COMP111: Artificial Intelligence

Section 7. Knowledge Representation and Reasoning (KR&R)

Frank Wolter

Content

- Basic idea behind KR and R.
- Rule-Based KR and R.
- Propositional Logic for KR and R.

Knowledge Representation and Reasoning

- An intelligent agent needs to be able to perform several tasks:
  - Perception: what is my state?
  - Deliberation: what action should I take?
  - Action: how do I execute the action?

Knowledge Representation and Reasoning

- An intelligent agent needs to be able to perform several tasks:
  - Perception: what is my state?
  - Deliberation: what action should I take?
  - Action: how do I execute the action?
- State recognition implies some form of representation of the knowledge about the state.
- Figuring out the right action requires some form of reasoning.

Knowledge Bases and Reasoning Engine

- Basic idea:
  - Tell the agent what it needs to know.
  - The agent uses reasoning to deduce relevant consequences.

Knowledge Bases and Reasoning Engine

- Basic idea:
  - Tell the agent what it needs to know.
  - The agent uses reasoning to deduce relevant consequences.
- This is the declarative approach to building agents. Agents have two parts:
  - A knowledge base which contains facts and general knowledge about a domain in some formal language.
  - A reasoning engine that produces relevant consequences of the knowledge base.

Example 1

Consider the following knowledge base:

- If I have an AI lecture today, then it is Tuesday or Friday.
- It is not Tuesday.
- I have an AI lecture today or I have no class today.
- If I have no class today, then I am sad.
- I am not sad.

Example 1

Consider the following knowledge base:

- If I have an AI lecture today, then it is Tuesday or Friday.
- It is not Tuesday.
- I have an AI lecture today or I have no class today.
- If I have no class today, then I am sad.
- I am not sad.

Can you infer what day it is?

Example 2: Medical Ontologies

Consider the following medical knowledge base:

- Pericardium is a tissue contained in the heart
- Pericarditis is an inflammation located in the pericardium
- Inflammation is a disease that acts on tissue
- A disease located in something contained in the heart is a heartdisease

Example 2: Medical Ontologies

Consider the following medical knowledge base:

- Pericardium is a tissue contained in the heart
- Pericarditis is an inflammation located in the pericardium
- Inflammation is a disease that acts on tissue
- A disease located in something contained in the heart is a heartdisease

Can you infer that pericarditis is a heartdisease?

More on Example 2

Example 2 contains propositions in SNOMED CT, a medical and healthcare knowledge base (called ontology) containing more than 300 000 propositions that define the meaning of terms used in medicine and healthcare in such a way that it can be processed by machines.

More on Example 2

Example 2 contains propositions in SNOMED CT, a medical and healthcare knowledge base (called ontology) containing more than 300 000 propositions that define the meaning of terms used in medicine and healthcare in such a way that it can be processed by machines.

SNOMED CT is used as a knowledge base by the NHS and the healthcare systems of more than 20 countries.

More on Example 2

Example 2 contains propositions in SNOMED CT, a medical and healthcare knowledge base (called ontology) containing more than 300 000 propositions that define the meaning of terms used in medicine and healthcare in such a way that it can be processed by machines.

SNOMED CT is used as a knowledge base by the NHS and the healthcare systems of more than 20 countries.

SNOMED CT is used to process electronic medical records containing test results, medications, and treatments (your GP uses SNOMED CT terms to fill in your electronic medical record).

Example 3: Mathematical Knowledge

Consider a mathematical problem such as:

for which x does $ax^2 + bx + c = 0$ hold?

There are infinitely many equations. It is not possible to store the answers to all these problems in a database or knowledge base.

Example 3: Mathematical Knowledge

Consider a mathematical problem such as:

$$\text{for which } x \text{ does } ax^2 + bx + c = 0 \text{ hold?}$$

There are infinitely many equations. It is not possible to store the answers to all these problems in a database or knowledge base.

Instead:

- Store axioms of arithmetic such as

  $$x + y = y + x, \quad x(y + z) = xy + xz$$

  in a knowledge base.

Example 3: Mathematical Knowledge

Consider a mathematical problem such as:

$$\text{for which } x \text{ does } ax^2 + bx + c = 0 \text{ hold?}$$

There are infinitely many equations. It is not possible to store the answers to all these problems in a database or knowledge base.

Instead:

- Store axioms of arithmetic such as

  $$x + y = y + x, \quad x(y + z) = xy + xz$$

  in a knowledge base.
- Use reasoning algorithms to solve particular problems using the axioms in the knowledge base.

Example 4: Knowledge Graph

When you search for Liverpool you receive structured information about Liverpool FC. In particular:

- Liverpool Football Club is a professional association football club based in Liverpool, Merseyside, England. They compete in the Premier League, the top tier of English football.
- Nickname: The Reds
- Manager: Arne Slot
- Arena/Stadium: Anfield
- Customer service: 0151 264 2500
- Parent organization: Fenway Sports Group

This information is stored in the Google knowledge graph (which is a knowledge base).

Example 4: Reasoning using Knowledge Graph

Many facts are not stored in the knowledge graph. Possibly the following:

- Liverpool FC competes in the FA Cup;
- Arne Slot is employed by Liverpool FC.

Example 4: Reasoning using Knowledge Graph

Many facts are not stored in the knowledge graph. Possibly the following:

- Liverpool FC competes in the FA Cup;
- Arne Slot is employed by Liverpool FC.

It is, however, not unlikely that the knowledge graph contains:

- If a club competes in the Premier League, then it competes in the FA Cup;
- A manager of a football club is employed by the football club.

Example 4: Reasoning using Knowledge Graph

Many facts are not stored in the knowledge graph. Possibly the following:

- Liverpool FC competes in the FA Cup;
- Arne Slot is employed by Liverpool FC.

