CS60045 Artificial Intelligence
Autumn 2023

Predicate Logic

From propositional to predicate logic

- Propositional logic is declarative.
- Every atom is a piece of truth or falsehood.
- There is no way to make generic statements about objects in a collection.
- Suppose that we want to declare prime numbers.
  - We will have one atom for each natural number.
  - 1_is_prime (false), 2_is_prime (true), 3_is_prime (true), 4_is_prime (false), 5_is_prime (true), 6_is_prime (false), and so on.
  - With these propositions, we cannot describe the usual definition of primes.
  - There are infinitely many natural numbers, so we need have infinitely many such propositions.
  - Even if we restrict to a limited (finite) set of numbers (like 32-bit unsigned integers), this representation is impractical.
  - For infinite domains, this representation is impossible.
- We need a logical language more expressive than propositional logic.
- Example: Think about a declaration like is_prime(n) with a *variable* n. Plug in different values of n to get different "propositions" with their own truth values.

Predicate logic: The basic players

- Also called **First-Order Logic**.
- **Objects** are constant members of *non-empty* sets (also called worlds or universes).
  Examples: nodes in a tree, natural numbers, real numbers, sets, lists.
- **Functions** take one or more objects and return objects.
  Examples: level of a node in a tree, sum of two numbers, union of sets.
- **Relations** take one or more objects and return true/false.
  Examples: is_ancestor is a relation on two nodes in a tree, is_prime and greater_than are unary and binary relations on natural numbers, containment is a relation on sets.

Proof of compositeness of 35:

$$\text{greater\_than}(5, 1) \wedge \text{less\_than}(5, 35) \wedge \text{divides}(5{,}35) \Rightarrow \neg\text{is\_prime}(35).$$

In a more conventional language:

$$(5 > 1) \wedge (5 < 35) \wedge (5|35) \Rightarrow \neg\text{is\_prime}(35).$$

Number of arguments of a function or relation is called **arity**. Unary relations are called **properties**.

Terms

- An object constant
- A variable (to be introduced soon)
- A function with each argument a term

Examples

- 5, v (a given node in a tree), the set $\{1, 2, 3\}$ (in the set of all subsets of natural numbers)
- x, y, S
- $sum(5, 7)$ (also written as $5 + 7$), LCA(u, v, T) (the lowest common ancestor of two nodes u and v in a tree T), $absval(f(x) + g(y))$ (also written as $|f(x) + g(y)|$), $f(g(x, h(y) + 5), 3, z + 5)$.

A term without any variable is called a **ground term**.

Predicates

A **predicate** is one of the following.

1. The constants *true* and *false*
2. A proposition
3. A relation on terms

Examples

- 2_is_prime, 6_is_prime, it_is_raining, the_marble_road_leads_to_the_center
- is_prime(2), is_prime(6), is_prime$(g(2)+5)$, $f(5x+g(y)) > (u+v)/w$, $S = \emptyset$, $q = \mathrm{LCA}(u, v, T)$

A predicate is also called an **atom** or an **atomic sentence**.

A **literal** is either a predicate or its negation.

Well-formed formulas (wffs)

Atoms are wffs.

Other wffs (or **composite sentences**) can be obtained from atoms by using the logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$. Parentheses can be used for disambiguation or precedence overriding.

Examples

- $(5 > 1) \wedge (5 < 35) \wedge (5|35) \Rightarrow \neg \text{is\_prime}(35)$
- $(5 > 1) \wedge (5 < 35) \wedge (5|35) \Rightarrow \text{is\_prime}(35)$
- $(5 > 1) \wedge (5 < 35) \wedge (5|36) \Rightarrow \neg \text{is\_prime}(36)$
- $(5 > 1) \wedge (5 < 35) \wedge (5|36) \Rightarrow \text{is\_prime}(36)$
- $(d > 1) \wedge (d < n) \wedge (d|n) \Rightarrow \neg \text{is\_prime}(n)$
- $\text{is\_sibling}(u, v) \Leftrightarrow \text{parent}(u) = \text{parent}(v)$
- $\big((A \subseteq B) \vee (B \vee A)\big) \wedge \neg(C \cap D = \emptyset)$

But, this is not all.

Variables

Writing predicates for a large number of tuples of objects is impractical.

It is impossible if there is an infinite number of tuples of objects.

All primes $\leqslant 6$ can be explicitly written as:

$$\neg\text{is\_prime}(1) \land \text{is\_prime}(2) \land \text{is\_prime}(3) \land \neg\text{is\_prime}(4) \land \text{is\_prime}(5) \land \neg\text{is\_prime}(6)$$

What about all primes $\leqslant 100$ or all primes that fit in 32-bit unsigned integers or all primes no matter how large (there are infinitely many primes)?

We introduce a variable n whose domain is the set of natural numbers.

We also introduce the predicate $\text{is\_prime}(n)$.

Another example: Let the world be the set of all subsets of $\{1, 2, 3, \ldots, 100\}$. The world contains 2^{100} objects. Specifying the disjoint-ness for all pairs is impractical.

Predicates like $\text{is\_prime}(n)$ and $\text{are\_disjoint}(A, B)$ are still useless unless they can be defined precisely and concisely.

Quantifiers

Let x be a variable with domain $\mathcal{D}_x$.

