ is called unsatisfiable.

- **Tautology**: a formula A is a tautology if and only if it's true under all truth valuations, ie, iff for any truth valuation t, we have that $A^t = 1$.
- **Contradiction**: a formula A is a contradiction if and only if it is false under all truth valuations, ie, iff for any truth valuation t, we have that $A^t = 0$.
- **Contingent**: a formula that is neither a tautology nor a contradiction is called contingent, ie, it is sometimes true and sometimes false.

- Important Tautology: **Law of the Excluded Middle**
  ↳ $p \vee \neg p$ is always true!

- if A is a tautology that contains proposition symbol p, we can determine a new expression by replacing all instances of p by an arbitrary formula. The resulting formula A' is also a tautology!

- Important Contradiction: **Law of Contradiction**
  ↳ $\neg(p \vee \neg p)$, or equivalently, $p \wedge \neg p$ is always false!

set of formulas, aka, premises — conclusion

Tautological Consequence: suppose $\Sigma \subseteq Form(\mathcal{L}^P)$ and $A \in Form(\mathcal{L}^P)$. A is a tautological consequence of Σ (that is, of the formulas in Σ), written as $\Sigma \models A$, iff for any truth valuations t, we have that $\Sigma^t = 1$ implies $A^t = 1$.

↳ Observations:

- $\models$ is not a symbol of the formal propositional language and $\Sigma \models A$ is not a formula.
- $\Sigma \models A$ is a statement (in the metalanguage) about Σ and A.
- we write $\Sigma \not\models A$ for "not $\Sigma \models A$"
- if $\Sigma \models A$, we say that the formulas in Σ (tauto)logically imply formula A.

- **Special Case:** $\emptyset \models A$.

  ↳ vacuously, $\emptyset^t = 1$ for any truth valuation t.

  Therefore, A must be a tautology!

- The following are equivalent:
  - The argument with premises $A_1, A_2, \ldots, A_n$ and conclusion C is valid
  - $(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \rightarrow C$ is a tautology
  - $(A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \neg C)$ is a contradiction
  - The formula $(A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \neg C)$ is not satisfiable
  - The set $\{A_1, A_2, \ldots, A_n, \neg C\}$ is not satisfiable
  - C is a tautological consequence of Σ, ie, $\{A_1, A_2, \ldots, A_n\} \models C$.

Valid Argument: an argument that is both sound and correct

↳ a conclusion is only guaranteed to be true when the argument is valid AND the premises are all true.

Tautological Equivalence: for two formulas we write $A \dashv\vdash B$ to denote "$A \vDash B$" and "$B \vDash A$".

↳ A and B are said to be tautologically equivalent (or simply equivalent) if $A \dashv\vdash B$ holds.

↳ note: tautological equivalence is weaker than equality of formulas.

↳ $A \dashv\vdash B$ if and only if $A \leftrightarrow B$ is a tautology (or, $\emptyset \vDash A \leftrightarrow B$)

Proving Argument Validity with Truth Tables:

↳ example: show that $\{p \to q,\ q \to r\} \vDash (p \to r)$

p	q	r	$p \to q$	$q \to r$	$A_1 \wedge A_2$	$p \to r$ (conclusion)
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	0	0	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	0
1	1	1	1	1	1	1

↳ since the condusion $(p \to r)$ is always true when the premises $(p \to q,\ q \to r)$ is true, the argument is valid!

- example: prove that $(p \to q) \vee (p \to r) \nvDash p \to (q \wedge r)$

↳ solution: find at least one row in the truth table in which the premises are true but the conclusion is false.

the row in the truth table that corresponds to the truth valuation t which assigns $p^t = 1$, $q^t = 1$, and $r^t = 0$ is one such counterexample.

Proving Arguments without Truth Tables:

- **Proof by Contradiction**: assume the contrary and show that it's impossible.

example: show that $\{A \to B,\ B \to C\} \models (A \to C)$.

let's assume the contrary, that is, $\{A \to B,\ B \to C\} \not\models (A \to C)$.

this means that there must be a truth valuation t that makes all premises true but the conclusion false. that is: $(A \to B)^t = 1$, $(B \to C)^t = 1$, and $(A \to C)^t = 0$.

since $(A \to C)^t = 0$, we have that $A^t = 1$ and $C^t = 0$.

$\therefore$, since $A^t = 1$, we have $(1 \to B)^t = 1$, so $B^t = 1$.

$\therefore$, since $B^t = 1$, we have $(1 \to C)^t = 1$, so $C^t = 1$. ← contradiction!

since we have shown C^t is both 0 and 1 simultaneously, the contrary argument $\{A \to B,\ B \to C\} \not\models (A \to C)$ does not hold.

Therefore, the original argument $\{A \to B,\ B \to C\} \models (A \to C)$ must hold by a proof by contradiction.

Important (Tauto)logical Equivalences:

De Morgan's Laws

↳ $\neg(p \vee q)$ is equivalent to $\neg p \wedge \neg q$, and $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$!

Contrapositive

↳ $p \to q$ is equivalent to $\neg q \to \neg p$!

↳ Converse vs Contrapositive

- Suppose we have an implication of the form $(P \to Q)$
- Its converse is $(Q \to P)$, and its contrapositive is $(\neg Q \to \neg P)$
- Contrapositives are equivalent, $(P \to Q) \dashv\vdash (\neg Q \to \neg P)$
- The converse of an implication is NOT necessarily equivalent to it, (unless $P \dashv\vdash Q$).

- **Duality:** suppose A is a formula composed of only atoms and the

connectives $\neg$, $\wedge$, and $\vee$. Suppose $\Delta(A)$ results from simultaneously replacing in A all occurences of $\wedge$ with $\vee$, all occurences of $\vee$ with $\wedge$, and each atom with its negation. Then $\neg A \models \Delta(A)$.

- **Removing connectives**
  - ↳ $A \Rightarrow B$ is equivalent to $\neg A \vee B$
  - ↳ $A \Leftrightarrow B$ is equivalent to $(A \wedge B) \vee (\neg A \wedge \neg B)$, OR, $(\neg A \vee B) \wedge (\neg B \vee A)$ ($A \Rightarrow B$, $B \Rightarrow A$)
  - ↳ example: remove the $\Rightarrow$ and $\Leftrightarrow$ from $(p \Rightarrow q \wedge r) \vee ((r \Leftrightarrow s) \wedge (q \vee s))$
    - $(\neg p \vee q \wedge r) \vee (((\neg r \vee s) \wedge (\neg s \vee r)) \wedge (q \vee s))$
- **Essential Laws for Propositional Logic**
  - $A \vee \neg A \models 1$ — law of the excluded middle
  - $A \wedge \neg A \models 0$ — law of contradiction
  - $A \vee 0 \models A$, $A \wedge 1 \models A$ — identity laws
  - $A \vee 1 \models 1$, $A \wedge 0 \models 0$ — domination laws
  - $A \vee A \models A$, $A \wedge A \models A$ — idempotent laws
  - $\neg(\neg A) \models A$ — double-negation law
  - $A \vee B \models B \vee A$, $A \wedge B \models B \wedge A$ — commutativity laws
  - $(A \vee B) \vee C \models A \vee (B \vee C)$, $(A \wedge B) \wedge C \models A \wedge (B \wedge C)$ — associativity laws
  - $A \vee (B \wedge C) \models (A \vee B) \wedge (A \vee C)$, $A \wedge (B \vee C) \models (A \wedge B) \vee (A \wedge C)$ — distributivity laws
  - $\neg(A \wedge B) \models \neg A \vee \neg B$, $\neg(A \vee B) \models \neg A \wedge \neg B$ — de morgan's laws
  - $A \vee (A \wedge B) \models A$, $A \wedge (A \vee B) \models A$ — absorption laws
  - $(A \wedge B) \vee (\neg A \wedge B) \models B$, $(A \vee B) \wedge (\neg A \vee B) \models B$
- **Dual Pairs**
  - ↳ all laws come in pairs, called dual pairs (↗ except double negation!)
  - ↳ for each formula depending only on the connectives $\neg$, $\wedge$, and $\vee$, the dual is found by replacing all 1s by 0s, all 0s by 1s, all $\vee$s by $\wedge$s, and all $\wedge$s with $\vee$s.
- **Literal**: a formula is called a literal if it is of the form p or $\neg p$,

where p is a proposition symbol.

↳ the two formulas p and $\neg p$ are called complementary literals.

Normal Forms

- Disjunctive Clause: a disjunction with literals as disjuncts
  - ↳ eg: $(p \vee q \vee \neg r)$
- Conjunctive Clause: a conjunction with literals as conjucts
  - ↳ eg: $(\neg p \wedge s \wedge \neg q)$

- Disjunctive Normal Form (DNF): a disjunction with conjunctive clauses.
  - ↳ eg: $(p \wedge q) \vee (p \wedge \neg q) \vee p$, $p \vee (q \wedge r)$, $\neg p \vee t$
  - ↳ eg: $\neg(p \wedge q) \vee r$ is NOT in DNF because of the $\neg$!
- Conjunctive Normal Form (CNF): a conjunction with disjunctive clauses.
  - ↳ eg: $p \wedge (q \vee r) \wedge (\neg q \vee r)$, $p \wedge q$
  - ↳ eg: $p \wedge (r \vee (p \wedge q))$ is NOT in CNF because of the nested brackets.

- example: is $\neg p \wedge q \wedge \neg r$ in CNF or DNF?
  - ↳ CNF of degenerative literals, but can also think of this as one conjunctive clause as a formula in DNF!

- How to obtain Normal Forms?
  - ↳ use: $A \Rightarrow B \quad \Vdash \quad \neg A \vee B$

    $A \Leftrightarrow B \quad \Vdash \quad (\neg A \vee B) \wedge (\neg B \vee A)$

    $A \Leftrightarrow B \quad \Vdash \quad (A \wedge B) \vee (\neg A \wedge \neg B)$

    $\neg(\neg A) \quad \Vdash \quad A$

    de morgan's laws

    distributivity laws

- example: convert the following formula to CNF: $\neg((p \vee \neg q) \wedge \neg r)$

$\neg((p \vee \neg q) \wedge \neg r) \models \neg(p \vee \neg q) \vee \neg\neg r$ de morgan's

$\models \neg(p \vee \neg q) \vee r$ double negation

$\models (\neg p \wedge \neg\neg q) \vee r$ de morgan's

$\models (\neg p \wedge q) \vee r$ double negation

$\models (\neg p \vee r) \wedge (q \vee r)$ distributivity

↳ $(\neg p \vee r) \wedge (q \vee r)$ is in CNF!

- Formulas for CNF / DNF from truth tables

- to get the DNF

↳ find the rows in the truth table for which the function outputs 1, and specify each disjunctive clause specifically.

↳ example: find the DNF from the following truth table:

p	q	r	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

based on lines 2, 6, and 8, we see that f is only 1 when p/q/r take values 0/0/1, 1/0/1, 1/1/1.

∴, the DNF is $(\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge r) \vee (p \wedge q \wedge r)$

- to get the CNF:

↳ determine the DNF for $\neg f$

compute $\Delta(\neg f)$ ($\models \neg\neg f \models f$), as $\Delta(\neg f)$ can be easily converted to CNF.

↳ example: find the CNF of the function f given by the following truth table:

p	q	r	f	¬f
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

- first, add a ¬f column (see on truth table in green)

DNF of $\neg f = (p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r)$

$\Delta(\neg f) = (\neg p \vee \neg\neg q \vee \neg r) \wedge (\neg p \vee \neg\neg q \vee \neg\neg r) \wedge (\neg\neg p \vee \neg\neg q \vee \neg r)$

$= (\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee q \vee \neg r) = f$

- as $\Delta(\neg f) == \neg\neg f = f$, the CNF for f is $(\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee q \vee \neg r)$.

- **Adequate:** any set of connectives with the capability to express any truth table is said to be adequate.
  - ↳ formally, a set S of connectives is called adequate if and only if any n-ary connective can be defined in terms of the connectives in S.
  - ↳ the set of the five standard connectives $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ is adequate!
  - ↳ but, the set $\{\neg, \wedge, \vee\}$ is also adequate!
    - proof: we can recreate any truth table in DNF form using only these connectives.
- to show if a set of connectives is adequate, we must show that each connective in a known adequate set can be written in terms of the connectives in the new set.
- example: show that $\{\neg, \wedge\}$ is an adequate set:

  we know that $\{\neg, \wedge, \vee\}$ is an adequate set.
  - ¬ and ∧ are in both sets, so no changes necessary there.
  - ∨ can be re-written in terms of ∧: $B \vee C \models \neg(\neg B \wedge \neg C)$
  - since every formula using the connectives in the adequate set $\{\neg, \vee, \wedge\}$ can be written using only the connectives in $\{\neg, \wedge\}$, $\{\neg, \wedge\}$ is

also an adequate set!

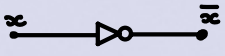
- interestingly, NOR ($\downarrow$) by itself is adequate!
  - ↳ proof:
    - $\{\neg, \wedge, \vee\}$ is adequate
    - $\neg p \equiv p \downarrow p$
    - $p \wedge q \equiv (p \downarrow p) \downarrow (q \downarrow q)$
    - $p \vee q \equiv (p \downarrow q) \downarrow (p \downarrow q)$
    - each connective in the adequate set $\{\neg, \vee, \wedge\}$ can be written solely in terms of $\downarrow$, so $\{\downarrow\}$ is an adequate set!
- additionally, NAND ($|$) is also adequate on it's own!
  - ↳ proof:
    - $\{\neg, \wedge, \vee\}$ is adequate
    - $\neg p \equiv p \mid p$
    - $p \wedge q \equiv (p \mid q) \mid (p \mid q)$
    - $p \vee q \equiv (p \mid p) \mid (q \mid q)$
    - each connective in the adequate set $\{\neg, \vee, \wedge\}$ can be written solely in terms of $|$, so $\{|\}$ is an adequate set!
- to show that a set of connectives is not adequate, we must show that a connective in a set of connectives that is known to be adequate cannot be expressed in terms of the connectives in the new set.
  - ↳ example: show that $\{\wedge\}$ is not adequate:
    - we know that $\{\neg, \vee, \wedge\}$ is adequate.
    - however, $\neg$ cannot be expressed using only $\wedge$
      - ↳ $p \wedge p \ldots \wedge p$ can only ever be value p, not $\neg p$!

∴, $\{\wedge\}$ is not adequate!