It is, however, not unlikely that the knowledge graph contains:

- If a club competes in the Premier League, then it competes in the FA Cup;
- A manager of a football club is employed by the football club.

Thus, the facts above can be deduced from the knowledge graph using reasoning.

Languages for KR&R

- To store knowledge in a knowledge base and do reasoning, one has to represent the knowledge in a formal language that can be processed by machines.

Languages for KR&R

- To store knowledge in a knowledge base and do reasoning, one has to represent the knowledge in a formal language that can be processed by machines.
- Many KR&R languages were first developed in logic, a subdiscipline of philosophy and mathematics, and now also of computer science.

Languages for KR&R

- To store knowledge in a knowledge base and do reasoning, one has to represent the knowledge in a formal language that can be processed by machines.
- Many KR&R languages were first developed in logic, a subdiscipline of philosophy and mathematics, and now also of computer science.
- We consider rule-based languages and propositional logic as KR&R languages. Both are important in computer science in general.

Rule-Based Languages

Syntax

- An individual name denotes an individual object. They are often also called constant symbols. Examples:
  - LiverpoolFC;
  - ArneSlot;
  - England;
  - we also often use lower case letters such as a, b, c, a_1, a_2 as individual names.

Syntax

- An individual name denotes an individual object. They are often also called constant symbols. Examples:
  - LiverpoolFC;
  - ArneSlot;
  - England;
  - we also often use lower case letters such as a, b, c, a_1, a_2 as individual names.
- An individual variable is a placeholder for an individual name. They are denoted by lower case letters x, y, z, x_1, and so on.

Syntax

- An individual name denotes an individual object. They are often also called constant symbols. Examples:
  - LiverpoolFC;
  - ArneSlot;
  - England;
  - we also often use lower case letters such as a, b, c, a_1, a_2 as individual names.
- An individual variable is a placeholder for an individual name. They are denoted by lower case letters x, y, z, x_1, and so on.
- A class name denotes a set of individual objects. They are often also called unary predicate symbols. Examples:
  - CompetesInPremierLeague;
  - FootballClub;.
  - Human_being;
  - we also often use upper case letters such as A, B, C, A_1, A_2 as class names.

Syntax: atomic assertion

An atomic assertion takes the form $A(b)$ and states that b is in the class A.

Example:

- The assertion

$$\mathsf{CompetesInPremierLeague}(\mathsf{LiverpoolFC})$$

  states that Liverpool FC competes in the Premier League.

Syntax: atomic assertion

An atomic assertion takes the form $A(b)$ and states that b is in the class A.

Example:

- The assertion

$$\text{CompetesInPremierLeague}(\text{LiverpoolFC})$$

  states that Liverpool FC competes in the Premier League.
- The assertion

$$\text{Manager}(\text{Slot})$$

  states that Slot is a manager.

Syntax: unary rule

A unary rule takes the form

$$A_1(x) \land \cdots \land A_n(x) \to A(x)$$

where $A_1, \ldots, A_n$ and A are class names and x is an individual variable.

The rule states that everything in all classes $A_1, A_2, \ldots, A_n$ is in the class A.

Syntax: unary rule

A unary rule takes the form

$$A_1(x) \wedge \cdots \wedge A_n(x) \rightarrow A(x)$$

where $A_1, \ldots, A_n$ and A are class names and x is an individual variable.

The rule states that everything in all classes $A_1, A_2, \ldots, A_n$ is in the class A.

Example:

- The rule

  $$\text{CompetesInPremierLeague}(x) \rightarrow \text{CompetesInFACup}(x)$$

  states that everything that competes in the Premier League competes in the FA Cup.
- The rule

  $$\text{Disease}(x) \wedge \text{LocatedInHeart}(x) \rightarrow \text{Heartdisease}(x)$$

  states that every disease located in the heart is a heartdisease.

Unary Rule-Based Systems

A unary rule-based knowledge base K is a collection K_a of atomic assertions and K_r of unary rules.

Unary Rule-Based Systems

A unary rule-based knowledge base K is a collection K_a of atomic assertions and K_r of unary rules.

Example: Let K_a contain the following atomic assertions:

- CompetesInPremierLeague(LiverpoolFC);
- CompetesInPremierLeague(Everton) and so on for all clubs competing in the Premier League.

Unary Rule-Based Systems

A unary rule-based knowledge base K is a collection K_a of atomic assertions and K_r of unary rules.

Example: Let K_a contain the following atomic assertions:

- CompetesInPremierLeague(LiverpoolFC);
- CompetesInPremierLeague(Everton) and so on for all clubs competing in the Premier League.

Let K_r contain the following rules:

- CompetesInPremierLeague(x) $\rightarrow$ CompetesInFACup(x);
- CompetesInPremierLeague(x) $\rightarrow$ FootballClub(x);
- CompetesInPremierLeague(x) $\rightarrow$ BasedInEngland(x).

Unary Rule-Based Systems

A unary rule-based knowledge base K is a collection K_a of atomic assertions and K_r of unary rules.

Example: Let K_a contain the following atomic assertions:

- CompetesInPremierLeague(LiverpoolFC);
- CompetesInPremierLeague(Everton) and so on for all clubs competing in the Premier League.

Let K_r contain the following rules:

- CompetesInPremierLeague(x) $\rightarrow$ CompetesInFACup(x);
- CompetesInPremierLeague(x) $\rightarrow$ FootballClub(x);
- CompetesInPremierLeague(x) $\rightarrow$ BasedInEngland(x).

The following atomic assertions follow from K:

- CompetesInFACup(LiverpoolFC)
- FootballClub(Everton), and so on.

Unary Rule-Based Systems

A unary rule-based knowledge base K is a collection K_a of atomic assertions and K_r of unary rules.