Universal quantification: The wff $\forall x\ P(x)$ is true if and only if $P(x)$ is true for all $x \in \mathcal{D}_x$.

Existential quantification: The wff $\exists x\ P(x)$ is true if and only if $P(x)$ is true for at least one $x \in \mathcal{D}_x$.

Examples

- All primes other than two are odd:

$$\forall n\ \text{is\_prime}(n) \Rightarrow (n = 2) \vee \neg(2|n)$$

- There exists a prime larger than 100:

$$\exists n\ [(n > 100) \wedge \text{is\_prime}(n)]$$

- There exists a prime number larger that any positive integer:

$$\forall n\ \exists p\ [(p > n) \wedge \text{is\_prime}(p)]$$

- A (non-empty) tree has a leaf node:

$$\exists u\ \forall v\ \neg(\text{is\_parent}(v) = u)$$

Quantifiers: Examples

Domain of all variables (x, y, z, n, d) is $\mathbb{N}$ (the set of positive integers).

If $\frac{x^2+y^2}{xy-1}$ is an integer, then it is 5:

$$\forall x\ \forall y\ \forall z\ \left[\text{equals}(\text{sum}(\text{sqr}(x), \text{sqr}(y)), \text{mul}(z, \text{sub}(\text{mul}(x, y), 1))) \Rightarrow \text{equals}(z, 5)\right]$$

A more readable way to represent the same wff is:

$$[\forall x\ \forall y\ \forall z\ \left[(x^2+y^2 = z(xy-1)) \Rightarrow (z=5)\right]$$

Exercise: If $\frac{x^2+y^2}{xy+1}$ is an integer, then it is a perfect square.

Definition: n is a prime if (and only if) it does not contain non-trivial divisors:

$$\forall n\ \left[\text{is\_prime}(n) \Leftrightarrow \neg\exists d\ [(d>1) \wedge (d<n) \wedge (d|n)]\right]$$

Definition: $n > 1$ is a prime if (and only if) its only divisor are 1 and n:

$$\forall n\ \left[\text{is\_prime}(n) \Leftrightarrow (n>1) \wedge \forall d\ [(d|n) \Rightarrow (d=1) \vee (d=n)]\right]$$

Quantifiers: Examples

Let S the set of all functions on a set A. The domain for function variables (like f) is S, and the domain for element variables (like x, y) are A.

A given function g (a constant member of S) is non-constant:

$$\exists x\ \exists y \Big[\neg (g(x) = g(y)) \Big]$$

Define bijective functions:

$$\forall f\ \Big[\text{is\_bijective}(f) \Leftrightarrow \Big(\forall x\ \forall y\ (f(x) = f(y)) \Rightarrow (x = y)\Big) \wedge \Big(\forall y\ \exists x\ (f(x) = y)\Big)\Big]$$

Define everywhere continuous functions for $S = \mathbb{R}$. The domain for δ and ϵ are $\mathbb{R}^+$.

$$\forall f\ \Big[\text{is\_continuous}(f) \Leftrightarrow \forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [(|y - x| < \delta) \Rightarrow (|f(y) - f(x)| < \epsilon)]\Big]$$

Quantifiers: Examples

Let T be a rooted tree given in the parent-pointer representation. The parent of the root node is NULL. The domain of each variable is $V(T)$ (the vertex set of T).

T contains a leaf node:

$$\exists u \; \forall v \; \neg(\text{parent}(v) = u)$$

T contains an internal node:

$$\exists u \; \exists v \; (\text{parent}(v) = u)$$

A (constant) node w has at least one sibling:

$$\exists u \; \left[\neg(u = w) \wedge (\text{parent}(u) = \text{parent}(w))\right]$$

A (constant) node w has exactly one sibling:

$$\exists u \; \Big[\neg(u = w) \wedge (\text{parent}(u) = \text{parent}(w)) \wedge \forall v \; \left[(\text{parent}(v) = \text{parent}(w)) \Rightarrow (v = w) \vee (v = u)\right]\Big]$$

Quantifiers: Examples

Let S be the set of all rooted trees (presented by parent pointers). The domain of tree variables (like T) is S, and the domain of node variables are vertex sets of trees.

Define ancestor in all trees:

$$\forall T\ \forall u\ \forall q \Big[\text{is\_ancestor}(q, u, T) \Leftrightarrow (q = u) \vee \text{is\_ancestor}(q, \text{parent}(u), T)\Big]$$

Define common ancestor in all trees:

$$\forall T\ \forall u\ \forall v\ \forall q\ \Big[\text{is\_CA}(q, u, v, T) \Leftrightarrow \text{is\_ancestor}(q, u, T) \wedge \text{is\_ancestor}(q, v, T)\Big]$$

Define lowest common ancestor in all trees:

$$\forall T\ \forall u\ \forall v\ \forall q\ \Big[q = \text{LCA}(u, v, T) \Leftrightarrow \Big(\text{is\_CA}(q, u, v, T) \wedge \forall r\ \big[\text{is\_CA}(r, u, v, T) \Rightarrow \text{is\_ancestor}(r, q, T)\big]\Big)\Big]$$

Lowest common ancestor is unique:

$$\forall T\ \forall u\ \forall v\ \forall q\ \forall q'\ \Big[(q = \text{LCA}(u, v, T)) \wedge (q' = \text{LCA}(u, v, T)) \Rightarrow (q = q')\Big]$$

Quantifiers: Examples

Two particular trees T and T' are isomorphic:

$$\exists f \left[\text{is\_bijection}(f) \wedge \forall u \left[f(\text{parent}(u, T)) = \text{parent}(f(u), T')\right]\right]$$

We have:

- The domain of f is the set of all functions $V(T) \to V(T')$.
- The domain of u is $V(T)$.

