University of
Central Lancashire
UCLan

CO2412
Computational Thinking

Resolution (tutorial 4.5)
Knowledge graph (tutorial 4.6)
Predicate logic
Quantifiers and first-order logic

Where opportunity creates success

Resolution:
Tutorial 4.5 problem

CO2412

15th March 2022

4.5.1 Concepts

Literals: Atomic statements p, q, …, and their negations $\neg p$, $\neg q$, …

Clauses: A conjunction (“and”) of literals, such as $p \land \neg q \land \neg r$, is a conjunctive clause. A disjunction (“or”) of literals, such as $\neg p \lor q \lor r$, is a disjunctive clause.

Conjunctive normal form (CNF):

- A statement is in CNF if it is a **conjunction of disjunctive clauses**.
- It is in **full CNF** if all atomic statements appear in all disjunctive clauses.
- The full CNF version of a truth table has one clause per **False** valuation.

Entailment: R entails S if and only if every model of R is a model of S. ($R \vDash S$.)

Inference: Deduction of an entailment following a rule or a system of rules.

Resolution: Inference technique applied to CNF statements based on the rule

$$(p \lor L_0 \lor L_1 \lor \ldots) \land (\neg p \lor M_0 \lor M_1 \lor \ldots) \vDash (L_0 \lor L_1 \lor \ldots \lor M_0 \lor M_1 \lor \ldots).$$

4.5.1 Resolution (completeness for satisfiability)

Completeness of resolution:

- If a statement in CNF is a **contradiction**, an algorithm implementing resolution as an inference method **succeeds at proving this in all cases;** *i.e.*, two clauses p_i and $\neg p_i$ for the same atomic statement are deduced.
- The same applies to proving that multiple statements are **inconsistent**.
- If resolution does not detect a contradiction, the statement is **satisfiable**.
- To check whether R is a **tautology**, resolution can be applied to $\neg R$.

Entailment: R entails S if and only if every model of R is a model of S. ($R \vDash S$.)

Inference: Deduction of an entailment following a rule or a system of rules.

Resolution: Inference technique applied to CNF statements based on the rule

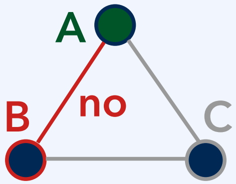
$(p \lor L_0 \lor L_1 \lor \ldots) \land (\neg p \lor M_0 \lor M_1 \lor \ldots) \vDash (L_0 \lor L_1 \lor \ldots \lor M_0 \lor M_1 \lor \ldots)$.

4.5.2 From logic to graphs

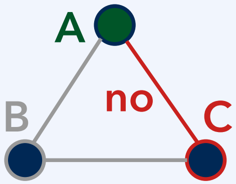
Undirected graph; p_{AB} representing “there is an edge between A and B,” *etc.*

How would we paraphrase the meaning of the propositional logic statements:

$S_{\mathrm{A}} = \neg p_{\mathrm{AB}} \vee \neg p_{\mathrm{AC}}$?



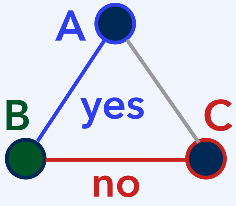
or



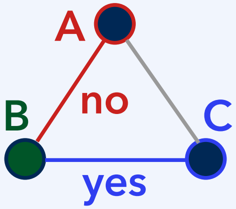
“Vertex A does not have degree 2.”

“Vertex A has degree 0 or 1.”

$S_{\mathrm{B}} = p_{\mathrm{AB}} \leftrightarrow \neg p_{\mathrm{BC}}$?

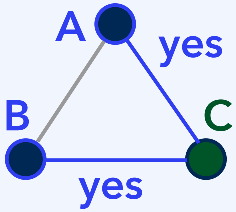


or

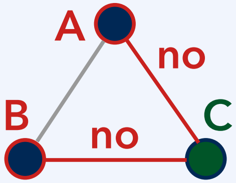


“Vertex B has degree 1.”

$S_{\mathrm{C}} = p_{\mathrm{AC}} \leftrightarrow p_{\mathrm{BC}}$?



or



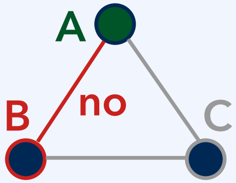
“Vertex C has degree 0 or 2.”

How many literals are there? Six: p_{AB}, $\neg p_{\mathrm{AB}}$, p_{AC}, $\neg p_{\mathrm{AC}}$, p_{BC}, and $\neg p_{\mathrm{BC}}$.

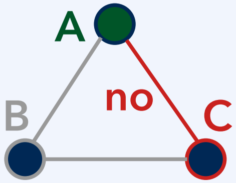
4.5.3 Conjunctive normal form

Transformation to CNF. Rule: $(R \leftrightarrow S) \equiv (R \lor \neg S) \land (\neg R \lor S)$.

$S_A = \neg p_{AB} \lor \neg p_{AC}$

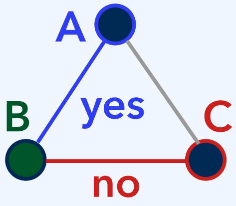


or

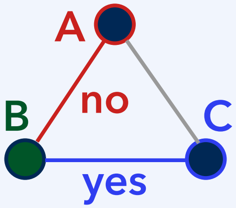


(already in CNF)

$S_B = p_{AB} \leftrightarrow \neg p_{BC}$

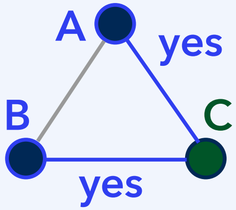


or

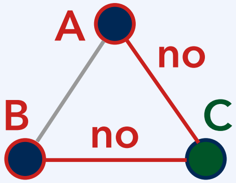


$$
\begin{aligned}
& p_{AB} \leftrightarrow \neg p_{BC} \\
& \quad \equiv (p_{AB} \lor \neg\neg p_{BC}) \land (\neg p_{AB} \lor \neg p_{BC}) \\
& \quad \equiv (p_{AB} \lor p_{BC}) \land (\neg p_{AB} \lor \neg p_{BC})
\end{aligned}
$$

$S_C = p_{AC} \leftrightarrow p_{BC}$



or



$$
\begin{aligned}
& p_{AC} \leftrightarrow \neg p_{BC} \\
& \quad \equiv (p_{AC} \lor \neg p_{BC}) \land (\neg p_{AC} \lor p_{BC})
\end{aligned}
$$

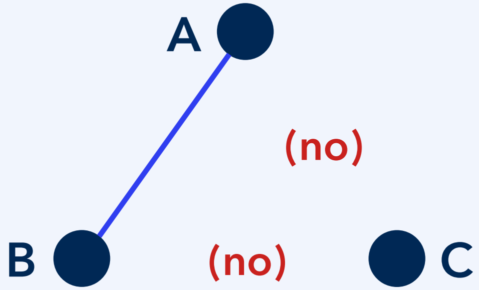
Clauses: 0) $\neg p_{AB} \lor \neg p_{AC}$ 1) $p_{AB} \lor p_{BC}$ 2) $\neg p_{AB} \lor \neg p_{BC}$ 3) $p_{AC} \lor \neg p_{BC}$ 4) $\neg p_{AC} \lor p_{BC}$

4.5.4 Consistency: Common model

If multiple statements are consistent, they have a common model.