Example: Let K_a contain the following atomic assertions:

- CompetesInPremierLeague(LiverpoolFC);
- CompetesInPremierLeague(Everton) and so on for all clubs competing in the Premier League.

Let K_r contain the following rules:

- CompetesInPremierLeague(x) $\rightarrow$ CompetesInFACup(x);
- CompetesInPremierLeague(x) $\rightarrow$ FootballClub(x);
- CompetesInPremierLeague(x) $\rightarrow$ BasedInEngland(x).

The following atomic assertions follow from K:

- CompetesInFACup(LiverpoolFC)
- FootballClub(Everton), and so on.

Thus, from 20 facts and 3 rules, we can deduce 60 additional facts.

Reasoning in Rule-Based Systems

Let K be a knowledge base and $A(b)$ an atomic assertion. Then $A(b)$ follows from K, in symbols,

$$K \models A(b)$$

if whenever K is true, then $A(b)$ is true. We write $K \not\models A(b)$ if this is not the case.

Reasoning in Rule-Based Systems

Let K be a knowledge base and $A(b)$ an atomic assertion. Then $A(b)$ follows from K, in symbols,

$$K \models A(b)$$

if whenever K is true, then $A(b)$ is true. We write $K \not\models A(b)$ if this is not the case.

In the example K from the previous slide, we have

$$K \models \mathrm{CompetesInFACup}(\mathrm{LiverpoolFC})$$

Reasoning in Rule-Based Systems

Let K be a knowledge base and $A(b)$ an atomic assertion. Then $A(b)$ follows from K, in symbols,

$$K \models A(b)$$

if whenever K is true, then $A(b)$ is true. We write $K \not\models A(b)$ if this is not the case.

In the example K from the previous slide, we have

$$K \models \text{CompetesInFACup}(\text{LiverpoolFC})$$

In fact, we have already for

$$K'_a = \{\text{CompetesInPremierLeague}(\text{LiverpoolFC})\}$$

and

$$K'_r = \{\text{CompetesInPremierLeague}(x) \rightarrow \text{CompetesInFACup}(x)\}$$

that for K' containing K'_a and K'_r,

$$K' \models \text{CompetesInFACup}(\text{LiverpoolFC})$$

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \rightarrow A_3(x)\}$.

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \to A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \rightarrow A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;
- $K \models A_2(a)$;

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \to A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;
- $K \models A_2(a)$;
- $K \models A_3(a)$.

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \wedge B(x) \to A_3(x)\}$.

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \wedge B(x) \to A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \wedge B(x) \rightarrow A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;
- $K \models A_2(a)$;

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \wedge B(x) \rightarrow A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;
- $K \models A_2(a)$;
- $K \not\models A_3(a)$;

More examples for $K \models A(b)$

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \wedge B(x) \rightarrow A_3(x)\}$.

Let K contain K_a and K_r. Then

- $K \models A_1(a)$;
- $K \models A_2(a)$;
- $K \not\models A_3(a)$;
- $K \not\models B(a)$.

How do we check that $K \models A(b)$?

The following algorithm takes as input the knowledge base K containing K_a and K_r and computes all assertions $E(a)$ with E a class name and a an individual name such that $K \models E(a)$. This set is stored in

$$DerivedAssertions$$

It only remains to check whether $A(b)$ is in *DerivedAssertions*.

How do we check that $K \models A(b)$?

The following algorithm takes as input the knowledge base K containing K_a and K_r and computes all assertions $E(a)$ with E a class name and a an individual name such that $K \models E(a)$. This set is stored in

$$\mathit{DerivedAssertions}$$

It only remains to check whether $A(b)$ is in $\mathit{DerivedAssertions}$.

The algorithm computes the set $\mathit{DerivedAssertions}$ by starting with K_a and then applying the rules in K_r exhaustively to already derived atomic assertions.

Computing *DerivedAssertions*

Input: a knowledge base K containing
assertions K_a and rules K_r

DerivedAssertions $:= K_a$
repeat
  if there exist $E(a)$ not in *DerivedAssertions*
    and $A_1(x) \wedge \cdots \wedge A_n(x) \rightarrow E(x)$ in K_r
    and $A_1(a), \cdots, A_n(a)$ in *DerivedAssertions*
  then add $E(a)$ to *DerivedAssertions*
    NewAssertion := true
  else NewAssertion := false
  endif
until NewAssertion = false
return *DerivedAssertions*

Rule application

In the algorithm above we say that:

$E(a)$ is added to *DerivedAssertions* by applying the rule

$$A_1(x) \wedge \cdots \wedge A_n(x) \rightarrow E(x)$$

to the atomic assertions

$$A_1(a), \ldots, A_n(a) \text{ in } \mathit{DerivedAssertions}$$

Example

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \to A_3(x)\}$.

Example

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \to A_3(x)\}$.

- First *DerivedAssertions* contains K_a only.

Example

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \rightarrow A_3(x)\}$.

- First *DerivedAssertions* contains K_a only.
- Then an application of $A_1(x) \rightarrow A_2(x)$ to $A_1(a)$ adds $A_2(a)$ to *DerivedAssertions*.

Example

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \rightarrow A_2(x), A_2(x) \rightarrow A_3(x)\}$.

- First *DerivedAssertions* contains K_a only.
- Then an application of $A_1(x) \rightarrow A_2(x)$ to $A_1(a)$ adds $A_2(a)$ to *DerivedAssertions*.
- Then an application of $A_2(x) \rightarrow A_3(x)$ to $A_2(a)$ adds $A_3(a)$ to *DerivedAssertions*.

Example

Let

- $K_a = \{A_1(a)\}$;
- $K_r = \{A_1(x) \to A_2(x), A_2(x) \to A_3(x)\}$.