Boolean Algebra

- A boolean algebra is a set B, together with two binary operations $+$ and $\cdot$, and a unary operation $\bar{}$. The set B contains elements 0 and 1, is closed under the application of $+$, $\cdot$, and $\bar{}$, and the following properties hold for all x, y, z in B:
  - identity laws: $x + 0 = x$ and $x \cdot 1 = x$.
  - complement laws: $x + \bar{x} = 1$ and $x \cdot \bar{x} = 0$
  - associativity laws: $(x+y)+z = x+(y+z)$ and $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
  - commutativity laws: $x+y = y+x$ and $x \cdot y = y \cdot x$
  - distributivity laws: $x+(y \cdot z) = (x+y) \cdot (x+z)$ and $x \cdot (y+z) = (x+z) \cdot (x+y)$

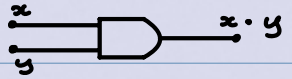
- logic gate: an electronic device that operates on a collection of bits and produces one binary output.
  - ↳ physically implemented by transistors
- Basic Logic Gates:
  - The Invertor (NOT)



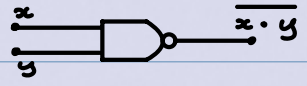
  - The OR gate

  - The NOR gate

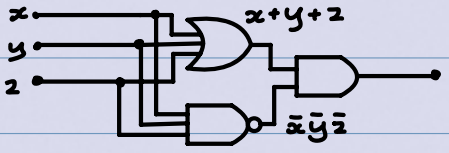
    - ↳ $x \oplus y$ is 1 only if both inputs x and y are 0.
  - The AND gate



  - The NAND gate



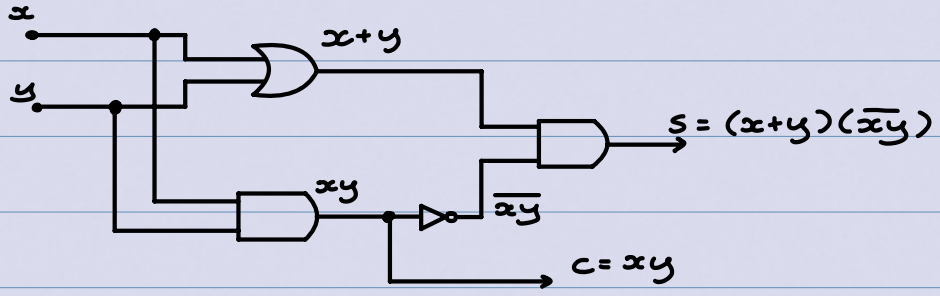
- Combinational Logic Circuits (Combinatorial Circuits): memoryless digital logic circuits whose output is a function of the present value of the inputs only.
  - ↳ implemented as a combination of NOT, OR, and AND gates.
- example: design a circuit to produce the output $(x+y+z)(\bar{x}\bar{y}\bar{z})$



- Example of useful circuits: **Adders**
  - ↳ Logic circuits can be used to carry out addition of two positive integers from their binary expansions
- **Half-Adder**
  - input will be two bits, x and y
  - output will be two bits, s (sum) and c (carry)
  - adds two bits without considering a carry from a previous operation.
  - truth table:

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

  - from this truth table, we observe that $S = x\bar{y} + \bar{x}y$ and $C = xy$.
    - ↳ note: $S = x\bar{y} + \bar{x}y = (x+y)(\overline{xy})$, which takes 4 gates instead of 6.
  - half-adder circuit:



- **Full Adder**
  - input bits: x, y, and carry bit from the previous operation, c_i
  - output bits: the sum bit s and the carry bit c_{i+1}
  - truth table:

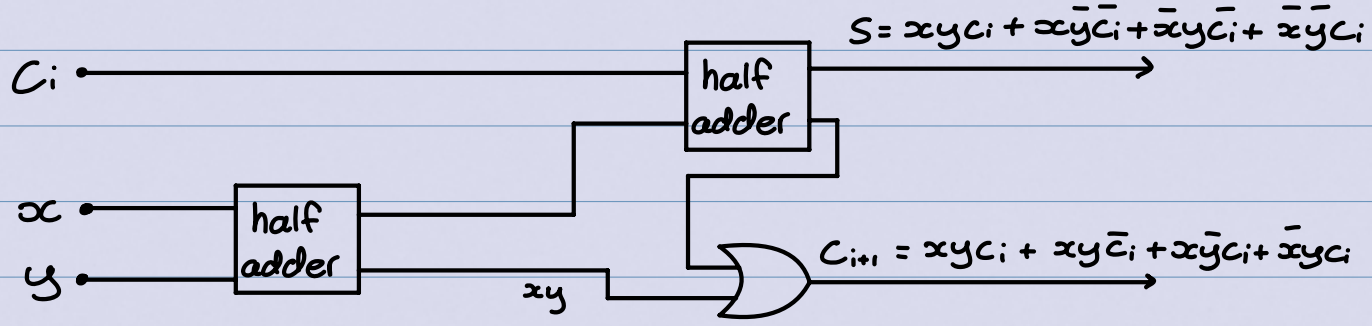
x	y	c_i	S	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

  - from this truth table, we observe that:

↳ $S = xyc_i + x\bar{y}\bar{c}_i + \bar{x}y\bar{c}_i + \bar{x}\bar{y}c_i$

$c_{i+1} = xyc_i + xy\bar{c}_i + x\bar{y}c_i + \bar{x}yc_i$

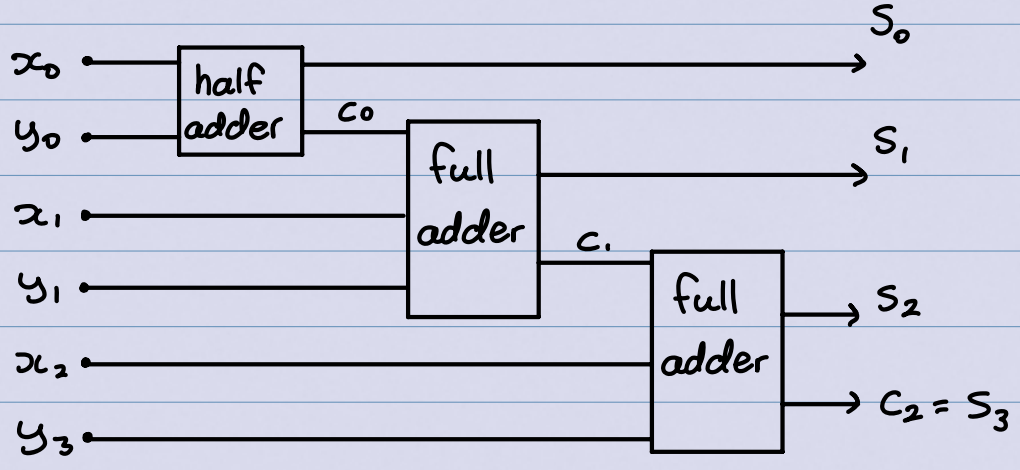
- Circuit for full-adders using half-adders:



- Adding 3-bit integers:

↳ full and half-adders can now be combined to add the 3-bit integers $(x_2 x_1 x_0)_2$ and $(y_2 y_1 y_0)_2$ to produce the sum $(S_3 S_2 S_1 S_0)_2$.

↳ note: S_3 is given by the carry c_2!

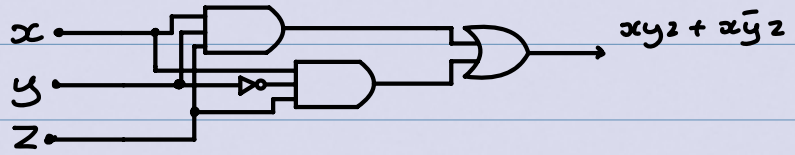


Circuit Minimization Through Formula Simplification

↳ example: consider the circuit that has output 1 if and only if $x=y=z=1$ or $x=z=1$ and $y=0$. Design a circuit for this.

- the formula corresponding to this circuit's truth table is $xyz + x\bar{y}z$.

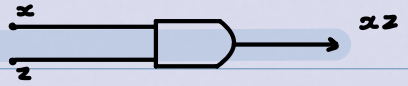
This is drawn as such:



however, we can significantly simplify this circuit:

$xyz + x\bar{y}z = (y+\bar{y})(xz) = 1(xz) = xz.$

∴ our final circuit is simply:



- example: minimize the circuit implementing the following truth table:

x	y	z	A
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

we get the DNF from observation: $A = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z.$

While this is technically correct, we can further simplify it:

↳ $\bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z = (\bar{y}+y)(\bar{x}z) + (\bar{z}+z)(x\bar{y})$

$= (1)(\bar{x}z) + (1)(x\bar{y}) = \bar{x}z + x\bar{y}$

Formal Deducability

- denoted by $\vdash$ to signify argument validity
- A relation between a set of formulas Σ (the premises) and a formula A (the conclusion).
- We write $\Sigma \vdash A$ to mean A is formally deducible (or provable) from Σ.
- The 11 rules of formal deduction:

1) $A \vdash A$ (Ref / Reflexivity)
2) if $\Sigma \vdash A$, then $\Sigma, \Sigma' \vdash A$ (+ / Addition of Premises)
3) if $\Sigma, \neg A \vdash B$ and $\Sigma, \neg A \vdash B'$, then $\Sigma \vdash A$ ($\neg$- / $\neg$ elimination)
4) if $\Sigma \vdash A \Rightarrow B$ and $\Sigma \vdash A$, then $\Sigma \vdash B$ ($\Rightarrow$- / $\Rightarrow$ elimination)
5) if $\Sigma, A \vdash B$ then $\Sigma \vdash A \Rightarrow B$ ($\Rightarrow$+ / $\Rightarrow$ introduction)
6) if $\Sigma \vdash A \wedge B$, then $\Sigma \vdash A$ and $\Sigma \vdash B$ ($\wedge$- / $\wedge$ elimination)
7) if $\Sigma \vdash A$ and $\Sigma \vdash B$, then $\Sigma \vdash A \wedge B$ ($\wedge$+ / $\wedge$ introduction)

8) if $\Sigma, A \vdash C$ and $\Sigma, B \vdash C$, then $\Sigma, A \vee B \vdash C$ ($\vee-$ / $\vee$ elimination)

9) if $\Sigma \vdash A$ then $\Sigma \vdash A \vee B$ and $\Sigma \vdash B \vee A$ ($\vee+$ / $\vee$ introduction)

10) if $\Sigma \vdash A \Leftrightarrow B$ and $\Sigma \vdash A$, then $\Sigma \vdash B$. ($\Leftrightarrow-$ / $\Leftrightarrow$ elimination)

if $\Sigma \vdash A \Leftrightarrow B$ and $\Sigma \vdash B$, then $\Sigma \vdash A$ ($\Leftrightarrow-$ / $\Leftrightarrow$ elimination)

11) if $\Sigma, A \vdash B$ and $\Sigma, B \vdash A$, then $\Sigma \vdash A \Leftrightarrow B$ ($\Leftrightarrow+$ / $\Leftrightarrow$ introduction)

- example: prove the "membership rule" that if $A \in \Sigma$, then $\Sigma \vdash A$.

suppose $A \in \Sigma$ and $\Sigma = A, \Sigma'$.

then, 1) $A \vdash A$ by ref

2) $A, \Sigma' \vdash A$ by +, 1

since $\Sigma = A, \Sigma'$, we have shown that $\Sigma \vdash A$!

- this is a new theorem we can use!

↳ if $A \in \Sigma$, then $\Sigma \vdash A$ ($\in$ / membership)

↳ aka, hypothetical syllogism!

- example: prove that $A \Rightarrow B, B \Rightarrow C \vdash A \Rightarrow C$

1) $A \to B, B \to C, A \vdash A$ by $\in$

2) $A \to B, B \to C, A \vdash A \to B$ by $\in$

3) $A \to B, B \to C, A \vdash B$ by $\to-$ on 1,2

4) $A \to B, B \to C, A \vdash B \to C$ by $\in$

5) $A \to B, B \to C, A \vdash C$ by $\to-$ on 3,4

6) $A \to B, B \to C \vdash A \to C$ by $\to+$ on 5

↳ important to note: we do not have to begin by using only the given premises, but we must end with them!

- theorem: if $A \to B$ and $B \to C$, then $A \to C$ (hypothetical syllogism)

- note: if $\Sigma, \neg A \vdash B$ and $\Sigma, \neg A \vdash \neg B$, then $\Sigma \vdash A$ is proof by contradiction!

if $\Sigma, A \vdash C$ and $\Sigma, B \vdash C$, then $\Sigma, A \vee B \vdash C$ is proof by cases

if $\Sigma, A \vdash B$ then $\Sigma \vdash A \to B$ is "if A then B"

- **Formal Deduction Proof Strategies**
  - If the conclusion has an implication (ie trying to prove that $\Sigma \vdash A \to B$), then try using $\to +$!
    - ↳ add A to the premises and try to deduce B (ie, just prove $\Sigma, A \vdash B$ first). Then the $\to +$ will follow.
      - ↳ note: this is very similar to assum $[A \ldots B] \vdash A \to B$ as in SE212.
  - If the premises have a disjunction (ie trying to prove something like $\Sigma, A \vee B \vdash C$), then try proof by cases.
    - ↳ in other words, seperately prove $\Sigma, A \vdash C$ (case 1) and $\Sigma, B \vdash C$ (case 2). Then, put these together with one application of $\vee -$ to obtain $\Sigma, A \vee B \vdash C$.
  - If we have to prove $A \vdash B$ and the direct proof does not work, try proving the contrapositive!
    - ↳ ie, try to prove $\neg B \vdash \neg A$.
  - If everything else fails, try proof by contradiction $(\neg -)$!
    - ↳ if we want to prove $\Sigma \vdash B$, start with the modified premises $\Sigma, \neg B$ and try to prove both $\Sigma, \neg B \vdash C$ and $\Sigma, \neg B \vdash \neg C$.
    - ↳ this means that we reached a contradiction, so our initial assumption of $\neg B$ must be false, and therefore B must hold!

- **Finiteness of Premise Set**: if $\Sigma \vdash A$, then there exists a finite $\Sigma^0 \subseteq \Sigma$ such that $\Sigma^0 \vdash A$.
  - ↳ ie, if $\Sigma \vdash A$, there is always a finite subset $\Sigma^0 \subseteq \Sigma$ that will also deduce A

- Theorem - **Transitivity of Deducibility (Tr)**: let $\Sigma, \Sigma' \subseteq \text{Form}(\mathcal{L}^p)$. If $\Sigma \vdash \Sigma'$ and $\Sigma' \vdash A$, then $\Sigma \vdash A$.

↳ note: $\Sigma \vdash \Sigma'$ means that $\forall\ B \in \Sigma',\ \Sigma \vdash B$.

↳ proof:

Σ' must be finite by finiteness of a premise set theorem!

1) $A_1, A_2, \dots, A_n \vdash A$ where $\{A_1, A_2, \dots, A_n\} = \Sigma'$.