$S_A = \neg p_{AB} \lor \neg p_{AC}$

$(\neg p_{AB} \lor \neg p_{AC})$

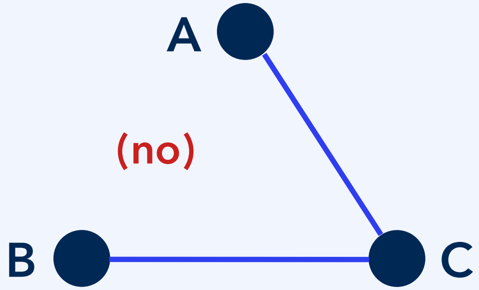


"Vertex A has degree 0 or 1."

$S_B = p_{AB} \leftrightarrow \neg p_{BC}$

$(p_{AB} \lor p_{BC}) \land (\neg p_{AB} \lor \neg p_{BC})$

"Vertex B has degree 1."



$S_C = p_{AC} \leftrightarrow p_{BC}$

$(p_{AC} \lor \neg p_{BC}) \land (\neg p_{AC} \lor p_{BC})$

"Vertex C has degree 0 or 2."

Clauses: 0) $\neg p_{AB} \lor \neg p_{AC}$ 1) $p_{AB} \lor p_{BC}$ 2) $\neg p_{AB} \lor \neg p_{BC}$ 3) $p_{AC} \lor \neg p_{BC}$ 4) $\neg p_{AC} \lor p_{BC}$

4.5.4 Consistency: Resolution

$$\boxed{(p \lor L_0 \lor L_1 \lor \ldots) \land (\neg p \lor M_0 \lor M_1 \lor \ldots) \models (L_0 \lor L_1 \lor \ldots \lor M_0 \lor M_1 \lor \ldots).}$$

0) $\neg p_{AB} \lor \neg p_{AC}$ with 1) $p_{AB} \lor p_{BC}$ | resolves to 4) $\neg p_{AC} \lor p_{BC}$

1) $p_{AB} \lor p_{BC}$ with 2) $\neg p_{AB} \lor \neg p_{BC}$ | resolves to $p_{AB} \lor \neg p_{AB}$ and $p_{BC} \lor \neg p_{BC}$

```
i = 1
while i < len(clauses):
    for j in range(i):
        if direct_contradiction(clauses[i], clauses[j]):
            return False
        resolved_clauses = resolve(clauses[i], clauses[j])
        for c in resolved_clauses:
            if should_be_appended(c, clauses):
                clauses.append(c)
    i += 1
return True
```

contradiction found if the two clauses are single opposite literals, such as p_2 and $\neg p_2$

append if the resolved clause is not redundant and not tautological

Clauses: 0) $\neg p_{AB} \lor \neg p_{AC}$ 1) $p_{AB} \lor p_{BC}$ 2) $\neg p_{AB} \lor \neg p_{BC}$ 3) $p_{AC} \lor \neg p_{BC}$ 4) $\neg p_{AC} \lor p_{BC}$

4.5.4 Consistency: Resolution

$$\boxed{(p \lor L_0 \lor L_1 \lor \ldots) \land (\neg p \lor M_0 \lor M_1 \lor \ldots) \models (L_0 \lor L_1 \lor \ldots \lor M_0 \lor M_1 \lor \ldots).}$$

0) $\neg p_{AB} \lor \neg p_{AC}$ with 1) $p_{AB} \lor p_{BC}$ resolves to 4) $\neg p_{AC} \lor p_{BC}$

1) $p_{AB} \lor p_{BC}$ with 2) $\neg p_{AB} \lor \neg p_{BC}$ resolves to $p_{AB} \lor \neg p_{AB}$ and $p_{BC} \lor \neg p_{BC}$

0) $\neg p_{AB} \lor \neg p_{AC}$ with 3) $p_{AC} \lor \neg p_{BC}$ resolves to 2) $\neg p_{AB} \lor \neg p_{BC}$

1) $\boldsymbol{p_{AB} \lor p_{BC}}$ with 3) $\boldsymbol{p_{AC} \lor \neg p_{BC}}$ resolves to 5) $\boldsymbol{p_{AB} \lor p_{AC}}$

2) $\neg p_{AB} \lor \neg p_{BC}$ with 4) $\neg p_{AC} \lor p_{BC}$ resolves to 0) $\neg p_{AB} \lor \neg p_{AC}$

3) $p_{AC} \lor \neg p_{BC}$ with 4) $\neg p_{AC} \lor p_{BC}$ resolves to $p_{AC} \lor \neg p_{AC}$ and $p_{BC} \lor \neg p_{BC}$

0) $\neg p_{AB} \lor \neg p_{AC}$ with 5) $p_{AB} \lor p_{AC}$ resolves to $p_{AB} \lor \neg p_{AB}$ and $p_{AC} \lor \neg p_{AC}$

2) $\neg p_{AB} \lor \neg p_{BC}$ with 5) $p_{AB} \lor p_{AC}$ resolves to 3) $p_{AC} \lor \neg p_{BC}$

4) $\neg p_{AC} \lor p_{BC}$ with 5) $p_{AB} \lor p_{AC}$ resolves to 1) $p_{AB} \lor p_{BC}$

Clauses: 0) $\neg p_{AB} \lor \neg p_{AC}$ 1) $p_{AB} \lor p_{BC}$ 2) $\neg p_{AB} \lor \neg p_{BC}$ 3) $p_{AC} \lor \neg p_{BC}$ 4) $\neg p_{AC} \lor p_{BC}$