- First *DerivedAssertions* contains K_a only.
- Then an application of $A_1(x) \to A_2(x)$ to $A_1(a)$ adds $A_2(a)$ to *DerivedAssertions*.
- Then an application of $A_2(x) \to A_3(x)$ to $A_2(a)$ adds $A_3(a)$ to *DerivedAssertions*.
- Now no rule is applicable. Thus

$$\mathit{DerivedAssertions} = \{A_1(a), A_2(a), A_3(a)\}$$

is returned.

Example

Let K_r contain:

- $A_1(x) \rightarrow A_2(x)$
- $A_2(x) \wedge B(x) \rightarrow A_3(x)$

Let K_a contain:

- $A_1(a)$

Example

Let K_r contain:

- $A_1(x) \rightarrow A_2(x)$
- $A_2(x) \wedge B(x) \rightarrow A_3(x)$

Let K_a contain:

- $A_1(a)$

DerivedAssertions

- $A_1(a)$

Example

Let K_r contain:

- $A_1(x) \rightarrow A_2(x)$
- $A_2(x) \wedge B(x) \rightarrow A_3(x)$

Let K_a contain:

- $A_1(a)$

DerivedAssertions

- $A_1(a)$

Example

Let K_r contain:

- $A_1(x) \rightarrow A_2(x)$
- $A_2(x) \wedge B(x) \rightarrow A_3(x)$

Let K_a contain:

- $A_1(a)$

DerivedAssertions

- $A_1(a)$
- $A_2(a)$

Example

Let K_r contain:

- $A_1(x) \rightarrow A_2(x)$
- $A_2(x) \wedge B(x) \rightarrow A_3(x)$

Let K_a contain:

- $A_1(a)$

DerivedAssertions

- $A_1(a)$
- $A_2(a)$

Now no rule is applicable. Thus

$$DerivedAssertions = \{A_1(a), A_2(a)\}$$

is returned.

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$
- $A_4(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$
- $A_4(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$
- $A_4(c)$
- $A_5(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$
- $A_4(c)$
- $A_5(c)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$?
- $K \models A_5(c)$?

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$
- $A_4(c)$
- $A_5(c)$
- $A_4(b)$

Example

Let K_r contain:

- $A_1(x) \wedge A_2(x) \rightarrow A_3(x)$
- $A_3(x) \rightarrow A_4(x)$
- $A_4(x) \wedge A_1(x) \rightarrow A_5(x)$
- $A_2(x) \rightarrow A_4(x)$

Let K_a contain:

- $A_2(b)$, $A_1(c)$, $A_2(c)$

Which of the following hold:

- $K \models A_5(b)$? No – $K \not\models A_5(b)$
- $K \models A_5(c)$? Yes – $K \models A_5(c)$

DerivedAssertions

- $A_2(b)$
- $A_1(c)$
- $A_2(c)$
- $A_3(c)$
- $A_4(c)$
- $A_5(c)$
- $A_4(b)$

Time and Space Complexity

We consider the algorithm computing the set *DerivedAssertions*.

Time and Space Complexity

We consider the algorithm computing the set *DerivedAssertions*.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Time and Space Complexity

We consider the algorithm computing the set *DerivedAssertions*.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Let K consist of K_a and K_r. By I_{K_a} we denote the individual names in K_a. By $|M|$ we denote the number of elements of a set M. So

- $|K_a|$ is the number of atomic assertions in K_a;
- $|K_r|$ is the number of rules in K_r;
- $|I_{K_a}|$ is the number of individual names in K_a.

Time and Space Complexity

We consider the algorithm computing the set *DerivedAssertions*.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Let K consist of K_a and K_r. By I_{K_a} we denote the individual names in K_a. By $|M|$ we denote the number of elements of a set M. So

- $|K_a|$ is the number of atomic assertions in K_a;
- $|K_r|$ is the number of rules in K_r;
- $|I_{K_a}|$ is the number of individual names in K_a.

Then the number of times an assertion is added to *DerivedAssertions* is $\leq |I_{K_a}| \times |K_r|$.

Time and Space Complexity

We consider the algorithm computing the set *DerivedAssertions*.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Let K consist of K_a and K_r. By I_{K_a} we denote the individual names in K_a. By $|M|$ we denote the number of elements of a set M. So

- $|K_a|$ is the number of atomic assertions in K_a;
- $|K_r|$ is the number of rules in K_r;
- $|I_{K_a}|$ is the number of individual names in K_a.

Then the number of times an assertion is added to *DerivedAssertions* is $\leq |I_{K_a}| \times |K_r|$. The number of assertions in *DerivedAssertions* is $\leq |K_a| + |I_{K_a}| \times |K_r|$.

Towards non-unary rule-based systems

Suppose we want to reason as follows:

- Peter is a son of John
- John is a son of Joseph
- Thus, Peter is a grandson of Joseph.

Towards non-unary rule-based systems

Suppose we want to reason as follows:

- Peter is a son of John
- John is a son of Joseph
- Thus, Peter is a grandson of Joseph.

Then we require names for relations in addition to names for classes.

Towards non-unary rule-based systems

Suppose we want to reason as follows:

- Peter is a son of John
- John is a son of Joseph
- Thus, Peter is a grandson of Joseph.

Then we require names for relations in addition to names for classes.

- A relation name R denotes a set of pairs of individual objects. Relation names are also called binary predicates. Examples:
  - sonOf;
  - grandsonOf
  - we often denote relation name by the upper case letters R, S, R_1, R_2, and so on.

Towards non-unary rule-based systems

Suppose we want to reason as follows:

- Peter is a son of John
- John is a son of Joseph
- Thus, Peter is a grandson of Joseph.

Then we require names for relations in addition to names for classes.