2) $A_1, A_2, \dots, A_{n-1} \vdash A_n \to A$ by $\to +$ on 1

3) $A_1, A_2, \dots, A_{n-2} \vdash A_{n-1} \to (A_n \to A)$ by $\to +$ on 2

⋮

$n+1$) $\emptyset \vdash A_1 \to (A_2 \to C \cdots (A_n \to A) \dots)$ by $\to +$ on n

$n+2$) $\Sigma \vdash A_1 \to (A_2 \to C \cdots (A_n \to A) \dots)$ by $+$ on $n+1$

$n+3$) $\Sigma \vdash A_1$ given, as $\forall\ B \in \Sigma',\ \Sigma \vdash B$.

$n+4$) $\Sigma \vdash A_2 \to (\dots (A_n \to A) \dots)$ by $\to -$ on $n+2, n+3$

⋮

$3n+1$) $\Sigma \vdash A_n \to A$ by $\to -$ on $3n, 3n-1$

$3n+2$) $\Sigma \vdash A_n$ given

$3n+3$) $\Sigma \vdash A$ by $\to -$ on $3n+1, 3n+2$.

- Theorem - $\neg\neg A \vdash A$

↳ proof (by contradiction):

1) $\neg\neg A, \neg A \vdash \neg\neg A$ by $\in$

2) $\neg\neg A, \neg A \vdash \neg A$ by $\in$

3) $\neg\neg A \vdash A$ by $\neg -$ on 1, 2.

- Theorem - Reductio ad Absurdum ($\neg +$): if $\Sigma, A \vdash B$ and $\Sigma, A \vdash \neg B$, then $\Sigma \vdash \neg A$.

↳ proof:

1) $\Sigma, A \vdash B$ given premise

2) $\Sigma, \neg\neg A \vdash \Sigma$ by $\in$

3) $\neg\neg A \vdash A$ by previous theorem

4) $\Sigma, \neg\neg A \vdash A$ by $+$ on 3

5) $\Sigma, \neg\neg A \vdash B$ by Tr on 2, 4, 1 ↑ but using the other premise $\Sigma, A \vdash \neg B$!

6) $\Sigma, \neg\neg A \vdash \neg B$ using same subproof as 5

7) $\Sigma \vdash \neg A$ by $\neg-$ on 5, 6

- note: $A \models B$ is equivalent to "$A \to B$ is a tautology"

- Replaceability of Syntactically Equivalent Formulas (Repl.): Let $B \dashv\vdash C$. Then, for any A, let A' be constructed from A by replacing some (not necessarily all) occurances of B by C. Then $A \dashv\vdash A'$.

- Soundness: we should not be able to prove false statements
- Completeness: we should be able to prove every correct statement from first principles (the formal deduction rules).

↳ together, these make a proof system "good"!

- interesting example: find a formal proof for the tautology $A = \neg q \wedge (p \to q) \to \neg p$.

1) $p, q \vdash A$ (proven in subproof A)
2) $p, \neg q \vdash A$ (subproof B)
3) $\neg p, q \vdash A$ (subproof C)
4) $\neg p, \neg q \vdash A$ (subproof D)

} showing that A is true no matter the combination of p and q inputs.

5) $p, q \vee \neg q \vdash A$ by V- on 1, 2
6) $\neg p, q \vee \neg q \vdash A$ by V- on 3, 4
7) $p \vee \neg p, q \vee \neg q \vdash A$ by V- on 5, 6
8) $\emptyset \vdash p \vee \neg p$ by lem
9) $\emptyset \vdash q \vee \neg q$ by lem
10) $\emptyset \vdash A$ by Tr. on 8, 9, 7.

Subproof A: show that $p, q \vdash \neg q \wedge (p \to q) \to \neg p$ ↗ = A

1) $p, q, \neg q \wedge (p \to q) \vdash q$ by $\in$
2) $p, q, \neg q \wedge (p \to q) \vdash \neg q \wedge (p \to q)$ by $\in$
3) $p, q, \neg q \wedge (p \to q) \vdash \neg q$ by $\wedge$- on 2
4) $q, \neg q \wedge (p \to q) \vdash \neg p$ by $\neg +$ on 1,3
5) $p, q, \neg q \wedge (p \to q) \vdash \neg p$ by $+$ on 4
6) $p, q \vdash \neg q \wedge (p \to q) \to \neg p$ by $\to +$ on 5.

- Note: formal deduction cannot be used to prove that an argument is invalid!

Resolution

- Resolution theorem proving a method of formal derivation (formal deduction) that has the following features:
  - The only formulas allowed in resolution theorem proving are disjunctions of literals, such as $(p \vee q \vee \neg r)$.
  - There is essentially one rule of formal deduction, resolution.

- To prove that the argument $A_1, A_2, \ldots, A_n \vDash C$ is valid, we show that the set $\{A_1, A_2, \ldots, A_n, \neg C\}$ is not satisfiable!
  - ↳ we do this by proving that, for some formula B, we can formally derive both B and $\neg B$ (a contradiction).
    - ↳ by "derive" we mean repeated application of resolution!

- In general, we can convert any formula into one or more disjunctive clauses.
  - ↳ to do this, we convert the formula to CNF, and

each term of the conjunction is then made into a clause of its own.

↳ example: convert $p \rightarrow (q \wedge r)$ into clauses:

$$p \rightarrow (q \wedge r) \equiv \neg p \vee (q \wedge r)$$
$$\equiv (\neg p \vee q) \wedge (\neg p \vee r)$$

∴, the two clauses are $\neg p \vee q$ and $\neg p \vee r$.

- $C \vee p,\ D \vee \neg p \vdash_r C \vee D$ (resolution)
  - ↳ where C and D are disjunctive clauses and p is a literal.
  - $C \vee p$ and $D \vee \neg p$ are called parent clauses, and we say that we are resolving the two parent clauses over p.
  - $C \vee D$ is called the resolvent or the child
  - The resolvent of $p \vee \neg p$ is called the empty clause and is denoted by $\{\}$, ie, $p, \neg p \vdash_r \{\}$.
  - A resolution derivation from a set of clauses S is a finite sequence of clauses such that each clause is either in S or results from previous clauses in the sequence by resolution.

↳ the empty clause $\{\}$ is a notation signifying that the contradiction $p \wedge \neg p$ was reached.

↳ by definition, the empty clause is not satisfiable.

- example: find the resolvent of $p \vee q \vee r$ and $\neg s \vee q$.

↳ the parent clauses can be resolved over q, as q is negative in the first clause and positive in the second.

- the resolvent is the disjunction of $p \vee r$ with $\neg s$, which yields $p \vee r \vee \neg s$.

· Proving argument validity with resolution

- to prove that the argument $A_1, A_2, \ldots, A_n \models C$ is valid, we show that from the set $\{A_1, A_2, \ldots, A_n, \neg C\}$ we can derive, by $\vdash_r$, the empty clause $\{\}$ (a contradiction), as follows:
  - pre-process the input by transforming each of the formulas in $\{A_1, A_2, \ldots, A_n, \neg C\}$ into CNF.
  - make each disjunctive clause a distinct clause. These clauses are the input of the resolution procedure.
  - if the resolution procedure outputs the empty clause $\{\}$, this implies that $\{A_1, A_2, \ldots, A_n\}$ is not satisfiable, and therefore, the argument was valid.

· Resolution Procedure

↳ input: set of disjunctive clauses $S = \{D_1, D_2, \ldots, D_m\}$.

REPEAT trying to get the empty clause, $\{\}$:

- choose two clause, one with p and one with $\neg p$
- resolve and call the resolvent D
- If $D = \{\}$, then output "empty clause"
- Else add D to S.

example: prove that $p, p \rightarrow q \vdash_r q$.

1) p premise
2) $\neg p \vee q$ premise
3) $\neg q$ negation of conclusion
4) q resolvent of 1,2 over p
5) $\{\}$ resolvent of 3,4 over q

example: prove that $p \to q,\ q \to r \vdash_r p \to r$

1) $\neg p \vee q$ premise
2) $\neg q \vee r$ premise
3) $p \wedge \neg r$ negation of conclusion
4) p by $\wedge$- on 3
5) $\neg r$ by $\wedge$- on 3
6) q resolution on 1,4 over p
7) r resolution on 2,6 over q
8) $\{\}$ resolution on 5,7 over r.

Set-of-Support Strategy

- Partition all clauses into two sets, the set of support and the auxillary set.
- The formulas in the auxillary set are not contradictory
  - ↳ usually, contradiction only arises when adding the negation of the conclusion.
- ∴, the set of premises is often used as the auxillary set and the negation of the conclusion is often used as the initial set of support.
- when using the set of support strategy, each resolution takes at least one clause from the set of support.
- the resolvent is then added to the set of support.
- Note: resolution with the set of support is complete!

- example: prove p_4 from $p_1 \to p_2,\ \neg p_2,\ \neg p_1 \to p_3 \vee p_4,\ p_3 \to p_5,\ p_6 \to \neg p_5$, and p_6 by using the set-of-support strategy.

1) $\neg p_1 \vee p_2$ premise

2) $\neg p_2$ premise

3) $p_1 \vee p_3 \vee p_4$ premise

4) $\neg p_3 \vee p_5$ premise

5) $\neg p_6 \vee \neg p_5$ premise

6) p_6 premise

7) $\neg p_4$ negation of the conclusion $\Sigma = \{7\}$

8) $p_1 \vee p_3$ resolution on 3, 7 on p_4 $\Sigma = \{7, 8\}$

9) $p_2 \vee p_3$ resolvent of 1, 8 on p_1 $\Sigma = \{7, 8, 9\}$

10) p_3 resolvent of 2, 9 on p_2 $\Sigma = \{7, 8, 9, 10\}$

11) p_5 resolvent of 4, 10 on p_3 $\Sigma = \{7, 8, 9, 10, 11\}$

12) $\neg p_6$ resolvent of 5, 11 on p_5 $\Sigma = \{7, 8, 9, 10, 11, 12\}$

13) $\{\}$ resolvent of 6, 12 on p_6.

- Davis - Putnam Procedure (DPP) - preprocessing the input
  - any clause corresponds to the literals within it.
    - ↳ ie, the clause $p \vee \neg q \vee r$ corresponds to the set $\{p, \neg q, r\}$.
  - ∴, we frequently treat clauses as sets, which allows us to use terminology such as the union of two clauses.

- if clauses are represented as sets, we can write the resolvent, on p, of two clauses $C \cup \{p\}$ and $D \cup \{\neg p\}$, when neither C or D is empty, as: (\ = set difference!)

$$[(C \cup \{p\}) \cup (D \cup \{\neg p\})] \setminus \{p, \neg p\}$$

↳ in words, the resolvent is the union of all literals in the parent clauses except that the two literals involving p are omitted.

- DPP Procedure:
  - input: nonempty set of clauses in the propositional variables $p_1, \ldots, p_n$
  - repeat the following steps until there are no variables left:
    1) remove all clauses that have both a literal q and $\neg q$
    2) choose a variable p in one of the clauses
    3) add all possible resolvents using resolution on p to the set of clauses
    4) discard all clauses with p or $\neg p$ in them!
  - ↳ note: we refer to this sequence of steps as "eliminating the variable p".
  - ↳ if in some step we resolve $\{p\}$ and $\{\neg p\}$, then we get the empty clause, and it will be the only clause at the end of the procedure!
  - ↳ if we never get a pair $\{p\}$ and $\{\neg p\}$ to resolve, then all the clauses will be discarded and the output will be no clauses.
    - ↳ ie, DPP failed!

- example: apply the Davis-Putnam Procedure to the set of clauses $\{\neg p, q\}, \{\neg q, \neg r, s\}, \{p\}, \{r\}, \{\neg s\}$.

- eliminating p gives $\{\neg q, \neg r, s\}, \{r\}, \{\neg s\}, \{q\}$
  - ↳ we did resolution on $\{p\}, \{\neg p, q\}$ to get just $\{q\}$!
- eliminating q gives $\{r\}, \{\neg s\}, \{\neg r, s\}$
  - ↳ we did resolution on $\{\neg q, \neg r, s\}, \{q\}$ to get just $\{\neg r, s\}$.
- eliminating r gives $\{\neg s\}, \{s\}$
  - ↳ we did resolution on $\{r\}, \{\neg r, s\}$ to get just $\{s\}$
- eliminating s gives $\{\}$

↳ we did resolution on {¬s}, {s} to get just {}.

∴, the output is the empty clause, and the argument is therefore valid!

→ note: DPP is sound and complete!

First-Order Logic

↳ extension of propositional logic that allows us to be much more expressive.

- The Domain (aka Universe of Discourse): the collection of all persons, ideas, symbols, data structures, and so on, that affect the logical argument.
  - ↳ the truth of a statement may depend on the domain selected!
    - ↳ ie, the statement that "there is a smallest number" is true in the domain of naturals $\mathbb{N}$, but false in the domain of integers $\mathbb{Z}$.
  - the elements of the domain are called individuals.
    - ↳ a domain is always non-empty (has at least one individual)

- Generally, relations make statements about individuals.
- In a statement, the list of individuals is called the argument list.

- example: in the statement "Mary and Paul are siblings", the argument list is Mary and Paul (in that order), whereas the relation is described by the phrase "are siblings".

- example: to express that Joan is the mother of Mary,

we could choose an identifier Mother to express the relation "is mother of", and write Mother(Joan, Mary).
↳ can even shorten to M(j, m).

Arity: the number of elements in the argument list of a relation
↳ ie, Mother(Joan, Mary) has arity 2.
↳ "n-ary arity" means n inputs in the argument list to the relation:

- Atomic Formulas can be combined using logical connectives
  ↳ atomic formulas take true/false values:
    ↳ eg: human(x) = { 1, if x is human; 0, otherwise }.
  ↳ simplest formulas are atomic formulas!
    ↳ can be connected like: human(Socrates) → mortal(Socrates).

- Term: either an individual or a variable
  ↳ generally, a term is anything that can be used in place of an individual.

- Universal Quantifier
  - let A(u) represent a formula, and let u be a variable
  - If we want to indicate that A(u) is true for all possible values of u in the domain, we write $\forall x\, A(x)$
  - Here, $\forall x$ is the universal quantifier, and A(x) is the scope of the quantifier
  - The variable x is said to be bound by the quantifier
  - The symbol $\forall$ is pronounced "for all"

- Existential Quantifier
  - let $A(u)$ represent a formula, and let u be a variable
  - If we want to indicate that $A(u)$ is true for at least one value u in the domain (possibly, but not necessarily, more than one), we write $\exists x\, A(x)$.
- Here, $\forall x$ is the existential quantifier, and $A(x)$ is the scope of the quantifier
- The variable x is said to be bound by the quantifier
- The symbol $\exists$ is pronounced "there exists"

- note: any variable that is not bound is said to be free
- note: quantifiers have higher precedence than all binary connectives!