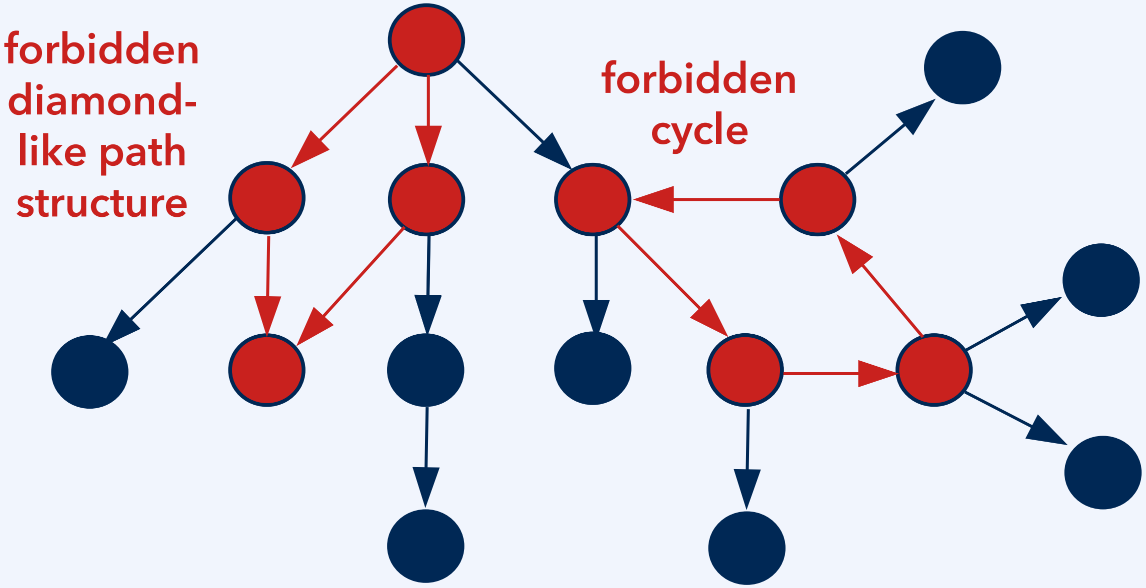
Knowledge graph: Tutorial 4.6 problem

CO2412

15th March 2022

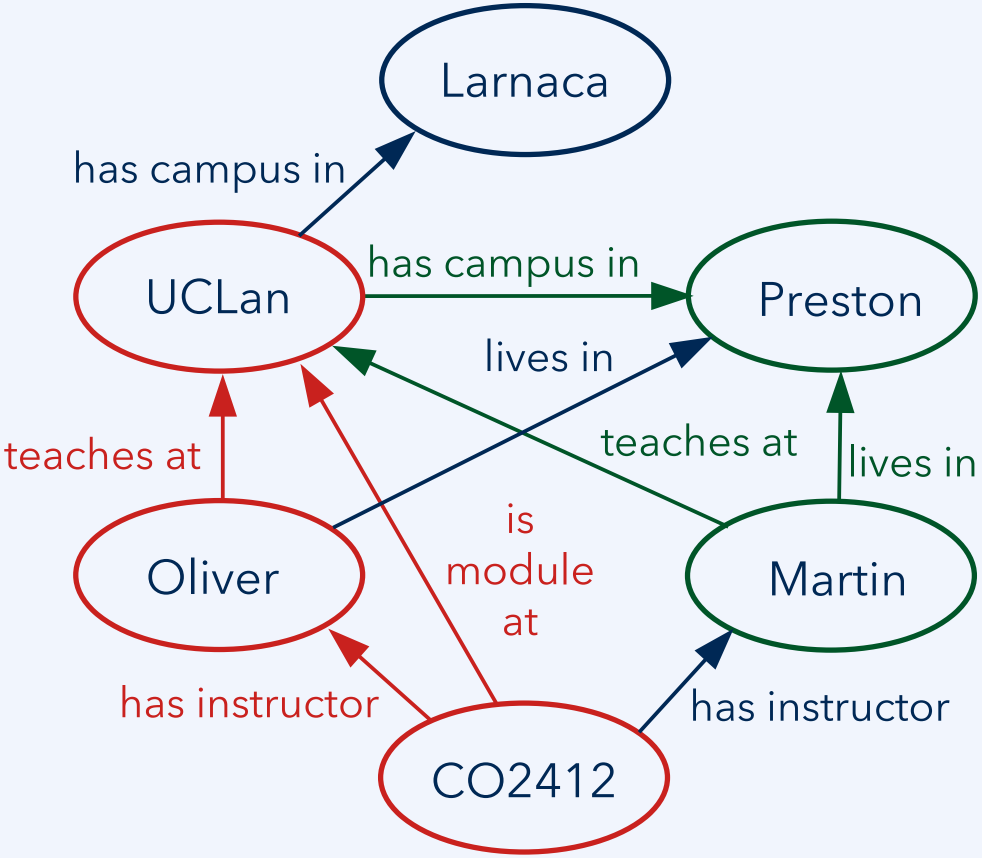
4.6.1 Paths and cycles in graphs

- A tree is **hierarchical**. Every link goes from one node to a subordinate node. There is only one path from the root to any of the nodes in the tree. No link goes back up again. In particular, there are **no cycles**.
- There is exactly one node, the **root** node, from which all nodes can be reached. Each node is either a **leaf**, or it is a root for its own subtree.



4.6.1 Paths and cycles in graphs

Example knowledge graph (see Jupyter notebook **knowledge-graph**).



How many cycles are there?

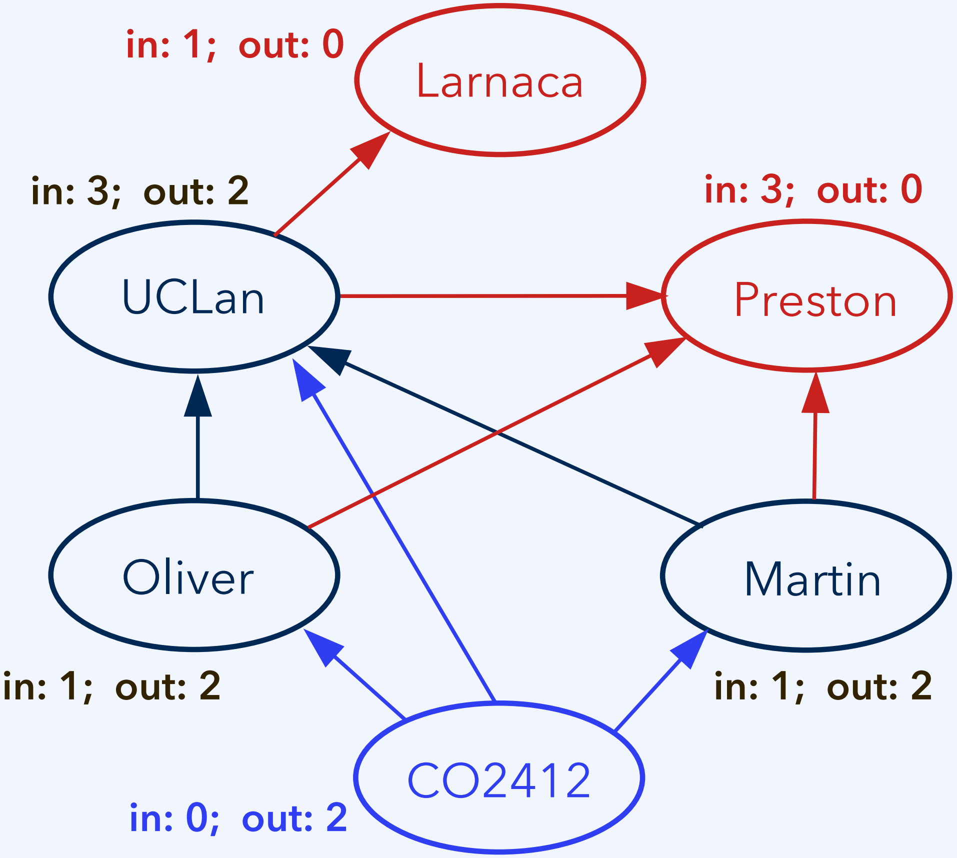
There are no cycles.