Define the isomorphism relation:

$$\forall T\ \forall T' \left[\text{are\_isomorphic}(T, T') \Leftrightarrow \exists f \left[\text{is\_bijection}(f) \wedge \forall u \left[f(\text{parent}(u, T)) = \text{parent}(f(u), T')\right]\right]\right]$$

Properties of quantifiers

- The variable used in a quantifier is a symbolic name, and can be changed if this does not lead to confusion. $\forall x \; P(x)$ can be rewritten as $\forall y \; P(y)$ provided that y does not appear in $P(x)$. Similarly for $\exists x \; p(x)$.
- Consecutive quantifiers of the same type can be interchanged. That is, $\forall x \; \forall y \; P(x, y)$ is the same as $\forall y \; \forall x \; P(x, y)$, and $\exists x \; \exists y \; P(x, y)$ is the same as $\exists y \; \exists x \; P(x, y)$. We often abbreviate as $\forall x, y \; P(x, y)$ and $\exists x, y \; P(x, y)$.
- Different types of quantifiers cannot be swapped. Example:
  - $\forall n \; \exists p \; [(p > n) \wedge \text{is\_prime}(p)]$ means every natural has a prime larger than it [*true*]
  - $\exists p \; \forall n \; [(p > n) \wedge \text{is\_prime}(p)]$ means there is a prime larger than all natural numbers [*false*]
- We have the following two logical equivalences:
  - $\neg(\forall x \; P(x)) \equiv \exists x \; \neg P(x)$
  - $\neg(\exists x \; P(x)) \equiv \forall x \; \neg P(x)$
- Rules of inference (a is a constant in the domain of x)
  - **Universal instantiation (EI):** $\forall x \; P(x)$ entails $P(a)$
  - **Existential generalization (UG):** $P(a)$ entails $\exists x \; P(x)$.

Predicate-logic theorem proving

Knowledge base: A set of wffs: $\phi_1, \phi_2, \ldots, \phi_n$

Goal or **Query:** A wff ψ.

The task is to decide whether $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$ entails ψ.

Example

- The knowledge base consists of the following.
  - A rooted tree T completely specified by the parent function $V(T) \to V(T) \cup \{NULL\}$.
  - The wff $\forall u\ \forall v \left[\text{is\_ancestor}(u, v) \Leftrightarrow (u = v) \vee \text{is\_ancestor}(u, \text{parent}(v))\right]$.
- Query: is_ancestor(a, b), where a and b are two constant nodes in T.
- Query: $\exists x\ \exists y \left[\neg(x = y) \wedge \neg\text{is\_ancestor}(x, y) \wedge \neg\text{is\_ancestor}(y, x)\right]$.

The equality predicate

Every knowledge base implicitly contains comparison predicates of constant objects.

The simplest comparison is equality check.

Let a and b be objects (not variables).

If a and b are the same object, then we have the predicate $(a = b)$.

If a and b are different objects, we have the predicate $\neg(a = b)$.

If a and b are from an ordered set, we may also talk about $(a \leqslant b)$, $(a < b)$, $(a \geqslant b)$, and $(a > b)$.

For example, if the objects are natural numbers, we have the implicit predicates $(5 < 7)$, $(5 \leqslant 7)$, $\neg(5 > 7)$, and $\neg(5 \geqslant 7)$.

Any other common predicates derivable from the properties of the objects can be included in the knowledge base.

Resolution refutation

We plan to show that $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n \wedge \neg\psi$ entails F.

This is similar to propositional theorem proving, but we need to deal with variables and quantifiers.

Steps

1. Eliminate implications and reduce the scopes of negation.
2. [Skolemization] Eliminate existential quantifiers.
3. [Prenex normal form] Move all the universal quantifiers to the beginning.
4. Eliminate the universal quantifiers.
5. Convert the formula to CNF.
6. Rename variables so that all the clauses have mutually distinct variables.
7. [Resolution] Keep on resolving clauses over literals.

   [Unification] You need to make safe substitutions in clauses for doing resolution.