- Restricting the Universal Quantifier:
  - Consider the statement "all dogs are mammals."
  - Since the quantifier should be restricted to dogs, we rephrase as "if u is a dog, then u is a mammal."
  - ∴, this leads to the formula: $\forall x\,(dog(x) \rightarrow mammal(x))$!

- Restricting the Existential Quantifier
  - Consider the statement "some dogs are brown."
  - This means that there are some animals that are dogs and that are brown.
  - The statement that u is a dog and u is brown can be translated as $dog(u) \wedge brown(u)$.

  ∴, this leads to the formula: $\exists x\,(dog(x) \wedge brown(x))$

- Translating Statements with "only"

- Consider statements such as "only dogs bark"
- This must be reworded as "it only barks if it is a dog", it's equivalent, "if it is not a dog, it does not bark", or it's contrapositive, "if it barks, then it is a dog".
- It's translation is therefore $\forall x\ (\text{barks}(x) \to \text{dog}(x))$.

Nested Quantifiers

↳ eg: "there is somebody who knows everyone": $\exists x\ (\forall y \cdot \text{Knows}(x, y))$

- $\forall$ and $\exists$ do not commute!!
  - ↳ ie, $\exists x \forall y\ A(x,y) \not\vDash \forall y \exists x\ A(x,y)$
  - ↳ but, $\forall$ and $\forall$ (and $\exists$ and $\exists$) do commute!
    - ↳ ie, $\forall x \forall y\ A(x,y) \vDash \forall y, x\ A(x,y)$.

Translating Statements with "nobody" / "nothing"

- Consider the statement "nobody is perfect"
- "Nobody" is the absence of an individual with a certain property.
- If $P(u)$ means that "u is perfect", then:
  - ↳ $\neg \exists x\ P(u)$ expresses "it is not the case that there is somebody is perfect"

Negating formulas with quantifiers

- $\neg(\forall x\ A(x)) \vDash \exists x\ \neg A(x)$
- $\neg(\exists x\ A(x)) \vDash \forall x\ \neg A(x)$

- note: relation symbols (F, G, H, ...) gives only boolean values as outputs, but function symbols (f, g, h, ...) can give other terms/individuals.

- There is a special binary relation symbol called the equality symbol, written as $\approx$.
  - ↳ we stipulate that the equality symbol is not in F, G, H, ...

- A term or formula is said to be closed if it contains no free variables.
  - ↳ aka, a sentence!
    - ↳ the set of sentences of $\mathcal{L}$ is denoted by $Sent(\mathcal{L})$.

- In First-Order Logic, we have valuations instead of truth valuations (as we did in propositional logic).
  - A valuation consists of an interpretation plus an assignment.
    - ↳ An interpretation consists of:
      - a non-empty set of individuals (objects) - aka the domain.
      - a specification, for each individual, relation, and the function symbol, of the actual individuals, relations, and functions they will denote.
    - ↳ An assignment assigns to each free variable a value in the domain.

↓

Formal Definition of a Valuation:

- A valuation for the first-order language $\mathcal{L}$ consists of:
  - A domain, often called D, which is a non-empty set,
  - A function, denoted by v, with the properties:
    1) for each individual constant symbol a, and free variable symbol u, we have that $a^v, u^v \in D$.
    2) for each n-ary relation symbol F, we have that F^v is an n-ary relation on D, that is, $F^v \subseteq D^n$.

3) for each m-ary function symbol f, we have that $f^{\mathcal{V}}$ is a total m-ary function of D into D, that is, $f^{\mathcal{V}}: D^m \to D$.

- Note: a function is "total" if it is never undefined.

Example: define a valuation for $\forall x\,(F(x) \vee H(x) \to G(x))$.

note: since there are no free variables, the valuation is just the interpretation (no assignment!)

- consider the valuation $\mathcal{V}$:
  - the domain D is the set of all ships
  - $F^{\mathcal{V}}$ is the unary relation over D defined by $F^{\mathcal{V}} = \{u \mid u \text{ is on fire}\}$
  - $H^{\mathcal{V}}$ is the unary relation over D defined by $H^{\mathcal{V}} = \{u \mid u \text{ has a hole}\}$
  - $G^{\mathcal{V}}$ is the unary relation over D defined by $G^{\mathcal{V}} = \{u \mid u \text{ sinks}\}$

- For any valuation $\mathcal{V}$, free variable u, and individual $d \in D$, we write $\mathcal{V}(u/d)$ to denote a valuation which is exactly the same as $\mathcal{V}$ except that $u^{\mathcal{V}(u/d)} = d$.

Value of a Quantified Formula: let $\forall x\,A(x)$ and $\exists x\,A(x)$, and let u be a free variable not in $A(x)$. The values of $\forall x\,A(x)$ and $\exists x\,A(x)$ under a valuation $\mathcal{V}$ with domain D are given by:

$$\forall x\,A(x)^{\mathcal{V}} = \begin{cases} 1 & \text{if } A(u)^{\mathcal{V}(u/d)} = 1 \text{ for every } d \in D \\ 0 & \text{otherwise} \end{cases}$$

$$\exists x\,A(x)^{\mathcal{V}} = \begin{cases} 1 & \text{if } A(u)^{\mathcal{V}(u/d)} = 1 \text{ for some } d \in D \\ 0 & \text{otherwise} \end{cases}$$

- (Informally) Proving and Disproving Argument Validity in First-Order Logic

- prove that $\forall x \neg A(x) \models \neg \exists x A(x)$.

we will try proof by contradiction. ie, we will suppose that there is some valuation v over D such that:

1) $(\forall x \neg A(x))^v = 1$

2) $(\neg \exists x A(x))^v = 0$

by negating equation 2 (informally), we get:

3) $(\exists x A(x))^v = 1$

let's assume that the $x \in D$ that this statement is true for is $d \in D$. Then,

4) $(A(d))^v = 1$.

But from inspecting (1), we see that the statement is quantified over ALL $x \in D$. So, we can conclude that the statement also holds for $d \in D$. Therefore:

5) $(\neg A(d))^v = 0$.

We see that statements (4) and (5) are directly contradictory, which means that our assumption was false. Therefore, the initial argument must be valid!

- prove that $\emptyset \models \forall x F(x) \vee \forall x G(x) \rightarrow \forall x (F(x) \vee G(x))$

let's try proof by contradiction. ie, we will assume that there exists some valuation v over domain D such that $(\forall x F(x) \vee \forall x G(x) \rightarrow \forall x (F(x) \vee G(x)))^v = 0$.

Since this implication is false, we see that:

1) $(\forall x F(x) \vee \forall x G(x))^v = 1$

2) $(\forall x (F(x) \vee G(x)))^v = 0$

negating equation (2):

3) $(\exists x (\neg F(x) \wedge \neg G(x)))^v = 1$

This implies that there exists an individual $d \in D$ such that

4) $(\neg F(d) \wedge \neg G(d))^v = 1$, further implying:

5) $F(d) = G(d) = 0$.

since we know $F(x)$ and $G(x)$ are 0 for d, we see that they both are not true for all $x \in D$.

6) $(\forall x\, F(x))^{v} = 0$

7) $(\forall x\, G(x))^{v} = 0$

8) $(\forall x\, F(x) \vee \forall x\, G(x)) = 0$

This directly contradicts our premise, meaning our assumption was false, and therefore the argument must be valid!

Lemma - Logical Equivalences

let $A \dashv\vdash A'$. $B \dashv\vdash B'$, and $C(u) \dashv\vdash C'(u)$. Then,

1) $\neg A \dashv\vdash \neg A'$,
2) $A \wedge B \dashv\vdash A' \wedge B'$,
3) $A \vee B \dashv\vdash A' \vee B'$,
4) $A \to B \dashv\vdash A' \to B'$,
5) $A \leftrightarrow B \dashv\vdash A' \leftrightarrow B'$,
6) $\forall x\, C(x) \dashv\vdash \forall x\, C'(u)$,
7) $\exists x\, C(x) \dashv\vdash \exists x\, C'(x)$.

↳ these basically mean that if you have equivalent formulas, you can substitute them.

Additional Rules of Formal Deduction (for First-Order Logic)

12) if $\Sigma \vdash \forall x\, A(x)$, then $\Sigma \vdash A(t)$ — ($\forall -$)
(t can be any term!)

13) if $\Sigma \vdash A(u)$ where u is not in Σ, then $\Sigma \vdash \forall x\, A(x)$ — ($\forall +$)
(essential!!)

14) if $\Sigma, A(u) \vdash B$ and u is not in Σ or B, then $\Sigma, \exists x\, A(x) \vdash B$ — ($\exists -$)

15) if $\Sigma \vdash A(t)$, then $\Sigma \vdash \exists x\, A(x)$ — ($\exists +$)

where $A(x)$ results from $A(t)$ by replacing some (not necessarily all) occurances of t by x.

16) if $\Sigma \vdash A(t_1)$ and $\Sigma \vdash t_1 \approx t_2$, then $\Sigma \vdash A(t_2)$ $(\approx -)$

where $A(t_1)$ results from $A(t_2)$ by replacing some (not necessarily all) occurances of t_1 by t_2.

17) $\varnothing \vdash u \approx u$ $(\approx +)$

$\forall -$

example: from the premises "all humans are mortal" and "Socrates is a human", prove that "Socrates is mortal".

let $H(x)$ mean x is human, $M(x)$ mean x is mortal, and s represent Socrates. Then,

$\Sigma = \{\forall x (H(x) \to M(x)), H(s)\}$, and we want to show that $\Sigma \vdash M(s)$.

1) $\Sigma \vdash \forall x (H(x) \to M(x)$ by $\in$
2) $\Sigma \vdash H(s)$ by $\in$
3) $\Sigma \vdash H(s) \to M(s)$ by $\forall -$ on 1
4) $\Sigma \vdash M(s)$ by $\to -$ on 2, 3.

$\forall +$

example: consider a problem whose domain is the set of all students in this class. Assume that all students in this class are computer science students, and that computer science students like coding. Prove that all students in this class like coding.

let $CS(x)$ mean x is a computer science student and $LC(x)$ mean x likes coding. Then,

$\Sigma = \{\forall x\, CS(x), \forall x (CS(x) \to LC(x))$, and we want to show that $\Sigma \vdash \forall x\, LC(x)$.

1) $\Sigma \vdash \forall x\, CS(x)$ by $\in$
2) $\Sigma \vdash \forall x (CS(x) \to LC(x))$ by $\in$
3) $\Sigma \vdash CS(u)$ by $\forall -$ on 1

4) $\Sigma \vdash CS(u) \to LC(u)$ by $\forall$- on 2

5) $\Sigma \vdash LC(u)$ by $\to$- on 3,4

6) $\Sigma \vdash \forall x\, LC(x)$ by $\forall$+ on 5

↗ $\exists$+

example: from the premises "everybody who has won a million dollars is rich" and "mary has won a million dollars", prove that "there is somebody who is rich".

let the domain D be the set of people, $W(u)$ means that u has won a million dollars, $R(u)$ means that u is rich, and m represent Mary. Then, $\Sigma = \{\forall x\,(W(x) \to R(x)),\ W(m)\}$. We want to show that $\Sigma \vdash \exists x\, R(x)$.

1) $\Sigma \vdash \forall x\,(W(x) \to R(x))$ by $\in$

2) $\Sigma \vdash W(m)$ by $\in$

3) $\Sigma \vdash W(m) \to R(m)$ by $\forall$- on 1

4) $\Sigma \vdash R(m)$ by $\to$- on 2, 3

5) $\Sigma \vdash \exists x\, R(x)$ by $\exists$+ on 4.

↗ $\exists$-

example: from the premises "someone has won a million dollars" and "everybody who has won a million dollars is rich", prove that "there is someone who is rich".

let the domain D be the set of people, $W(u)$ means that u has won a million dollars, and $R(u)$ means that u is rich. Then, $\Sigma = \{\exists x\, W(x),\ \forall x\,(W(x) \to R(x))\}$. We want to show that $\Sigma \vdash \exists x\, R(x)$.

note that we have an existential quantifier in the premises.

Let's give it a specific term, so we can reference the specific existential term elsewhere.

↳ let $\Sigma'' = \{\forall x\,(W(x) \to R(x))\}$. That is, $\Sigma = \{\exists x\, W(x), \Sigma''\}$.

1) $W(u), \Sigma'' \vdash \forall x\,(W(x) \to R(x))$ by $\in$

2) $W(u), \Sigma'' \vdash W(u) \to R(u)$ by $\forall$- on 1

3) $W(u), \Sigma'' \vdash W(u)$ by $\in$

4) $W(u), \Sigma'' \vdash R(u)$ by $\to$- on 2, 3

5) $W(u), \Sigma'' \vdash \exists x\, R(x)$ by $\exists$+ on 4

6) $\exists x\, W(x), \Sigma'' \vdash \exists x\, R(x)$ by $\exists$- on 5

$\exists x\, W(x), \Sigma''$: Σ!

↳ as u not in Σ or $\exists x R(x)$

↳ note: this is generally the strategy used to deal with existential quantifiers in the premises!

Another Useful Rule: if $A \vdash B$, then $\neg B \vdash \neg A$ (flip-flop rule)

- Useful Proof: $\neg\forall x\, A(x) \dashv\vdash \exists x \neg A(x)$ (negation!)

note: the other negation is proved in the exercise note!

1) $\neg A(u) \vdash \neg A(u)$ by ref

2) $\neg A(u) \vdash \exists x \neg A(x)$ by $\exists$+ on 1

3) $\neg\exists x \neg A(x) \vdash A(u)$ by flip-flop on 2

4) $\neg\exists x \neg A(x) \vdash \forall x\, A(x)$ by $\forall$+ on 3

5) $\neg\forall x\, A(x) \vdash \exists x \neg A(x)$ by flip-flop on 4

and the other way:

1) $\forall x\, A(x) \vdash \forall x\, A(x)$ by ref

2) $\forall x\, A(x) \vdash A(u)$ by $\forall$- on 1

3) $\neg A(u) \vdash \neg\forall x\, A(x)$ by flip-flop on 2

4) $\exists x \neg A(x) \vdash \neg\forall x\, A(x)$ by $\exists$- on 3 (u does not occur elsewhere)

Proof Strategies for Formal Deduction in First-Order Logic

1a) "ignore" the quantifiers and imagine how the proof would look in propositional logic.

1b) after getting an idea for the general "shape" of the proof, we:

- remove quantifiers (eg with $\exists-$ and $\forall-$)
- carry the proof with formulas in propositional logic
- re-introduce quantifiers as needed (eg with $\exists+$ and $\forall+$)

2) if one of the premises in Σ is existentially quantified, the way to remove the $\exists$ is to:

- replace the premise $\exists x\, A(x)$ in Σ by $A(u)$, resulting in Σ'
- carry on the entire proof with the modified set of premises Σ'
- at the end, use $\exists$ to re-introduce the $\exists$ back. make sure u does not appear in any of the other premises or the conclusion.