Why is the example knowledge graph not a tree?

There are multiple diamond-like structures (multiple paths from one node to another).

4.6.1 Paths and cycles in graphs

What are the indegrees and outdegrees of the nodes? (See below.)

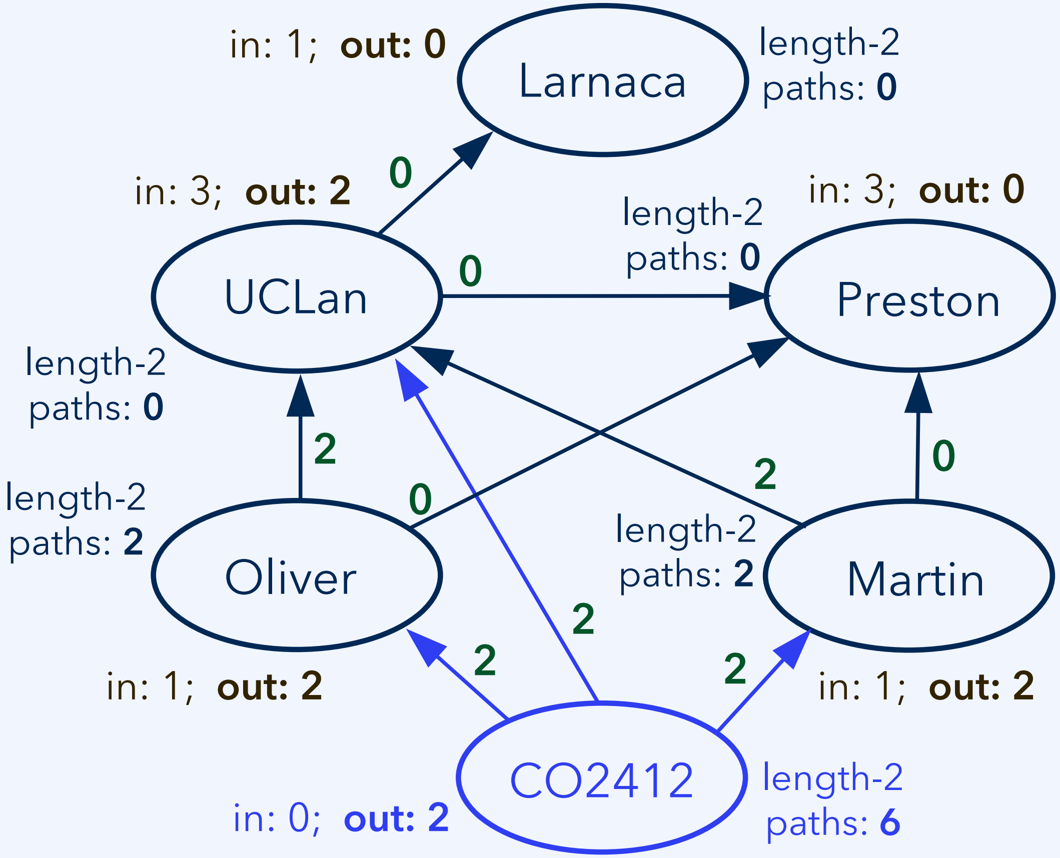


How many length-5 paths?

There are no cycles.
(And no paths with length 5.)

4.6.1 Paths and cycles in graphs

What are the indegrees and outdegrees of the nodes? (See below.)



How many length-5 paths?

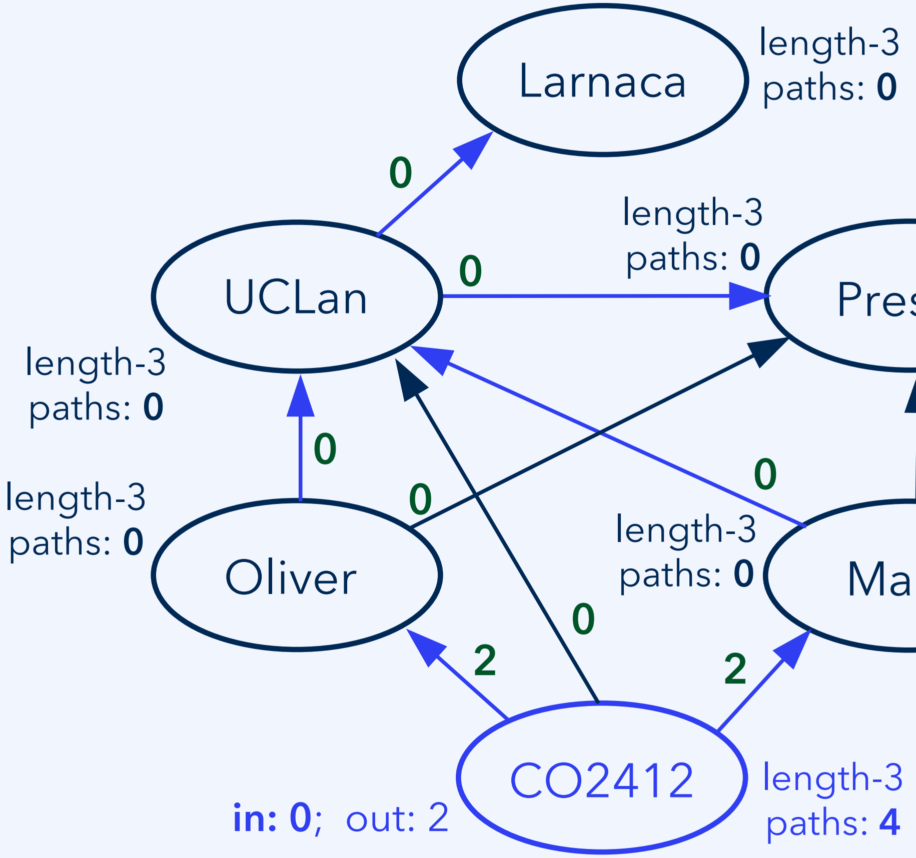
There are no cycles.
(And no paths with length 5.)

Paths of length 2 from each vertex:

Sum of the outdegrees of all successor vertices.

4.6.1 Paths and cycles in graphs

Algorithm for computing the no. of paths with length *m* (consisting of *m* edges):



<u>**Initialization**</u>

```
for v in vertices:
    v.count = 1
```

time?