Step 1: Handling implications and negations

$$\forall f \left[\text{is\_continuous}(f) \Leftrightarrow \forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [(|y-x| < \delta) \Rightarrow (|f(y) - f(x)| < \epsilon)]\right]$$

[Eliminate $\Leftrightarrow$]

$$\equiv \quad \forall f \Big[\Big(\text{is\_continuous}(f) \Rightarrow \forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [(|y-x| < \delta) \Rightarrow (|f(y) - f(x)| < \epsilon)]\Big)$$
$$\wedge \ \Big(\forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [(|y-x| < \delta) \Rightarrow (|f(y) - f(x)| < \epsilon)] \Rightarrow \text{is\_continuous}(f)\Big)\Big]$$

[Eliminate $\Rightarrow$]

$$\equiv \quad \forall f \Big[\Big(\neg\text{is\_continuous}(f) \vee \forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [\neg(|y-x| < \delta) \vee (|f(y) - f(x)| < \epsilon)]\Big)$$
$$\wedge \ \Big(\neg\forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [\neg(|y-x| < \delta) \vee (|f(y) - f(x)| < \epsilon)] \vee \text{is\_continuous}(f)\Big)\Big]$$

[Move $\neg$ to the innermost scope]

$$\equiv \quad \forall f \Big[\Big(\neg\text{is\_continuous}(f) \vee \forall x\ \forall \epsilon\ \exists \delta\ \forall y\ [\neg(|y-x| < \delta) \vee (|f(y) - f(x)| < \epsilon)]\Big)$$
$$\wedge \ \Big(\exists x\ \exists \epsilon\ \forall \delta\ \exists y\ [(|y-x| < \delta) \wedge \neg(|f(y) - f(x)| < \epsilon)] \vee \text{is\_continuous}(f)\Big)\Big]$$

Step 2: Skolemization

Named after the Norwegian logician Thoralf Albert Skolem (1887–1963).

If something is true for some value of a variable, give that value a name. That name specifies an *unknown* constant in the domain of the variable. That constant may depend on other variables. The name must not interfere with other names and variables appearing in the formula.

Skolem constant

Consider the wff for the existence of a leaf node in a tree: $\exists u \; \forall v \; \neg\big(\text{parent}(v) = u\big)$

Give that leaf node a name like L.

We have: $\forall v \; \neg\big(\text{parent}(v) = L\big)$

Skolem function

$\forall n \; \exists p \; \big[(p > n) \wedge \text{is\_prime}(p)\big]$

The prime p is a function of n—call it $P(n)$

We have: $\forall n \; \big[(P(n) > n) \wedge \text{is\_prime}(P(n))\big]$

Skolem functions: More examples

- Replace

$$\forall f \left[\neg \text{is\_continuous}(f) \vee \forall x\, \forall \epsilon\, \exists \delta\, \forall y\, \left[\neg(|y-x| < \delta) \vee (|f(y) - f(x)| < \epsilon)\right]\right]$$

by

$$\forall f \left[\neg \text{is\_continuous}(f) \vee \forall x\, \forall \epsilon\, \forall y\, \left[\neg(|y-x| < \Delta(f,x,\epsilon)) \vee (|f(y) - f(x)| < \epsilon)\right]\right]$$

- Replace

$$\forall x\, \exists y \left[\neg P(x,y) \vee \exists u\, \forall v\, \exists w\, Q(u,v,w,x,y)\right]$$

by

$$\forall x \left[\neg P(x, Y(x)) \vee \forall v\, Q\Big(U(x), v, W(x,v), x, Y(x)\Big)\right]$$

Step 3: Prenex normal form

After skolemization, there are no existential quantifiers.

We need to move all the universal quantifiers to the beginning, and let the scope of each quantifier extend to the entire formula.

Just moving the quantifiers to the beginning is wrong:

$$\forall x\, P(x) \lor \forall x\, \forall y\, Q(x, y) \text{ is not equivalent to } \forall x\, \forall y\, [P(x) \lor Q(x, y)].$$

Rename the variables so that each quantifier has its own variable, that is, no two quantifiers have the same variable.

After this, all the quantifiers can be relocated to the front.

Example: Convert

$$\forall x\, P(x) \lor \forall x\, \forall y\, Q(x, y)$$

to

$$\forall x\, \forall y\, \forall z\, [P(x) \lor Q(y, z)].$$

Step 4: Eliminate all the quantifiers

Doing this does not require any special operation.

But we need to keep in mind that the formula is universally valid.

This means that:

- We can rename variables in future without introducing conflicts.
- We can substitute a variable by a constant object.
- We can substitute a variable by a function of constant objects.
- We can also substitute a variable by a function of other variables, provided again that no conflict ensues out of this.

Steps 5 and 6: Convert to CNF, and rename variables

This procedure is exactly the same as in the case of propositional-logic formulas.

After this conversion, the formula is a conjunction of clauses (each clause is a disjunction of literals, each literal is either an atom or its complement).

Since the formula is universally valid, all the clauses are universally valid too.

So variable substitution is each clause is allowed so long as no conflict is introduced.

We rename the variables so that no two clauses share any common variable.

Example:

$$P(x) \vee \Big(Q(x,y) \wedge (R(y,z) \vee \neg S(x))\Big) \equiv \Big(P(x) \vee Q(x,y)\Big) \wedge \Big(P(x) \vee R(y,z) \vee \neg S(x)\Big)$$

The two clauses share the variables x and y. We rename these variables in one of the clauses (say, the second), and get the two clauses

$$P(x) \vee Q(x,y)$$

$$P(x') \vee R(y',z) \vee \neg S(x')$$

Step 7: Resolution

In order to resolve two clauses, we need a predicate appearing in positive form in one clause and in negative form in the other.

But the predicate may have arguments, and the arguments may fail to match in the two instances.

Take $P(a_1, a_2, \ldots, a_k)$ in one clause and $\neg P(b_1, b_2, \ldots, b_k)$ in another, where k is the arity of P.