Resolution for First-Order Logic

- **Prenex Normal Form**: a formula is in prenex normal form if it is of the form:

$$Q_1 x_1\ Q_2 x_2 \ldots Q_n x_n\ B$$

where $n \geq 1$, Q_i is $\forall$ or $\exists$, for $1 \leq i \leq n$, and the expression B has no quantifiers.

↳ the string $Q_1 x_1\ Q_2 x_2 \ldots Q_n x_n$ is called **the prefix** and B is called **the matrix**.

↳ a formula with no quantifiers ($n = 0$) is considered a trivial case of a prenex normal form.

↳ basically, prenex normal form is when all of the quantifiers are at the beginning.

- eg: $\forall x\, P(x) \vee \forall x\, Q(x)$ is not in prenex normal form.
  $\forall x \forall y \neg(P(x) \to Q(y))$ is in prenex normal form
  $\forall x \exists y\, R(x,y)$ is in prenex normal form.
  $\neg \forall x\, R(x,y)$ is not in prenex normal form.
  $R(u,v)$ is in prenex normal form trivially (bc no quantifiers)

- Algorithm to convert any formula in Form(L) into prenex normal form:
  1) eliminate all occurences of $\to$ and $\leftrightarrow$.
  2) "move all negations inwards" such that, in the end, negations only appear as part of literals. (↳ an atom, or it's negation.)
  3) standardize the variables apart, when needed
     ↳ distinct variables should be renamed to have unique names
  4) the prenex normal form can be obtained by moving all quantifiers to the front of the formula.

↳
- for step 1 (eliminating $\to$ and $\leftrightarrow$), use:
  ↳ $A \to B \models \neg A \vee B$
  $A \leftrightarrow B \models (\neg A \vee B) \wedge (\neg B \vee A)$
  $A \leftrightarrow B \models (A \wedge B) \vee (\neg A \wedge \neg B)$.
- for step 2 (moving negations inwards), use:
  ↳ de morgan's laws
  $\neg\neg A \models A$
  $\neg \exists x\, A(x) \models \forall x \neg A(x)$
  $\neg \forall x\, A(x) \models \exists x \neg A(x)$.
- for step 3 (standardizing the variables apart):

↳ eg: $\forall x\,(A(x) \vee B(x)) \vee \exists x\, C(x)$. note that the x's in A and B are different to the x in C, and if we remove the quantifiers, they will be indistinguishable.

ie: standardize all variables apart in:

$\forall x\,(P(x) \to Q(x)) \wedge \exists x\, Q(x) \wedge \exists z\, P(z) \wedge \exists z\,(Q(z) \to R(z))$

↳ use y for x in $\forall x\,(P(x) \to Q(x))$, u for x in $\exists x\, Q(x)$, w for z in $\exists z\, P(z)$, and a for z in $\exists z\,(Q(z) \to R(z))$.

$\forall y\,(P(y) \to Q(y)) \wedge \exists u\, Q(u) \wedge \exists w\, P(w) \wedge \exists a\,(Q(a) \to R(a))$.

- for step 4 (moving quantifiers):

↳ use
$A \wedge \exists x B(x) \vDash \exists x\,(A \wedge B(x))$, x not in A.
$A \wedge \forall x\, B(x) \vDash \forall x\,(A \wedge B(x))$, x not in A.
$A \vee \exists x B(x) \vDash \exists x\,(A \vee B(x))$, x not in A.
$A \vee \forall x\, B(x) \vDash \forall x\,(A \vee B(x))$, x not in A.

important two!!
$\forall x A(x) \wedge \forall x B(x) \vDash \forall x\,(A(x) \wedge B(x))$
$\exists x A(x) \vee \exists x B(x) \vDash \exists x\,(A(x) \vee B(x))$.

$\forall x \forall y A(x,y) \vDash \forall y \forall x A(x,y)$
$\exists x \exists y A(x,y) \vDash \exists y \exists x A(x,y)$

different from above two bc # quantifiers doesn't go down!
$Q_1 x A(x) \wedge Q_2 y B(y) \vDash Q_1 x\, Q_2 y\,(A(x) \wedge B(y))$.
$Q_1 x A(x) \vee Q_2 y B(y) \vDash Q_1 x\, Q_2 y\,(A(x) \vee B(y))$.

↱ x not in $B(y)$, y not in $A(x)$.

- example: find the prenex normal form of:

$\forall x\,(\exists y R(x,y) \wedge \forall y \neg S(x,y) \to \neg \exists y \neg Q(x,y))$.

1) eliminate $\to$:

$\forall x(\neg(\exists y R(x,y) \wedge \forall y \neg S(x,y)) \vee \neg \exists y \neg Q(x,y))$.

2) move negations inwards:

$\forall x\,(\forall y \neg R(x,y) \vee \exists y S(x,y)) \vee \forall y\, Q(x,y))$

3) standardize all variables apart:

$\forall x\,(\forall y_1 \neg R(x,y_1) \vee \exists y_2 S(x,y_2)) \vee \forall y_3 Q(x,y_3))$.

4) move quantifiers to the front:

$\forall x \forall y_1 \exists y_2 \forall y_3 (\neg R(x, y_1) \vee S(x, y_2) \vee Q(x, y_3))$.

- example: find the prenex normal form of:

$\neg [\forall x \exists y F(u, x, y) \rightarrow \exists x (\neg \forall y G(y, w) \rightarrow H(x))]$.

1) eliminate the $\rightarrow$s:

$\neg [\neg (\forall x \exists y F(u, x, y)) \vee \exists x (\neg \forall y G(y, w) \rightarrow H(x))]$.

$\neg [\neg (\forall x \exists y F(u, x, y)) \vee \exists x (\neg (\neg \forall y G(y, w)) \vee H(x))]$

2) moving negations inwards:

$\neg [\exists x \forall y \neg F(u, x, y) \vee \exists x (\neg (\exists y \neg G(y, w)) \vee H(x))]$

$\neg [\exists x \forall y \neg F(u, x, y) \vee \exists x (\forall y G(y, w) \vee H(x))]$

$\forall x \exists y F(u, x, y) \wedge \forall x \neg (\forall y G(y, w) \vee H(x))]$

$\forall x \exists y F(u, x, y) \wedge \forall x \exists y \neg G(y, w) \wedge \neg H(x)]$

3) standardizing all variables apart:

$\forall x \exists y F(u, x, y) \wedge \forall x_1 \exists y_1 \neg G(y_1, w) \wedge \neg H(x_1)]$

4) moving quantifiers to the front:

$\forall x \exists y \forall x_1 \exists y_1 [F(u, x, y) \wedge \neg G(y_1, w) \wedge \neg H(x_1)]$.

- $\exists$-free prenex normal form

↳ if A is an expression, the individual generated by $\exists y A$ is a function of $x_1, \ldots, x_n$, which can be expressed by using some $f(x_1, x_2, \ldots, x_n)$.

↳ This function f is called a Skolem function.

- Skolem Functions: allow us to remove all existential quantifiers

the skolemized version of $\forall x_1 \forall x_2 \ldots \forall x_n \exists y A$ is

$\forall x \forall x_2 \ldots \forall x_n f(x_1, x_2, \ldots, x_n)$.

where $n \geq 0$, and A' is the expression obtained from A by substituting

each occurence of y by $f(x_1, x_2, \dots, x_n)$.

- example: let the domain be $\mathbb{Z}$, and consider $\forall x \exists y (x + y = 0)$.

each x, say $x = d \in D$, generates a corresponding $y = -d$ to satisfy the formula. $\therefore$, we define $y = f(x) = -x$.

the Skolemized version of the formula is $\forall x (x + f(x) = 0)$, where $f(x) = -x$.

↳ in this example, finding $f(x)$ was easy. However, usually, it will be very hard to find, but we know it is guaranteed to exist. Therefore, we just use $f(x, \dots, x_n)$ and don't define it's function

↳ basically we just say $\forall x \exists y \, P(x, y) \rightarrow \forall x \, P(x, f(x))$.

- note: the sentence obtained by using Skolem functions is not, in general, logically equivalent to the original sentence!

- if an existential quantifier is not preceeded by any universal quantifiers, we just use a constant as it's Skolem function.

- Algorithm to convert any formula in Form($\mathcal{L}$) into $\exists$-free prenex normal form:

1) Transform the input $A_0 \in$ Form($\mathcal{L}$) into A_i in prenex normal form. Set $i = 0$.

2) Repeat until all existential quantifiers are removed:

- Assume A_i is of the form $A_i = \forall x_1 \forall x_2 \dots \forall x_n \exists y \, A$, where A is an expression possibly containing quantifiers.
- if $n = 0$, then A_i is of the form $\exists y \, A$. Then $A_{i+1} = A'$, where A' is obtained from A by replacing all occurences of y by the individual symbol c, where c doesn't occur in A_i.

- if $n > 0$, then $A_{i+1} = \forall x_1 \forall x_2 \dots \forall x_n A'$, where A' is the expression obtained from A by replacing all occurences of y by $f(x_1, x_2, \dots, x_n)$, where f is a new function symbol.

- example: transform the following sentence to $\exists$-free prenex normal form: $\exists x \forall y \forall z \exists s \, P(x, y, z, s)$.

since the leftmost symbol is $\exists x$ (which is preceeded by no $\forall$s), x can be replaced by a Skolem function with 0 arguments (a constant). Use a for this. Then,

$A_2 = \forall y \forall z \exists s \, P(a, y, z, s)$.

there are now two $\forall$s preceeding the leftmost $\exists$. $\therefore$, the Skolem function corresponding to s must have two arguments, y and z. We will use $g(y, z)$ for this Skolem function.

Then, $A_3 = P(a, y, z, g(y, z))$.

- Notation: after all $\exists$s have been removed, we can "drop" the $\forall$s. "drop" meaning we don't write them everytime, but just treat all variables implicitly as universally quantified.

- Theorem: given a sentence F in $\exists$-free prenex normal form, we can construct a finite set C_F of disjunctive clauses such that F is satisfiable if and only if the set C_F of clauses is satisfiable.

↳ example: construct the set of clauses C_F for:

$$F = \forall x \forall y \forall z \, (R(x, y) \to (R(x, z) \wedge R(z, y)))$$

first, we put the matrix of F into CNF:

$$R(x, y) \to (R(x, z) \wedge R(z, y)) \models \neg R(x, y) \vee (R(x, z) \wedge R(z, y))$$
$$\models (\neg R(x, y) \vee R(x, z)) \wedge (\neg R(x, y) \vee R(z, y))$$

$\therefore$, $C_F = \{\neg R(x,y) \vee R(x,z),\ \neg R(x,y) \vee R(z,y)\}$.

- To pre-process the formulas in an argument for resolution, we have 3 steps:
  1) obtain the prenex normal form of the formulas.
  2) obtain the $\exists$-free prenex normal form of the formulas.
  3) change the matrix into CNF and create the set C_F.

- An argument in First-Order Logic is valid if and only if the set of clauses consisting of the union of:
  - $U_{F \in \Sigma} C_F$: the set of clauses obtained from each premise F in Σ, and
  - $C_{\neg A}$: the set of clauses generated by the negation of the conclusion A

  is not satisfiable.

- example: from argument to clauses!

let the set of premises be $\Sigma = \{\forall x\, R(x,x),\ \forall x \forall y\,(R(x,y) \to R(y,x)),\ \forall x \forall y \forall z\,((R(x,y) \wedge R(y,z)) \to R(x,z))\}$.

and the conclusion be $A = \forall x \forall y\,(\neg R(x,y) \to \forall s\,(R(x,s) \to \neg R(y,s)))$.

find the set of clauses $C_{\Sigma, \neg A}$ that is not satisfiable iff the argument $\Sigma \vDash A$ is valid.

- the negation of the conclusion is:

$$\neg A = \neg \forall x \forall y\,(\neg R(x,y) \to \forall s\,(R(x,s) \to \neg R(y,s)))$$
$$\vDash \neg \forall x \forall y\,(R(x,y) \vee \forall s\,(\neg R(x,s) \vee \neg R(y,s)))$$
$$\vDash \exists x \exists y\,(\neg R(x,y) \wedge \exists s\,(R(x,s) \wedge R(y,s))).$$

- putting $\neg A$ in prenex normal form:

$\exists x \exists y \exists s (\neg R(x,y) \wedge R(x,s) \wedge R(y,s))$.

- now into $\exists$-free prenex normal form:

$\neg R(a,b) \wedge R(a,c) \wedge R(b,c)$.

- therefore, the set of clauses $C_{\Sigma,\neg A}$ consists of:

$\{R(x,x), \neg R(x,y) \vee R(y,x), \neg R(x,y) \vee \neg R(y,z) \vee R(x,z), \neg R(a,b), R(a,c), R(b,c)\}$.

from: $\forall x R(x,x)$; $\forall x \forall y (R(x,y) \rightarrow R(y,x))$; $\forall x \forall y \forall z ((R(x,y), R(y,z)) \rightarrow R(x,z))$; negation of conclusion

↳ the set of clauses $C_{\Sigma,\neg A}$ is not satisfiable iff the argument $\Sigma \models A$ is valid. ∴, to show the argument is valid, we must show that the set $C_{\Sigma,\neg A}$ is not satisfiable!

- Unification and Resolution
  - Instantiation is an assignment to a variable x_i of a quasi-term t_i'. We write $x_i := t_i'$.
    ↳ quasi-term: either an individual symbol, a variable symbol, or a function symbol applied to individual symbols or variable symbols.
  - Two formulas are said to unify if there are instantiations that make the formulas identical.
    ↳ unifying is called unification
    ↳ the instantiation that unifies the formulas is called a unifier.
  - example: assume that $Q(a,y,z)$ and $Q(y,b,c)$ are expressions appearing on two different lines on a resolution proof. Show that the two expressions unify, and give a unifier.

    since the y in $Q(a,y,z)$ is a different y than the y in $Q(y,b,c)$, let's rename them for clarity: $Q(a,y,z)$ and $Q(y_1,b,c)$.
    - an instance of $Q(a,y,z)$ is $Q(a,b,c)$ (given by $y:=b$, $z:=c$).
    - an instance of $Q(y_1,b,c)$ is $Q(a,b,c)$ (given by $y_1 := a$).

since these two instances are identical, $Q(a, y, z)$ and $Q(y_1, b, c)$ unify, with unifier $y_1 := a$, $y := b$, $z := c$.

- the idea is now to create complementary literals by means of unification, and determine the resolvent.
  ↳ because resolution can only be applied to expressions that contain complementary literals.