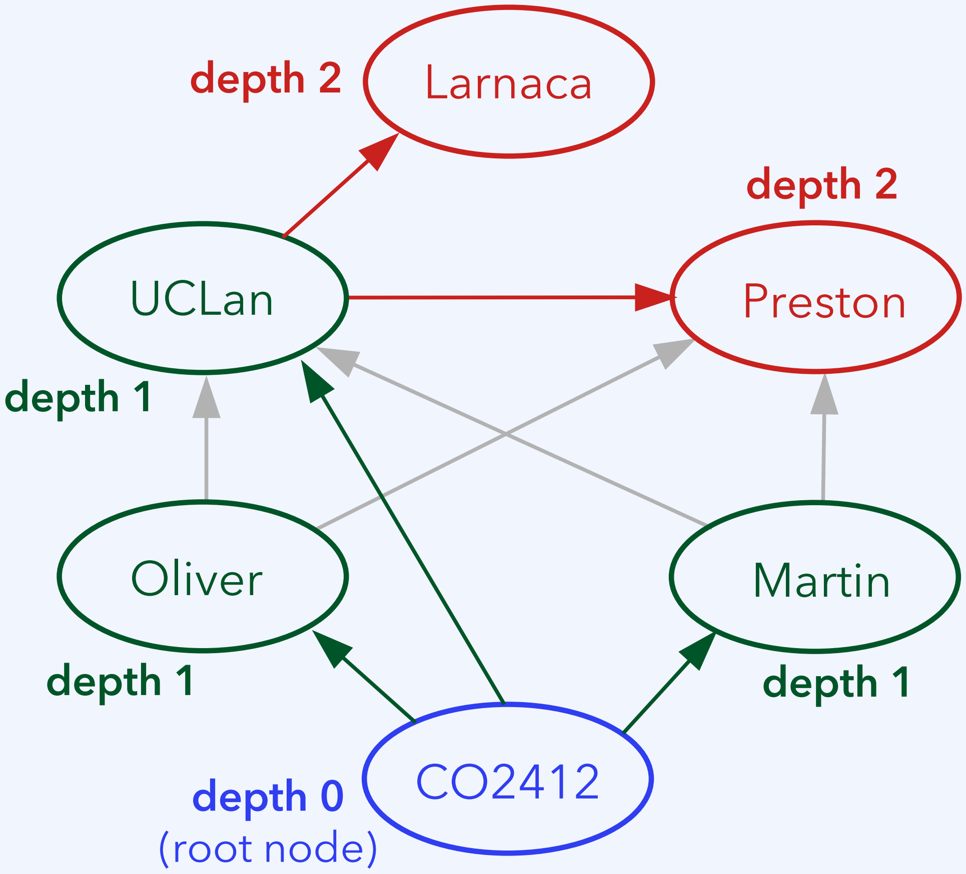
space?

<u>**Iteration**</u>

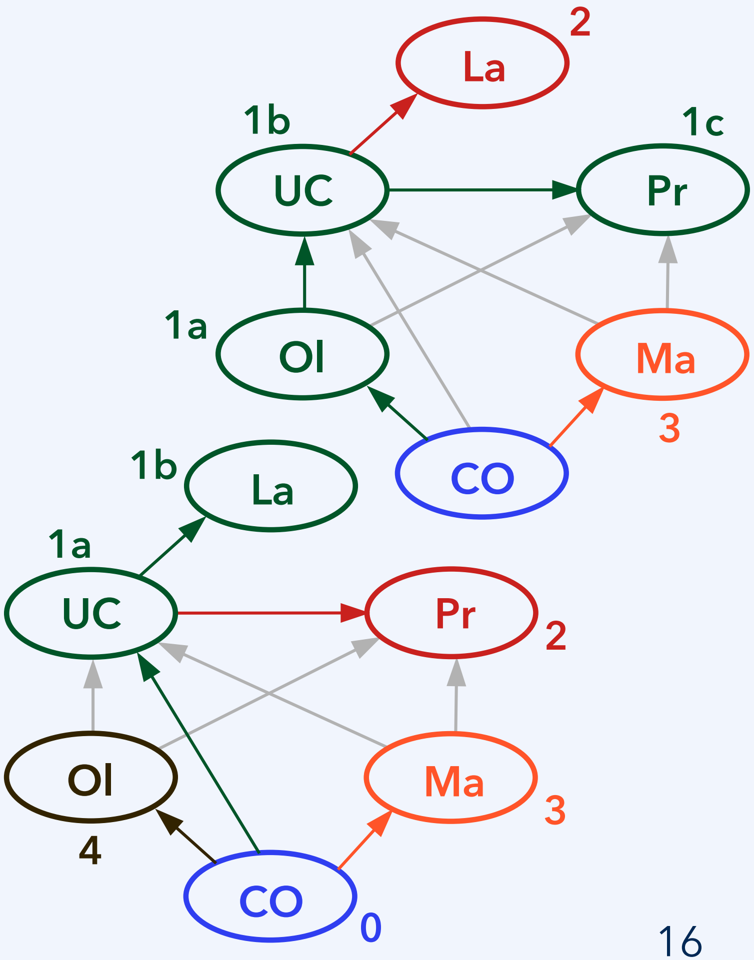
```
for i in range(m):
    for v in vertices:
        v.prev = v.count
        v.count = 0
    for e in edges:
        e.source.count += e.target.prev
```

4.6.2 Spanning trees

breadth-first tree (one out of three)