Unification: Use a set of substitutions to convert the two instances of $P()$ to the same predicate $P(c_1, c_2, \ldots, c_k)$. If this is possible, the two (initial) instances of $P()$ are called **unifiable**.

Only variables can be substituted by terms (that is, constants or other variables or functions of constants and/or other variables).

Substitution fails if the same variable needs to be substituted by multiple expressions.

The same substitutions must be applied in the rest of the clauses.

Unification algorithm

Input: Two instances I and J of the same predicate with (possibly) different arguments

Output: A set S of substitutions, or failure

Steps

```
Initialize $S$ to empty.
Repeat:
    Identify the first token where $I$ and $J$ disagree.
    If no such token is found, return $S$.
    If the disagreement is between a variable $v$ and a term $t$, and if $v$ does not appear in $t$:
        Substitute $v$ by $t$ throughout in $I$ and $J$.
        Append $S$ by $v/t$.
    Otherwise
        Return failure.
```

Unification success: Examples

- $I = P(A, f(B), y), J = P(x, f(B), z)$.
  - x/A $\quad I = P(A, f(B), y), J = (A, f(B), z)$.
  - y/z $\quad I = P(A, f(B), z), J = (A, f(B), z)$.
- $I = P(A, f(z), y), J = P(x, f(B), z)$.
  - x/A $\quad I = P(A, f(z), y), J = (A, f(B), z)$.
  - z/B $\quad I = P(A, f(B), y), J = (A, f(B), B)$.
  - y/B $\quad I = P(A, f(B), B), J = (A, f(B), B)$.
- $I = P(f(x), y, g(x)), J = P(f(y), x, g(z))$.
  - x/y $\quad I = P(f(y), y, g(y)), J = (f(y), y, g(z))$.
  - y/z $\quad I = P(f(z), z, g(z)), J = (f(z), z, g(z))$.
- $I = P(f(x), A, x), J = P(y, z, g(z))$.
  - $y/f(x)$ $\quad I = P(f(x), A, x), J = P(f(x), z, g(z))$.
  - z/A $\quad I = P(f(x), A, x), J = P(f(x), A, g(A))$.
  - $x/g(A)$ $\quad I = P(f(g(A)), A, g(A)), J = P(f(g(A)), A, g(A))$.

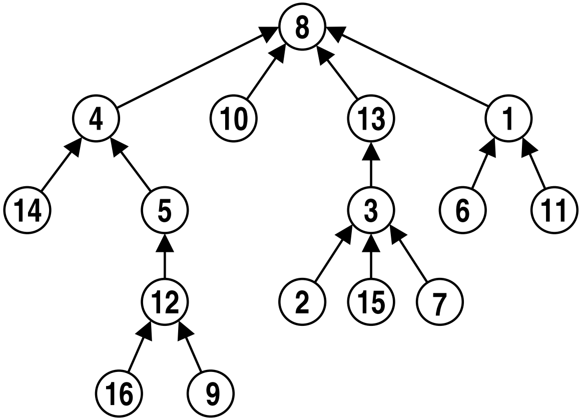
Unification failure: Examples

- $I = P(x, B, f(C)), J = P(A, y, f(x))$.
  - x/A $\quad I = P(A, B, f(C)), J = P(A, y, f(A))$.
  - y/B $\quad I = P(A, B, f(C)), J = P(A, B, f(A))$.
  - Failure $\quad$ Constant values cannot be substituted (unless $f(A) = f(C)$).
- $I = P(x, y, C), J = (A, z, f(z))$.
  - x/A $\quad I = P(A, y, C), J = P(A, z, f(z))$.
  - y/z $\quad I = P(A, z, C), J = P(A, z, f(z))$.
  - Failure $\quad f(z)$ is not a variable.
- $I = P(f(z), z, x), J = P(y, x, g(y))$.
  - $y/f(z)$ $\quad I = P(f(z), z, x), J = P(f(z), x, g(f(z)))$.
  - z/x $\quad I = P(f(x), x, x), J = P(f(x), x, g(f(x)))$.
  - Failure $\quad g(f(x))$ contains x.

Example of predicate-logic proof

Parent function

P(1) = 8
P(2) = 3
P(3) = 13
P(4) = 8
P(5) = 4
P(6) = 1
P(7) = 3
P(8) = 0
P(9) = 12
P(10) = 8
P(11) = 1
P(12) = 5
P(13) = 8
P(14) = 4
P(15) = 3
P(16) = 12



Domain of node variables: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$

Example: Knowledge base

Definition of ancestors:

$$\forall u\ \forall v\ \Big[A(u,v) \Leftrightarrow (u = v) \lor A\big(u, P(v)\big)\Big]$$

This is a recursive definition, and requires some basis conditions:

$$\forall u\ \neg A(u,0)$$

Implicit equality predicates on nodes:

$$(1 = 1), (2 = 2), \neg(1 = 2), \ldots$$

Note that the nodes have symbolic integer names. Integer operations (except equality) do not apply on them. For example, you cannot compare nodes like $u \leqslant v$, or add nodes like $u + v$.