- example: find the resolvent of the following pre-processed clauses: $G(a, x, y) \vee H(y, x) \vee D(z)$ and $\neg G(x, c, y) \vee H(f(x), b) \vee E(a)$.

by notation, we assume x, y, and z are universally quantified variables, and a, b, and c are constant individual symbols.

note that only the G literal can resolve (as it is the only complimentary literal!). ∴, we try to unify it:

1st clause: $x := c$ gives $G(a, c, y) \vee H(y, c) \vee D(z)$.

2nd clause: $x := a$ gives $\neg G(a, c, y) \vee H(f(a), b) \vee E(a)$.

∴, resolving gives $H(y, c) \vee D(z) \vee H(f(a), b) \vee E(a)$.

- note: not all expressions can be unified!
  ↳ eg: $Q(a, b, y)$ and $Q(c, b, y)$, as we cannot change one constant individual symbol (eg, a) into another (eg, c)!

- recall: soundness of resolution: if resolution with input S outputs the empty clause, then the set S is not satisfiable
  completeness of resolution: if the set S is not satisfiable, then resolution with input S outputs the empty set.

• full example: prove using resolution that "everybody has a grandparent, provided everybody has a parent".

let the domain D be the set of all people and let $P(x,y)$ represent x is a parent of y. The premise can then be stated as $\forall x \exists y\, P(y,x)$.

we can express the conclusion as $\forall x \exists y \exists z\, (P(z,y) \wedge P(y,x))$.

∴, we want to prove $\forall x \exists y\, P(y,x) \models \forall x \exists y \exists z\, (P(z,y) \wedge P(y,x))$.

• first, add the negation of the conclusion to the set of premises:

$\forall x \exists y\, P(y,x)$, $\exists x \forall y \forall z (\neg P(z,y) \vee \neg P(y,x))$.

• this is already in prenex normal form, so now we convert to $\exists$-free prenex normal form:

$\forall x\, P(f(x),x)$, $\forall y \forall z (\neg P(z,y) \vee \neg P(y,a))$.

• then, we can "drop" the universal quantifiers:

$P(f(x),x)$, $\neg P(z,y) \vee \neg P(y,a)$.

• now we can do resolution:

1) $P(f(x),x)$ from premise

2) $\neg P(z,y) \vee \neg P(y,a)$ from negation of conclusion

3) $P(f(a),a)$ by $x := a$ on 1

4) $\neg P(z,f(a)) \vee \neg P(f(a),a)$ by $y := f(a)$ on 2

5) $\neg P(z,f(a))$ by resolving 3 and 4

6) $P(f(f(a)),f(a))$ by $x := f(a)$ on 1

7) $\neg P(f(f(a)),f(a))$ by $z := f(f(a))$ on 5

8) $\{\}$ by resolving 6 and 7.

Since resolution outputted the empty clause, we know that the

clauses were not satisfiable and therefore the argument was valid by the soundness of resolution.

Comments on Resolution

- Any clause can be used multiple times as a parent
- Any clause with variables can be instantiated multiple times
- In any clause, we can remove duplicate literals
- resolutions that result in formulas that are universally valid should be avoided (eg, $P \vee \neg P$).
- resolution is NOT an algorithm!

Turing Machines

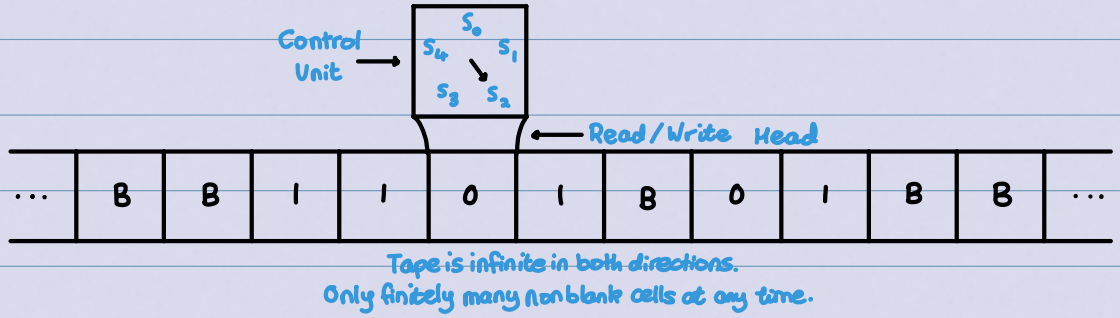
- A Turing Machine is a simple mathemetical model of a computation / computer. It consists of:
  - A two-way infinite tape, divided into cells.
  - A finite control unit with a read-write head, which can move along the tape, and can be in any state from a finite state.
  - Read / Write capabilities on the tape, as the finite control unit moves back and forth on the tape, changing states depending on:
    a) the tape symbol currently being read
    b) its current state.

- Formally, a Turing Machine $T = (S, I, f, s_0)$ consists of:
  - S: a finite set of states
  - I: an input alphabet (finite set of symbols / letters)

containing the blank symbol B

- $s_0 \in S$: the start state
- $f : S \times I \to S \times I \times \{L, R\}$: a partial function called the transition function, where L and R stand for "Left" and "Right".

- to interpret this definition in terms of a machine, consider a control unit and a tape divided into cells, infinite in both directions, having only a finite number of nonblank symbols on it at any given time:



- given a string, to write on the tape means that we write consecutive symbols in this string in consecutive cells.
- the action of the Turing machine at each step of its operation depends on the value of the transition function f for the current state and current tape symbol being read by the control unit.

How a Turing Machine works

- At each step, the control unit reads the current tape symbol x.
- If the control unit is in state s and the partial function f is defined for the pair (s, x), by $f(s, x) = (s', x', d')$, the control unit:

1) enters the state s',

2) writes the symbol x' in the current cell, erasing x,

3) moves right by one cell if $d=R$ or moves left one cell if $d=L$.

- we write this step as a 5-tuple (s, x, s', x', d), and call it a transition rule of the TM.
- if the transition function f is undefined for the pair (s, x), then the Turing machine T will halt.

- an alphabet Σ is a finite non-empty set of symbols, also called letters.
  - ↳ eg, $\Sigma = \{0, 1\}$ is the binary alphabet.
- Σ^* denotes the set of all possible strings written with letters from Σ, including the empty string λ.
  - ↳ eg, if $\Sigma = \{0, 1\}$, then $\Sigma^* = \{\lambda, 0, 1, 00, 01, 1100111010, \ldots\}$.
- a language L over Σ is a subset of Σ^*.
  - ↳ eg, if $\Sigma = \{0, 1\}$, then $L = \{w \in \{0,1\}^* \mid w \text{ has equally many 1s / 0s}\}$ is a possible language over Σ.
- TMs can be used to accept / recognize a language.

- A final state of a Turing machine $T = (S, I, f, s_0)$ is any state $s_f \in S$ that is not the first state in any five-tuple in the description of T using five-tuples.

- Language Accepted by a TM: a turing machine $T = (S, I, f, s_0)$ accepts/recognizes a string $x \in V^*$, where $V \subseteq I$, if and only if T, starting in the initial position when x is written on the tape, halts in a final state. T is said to accept/recognize a language L over V, if x is

accepted by T if and only if x belongs to L.

- A Turing machine T does not accept a string x in V^* if, when started in the initial position with x written on the tape:
  - either: T does not halt (infinite loop), or
  - T halts in a non-final state.

- Transitions
  - A common way to define a TM is to specify its transition rules as a set of 5-tuples of the form (s, x, s', x', d).
  - Another way to define a TM is by a transition diagram, where:
    - ↳ each state is represented by a node
    - ↳ the start and final states are specified (incoming arrow / double ring)
    - ↳ a transition rule (s, x, s', x', d) is symbolized by an arrow between node s and the node s' that is labelled by the triplet $x/x', d$ (ie, current symbol, new symbol, move direction).

- Decider / Total Turing Machine: a TM that always halts on every input.

- Decision Problem: a yes or no question on an infinite set of inputs. Each input is an instance of the problem.

- A decision problem for which there exists a terminating algorithm that solves it is called decidable (solveable)
  If no such algorithm exists, it is undecidable (unsolveable)

- Let $S \subseteq \mathbb{N}$ be any subset. The S-membership problem asks

"for any arbitrary $x \in \mathbb{N}$, is $x \in S$?"

- A set $S \subseteq \mathbb{N}$ is decidable (and respectively undecideable) if the S-membership problem is decideable (and respectively undecideable).
  ↳ we use reduction to determine if a problem is decidable.

- if we have an algorithm to convert any instance of the problem P_1 into an instance of the problem P_2 with the same yes/no answer, then we say that P_1 is reduced to P_2. NOTE: DO TM UNDECIDABILITY REDUCTION EXAMPLES!
- if problem P_1 is reducible to P_2, then if P_1 is undecidable, P_2 must also be undecidable.
  ↳ if we want to prove P_2 undecidable, we must reduce another known undecidable problem P_1 to our P_2.

Peano Arithmetic (PA)

let's start by proving properties of equality (using $\approx -$ and $\approx +$).

- Reflexivity: prove $\emptyset \vdash \forall x\,(x = x)$
  1) $\emptyset \vdash u = u$ by $\approx +$
  2) $\emptyset \vdash \forall x\,(x = x)$ by $\forall +$ on 1, u not elsewhere

- Symmetry: prove $\emptyset \vdash \forall x \forall y\,((x = y) \to (y = x))$
  1) $u = v \vdash u = v$ by $\in$
  2) $\emptyset \vdash u = u$ by $\approx +$
  3) $u = v \vdash u = u$ by $+$ on 2
  4) $u = v \vdash v = u$ by $\approx$ on 1, 3
  5) $\emptyset \vdash u = v \to v = u$ by $\to +$ on 4
  6) $\emptyset \vdash \forall x\,((x = v) \to (v = x))$ by $\forall +$ on 5, u not elsewhere

7) $\emptyset \vdash \forall x \forall y ((x=y) \to (y=x))$ by $\forall+$ on 6, v not elsewhere

- Note, a variant of $\approx-$: (difference is $t_2=t_1$ instead of $t_1=t_2$)
  if $\Sigma \vdash A(t_1)$ and $\Sigma \vdash t_2=t_1$, then $\Sigma \vdash A'(t_2)$ $(\approx-)'$
  ↳ can otherwise use the symmetry of equality to get here.

- Transitivity: prove $\emptyset \vdash \forall x \forall y \forall z ((x=y) \wedge (y=z) \to (x=z))$
  1) $(u=v) \wedge (v=w) \vdash (u=v) \wedge (v=w)$ by $\in$
  2) $(u=v) \wedge (v=w) \vdash (u=v)$ by $\wedge-$ on 1
  3) $(u=v) \wedge (v=w) \vdash (v=w)$ by $\wedge-$ on 1
  4) $(u=v) \wedge (v=w) \vdash (u=w)$ by $\approx-$ on 2, 3
  5) $\emptyset \vdash ((u=v) \wedge (v=w) \to (u=w))$ by $\to+$ on 4
  6) $\emptyset \vdash \forall x ((x=v) \wedge (v=w) \to (x=w))$ by $\forall+$ on 5, u not elsewhere
  7) $\emptyset \vdash \forall x \forall y ((x=y) \wedge (y=w) \to (x=w))$ by $\forall+$ on 6, v not elsewhere
  8) $\emptyset \vdash \forall x \forall y \forall z ((x=y) \wedge (y=z) \to (x=z))$ by $\forall+$ on 7, w not elsewhere.

- The set A of domain axioms is a set of first-order logic formulas which we accept/assume to always be true in the specific domain/theory.
  ↳ A should be decidable - ie, there should exist a terminating algorithm to determine if a given formula is a domain axiom.
  ↳ A should be consistent
  ↳ A should be syntactically complete - ie, for any formula F describable in the language of the system, either F or $\neg F$ should be provable from A.

A set of formulas Σ is consistent if there is no formula F, such that $\Sigma \vdash F$ and $\Sigma \vdash \neg F$. Otherwise, Σ is inconsistent.

→ note: A set of logic formulas Σ is satisfiable if and only if Σ is consistent.

- **Peano Arithmetic**
  - **Non-Logical Symbols:**
    - individual (constant): 0
    - functions: successor (s), $+$, $\cdot$
    - relation: equality
  - **Axioms** defining the unary function **successor**, and the binary functions **addition** and **multiplication**.
  - **Axiom** for **induction**

↳ PA 1: $\forall x \neg(s(x) = 0)$
PA 2: $\forall x \forall y ((S(x) = S(y)) \rightarrow x = y)$
} sufficient to describe successor

PA 3: $\forall x (x + 0 = x)$
PA 4: $\forall x \forall y (x + S(y) = S(x+y))$
} sufficient to recursively define addition

PA 5: $\forall x (x \cdot 0 = 0)$
PA 6: $\forall x \forall y (x \cdot S(y) = x \cdot y + x)$
} sufficient to recursively define multiplication

PA 7: $(A(0) \wedge \forall x (A(x) \rightarrow A(S(x)) \rightarrow \forall x A x$ sufficient to define induction
(for each formula $A(u)$ with free variable u.

↳ we denote the set of Peano Axioms $\{P_1, P_2, \ldots, P_7\}$ as **PA**.
in a Peano Arithmetic proof, the set **PA** is implicitly in the premises.

- Notation: given a theory T, with an associated set of Axioms A_T, we use $\Sigma \vdash_{A_T} C$ to denote $\Sigma, A_T \vdash C$.
  ↳ in particular for Peano Arithmetic, we will use $\Sigma \vdash_{PA} C$.

Proofs in Peano Arithmetic

- for these proofs:
  - k denotes a free variable
  - red line numbers denote steps that prove the base case
  - blue line numbers denote steps that prove the inductive step
  - green line numbers denote the application of induction (PA 7).

- Show that $\forall x\,(s(x) \neq x)$

↳ formally, we want to prove $\emptyset \vdash_{PA} \forall x\,(s(x) \neq x)$

idea: let's do induction on $A(u) = s(u) \neq u$!

use PA7: $A(0) \wedge \forall x\,(A(x) \to A(s(x))) \to \forall x\,A(x)$.