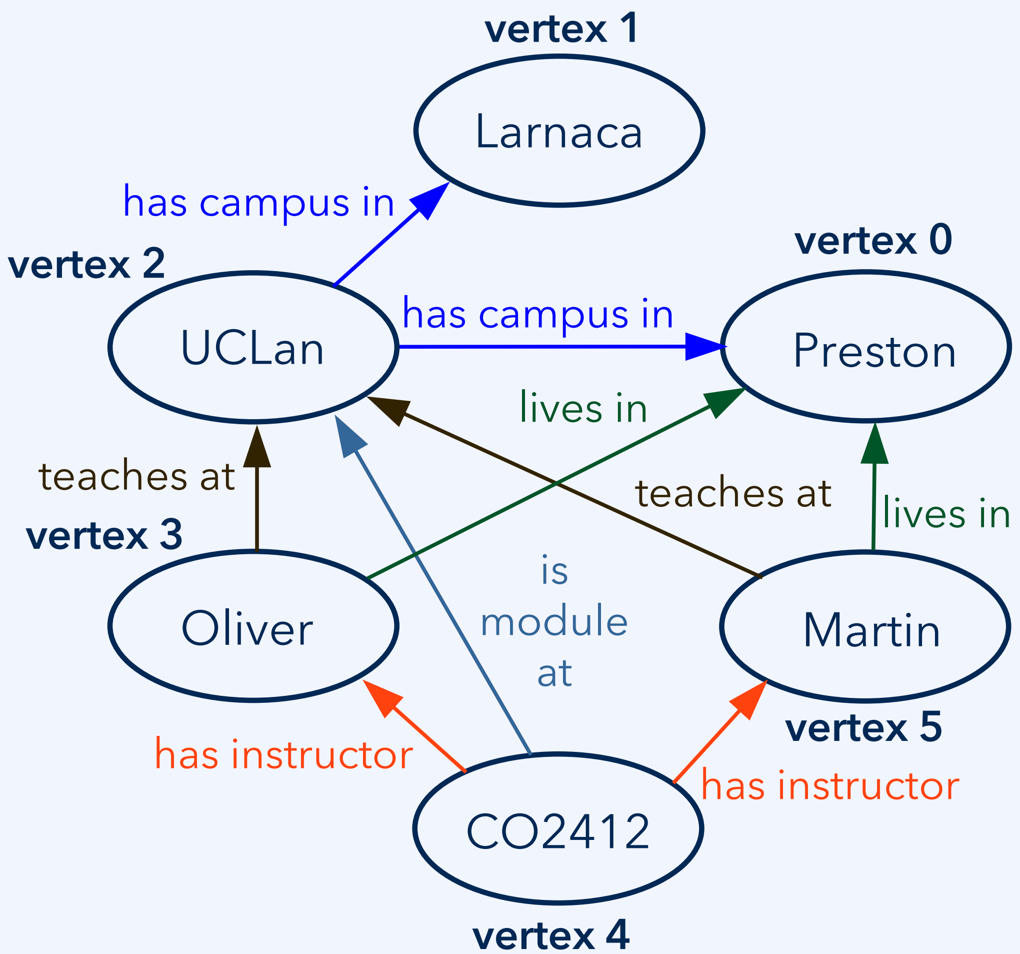


depth-first trees (examples)



4.6.3 Knowledge graphs and predicates

Introduction of quantifiers (“exists”, “for all”):



Are there any nodes x, y
for which the logical expression
edge(x, y) ∧ label(y, "Larnaca")
becomes True?

Yes!
A node **x exists**, and a node **y exists**,
such that this is True:
x = vertex 2; *y* = vertex 1.

Are there any nodes x, y
for which the logical expression
has_campus_in(x, y) → label(x, "UCLan")
becomes False?

No!
The expression is True **for all x and y**.

Predicate logic

CO2412

15th March 2022

Predicate logic

Statements in propositional logic are constructed from a finite number of atomic statements in combination with the five logical operators for conjunction, disjunction, negation, implication, and equivalence.

To extend this to predicate logic, we replace atomic statements by **predicates** in combination with **variables** (x, y, …) and **values** that can be used as arguments of the predicates. Predicates are functions with boolean return values.

Example (Erciyes): Define $P(x, y, z)$ by the truth criterion $x + y = z^2$.

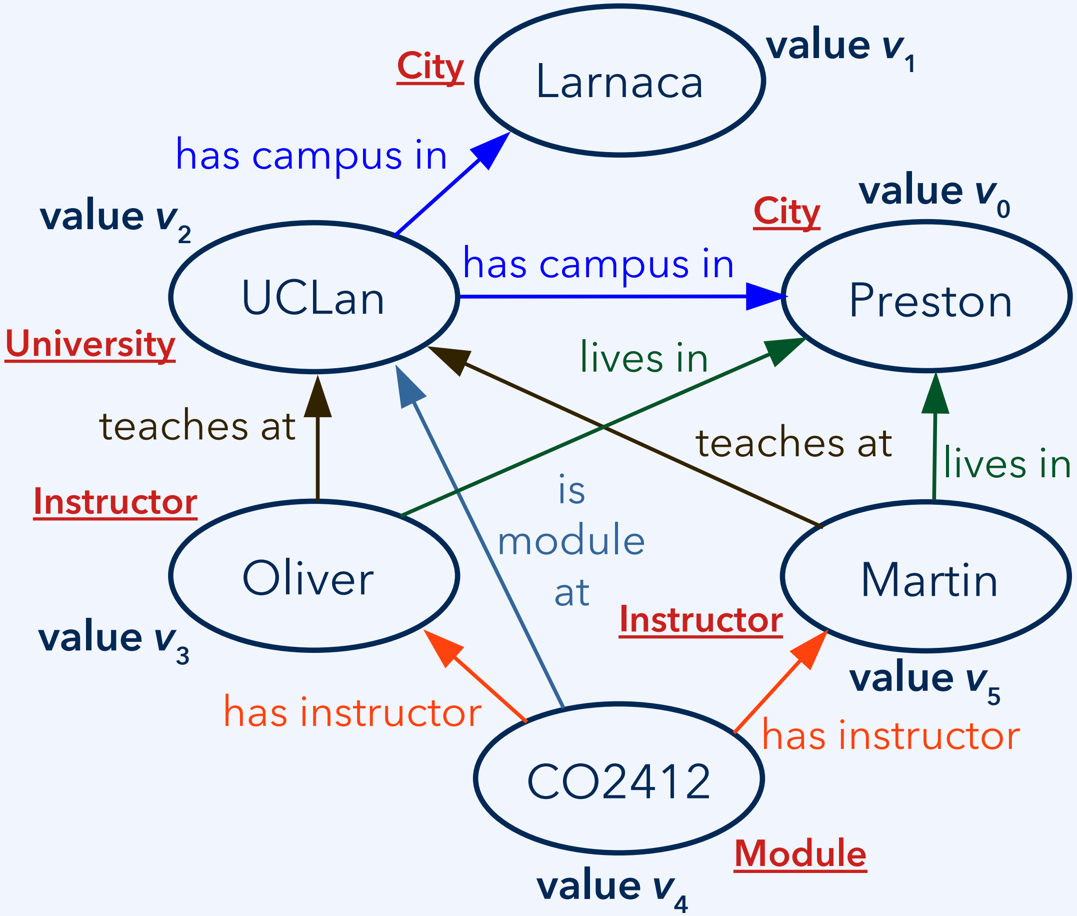
$P(2, 7, 3)$	evaluates to True;
$P(5, 11, 4)$	evaluates to True;
$P(3, 2, 4)$	evaluates to False.

$P(5, 11, 4) \rightarrow P(3, 2, 4)$
evalutes to False.

There, P is a **ternary predicate** (three arguments).

Unary and binary predicates

"Model" or knowledge graph *K*



Knowledge graphs are best suitable to visualize unary predicates (one argument) and binary predicates (two arguments).

Binary predicates, visualized as edges, represent relations between two objects.

teaches_at(v_3, v_2) ∧ ¬teaches_at(v_2, v_3)

Unary predicates can represent properties, types, or similar features of single objects.

Module(v_4) ∧ label(v_4, "CO2412")

Models of predicate logic statements

“Model” or knowledge graph *K*

K satisfies the predicate logic **statements**:

S_0 = teaches_at(v_3, v_2) ∧ ¬teaches_at(v_2, v_3)

S_1 = Module(v_4) ∧ label(v_4, “CO2412”)

We say, “*K* models S_0” or “*K* is a model of S_1”.

Notation: $K \vDash S_0$ and $K \vDash S_1$.