Example: Preprocessing

$$\begin{aligned}
& \forall u\, \forall v\, \Big[A(u,v) \Leftrightarrow (u=v) \vee A\big(u,P(v)\big)\Big] \\
\equiv\; & \forall u\, \forall v\, \bigg[\Big(A(u,v) \Rightarrow (u=v) \vee A\big(u,P(v)\big)\Big) \wedge \Big(\big((u=v) \vee A\big(u,P(v)\big)\big) \Rightarrow A(u,v)\Big)\bigg] \\
\equiv\; & \forall u\, \forall v\, \bigg[\Big(\neg A(u,v) \vee (u=v) \vee A\big(u,P(v)\big)\Big) \wedge \Big(\neg\big((u=v) \vee A\big(u,P(v)\big)\big) \vee A(u,v)\Big)\bigg] \\
\equiv\; & \forall u\, \forall v\, \bigg[\Big(\neg A(u,v) \vee (u=v) \vee A\big(u,P(v)\big)\Big) \wedge \Big(\big(\neg(u=v) \wedge \neg A\big(u,P(v)\big)\big) \vee A(u,v)\Big)\bigg] \\
\equiv\; & \forall u\, \forall v\, \bigg[\Big(\neg A(u,v) \vee (u=v) \vee A\big(u,P(v)\big)\Big) \wedge \Big(\neg(u=v) \vee A(u,v)\Big) \wedge \Big(\neg A\big(u,P(v)\big) \vee A(u,v)\Big)\bigg]
\end{aligned}$$

(C1) $\neg A(u,v) \vee (u=v) \vee A\big(u,P(v)\big)$

(C2) $\neg(w=x) \vee A(w,x)$

(C3) $\neg A\big(y,P(z)\big) \vee A(y,z)$

(C4) $\neg A(t,0)$

(C5) All the equality predicates

Proof 1: $A(4{,}9)$

Add the refutation clause

(C6) $\neg A(4, 9)$

Substitute $(y/4, z/9)$ in (C3) and resolve with (C6) to get

(C7) $\neg A(4, 12)$

Substitute $(y/4, z/12)$ in (C3) and resolve with (C7) to get

(C8) $\neg A(4, 5)$

Substitute $(y/4, z/5)$ in (C3) and resolve with (C8) to get

(C9) $\neg A(4, 4)$

Substitute $(y/4, z/4)$ in (C2) and resolve with (C9) to get

(C10) $\neg(4 = 4)$

Resolve (C10) with the (C5) clause $(4 = 4)$ to get the empty clause (F).

Proof 2: There exist *u* and *v*, neither of which is an ancestor of the other

We want to prove $\exists u\ \exists v\ [\neg A(u,v) \wedge \neg A(v,u)]$.

Using the negation of this in the preprocessing stage gives the clause	$A(r,s) \vee A(s,r)$	(C11)
Substitute $r/u, s/v$ in (C11) and resolve with (C1):	$(u=v) \vee A(u,P(v)) \vee A(v,u)$	(C12)
Substitute (C12) by $u/5, v/7$ and resolve with $\neg(5=7)$:	$A(5,3) \vee A(7,5)$	(C13)
Substitute (C1) by $u/5, v/3$ and resolve with (C13):	$(5=3) \vee A(5,13) \vee A(7,5)$	(C14)
Resolve (C14) with $\neg(5=3)$:	$A(5,13) \vee A(7,5)$	(C15)
Substitute (C1) by $u/5, v/13$ and resolve with (C15):	$(5=13) \vee A(5,8) \vee A(7,5)$	(C16)
Resolve (C16) with $\neg(5=13)$:	$A(5,8) \vee A(7,5)$	(C17)
Substitute (C1) by $u/5, v/8$ and resolve with (C17):	$(5=8) \vee A(5,0) \vee A(7,5)$	(C18)
Resolve (C16) with $\neg(5=8)$:	$A(5,0) \vee A(7,5)$	(C19)
Substitute $t/5$ in (C4) and resolve with (C19):	$A(7,5)$	(C20)
Substitute (C1) by $u/7, v/5$ and resolve with (C20):	$(7=5) \vee A(7,4)$	(C21)
Resolve (C21) with $\neg(7=5)$:	$A(7,4)$	(C22)
Substitute (C1) by $u/7, v/4$ and resolve with (C22):	$(7=4) \vee A(7,8)$	(C23)
Resolve (C23) with $\neg(7=4)$:	$A(7,8)$	(C24)
Substitute (C1) by $u/7, v/8$ and resolve with (C24):	$(7=8) \vee A(7,0)$	(C25)
Resolve (C23) with $\neg(7=8)$:	$A(7,0)$	(C26)
Substitute $t/7$ in (C4) and resolve with (C26):	Empty clause (F)	

Proof 3: Every node's grandparent (if it exists) is an ancestor

We want to prove $\forall u \left[\Big(\neg(P(u)=0) \wedge \neg(P(P(u))=0)\Big) \Rightarrow A\big(P(P(u)),u\big)\right]$.

The negation of this is $\exists u \left[\neg(P(u)=0) \wedge \neg(P(P(u))=0) \wedge \neg A\big(P(P(u)),u\big)\right]$.