- A relation name R denotes a set of pairs of individual objects. Relation names are also called binary predicates. Examples:
  - sonOf;
  - grandsonOf
  - we often denote relation name by the upper case letters R, S, R_1, R_2, and so on.

To express that an individual object a is in the relation R to an individual object b, we write $R(a, b)$. $R(a, b)$ is (again) called an atomic assertion. Example:

- sonOf(Peter, John).

Rule-based knowledge bases

A rule has the form

$$R_1(x_1, y_1) \wedge \cdots \wedge R_n(x_n, y_n) \wedge A_1(x_{n+1}) \wedge \ldots \wedge A_m(x_{n+m}) \to R(x, y)$$

or

$$R_1(x_1, y_1) \wedge \cdots \wedge R_n(x_n, y_n) \wedge A_1(x_{n+1}) \wedge \ldots \wedge A_m(x_{n+m}) \to A(x)$$

where

- $R_1, \ldots, R_n$ and R are relation names,
- $A_1, \ldots, A_n$ and A are class names,
- $x_1, y_1, \ldots, x_n, y_n, x_{n+1}, \ldots, x_{n+m}, x, y$ are individual variables

Rule-based knowledge bases

A rule has the form

$$R_1(x_1, y_1) \wedge \cdots \wedge R_n(x_n, y_n) \wedge A_1(x_{n+1}) \wedge \ldots \wedge A_m(x_{n+m}) \rightarrow R(x, y)$$

or

$$R_1(x_1, y_1) \wedge \cdots \wedge R_n(x_n, y_n) \wedge A_1(x_{n+1}) \wedge \ldots \wedge A_m(x_{n+m}) \rightarrow A(x)$$

where

- $R_1, \ldots, R_n$ and R are relation names,
- $A_1, \ldots, A_n$ and A are class names,
- $x_1, y_1, \ldots, x_n, y_n, x_{n+1}, \ldots, x_{n+m}, x, y$ are individual variables

A rule-based knowledge base K is a collection K_a of atomic assertions and K_r of rules.

Rule-based knowledge base: Example 1

Consider the following set K_a of atomic assertions:

- childOf(Peter, John)
- childOf(John, Joseph)

Rule-based knowledge base: Example 1

Consider the following set K_a of atomic assertions:

- $\mathrm{childOf}(\mathrm{Peter}, \mathrm{John})$
- $\mathrm{childOf}(\mathrm{John}, \mathrm{Joseph})$

Consider the following set K_r of rules:

- $\mathrm{childOf}(x, y) \wedge \mathrm{childOf}(y, z) \rightarrow \mathrm{grandchildOf}(x, z)$

Rule-based knowledge base: Example 1

Consider the following set K_a of atomic assertions:

- $\text{childOf}(\text{Peter}, \text{John})$
- $\text{childOf}(\text{John}, \text{Joseph})$

Consider the following set K_r of rules:

- $\text{childOf}(x, y) \land \text{childOf}(y, z) \rightarrow \text{grandchildOf}(x, z)$

Then $\text{grandchildOf}(\text{Peter}, \text{Joseph})$ follows from K, in symbols

$$K \models \text{grandchildOf}(\text{Peter}, \text{Joseph})$$

On the other hand, $\text{grandchildOf}(\text{Joseph}, \text{Peter})$ does not follow from K, in symbols

$$K \not\models \text{grandchildOf}(\text{Joseph}, \text{Peter})$$

Rule-based knowledge base: Example 2

Consider the following set K_a of atomic assertions:

- childOf(Peter, John)
- childOf(John, Joseph)

Rule-based knowledge base: Example 2

Consider the following set K_a of atomic assertions:

- $\text{childOf}(\text{Peter}, \text{John})$
- $\text{childOf}(\text{John}, \text{Joseph})$

Consider the following set K_r of rules:

- $\text{childOf}(x, y) \rightarrow \text{parentOf}(y, x)$

Rule-based knowledge base: Example 2

Consider the following set K_a of atomic assertions:

- childOf(Peter, John)
- childOf(John, Joseph)

Consider the following set K_r of rules:

- $\text{childOf}(x, y) \rightarrow \text{parentOf}(y, x)$

Then parentOf(John, Peter) follows from K, in symbols

$$K \models \text{parentOf}(\text{John}, \text{Peter})$$

On the other hand, parentOf(Joseph, Peter) does not follow from K, in symbols

$$K \not\models \text{parentOf}(\text{Joseph}, \text{Peter})$$

Computing *DerivedAssertions*

The algorithm for computing *DerivedAssertions* is very similar to the one for unary rules.

Examples:

Computing *DerivedAssertions*

The algorithm for computing *DerivedAssertions* is very similar to the one for unary rules.

Examples:

(1) If $R(x, y) \rightarrow R(y, x) \in K_r$ and $R(a, b)$ has already been derived, then $R(b, a)$ is added to the derived assertions.

Computing *DerivedAssertions*

The algorithm for computing *DerivedAssertions* is very similar to the one for unary rules.

Examples:

(1) If $R(x, y) \rightarrow R(y, x) \in K_r$ and $R(a, b)$ has already been derived, then $R(b, a)$ is added to the derived assertions.

(2) If $R(x, y) \wedge R(y, z) \rightarrow R(x, z) \in K_r$ and $R(a, b), R(b, c)$ have already been derived, then $R(a, c)$ is added to the derived assertions.

Computing *DerivedAssertions*

The algorithm for computing *DerivedAssertions* is very similar to the one for unary rules.

Examples:

(1) If $R(x, y) \rightarrow R(y, x) \in K_r$ and $R(a, b)$ has already been derived, then $R(b, a)$ is added to the derived assertions.