1) $\emptyset \vdash_{PA} \forall x\,(s(x) \neq 0)$ by PA1

2) $\emptyset \vdash_{PA} (s(0) \neq 0)$ by $\forall$- on 1

proof by contradiction, so adding $\neg A$ to premises

3) $s(k) \neq k,\ s(s(k)) = s(k) \vdash_{PA} s(s(k)) = s(k)$ by $\in$

4) $\emptyset \vdash_{PA} \forall x \forall y\,(s(x) = s(y) \to x = y)$ by PA2

5) $\emptyset \vdash_{PA} \forall y\,(s(s(k)) = s(y) \to s(k) = y)$ by $\forall$- on 4

6) $\emptyset \vdash_{PA} s(s(k)) = s(k) \to s(k) = k$ by $\forall$- on 5

7) $s(k) \neq k,\ s(s(k)) = s(k) \vdash_{PA} s(s(k)) = s(k) \to s(k) = k$ by + on 6

8) $s(k) \neq k,\ s(s(k)) = s(k) \vdash_{PA} s(k) = k$ by $\to$- on 3, 6

9) $s(k) \neq k,\ s(s(k)) = s(k) \vdash_{PA} s(k) \neq k$ by $\in$

10) $s(k) \neq k \vdash_{PA} s(s(k)) \neq s(k)$ by $\neg$+ on 7, 8

11) $\emptyset \vdash_{PA} s(k) \neq k \to s(s(k)) \neq s(k)$ by $\to$+ on 9

12) $\emptyset \vdash_{PA} \forall x\,(s(x) \neq x \to s(s(x)) \neq s(x))$ by $\forall$+ on 11 [u not elsewhere]

13) $\emptyset \vdash_{PA} s(0) \neq 0 \wedge \forall x\,(s(x) \neq x \to s(s(x)) \neq s(x))$ by $\wedge$+ on 2, 12

14) $\emptyset \vdash_{PA} (s(0) \neq 0 \wedge \forall x\,(s(x) \neq x \to s(s(x)) \neq s(x)) \to \forall x\,(s(x) \neq x)$ by PA7

15) $\emptyset \vdash_{PA} \forall x\,(s(x) \neq x)$ by $\to$- on 12, 13.

- Show that $\forall x\,(x = 0 \vee \exists y\,(s(y) = x))$

↳ formally, we want to prove $\emptyset \vdash_{PA} \forall x\,(x = 0 \vee \exists y\,(s(y) = x))$.

let's try induction on $A(u) = u=0 \vee \exists y\,(S(y)=u)$

using PA7: $A(0) \wedge \forall x(A(x) \to A(S(x)) \to \forall x\, A(x)$.

1) $\emptyset \vdash_{PA} 0=0$ by $\approx +$

2) $\emptyset \vdash_{PA} 0=0 \vee \exists y\,(S(y)=0)$ by $\vee +$ on 1

3) $\emptyset \vdash_{PA} S(k)=S(k)$ by $\approx +$

4) $\emptyset \vdash_{PA} \exists y\,(S(y)=S(k))$ by $\exists +$ on 3

5) $\emptyset \vdash_{PA} (S(k)=0) \vee \exists y\,(S(y)=S(k))$ by $\vee +$ on 4

6) $k=0 \vee \exists y(S(y)=k) \vdash_{PA} (S(k)=0) \vee \exists y(S(y)=S(k))$ by $+$ on 5

7) $\emptyset \vdash_{PA} (k=0) \vee \exists y\,(S(y)=k) \to (S(k)=0) \vee \exists y(S(y)=S(k))$ by $\to +$ on 6

8) $\emptyset \vdash_{PA} \forall x\,((x=0) \vee \exists y\,(S(y)=x) \to (S(x)=0) \vee \exists y\,(S(y)=S(x))$ by $\forall +$ on 7 [k not elsewhere]

9) $\emptyset \vdash_{PA} (0=0 \vee \exists y(S(y)=0)) \wedge \forall x\,(x=0 \vee \exists y(S(y)=x) \to (S(x)=0 \vee \exists y(S(y)=S(x))$ by $\wedge +$ on 2, 8

10) $\emptyset \vdash_{PA} (0=0 \vee \exists y(S(y)=0)) \wedge \forall x\,(x=0 \vee \exists y(S(y)=x) \to (S(x)=0 \vee \exists y(S(y)=S(x)) \to \forall x\,(x=0 \vee \exists y(S(y)=x))$ by PA7

11) $\emptyset \vdash_{PA} \forall x\,(x=0 \vee \exists y\,(S(y)=x))$ by $\to -$ on 9, 10

- prove that $\leq$ is transitive $(\forall x \forall y \forall z\,(x \leq y \wedge y \leq z \to x \leq z)$.

↳ $u \leq v \to \exists z\,(u+z=v)$.

we want to add the antecedent $(x \leq y \wedge y \leq z)$ to the premises, but call them some unquantified free variables u, v, w.

but since $\leq$ has a $\exists$, we want to remove that too.

∴, we add $u+a=v$ and $v+b=w$ to the premises!

1) $u+a=v,\ v+b=w \vdash_{PA} u+a=v$ by $\in$

2) $u+a=v,\ v+b=w \vdash_{PA} v+b=w$ by $\in$

3) $u+a=v,\ v+b=w \vdash_{PA} (u+a)+b=w$ by $(x-)'$ on 1, 2

4) $u+a=v,\ v+b=w \vdash_{PA} u+(a+b)=w$ by associativity of $+$ on 3

5) $u+a=v,\ v+b=w \vdash_{PA} \exists y\,(u+y=w)$ by $\exists +$ on 4

6) $u+a=v,\ v+b=w \vdash_{PA} u \leq w$ by defn of $\leq$.

7) $\exists y\,(u+y=v),\ \exists y(v+y=w) \vdash_{PA} u \leq w$ by $\exists -$ twice on 6 [a, b not elsewhere]

8) $u \leq v,\ v \leq w \vdash_{PA} u \leq w$ by defn of $\leq$ (twice)

→ this proves the Transitivity of ≤. now we must coerce the form:

9) $(u \leq v) \wedge (v \leq w) \vdash_{PA} (u \leq v) \wedge (v \leq w)$ by $\wedge$-

10) $(u \leq v) \wedge (v \leq w) \vdash_{PA} u \leq v$ by $\wedge$- on 9

11) $(u \leq v) \wedge (v \leq w) \vdash_{PA} v \leq w$ by $\wedge$- on 9

12) $(u \leq v) \wedge (v \leq w) \vdash_{PA} u \leq w$ by Transitivity on 8, 10, 11

13) $\emptyset \vdash_{PA} (u \leq v) \wedge (v \leq w) \rightarrow (u \leq w)$ by $\rightarrow$+ on 12

14) $\emptyset \vdash_{PA} \forall x \forall y \forall z ((x \leq y) \wedge (y \leq z) \rightarrow (x \leq z))$ by $\forall$+ (thrice) on 13 [u, v, w not elsewhere].

- Note: the Peano Axioms are decidable and consistent. But, we cannot, for every formula F, either formally prove F or disprove ¬F.

Gödel's Incompleteness Theorem: in any consistent formal theory T with a decidable set of axioms, there exists a statement/formula that can neither be proved nor disproved in the theory.

Program Verification

Program Correctness: does a given program satisfy its specification, ie, does it do what it's supposed to do?

↳ techniques for showing program correctness:

- Inspection (code walkthrough)
- Testing
  - Black-box Testing: tests designed independent of code
  - White-box Testing: tests designed based on code
- Formal Program Verification
  - formally state the specification of the problem (using the formalism of first-order logic)
  - prove that the program satisfies the specification for all inputs.

- The steps of formal (program) verification:
  1) convert the informal description R of requirements for an application into an "equivalent" formula Φ_R of some symbolic logic,
  2) write a program P which is meant to realize Φ_R in some given programming environment, and
  3) prove that program P satisfies the formula Φ_R.

we will only do step 3 in CS 245!!

we are verifying imperative, sequential, and transformational programs.

- sequential: no concurrency
- transformational: given inputs, compute outputs and terminate
- imperative:
  - manipulate the values of "variables"
  - the state of a program consists of a vector of the values of all the variables at a particular time in the execution of the program.
  - expressions are evaluated relative to the current state of the program
  - executing a statement/command changes the state of the program.

Hoare Triples

our assertions about programs will have the form $(|P|)\ C\ (|Q|)$ where $(|P|)$ is the precondition, C is the program/code, and $(|Q|)$ is the postcondition.

- Conditions P and Q are written in the first-order logic of integers (using relations $<, =$, functions $+, -, *$, and so on).

A specification of a program C is a Hoare triple

(| P |) C (| Q |).

↳ eg: express the following requirement as a Hoare triple: "if the input x is a positive number, compute a number whose square is less than x"

↳ $(| x > 0 |)$ C $(| y \cdot y < x |)$

but this can also just always output $y = 0$! making it more specific: $(| x > 0 |)$ C $(| y \cdot y < x \wedge \forall z\,(z \cdot z < x \rightarrow z \leq y) |)$

- if there's no precondition P, just make it (| true |).

A Hoare triple (| P |) C (| Q |) is satisfied under partial correctness, denoted $\models_{par}$ (| P |) C (| Q |), if and only if:
for every state s that satisfies condition P,
if the execution of the program starting in state S terminates in state s',
then the state s' satisfies condition Q.

↳ note: any program that does not terminate (infinite loop, eg while true {x=0;}) is satisfied under partial correctness!

A Hoare triple (| P |) C (| Q |) is satisfied under total correctness, denoted $\models_{tot}$ (| P |) C (| Q |), if and only if:
for every state s that satisfies condition P,
execution of program C starting from state s terminates,
and the resulting state s' satisfies Q.

- basically, the difference between partial and total correctness is that total correctness requires termination!

example: $(| x = 1 |)$ $y = x$ $(| y = 1 |)$

↳ satisfied under both partial and total correctness.

example: (|x = 1|) y = x (|y = 2|)

↳ not satisfied by partial or total correctness.

example: (| x = 1 |) while (true) { x = 0; } (| x > 0 |)

↳ never terminates, so satisfied under partial but not total correctness!

example: (| x ≥ 0 |)

y = 1; z = 0;

while (z != x) {z = z + 1; y = y · z; }

(|y = x ! |)

↳ satisfied under both partial and total correctness.

example: (|x ≥ 0 |)

y = 1;

while (x != 0) { y = y · x ; x = x - 1; }

(|y = x !|)

↳ not satisfied by partial or total correctness, because the input x is altered / consumed!

We can write the pre- and post-conditions for partial and total correctness in first-order logic:

- relation State (s) → "s is a program State"
- relation Condit (P) → "P is a condition"
- relation Code (c) → "C is a program"
- relation Satisfies (s, P) → "State s satisfies condition P"

- relation Terminates(C, S) → "Program C terminates when execution begins in state S"
- function result(C, S) → "the state that results from executing code C beginning in state S, if C terminates (undefined otherwise)."

Using these, Partial Correctness of a Hoare Triple:

$\forall s \forall P \forall C \forall Q [State(s) \wedge Condit(P) \wedge Code(C) \wedge Condit(Q) \rightarrow (Satisfies(s, P) \wedge Terminates(C, s) \rightarrow Satisfies(result(C, s), Q))]$

And, Total Correctness of a Hoare Triple:

$\forall s \forall P \forall C \forall Q [State(s) \wedge Condit(P) \wedge Code(C) \wedge Condit(Q) \rightarrow (Satisfies(s, P) \rightarrow Terminates(C, s) \wedge Satisfies(result(C, s), Q))]$

Partial Correctness Proofs

- an annotated program, with one or more condition(s) before and after each program statement.
- each program statement, together with the preceding and following condition, form a Hoare Triple:

  (|precondition|) program statement (|postcondition|)

logical / auxillary variable: "stores" the value of a precondition.

↳ eg: $(|x = x_0 \wedge x_0 > 0|)$

```
y=1; while (true) {y=y*x; x=x-1;}
```

$(|y = x_0!|)$

see here that this satisfies both partial and total completeness now that x_0 acts as a logical variable!

Inference Rule for Assignment: $\dfrac{}{(|Q[E/x]|)\ x = E\ (|Q|)}$ (assignment)

↳ note: to read these rules, if the condition(s) / Hoare triple(s) above the horizontal line are proved, then the Hoare triple under the horizontal line follows.

→ assignment rule: "Q(x) will hold after assigning (the value of) E to x, if Q(E) was true beforehand.

note: we read / as "in place of", so Q[E/x] is read as "Q with E in place of x".

• example: $(|y+1 = 7|)\ x = y+1\ (|x = 7|)$

↳ the partial correctness is proved by one application of the (sound) assignment inference rule, with Q(x) being "x = 7" and E being "y + 1".

the assignment rule is "applied backwards"!

example: find $(|P|)$ in: $(|P|)\ x = x+1;\ (|x = 2|)$

we simply substitute backwards. see that $x = 2$ and $x = x+1$, so $(|P|) = (|x+1 = 2|)$

example: find $(|P|)$ in: $(|P|)\ x = x+1;\ (|x = n+1|)$

see that $x = n+1$ and $x = x+1$, so $(|P|) = (|x+1 = n+1|)$

Implied Rule of "precondition strengthening":

$$\frac{P \to P' \qquad (|P'|)\ C\ (|Q|)}{(|P|)\ C\ (|Q|)}$$ (implied)

Implied Rule of "postcondition weakening":

$$\frac{(|P|)\ C\ (|Q'|) \qquad Q' \rightarrow Q}{(|P|)\ C\ (|Q|)} \text{ (implied)}$$

example: show that the program "$x = y + 1$" satisfies the specification $(|y = 6|)\ x = y+1\ (|x = 7|)$ under partial correctness.

$(|y = 6|)$
$x = y + 1$
$(|x = 7|)$

$\rightarrow$

$(|y = 6|)$
$(|y + 1 = 7|)$
$x = y + 1$
$(|x = 7|)$ assignment

$\rightarrow$

$(|y = 6|)$
$(|y + 1 = 7|)$ implied
$x = y + 1$
$(|x = 7|)$ assignment

example: show that the program "$x = y + 1$" satisfies the specification $(|y+1 = 7|)\ x = y+1\ (|x \leq 7|)$ under partial correctness.

$(|y+1 = 7|)$
$x = y+1$
$(|x \leq 7|)$

$\rightarrow$

$(|y+1 = 7|)$
$x = y+1$
$(|x = 7|)$ assignment
$(|x \leq 7|)$

$\rightarrow$

$(|y+1 = 7|)$
$x = y+1$
$(|x = 7|)$ assignment
$(|x \leq 7|)$ implied

Inference Rule for Composition of Instructions

$$\frac{(|P|)\ C_1\ (|Q|), \quad (|Q|)\ C_2\ (|R|)}{(|P|)\ C_1 ; C_2\ (|R|)} \text{ (composition)}$$

↳ we must find a midcondition Q for which we can prove:
$(|P|)\ C_1\ (|Q|)$ and $(|Q|)\ C_2\ (|R|)$.

Proof Format: Annotated Programs

- interweave program statements with assertions (= conditions), each justified by an inference rule.
- the composition rule is implicit
- each assertion should hold whenever the program reaches

that point in its execution.