Replace u by a Skolem constant α to get the three clauses:

$$\neg\big(P(\alpha)=0\big) \quad \text{(C27)}$$
$$\neg\big(P(P(\alpha))=0\big) \quad \text{(C28)}$$
$$\neg A\big(P(P(\alpha)),\alpha\big) \quad \text{(C29)}$$

Resolve (C29) with (C3) with the substitution $y/P(P(\alpha)), z/\alpha$:

$$\neg A\big(P(P(\alpha)),P(\alpha)\big) \quad \text{(C30)}$$

Resolve (C29) with (C3) with the substitution $y/P(P(\alpha)), z/P(\alpha)$:

$$\neg A\big(P(P(\alpha)),P(P(\alpha))\big) \quad \text{(C31)}$$

Resolve (C31) with (C2) with the substitution $w/P(P(\alpha)), x/P(P(\alpha))$:

$$\neg\big(P(P(\alpha))=P(P(\alpha))\big) \quad \text{(C32)}$$

But $P(P(\alpha))$ is a constant node (whatever it is), so we can use the (C5) clause $P(P(\alpha))=P(P(\alpha))$. Resolving this (C32) leads to contradiction (F).

Common proofs that will fail

The knowledge base is under-specified

Our knowledge base uses the parent function $P()$ as a black box.
It never uses the fact that $P()$ defines a *tree*.
We cannot prove that

$$\forall u\ \forall v\ \left[\Big(A(u,v) \wedge A(v,u)\Big) \Rightarrow (u = v)\right],$$

because the definition of $P()$ does not preclude a cycle.
Even if $P()$ defines a tree, the knowledge base is unable to exploit the tree-ness of $P()$.

Induction is not allowed

Mathematical induction does not follow from the building blocks of predicate logic.
In predicate logic, we cannot prove *obvious* statements like these:

$$\forall u\ \forall v\ \forall w\ \left[\Big(A(u,v) \wedge A(v,w)\Big) \Rightarrow A(u,w)\right]$$

$$\forall u\ \forall v\ \left[\Big(\neg(P(u) = 0) \wedge \neg(P(v) = 0) \wedge A(u,v)\Big) \Rightarrow A\big(P(u), P(v)\big)\right]$$

These statements make sense even in the presence of cycles.

Improving our knowledge base

Instead of working with trees only, let us work on forests f (collections of trees).

Iterated parent function

For all vertex u, and for all non-negative integer n, define

$$P^n(u,f) = \begin{cases} u & \text{if } n = 0, \\ P\left(P^{n-1}(u,f), f\right) & \text{if } n > 0 \text{ and } P^{n-1}(u,f) \neq 0 \\ 0 & \text{if } n > 0 \text{ and } P^{n-1}(u,f) = 0 \end{cases}$$

This function handles the induction business.

No cycles

$$\forall f \ \forall u \ \forall n \ \left[\neg(n = 0) \Rightarrow \neg\left(P^n(u,f) = u\right)\right]$$

Redefine ancestors

$$\forall f \ \forall u \ \forall v \ \left[A(u,v,f) \Leftrightarrow \exists n \ \left[u = P^n(v,f)\right]\right]$$

Preprocessing the new knowledge base

The no-cycles requirement gives:

$$\forall f\ \forall u\ \forall n\ \Big[(n = 0) \vee \neg\big(P^n(u,f) = u\big)\Big]$$

The new definition of ancestors gives:

$$\begin{aligned}
& \forall f\ \forall u\ \forall v\ \Big[A(u,v,f) \Leftrightarrow \exists n\ \big[u = P^n(v,f)\big]\Big] \\
\equiv\ & \forall f\ \forall u\ \forall v\ \Big[\Big(A(u,v,f) \Rightarrow \exists n\ \big[u = P^n(v,f)\big]\Big) \wedge \Big(\exists n\ \big[u = P^n(v,f)\big] \Rightarrow A(u,v,f)\Big)\Big] \\
\equiv\ & \forall f\ \forall u\ \forall v\ \Big[\Big(\neg A(u,v,f) \vee \exists n\ \big[u = P^n(v,f)\big]\Big) \wedge \Big(\forall n\ \neg\big(u = P^n(v,f)\big) \vee A(u,v,f)\Big)\Big]
\end{aligned}$$

This gives the following clauses.

$$\begin{aligned}
(n = 0) \vee \neg\big(u = P^n(u,f)\big) \quad & \text{(D1)} \\
\neg A(v,w,g) \vee \big(v = P^{\nu(v,w,g)}(w,g)\big) \quad & \text{(D2)} \\
\neg\big(x = P^m(y,h)\big) \vee A(x,y,h) \quad & \text{(D3)}
\end{aligned}$$

Proof 4: Transitivity of ancestors

We want to prove

$$\forall f \; \forall u \; \forall v \; \forall w \; \Big[\Big(A(u,v,f) \wedge A(v,w,f)\Big) \Rightarrow A(u,w,f)\Big].$$

Negating this gives the following clauses.

$$A(\alpha,\beta,\phi) \quad \text{(D4)}$$

$$A(\beta,\gamma,\phi) \quad \text{(D5)}$$

$$\neg A(\alpha,\gamma,\phi) \quad \text{(D6)}$$

Resolve (D4) with (D2): $\alpha = P^{\nu(\alpha,\beta,\phi)}(\beta,\phi)$ (D7)

Resolve (D5) with (D2): $\beta = P^{\nu(\beta,\gamma,\phi)}(\gamma,\phi)$ (D8)

Combining (D7) and (D8) by *common sense* yields $\alpha = P^{\nu(\alpha,\beta,\phi)+\nu(\beta,\gamma,\phi)}(\gamma,\phi)$ (D9)

Resolve (D3) and (D9): $A(\alpha,\gamma,\phi)$ (D10)

Resolve (D10) and (D6) to get contradiction.