(2) If $R(x, y) \wedge R(y, z) \rightarrow R(x, z) \in K_r$ and $R(a, b), R(b, c)$ have already been derived, then $R(a, c)$ is added to the derived assertions.

(3) If $R(x, y) \wedge R(y, z) \wedge A(y) \rightarrow A(z) \in K_r$ and $R(a, b), R(b, c), A(b)$ have already been derived, then $A(c)$ is added to the derived assertions.

Rule-based knowledge bases: also called knowledge graphs

Binary predicates allows us to talk about graphs.

Rule-based knowledge bases: also called knowledge graphs

Binary predicates allows us to talk about graphs.

Let K_r contain:

- $\text{childOf}(x, y) \rightarrow \text{descendantOf}(x, y)$
- $\text{descendantOf}(x, y) \wedge \text{descendantOf}(y, z) \rightarrow \text{descendantOf}(x, z)$

Rule-based knowledge bases: also called knowledge graphs

Binary predicates allows us to talk about graphs.

Let K_r contain:

- $\text{childOf}(x, y) \rightarrow \text{descendantOf}(x, y)$
- $\text{descendantOf}(x, y) \wedge \text{descendantOf}(y, z) \rightarrow \text{descendantOf}(x, z)$

Let K_a be $\{\text{childOf}(\text{Peter}, \text{John}), \text{childOf}(\text{John}, \text{Joseph})\}$

Rule-based knowledge bases: also called knowledge graphs

Binary predicates allows us to talk about graphs.

Let K_r contain:

- $\text{childOf}(x, y) \rightarrow \text{descendantOf}(x, y)$
- $\text{descendantOf}(x, y) \wedge \text{descendantOf}(y, z) \rightarrow \text{descendantOf}(x, z)$

Let K_a be $\{\text{childOf}(\text{Peter}, \text{John}), \text{childOf}(\text{John}, \text{Joseph})\}$

K_a can be seen as the following graph (individual names = nodes, relations = edges).

$$\text{Peter} \xrightarrow{\text{childOf}} \text{John} \xrightarrow{\text{childOf}} \text{Joseph}$$

Rule-based knowledge bases: also called knowledge graphs

Binary predicates allows us to talk about graphs.

Let K_r contain:

- $\text{childOf}(x, y) \rightarrow \text{descendantOf}(x, y)$
- $\text{descendantOf}(x, y) \land \text{descendantOf}(y, z) \rightarrow \text{descendantOf}(x, z)$

Let K_a be $\{\text{childOf}(\text{Peter}, \text{John}), \text{childOf}(\text{John}, \text{Joseph})\}$

K_a can be seen as the following graph (individual names = nodes, relations = edges).

$$\begin{tikzcd}
\text{Peter} \arrow[r, "\text{childOf}", "\text{descendantOf}"'] \arrow[rr, bend right, "\text{descendantOf}"'] & \text{John} \arrow[r, "\text{childOf}", "\text{descendantOf}"'] & \text{Joseph}
\end{tikzcd}$$

Then $K \models \text{descendantOf}(\text{Peter}, \text{Joseph})$. Computing *DerivedAssertions* corresponds to a graph completion.

Knowledge Graphs: Example 3

Consider the following set K_a of atomic assertions:

- $\mathsf{childOf}(\mathsf{Peter}, \mathsf{John})$
- $\mathsf{childOf}(\mathsf{John}, \mathsf{Joseph})$
- $\mathsf{childOf}(\mathsf{Joseph}, \mathsf{Paul})$

Knowledge Graphs: Example 3

Consider the following set K_a of atomic assertions:

- $\text{childOf}(\text{Peter}, \text{John})$
- $\text{childOf}(\text{John}, \text{Joseph})$
- $\text{childOf}(\text{Joseph}, \text{Paul})$

Consider the following set K_r of rules:

- $\text{childOf}(x, y) \rightarrow \text{descendantOf}(x, y)$
- $\text{descendantOf}(x, y) \wedge \text{descendantOf}(y, z) \rightarrow \text{descendantOf}(x, z)$

Knowledge Graphs: Example 3

Consider the following set K_a of atomic assertions:

- $\mathrm{childOf}(\mathrm{Peter}, \mathrm{John})$
- $\mathrm{childOf}(\mathrm{John}, \mathrm{Joseph})$
- $\mathrm{childOf}(\mathrm{Joseph}, \mathrm{Paul})$

Consider the following set K_r of rules:

- $\mathrm{childOf}(x, y) \rightarrow \mathrm{descendantOf}(x, y)$
- $\mathrm{descendantOf}(x, y) \wedge \mathrm{descendantOf}(y, z) \rightarrow \mathrm{descendantOf}(x, z)$

Then $\mathrm{descendantOf}(\mathrm{Peter}, \mathrm{Paul})$ follows from K, in symbols

$$K \models \mathrm{descendantOf}(\mathrm{Peter}, \mathrm{Paul})$$

Knowledge Graphs: Example 4

Consider the following set K_a of atomic assertions:

- $R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)$.

Knowledge Graphs: Example 4

Consider the following set K_a of atomic assertions:

- $R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)$.

Consider the following set K_r of rules:

- $R(x, y) \rightarrow \text{path}(x, y)$
- $\text{path}(x, y) \wedge \text{path}(y, z) \rightarrow \text{path}(x, z)$

Knowledge Graphs: Example 4

Consider the following set K_a of atomic assertions:

- $R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)$.

Consider the following set K_r of rules:

- $R(x, y) \rightarrow \text{path}(x, y)$
- $\text{path}(x, y) \wedge \text{path}(y, z) \rightarrow \text{path}(x, z)$

Then $\text{path}(a_i, a_j)$ follows from K, in symbols

$$K \models \text{path}(a_i, a_j)$$

if, and only if, $i < j$.