The **expression** has_campus_in(x, y) contains **free variables**: Variables with an unspecified value. Its truth value, even for a given *K*, depends on the values assigned to x and y.

K is **a model for** (*i.e.*, includes) all the predicates and values on the left, but **not a model of** all that can be said about them, e.g., $K \nvDash$ teaches_at(v_2, v_3).

Binary predicates, visualized as edges, represent relations between two objects.

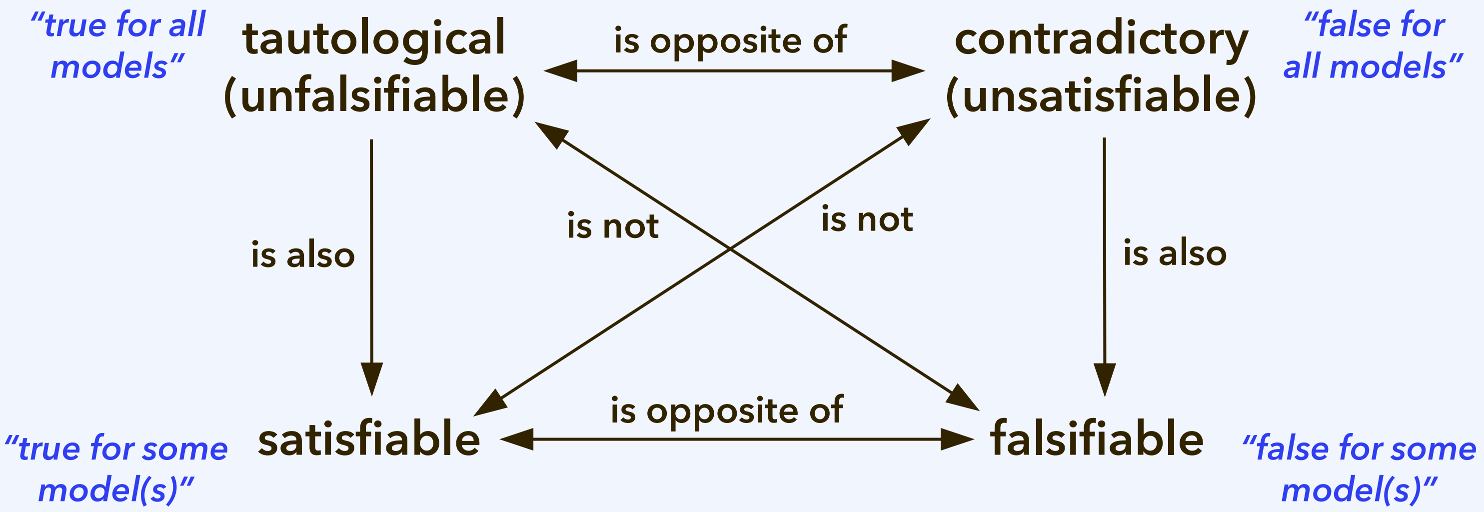
teaches_at(v_3, v_2) ∧ ¬teaches_at(v_2, v_3)

Unary predicates can represent properties, types, or similar features of single objects.

Module(v_4) ∧ label(v_4, “CO2412”)

What predicate logic retains from propositional logic

All observations on propositional logic continue to apply if we replace all the predicate-argument combinations by atomic statements.



Tautologies, contradictions, satisfiability, and falsifiability

$\models p \lor (p \rightarrow q)$

$$\begin{aligned} \models\ & (\text{is\_greater\_than}(v, w) \lor \\ & \quad (\text{is\_greater\_than}(v, w) \rightarrow \text{is\_even}(w)) \end{aligned}$$

What predicate logic retains from propositional logic

All observations on propositional logic continue to apply if we replace all the predicate-argument combinations by atomic statements.

Semantic equivalence and entailment

literals in predicate logic:
predicates and their negations

$p \leftrightarrow \neg q \;\equiv\; (p \lor q) \land (\neg p \lor \neg q)$

literals in propositional logic:
atomic statements and their negations

$\text{identical}(x, y) \leftrightarrow \neg\text{different}(x, y)$

$\equiv\; (\text{identical}(x, y) \lor \text{different}(x, y))$

$\land\; (\neg\text{identical}(x, y) \lor \neg\text{different}(x, y))$

$(\neg p \lor q), (\neg q \lor \neg r) \vDash \neg p \lor \neg r$

a disjunctive
clause

$\neg\text{is\_father\_of}(v, w) \lor \text{Human}(v),$
$\neg\text{Human}(v) \lor \neg\text{Robot}(v)$

a disjunctive
clause

$\vDash\; \neg\text{is\_father\_of}(v, w) \lor \neg\text{Robot}(v)$

Tautologies, contradictions, satisfiability, and falsifiability

$\vDash\; p \lor (p \to q)$

$\vDash\; (\text{is\_greater\_than}(v, w) \;\lor$
$(\text{is\_greater\_than}(v, w) \to \text{is\_even}(w))$

University of Central Lancashire UCLan

Quantifiers and first-order logic

CO2412

15th March 2022

Universal quantifier

The **universal quantifier**, denoted ∀ and read as “for all,” is applied to a variable that occurs as a free variable in a predicate logic expression.

expression with free variables	**statement with bound variables**
has_campus_in(*x*, *y*) → label(*x*, "UCLan")	**∀x ∀y (has_campus_in(*x*, *y*) → label(*x*, "UCLan"))**
“If x has a campus in y, the label of x is «UCLan».”	*“For all possible values of x and y, if x has a campus in y, the label of x is «UCLan».”*
The expression cannot be assigned a truth value based on a model; values for the variables would be required.	*K* models the statement if the expression is True for all potential values in *K* of the bound variables (all values from the **domain**, *e.g.*, all nodes in the knowledge graph).

In first-order predicate logic (usually just called **first-order logic**), the quantifiers “for all” (∀, universal quantifier) and “there is” (∃, existential quantifier) can be applied to variables that occur as arguments of predicates.

Existential quantifier

The **existential quantifier**, denoted ∃ and read “there is” or “there exists,” is applied to a variable that occurs as a free variable in a predicate logic expression.

expression with free variables

edge(*x*, *y*) ∧ label(*y*, "Larnaca")

“There is an edge from x to y and the label of y is «Larnaca».”