Proof 5: Antisymmetry of ancestors

To prove: If u is an ancestor of v, and v is an ancestor pf u, then $u = v$.

$$\forall f \; \forall u \; \forall v \left[\Big(A(u,v,f) \wedge A(v,u,f) \Big) \Rightarrow (u = v) \right]$$

Negating this gives the three clauses

$$A(\alpha, \beta, \phi) \quad \text{(D11)}$$
$$A(\beta, \alpha, \phi) \quad \text{(D12)}$$
$$\neg(\alpha = \beta) \quad \text{(D13)}$$

Proceeding as in Proof 4 (using some *common sense*), we can derive

$$\nu(\alpha, \beta, \phi) + \nu(\beta, \alpha, \phi) = 0 \quad \text{(D14)}$$

Apply more *common sense* to conclude $\nu(\alpha, \beta, \phi) = 0$ (D15)

and finally (form the definition of the iterated parent function) $\alpha = \beta$ (D16)

Resolve with (D13).

Proof 6: If *u* is an ancestor of *v*, then *P*(*u*) is an ancestor of *P*(*v*)

We need to prove $\forall f\ \forall u\ \forall v\ \Big[\Big(\neg(P(u)=0) \wedge \neg(P(v)=0) \wedge A(u,v,f)\Big) \Rightarrow A\big(P(u),P(v),f\big)\Big]$.

Negate this to get $\exists f\ \exists u\ \exists v\ \Big[\neg(P(u)=0) \wedge \neg(P(v)=0) \wedge A(u,v,f) \wedge \neg A\big(P(u),P(v),f\big)\Big]$.

Skolemization gives $\neg(P(\alpha)=0) \wedge \neg(P(\beta)=0) \wedge A(\alpha,\beta,\phi) \wedge \neg A\big(P(\alpha),P(\beta),\phi\big)$ for constants α,β,ϕ. This gives four clauses:

$$\neg\big(P(\alpha)=0\big) \quad \text{(D17)}$$
$$\neg\big(P(\beta)=0\big) \quad \text{(D18)}$$
$$A(\alpha,\beta,\phi) \quad \text{(D19)}$$
$$\neg A\big(P(\alpha),P(\beta),\phi\big) \quad \text{(D20)}$$

Resolve (D2) with (D19):
$$\alpha = P^{\nu(\alpha,\beta,\phi)}(\beta,\phi) \quad \text{(D21)}$$

Resolve (D3) with (D20):
$$\neg\Big(P(\alpha) = P^m(P(\beta),\phi)\Big) \quad \text{(D22)}$$

(D21) and (D22) cannot be unified by substitution. Once again, *common sense* prevails. We can apply P to (D21) to get $P(\alpha) = P\Big(P^{\nu(\alpha,\beta,\phi)}(\beta,\phi)\Big) = P^{\nu(\alpha,\beta,\phi)+1}(\beta,\phi) = P^{\nu(\alpha,\beta,\phi)}(P(\beta),\phi)$. This gives

$$P(\alpha) = P^{\nu(\alpha,\beta,\phi)}(P(\beta),\phi) \quad \text{(D23)}$$

Now, (D22) and (D23) can be unified by variable substitution $m/\nu(\alpha,\beta,\phi)$ to arrive at the desired contradiction.

Common sense is the most uncommon sense

- We do not use the iterated parent function as an infinite family of functions $P^0, P^1, P^2, \ldots$. Instead we introduce $Q(n, u, f)$ to stand for $P^n(u, f)$.
- In order to use the standard composition property, we may introduce the rule $\forall f\ \forall u\ \forall m\ \forall n\ \big[Q(n, Q(m, u, f)) = Q(m + n, u, f)\big]$.
- In a factored form, this rule can be restated as $\forall f\ \forall u\ \forall v\ \forall w\ \forall m\ \forall n\ \Big[\Big((v = Q(m, u, f)) \wedge (w = Q(n, v, f))\Big) \Rightarrow (w = Q(m+n, u, f))\Big]$.
- These rules never teach the knowledge base to apply P or P^n on an equality. We can introduce the additional rule $\forall f\ \forall u\ \forall v\ \forall n\ \Big[(u = v) \Rightarrow Q(n, u, f) = Q(n, v, f)\Big]$.
- However, in order to conclude $m = 0$ from $m + n = 0$, we need some equation-solving rules.
- Where do we stop? New proofs may demand other common-sense rules. It would be impractical, if possible at all, to embed all common sense in every knowledge base. Moreover, adding too many obvious rules slows down the resolution-refutation process.
- For example, nothing has so far prepared us to handle replacing equal arguments in a parameter list. That is, even if $R(u)$ and $(u = v)$ are proved, $R(v)$ does not follow.
- Some practical alternatives are available in the literature. We will not study these here.