Knowledge Graphs: Example 5

Consider the following set K_a of atomic assertions:

- $R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)$.

Knowledge Graphs: Example 5

Consider the following set K_a of atomic assertions:

- $R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)$.

Consider the following set K_r of rules:

- $R(x, y) \rightarrow \mathrm{path}(x, y)$
- $\mathrm{path}(x, y) \wedge \mathrm{path}(y, z) \rightarrow \mathrm{path}(x, z)$
- $R(x, y) \rightarrow R(y, x)$

Knowledge Graphs: Example 5

Consider the following set K_a of atomic assertions:

- $R(a_0, a_1), R(a_1, a_2), \ldots, R(a_{n-1}, a_n)$.

Consider the following set K_r of rules:

- $R(x, y) \rightarrow \text{path}(x, y)$
- $\text{path}(x, y) \wedge \text{path}(y, z) \rightarrow \text{path}(x, z)$
- $R(x, y) \rightarrow R(y, x)$

Then $\text{path}(a_i, a_j)$ follows from K, in symbols

$$K \models \text{path}(a_i, a_j)$$

for all i, j.

Knowledge Graphs: Example 6

Let K_r contain:

- $\text{childOf}(x, y) \wedge \text{childOf}(z, y) \wedge x \neq z \rightarrow \text{siblingOf}(x, z)$
- $\text{Female}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{sisterOf}(x, y)$
- $\text{Male}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{brotherOf}(x, y)$

Knowledge Graphs: Example 6

Let K_r contain:

- $\mathrm{childOf}(x, y) \wedge \mathrm{childOf}(z, y) \wedge x \neq z \to \mathrm{siblingOf}(x, z)$
- $\mathrm{Female}(x) \wedge \mathrm{siblingOf}(x, y) \to \mathrm{sisterOf}(x, y)$
- $\mathrm{Male}(x) \wedge \mathrm{siblingOf}(x, y) \to \mathrm{brotherOf}(x, y)$

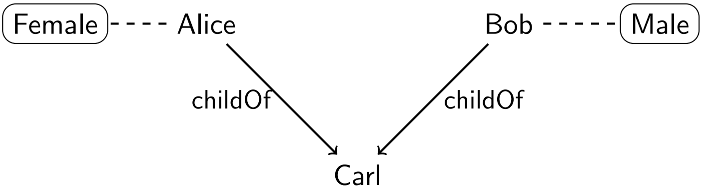
Let K_a be
$\{\mathrm{Female}(Alice), \mathrm{Male}(Bob), \mathrm{childOf}(Alice, Carl), \mathrm{childOf}(Bob, Carl)\}$

Knowledge Graphs: Example 6

Let K_r contain:

- $\text{childOf}(x, y) \wedge \text{childOf}(z, y) \wedge x \neq z \rightarrow \text{siblingOf}(x, z)$
- $\text{Female}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{sisterOf}(x, y)$
- $\text{Male}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{brotherOf}(x, y)$

Let K_a be
$\{\text{Female}(\mathit{Alice}), \text{Male}(\mathit{Bob}), \text{childOf}(\mathit{Alice}, \mathit{Carl}), \text{childOf}(\mathit{Bob}, \mathit{Carl})\}$

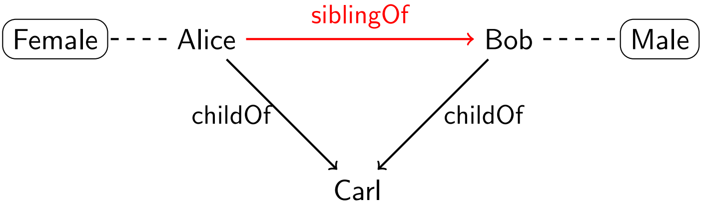


Knowledge Graphs: Example 6

Let K_r contain:

- $\text{childOf}(x, y) \wedge \text{childOf}(z, y) \wedge x \neq z \rightarrow \text{siblingOf}(x, z)$
- $\text{Female}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{sisterOf}(x, y)$
- $\text{Male}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{brotherOf}(x, y)$

Let K_a be
$\{\text{Female}(\textit{Alice}), \text{Male}(\textit{Bob}), \text{childOf}(\textit{Alice}, \textit{Carl}), \text{childOf}(\textit{Bob}, \textit{Carl})\}$

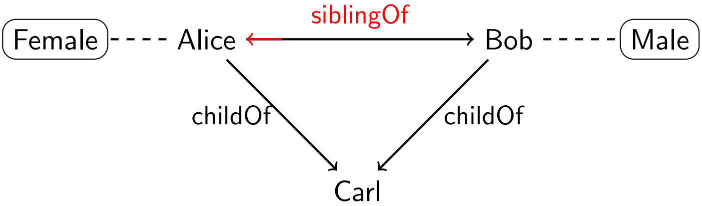


Knowledge Graphs: Example 6

Let K_r contain:

- $\text{childOf}(x, y) \wedge \text{childOf}(z, y) \wedge x \neq z \rightarrow \text{siblingOf}(x, z)$
- $\text{Female}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{sisterOf}(x, y)$
- $\text{Male}(x) \wedge \text{siblingOf}(x, y) \rightarrow \text{brotherOf}(x, y)$

Let K_a be
$\{\text{Female}(\mathit{Alice}), \text{Male}(\mathit{Bob}), \text{childOf}(\mathit{Alice}, \mathit{Carl}), \text{childOf}(\mathit{Bob}, \mathit{Carl})\}$