The expression cannot be assigned a truth value based on a model; values for the variables would be required.

statement with bound variables

∃*x* ∃*y* (edge(*x*, *y*) ∧ label(*y*, "Larnaca"))

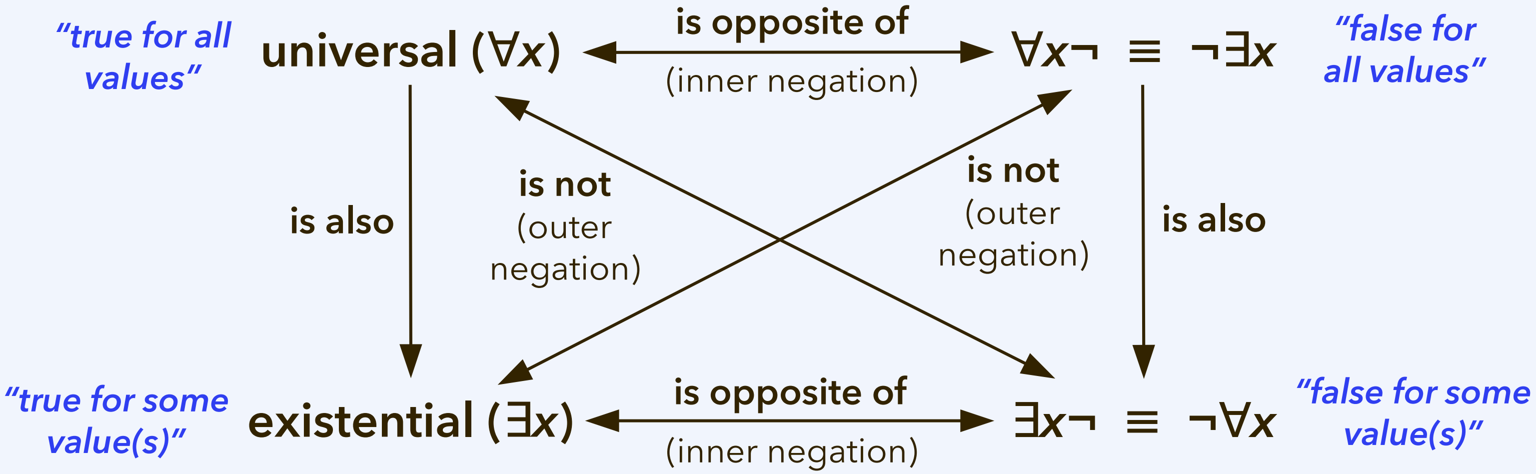
“There are (possible values of) x and y such that there is an edge from x to y and the label of y is «Larnaca».”

K models the statement if the expression is True for at least one potential value in *K* of the bound variables (all values from the **domain**, *e.g.*, all nodes in the knowledge graph).

In first-order predicate logic (usually just called **first-order logic**), the quantifiers “for all” (∀, universal quantifier) and “there is” (∃, existential quantifier) can be applied to variables that occur as arguments of predicates.

Square of opposition for quantifiers

Assume a given logical expression *F*(*x*) contains a free variable *x*. The square of opposition visualizes for modes of binding the variable by using quantifiers:



$\forall x\ F(x)$	$\equiv$	$\neg\exists x\ \neg F(x)$	"For all possible values of *x*, *F*(*x*) holds."
$\forall x\ \neg F(x)$	$\equiv$	$\neg\exists x\ F(x)$	"There is no value of *x* such that *F*(*x*) holds."
$\exists x\ F(x)$	$\equiv$	$\neg\forall x\ \neg F(x)$	"There is a value of *x* for which F(*x*) holds."
$\exists x\ \neg F(x)$	$\equiv$	$\neg\forall x\ F(x)$	"*F*(*x*) does not hold for all possible values of *x*."

De Morgan's laws for quantifiers

Example 4: a) Transform $\forall x\,(P(x) \land \exists y\, Q(x, y)) \rightarrow \neg\forall z\, P(z)$ into a semantically equivalent statement where only the negation, disjunction, and conjunction operators are used; negations should only occur within literals.

b) Simplify the statement as far as possible.

$$\begin{aligned}
\forall x\,(P(x) \land \exists y\, Q(x, y)) \rightarrow \neg\forall z\, P(z) \quad &\equiv \quad \neg\forall x\,(P(x) \land \exists y\, Q(x, y)) \;\lor\; \neg\forall z\, P(z) \\
&\equiv \quad \exists x\, \neg(P(x) \land \exists y\, Q(x, y)) \;\lor\; \exists z\, \neg P(z) \\
&\equiv \quad \exists x\,(\neg P(x) \lor \neg\exists y\, Q(x, y)) \lor \exists z\, \neg P(z) \\
&\equiv \quad \exists x\,(\neg P(x) \lor \forall y\, \neg Q(x, y)) \lor \exists z\, \neg P(z)
\end{aligned}$$

De Morgan's laws

$$\begin{aligned}
\forall x\, F(x) \quad &\equiv \quad \neg\exists x\, \neg F(x) \\
\forall x\, \neg F(x) \quad &\equiv \quad \neg\exists x\, F(x) \\
\exists x\, F(x) \quad &\equiv \quad \neg\forall x\, \neg F(x) \\
\exists x\, \neg F(x) \quad &\equiv \quad \neg\forall x\, F(x)
\end{aligned}$$

$$\begin{aligned}
\neg(R \lor S) \;&\equiv\; \neg R \land \neg S \\
\neg(R \land S) \;&\equiv\; \neg R \lor \neg S
\end{aligned}$$

De Morgan's laws for quantifiers

Example 4: a) Transform $\forall x\,(P(x) \land \exists y\, Q(x, y)) \rightarrow \neg\forall z\, P(z)$ into a semantically equivalent statement where only the negation, disjunction, and conjunction operators are used; negations should only occur within literals.

b) Simplify the statement as far as possible.

$$
\begin{aligned}
\forall x\,(P(x) \land \exists y\, Q(x, y)) \rightarrow \neg\forall z\, P(z) \quad &\equiv \quad \neg\forall x\,(P(x) \land \exists y\, Q(x, y)) \;\lor\; \neg\forall z\, P(z) \\
&\equiv \quad \exists x\,\neg(P(x) \land \exists y\, Q(x, y)) \;\lor\; \exists z\,\neg P(z) \\
&\equiv \quad \exists x\,(\neg P(x) \lor \neg\exists y\, Q(x, y)) \lor \exists z\,\neg P(z) \\
&\equiv \quad \exists x\,(\neg P(x) \lor \forall y\,\neg Q(x, y)) \lor \exists z\,\neg P(z) \\
&\equiv \quad \exists x\,(\neg P(x) \lor \forall y\,\neg Q(x, y)) \lor \exists x\,\neg P(x) \\
&\equiv \quad \exists x\,\neg P(x) \;\lor\; \exists x \forall y\,\neg Q(x, y)
\end{aligned}
$$

De Morgan's laws

$\forall x\, F(x)$	$\equiv$	$\neg\exists x\,\neg F(x)$
$\forall x\,\neg F(x)$	$\equiv$	$\neg\exists x\, F(x)$
$\exists x\, F(x)$	$\equiv$	$\neg\forall x\,\neg F(x)$
$\exists x\,\neg F(x)$	$\equiv$	$\neg\forall x\, F(x)$

University of
Central Lancashire
UCLan

CO2412
Computational Thinking

Resolution (tutorial 4.5)
Knowledge graph (tutorial 4.6)
Predicate logic
Quantifiers and first-order logic

Where opportunity creates success