- if the implied inference rule is used, we also need to provide a (first-order logic) formal proof of the implication $\emptyset \vdash P \to P'$ or $\emptyset \vdash Q' \to Q$.
  ↳ usually, we do these proofs seperately after annotating.

Example: show that the following Hoare Triple (whose program has 3 lines of code) is satisfied under partial correctness.

$(|x = x_0 \land y = y_0|)$
$t = x$
$x = y$
$y = t$
$(|x = y_0 \land y = x_0|)$

$\to$

$(|x = x_0 \land y = y_0|)$
$t = x$
$x = y$
$(|x = y_0 \land t = x_0|)$ assignment
$y = t$
$(|x = y_0 \land y = x_0|)$ assignment

$\to$

$(|x = x_0 \land y = y_0|)$
$t = x$
$(|y = y_0 \land t = x_0|)$ assignment
$x = y$
$(|x = y_0 \land t = x_0|)$ assignment
$y = t$
$(|x = y_0 \land y = x_0|)$ assignment

$(|x = x_0 \land y = y_0|)$
$(|y = y_0 \land x = x_0|)$
$t = x$
$(|y = y_0 \land t = x_0|)$ assignment
$x = y$
$(|x = y_0 \land t = x_0|)$ assignment
$y = t$
$(|x = y_0 \land y = x_0|)$ assignment

$\to$

$(|x = x_0 \land y = y_0|)$
$(|y = y_0 \land x = x_0|)$ implied
$t = x$
$(|y = y_0 \land t = x_0|)$ assignment
$x = y$
$(|x = y_0 \land t = x_0|)$ assignment
$y = t$
$(|x = y_0 \land y = x_0|)$ assignment

we also now need to seperately prove that:

$\emptyset \vdash x = x_0 \land y = y_0 \to y = y_0 \land x = x_0$

which is obvious.

Inference Rules for Conditionals

if then (without else):

$$\frac{\{P \wedge B\}\ C\ \{Q\} \qquad (P \wedge \neg B) \to Q}{\{P\}\ \text{if}\ (B)\ C\ \{Q\}}\ \text{(if-then)}$$

if-then-else:

$$\frac{\{P \wedge B\}\ C_1\ \{Q\} \qquad \{P \wedge \neg B\}\ C_2\ \{Q\}}{\{P\}\ \text{if}\ (B)\ C_1\ \text{else}\ C_2\ \{Q\}}\ \text{(if-then-else)}$$

annotated program template for if-then:

```
{P}                          {P}
if B {                       if B {
  C1                           {P ∧ B}  if-then
}                    →         C
{Q}                            {Q}      justify based on C
                             }
                             {Q}  if-then
                                  implied: proof of P ∧ ¬B → Q!
```

annotated program template for if-then-else:

```
{P}                          {P}
if B {                       if B {
  C1                           {P ∧ B}  if-then-else
} else {             →         C1
  C2                             {Q}    justify depending on C1
}                            } else {
{Q}                            {P ∧ ¬B}  if-then-else
                                 C2
                                 {Q}     justify depending on C2
                             }
                             {Q}  if-then-else
```

example: prove that the program below satisfies the specifications under partial correctness.

```
(| true |)
if (max < x) {
   max = x
}
(| max ≥ x |)
```

→

```
(| true |)
if (max < x) {
   (| true ∧ max < x |)        if-then
   max = x;
   (| max ≥ x |);              ← to be shown
}
(| max ≥ x |)                  if-then
```

implied: $(\text{true} \wedge \neg(\text{max} < x)) \rightarrow \text{max} \geq x$

→

```
(| true |)
if (max < x) {
   (| true ∧ max < x |)        if-then
   (x ≥ x);                    implied (a)
   max = x;
   (| max ≥ x |);              assignment
}
(| max ≥ x |)                  if-then
```

implied (b): $(\text{true} \wedge \neg(\text{max} < x)) \rightarrow \text{max} \geq x$

implied (a): $\emptyset \vdash (\text{true} \wedge \text{max} < x) \rightarrow x \geq x$

↳ clearly $x \geq x$ holds (basic arithmetic), and thus the required implication holds.

implied (b): $\emptyset \vdash (\text{true} \wedge \neg(\text{max} < x)) \rightarrow (\text{max} \geq x)$

1) $(\text{true} \wedge \neg(\text{max} < x)) \vdash (\text{true} \wedge \neg(\text{max} < x))$ by e

2) $(true \wedge \neg(max < a)) \vdash \neg(max < x)$ by $\wedge$- on 1

3) $(true \wedge \neg(max < x)) \vdash max \geq x$ by defn of $\geq$ on 2

4) $\emptyset \vdash (true \wedge \neg(max < x)) \rightarrow (max \geq x)$ by $\rightarrow +$ on 3.

example: prove that the program satisfies the specifications under partial correctness.

$(\!|\ true\ |\!)$

```
if (x > y) {
    max = x;
} else {
    max = y;
}
```

$(\!|(x > y \wedge max = x) \vee (x \leq y \wedge max = y)\ |\!)$

↓

$(\!|\ true\ |\!)$

```
if (x > y) {
```

$(\!|\ true \wedge x > y\ |\!)$ if-then-else

```
    max = x;
```

$(\!|(x > y \wedge max = x) \vee (x \leq y \wedge max = y)\ |\!)$ justify somehow

```
} else {
```

$(\!|(true \wedge \neg(x > y))$ if-then-else

```
    max = y;
```

$(\!|(x > y \wedge max = x) \vee (x \leq y \wedge max = y)\ |\!)$ justify somehow

```
}
```

$(\!|(x > y \wedge max = x) \vee (x \leq y \wedge max = y)\ |\!)$ if-then-else

↓

$(\!|\ true\ |\!)$

if (x > y) {

(| true ∧ x > y |) if-then-else

(|(x > y ∧ x = x) ∨ (x = y ∧ x = y)|) implied (a)

max = x;

(|(x > y ∧ max = x) ∨ (x ≤ y ∧ max = y)|) assignment ~~justify somehow~~

} else {

(|(true ∧ ¬(x > y)) if-then-else

(|(x > y ∧ y = x) ∨ (x ≤ y ∧ y = y)|) implied (b)

max = y;

(|(x > y ∧ max = x) ∨ (x ≤ y ∧ max = y)|) assignment ~~justify somehow~~

}

(|(x > y ∧ max = x) ∨ (x ≤ y ∧ max = y)|) if-then-else

implied (a): $\emptyset \vdash x > y \to (x > y \wedge x = x) \vee (x \leq y \wedge x = y)$

1) $x > y \vdash x > y$ by $\in$

2) $\emptyset \vdash x \approx x$ by $\approx+$

3) $x > y \vdash x \approx x$ by + on 2

4) $x > y \vdash x > y \wedge x \approx x$ by $\wedge+$ on 1,3

5) $x > y \vdash (x > y \wedge x \approx x) \vee (x \leq y \wedge x = y)$ by $\vee+$ on 4

6) $\emptyset \vdash (x > y) \to (x > y \wedge x = x) \vee (x \leq y \wedge x = y)$ by $\to+$ on 5

implied (b): $\emptyset \vdash x \leq y \to ((x > y \wedge x = x) \vee (x \leq y \wedge y = y))$

1) $x \leq y \vdash x \leq y$ by $\in$

2) $\emptyset \vdash y = y$ by $\approx+$

3) $x \leq y \vdash y = y$ by + on 2

4) $x \leq y \vdash (x \leq y \wedge y = y)$ by $\wedge+$ on 1,3

5) $x \leq y \vdash (x > y \wedge x = x) \vee (x \leq y \wedge y = y)$ by $\vee+$ on 4

6) $\emptyset \vdash (x \leq y) \to ((x > y \wedge x = x) \vee (x \leq y \wedge y = y))$ by $\to+$ on 5

While-Loops and Total Correctness

$$\frac{(\!|\ I \wedge B\ |\!)\ C\ (\!|\ I\ |\!)}{(\!|\ I\ |\!)\ \text{while}\ (B)\ \{\ C\ \}\ (\!|\ I \wedge \neg B\ |\!)}\ \text{(partial-while)}$$

- Condition **I** is called the **loop invariant.**

annotated program template for partial-while:

```
(| P |)                          (| P |)
while (B) {                      (| I |)                    implied (a)
   C                       →     while (| {
}                                   (| I ∧ B |)             partial-while
(| Q |)                             C
                                    (| I |)                 justify based on C
                                 }
                                 (| I ∧ ¬B |)               partial-while
                                 (| Q |)                    implied (b)
```

A **loop invariant** is an **assertion** (condition) that is **true** both **before** and **after** each **execution** of the body of a **loop.**

- true before the while loop begins
- true after the while loop ends
- expresses the relationship among the variables used within the body of the loop. Some of these variables will have their values changed within the loop.

example: finding a loop invariant

$(| x \geq 0 |)$

```
y = 1;
z = 0;
while (z != x) {
    z = z+1;
    y = y*z;
}
```

$(| y = x! |)$

→ let's trace this for some input $x=5$:

x	y	z	$z != x$
5	1	0	true
5	1	1	true
5	2	2	true
5	6	3	true
5	24	4	true
5	120	5	false.

notice here that a constant relationship between y and z is $y = z!$.

$(| x \geq 0 |)$

$y = 1;$

$z = 0;$

$(| y = z! |)$ justify

while $(z != x)$ {

$(|(y = z!) \wedge \neg(z = x)|)$ partial-while

$z = z+1;$

$y = y*z;$

$(| y = z! |)$ justify

}

$(| y = z! \wedge z = x |)$ partial-while

$(| y = x! |)$

→

$(| x \geq 0 |)$

$(|1 = 0!|)$ implied (a)

$y = 1;$

$(| y = 0! |)$ assignment

$z = 0;$

$(| y = z! |)$ assignment

while $(z != x)$ {

$(|(y = z!) \wedge \neg(z = x)|)$ partial-while

$(| y*(z+1) = (z+1)! |)$ implied (b)

$z = z+1;$

$(| y*z = z! |)$ assignment

$y = y*z;$

$(| y = z! |)$ assignment

}

...

implied (a): $(x \geq 0) \vdash (1 = 0!)$ $(\lfloor y = z! \wedge z = x \rfloor)$ partial-while

↳ true by defn. of factorial! $(\lfloor y = x! \rfloor)$ implied (c)

implied (b): $((y = z!) \wedge \neg(z = x)) \vdash (z+1)y = (z+1)!$

1) $y = z! \wedge z \neq x \vdash y = z! \wedge z \neq x$ by $\in$

2) $y = z! \wedge z \neq x \vdash y = z!$ by $\wedge$- on 1

3) $y = z! \wedge z \neq x \vdash (z+1)y = (z+1)z!$ by algebra on 2

4) $y = z! \wedge z \neq x \vdash (z+1)z! = (z+1)!$ by defn of factorial and +

5) $y = z! \wedge z \neq x \vdash (z+1)y = (z+1)!$ by transitivity of = on 3,4

implied (c): $((y = z!) \wedge (z = x)) \vdash (y = x!)$

1) $(y = z!) \wedge (z = x) \vdash (y = z!) \wedge (z = x)$ by $\in$

2) $(y = z!) \wedge (z = x) \vdash y = z!$ by $\wedge$- on 1

3) $(y = z!) \wedge (z = x) \vdash z = x$ by $\wedge$- on 1

4) $(y = z!) \wedge (z = x) \vdash y = x!$ by x- on 2,3

example: prove that the following Hoare triple is satisfied under partial correctness:

$(\lfloor n \geq 0 \wedge a \geq 0 \rfloor)$

```
s = 1;
i = 0;
while (i < n) {
   s = s * a;
   i = i + 1;
}
```

$(\lfloor s = a^n \rfloor)$

a	n	s	i	i < n
2	5	1	0	true
2	5	2	1	true
2	5	4	2	true
2	5	8	3	true
2	5	16	4	true
2	5	32	5	false

the loop invariant could be $s = a^i$.

→ $(| n \geq 0 \land a \geq 0 |)$

```
s = 1;
i = 0;
```

$(| s = a^i |)$ justify

```
while (i < n) {
```

$(| (s = a^i) \land (i < n) |)$ while-loop

```
  s = s * a;
  i = i + 1;
```

$(| s = a^i |)$ justify

```
}
```

$(| s = a^i \land \neg(i < n) |)$ while loop

$(| s = a^n |)$

→ $(| n \geq 0 \land a \geq 0 |)$

$(| 1 = a^0 |)$ implied (a)

```
s = 1;
```

$(| s = a^0 |)$ assignment

```
i = 0;
```

$(| s = a^i |)$ justify

```
while (i < n) {
```

$(| (s = a^i) \land (i < n) |)$ while-loop

$(| (s * a) = a^{i+1} |)$ implied (b)

```
  s = s * a;
```

$(| s = a^{i+1} |)$ assignment

```
  i = i + 1;
```

$(| s = a^i |)$ assignment

```
}
```

$(|\ s = a^i \wedge \neg(i < n)\ |)$ while loop

$|\ s = a^n\ |)$ implied (c)

implied (a): $n \geq 0 \wedge a \geq 0 \vdash 1 = a^0$

↳ true be defn. of power

implied (b): $(s = a^i) \wedge (i < n) \vdash (s \times a) = a^{i+1}$

1) $(s = a^i) \wedge (i < n) \vdash (s = a^i) \wedge (i < n)$ by $\in$

2) $(s = a^i) \wedge (i < n) \vdash s = a^i$ by $\wedge$- on 1

3) $(s = a^i) \wedge (i < n) \vdash i < n$ by $\wedge$- on 1

4) $(s = a^i) \wedge (i < n) \vdash a^i \times a = a^{i+1}$ by defn of power and +

5) $(s = a^i) \wedge (i < n) \vdash s \times a = a^{i+1}$ by $=$- on 2,4

implied (c): cannot be done with $\neg(i < n)$ as $i \geq n$! we want only $i = n$. so new invariant!

↳ $I_{new} = (s = a^i) \wedge (i \leq n)$

↗ in our programming language

→ note: only while-loops can be responsible for non-termination

Proving Termination: for each while-loop in the program, identify an integer expression which is always non-negative and whose value decreases every time through the while loop.

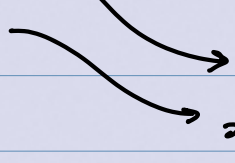
↳ eg: show that the following program terminates:

$(|\ x \geq 0\ |)$

```
y = 1;
z = 0;
while (z ≠ x) {
    z = z + 1;
}
```

→ see that the while loop continues until $z = x$, or, until $x - z = 0$. ∴ our variant is $x - z \neq 0$. at the start, $x - z = x \geq 0$.

```
    y = y * z;
}
(| y = x ! |)
```



$x-z$ decreases by 1

$x-z$ doesn't change

$\therefore$, eventually $x-z$ will reach 0 and the program will terminate!