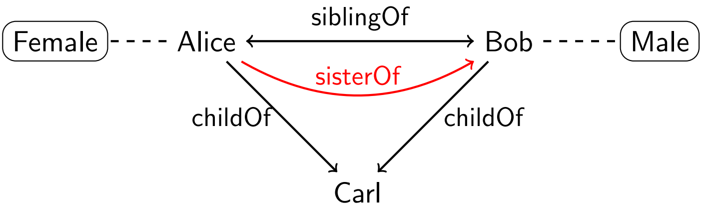


Knowledge Graphs: Example 6

Let K_r contain:

- $\text{childOf}(x, y) \land \text{childOf}(z, y) \land x \neq z \rightarrow \text{siblingOf}(x, z)$
- $\text{Female}(x) \land \text{siblingOf}(x, y) \rightarrow \text{sisterOf}(x, y)$
- $\text{Male}(x) \land \text{siblingOf}(x, y) \rightarrow \text{brotherOf}(x, y)$

Let K_a be
$\{\text{Female}(\mathit{Alice}), \text{Male}(\mathit{Bob}), \text{childOf}(\mathit{Alice}, \mathit{Carl}), \text{childOf}(\mathit{Bob}, \mathit{Carl})\}$

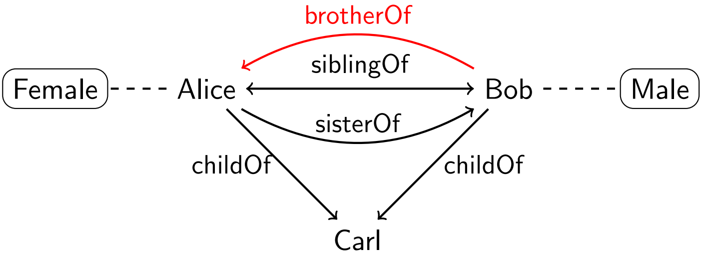


Knowledge Graphs: Example 6

Let K_r contain:

- $\mathsf{childOf}(x, y) \wedge \mathsf{childOf}(z, y) \wedge x \neq z \rightarrow \mathsf{siblingOf}(x, z)$
- $\mathsf{Female}(x) \wedge \mathsf{siblingOf}(x, y) \rightarrow \mathsf{sisterOf}(x, y)$
- $\mathsf{Male}(x) \wedge \mathsf{siblingOf}(x, y) \rightarrow \mathsf{brotherOf}(x, y)$

Let K_a be
$\{\mathsf{Female}(Alice), \mathsf{Male}(Bob), \mathsf{childOf}(Alice, Carl), \mathsf{childOf}(Bob, Carl)\}$



Time and Space Complexity: non-unary case

We consider the algorithm computing the set *DerivedAssertions* for

.

Time and Space Complexity: non-unary case

We consider the algorithm computing the set *DerivedAssertions* for

.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Time and Space Complexity: non-unary case

We consider the algorithm computing the set *DerivedAssertions* for

.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Let K consist of K_a and K_r. By I_{K_a} we denote the individual names in K_a. By $|M|$ we denote the number of elements of a set M.

Time and Space Complexity: non-unary case

We consider the algorithm computing the set *DerivedAssertions* for

.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Let K consist of K_a and K_r. By I_{K_a} we denote the individual names in K_a. By $|M|$ we denote the number of elements of a set M.

Then the number of times an assertion is added to *DerivedAssertions* is $\leq |I_{K_a}|^2 \times |K_r|$.

Time and Space Complexity: non-unary case

We consider the algorithm computing the set *DerivedAssertions* for

.

- Time complexity: the number of times a new assertion is added to *DerivedAssertions*. (We ignore checking rule applicability).
- Space complexity: The final size of *DerivedAssertions*. (Note that it only grows.)

Let K consist of K_a and K_r. By I_{K_a} we denote the individual names in K_a. By $|M|$ we denote the number of elements of a set M.

Then the number of times an assertion is added to *DerivedAssertions* is $\leq |I_{K_a}|^2 \times |K_r|$.

The number of assertions in *DerivedAssertions* is $\leq |K_a| + |I_{K_a}|^2 \times |K_r|$.

Formulating queries using rules (1)

Assume that atomic assertions in K_a are formulated using the binary relation childOf and the class name King. Define a set K_r of rules for the class name GrandchildK such that for any such set of atomic assertions K_a:

$$K \models \text{GrandchildK}(a) \text{ if and only if } a \text{ is a grandchild of a king}$$

Formulating queries using rules (1)

Assume that atomic assertions in K_a are formulated using the binary relation childOf and the class name King. Define a set K_r of rules for the class name GrandchildK such that for any such set of atomic assertions K_a:

$$K \models \text{GrandchildK}(a) \text{ if and only if } a \text{ is a grandchild of a king}$$

Answer. K_r consisting of the rule

$$\text{childOf}(x, y) \wedge \text{childOf}(y, z) \wedge \text{King}(z) \rightarrow \text{GrandchildK}(x)$$

is as required.

Formulating queries using rules (2)

Assume that atomic assertions in K_a are formulated using the binary relation childOf and the class name King. Define a set K_r of rules for the class name DescendantK such that for any such set of atomic assertions K_a:

$K \models \mathsf{DescendantK}(a)$ if and only if a is a descendant of a king

Formulating queries using rules (2)

Assume that atomic assertions in K_a are formulated using the binary relation childOf and the class name King. Define a set K_r of rules for the class name DescendantK such that for any such set of atomic assertions K_a:

$$K \models \text{DescendantK}(a) \text{ if and only if } a \text{ is a descendant of a king}$$

Answer. K_r consisting of the rules

$$\text{childOf}(x, y) \rightarrow \text{descendantOf}(x, y)$$

$$\text{descendantOf}(x, y) \wedge \text{descendantOf}(y, z) \rightarrow \text{descendantOf}(x, z)$$

$$\text{descendantOf}(x, y) \wedge \text{King}(y) \rightarrow \text{DescendantK}(x)$$

is as required.