Unification in Description Logics

Part I: Introduction

Oliver Fernández Gil

Chair of Automata Theory

TECHNISCHE UNIVERSITÄT DRESDEN

ESSLLI'19

Riga, August 2019

What is unification?

What is unification?

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x, g(a, b)) \qquad s = f(g(y, b), x)$$

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\ g(a\,,b)) \qquad\qquad s = f(g(y\,,b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\; g(a\,, b)) \qquad\qquad s = f(g(y\,, b),\; x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$$x \mapsto g(a, b), y \mapsto b \text{ is a solution of } s =^{?} t \text{ (a unifier).}$$

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\ g(a\,,b)) \qquad\qquad s = f(g(y\,,b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a, b), y \mapsto b$ is a solution of $s =^{?} t$ (a unifier).

Originally introduced in automated deduction

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\ g(a\,,b)) \qquad\qquad s = f(g(y\,,b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a, b), y \mapsto b$ is a solution of $s =^{?} t$ (a unifier).

Originally introduced in automated deduction

- Basic operation of J.A. Robinson's resolution inference principle [Rob65].

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\ ,\ g(a\ , b)) \qquad\qquad s = f(g(y\ , b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a, b), y \mapsto b$ is a solution of $s =^{?} t$ (a unifier).

Originally introduced in automated deduction

- Basic operation of J.A. Robinson's resolution inference principle [Rob65].
- Important! To compute a most general unifier (mgu).

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\ ,\ g(a\ ,b)) \qquad\qquad s = f(g(y\ ,b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a, b), y \mapsto b$ is a solution of $s =^? t$ (a unifier).

Originally introduced in automated deduction

- Basic operation of J.A. Robinson's resolution inference principle [Rob65].
- Important! To compute a most general unifier (mgu).

$f(x, y) =^? f(y, x)$ has many solutions: $x, y \mapsto f(x), x, y \mapsto f(f(x)), \ldots$

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\ g(a\,, b)) \qquad\qquad s = f(g(y\,, b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a, b), y \mapsto b$ is a solution of $s =^? t$ (a unifier).

Originally introduced in automated deduction

- Basic operation of J.A. Robinson's resolution inference principle [Rob65].
- Important! To compute a most general unifier (mgu).

$f(x, y) =^? f(y, x)$ has many solutions: $x, y \mapsto f(x), x, y \mapsto f(f(x)), \ldots$

$x \mapsto y$ generates all of them, i.e., it is a mgu

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\ g(a\,,b)) \qquad\qquad s = f(g(y\,,b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a,b), y \mapsto b$ is a solution of $s =^{?} t$ (a unifier).

Originally introduced in automated deduction

- Basic operation of J.A. Robinson's resolution inference principle [Rob65].
- Important! To compute a most general unifier (mgu).

$f(x,y) =^{?} f(y,x)$ has many solutions: $x, y \mapsto f(x), x, y \mapsto f(f(x)), \ldots$

$x \mapsto y$ generates all of them, i.e., it is a mgu

Rediscovered in the area of term rewriting systems.

What is unification?

Unification problem: make two given first-order logic terms syntactically equal.

$$t = f(x\,,\ g(a\,,b)) \qquad\qquad s = f(g(y\,,b),\ x)$$

Q: can x and y be substituted in s and t by terms such that the resulting terms are "identical"?

$$\downarrow$$

$x \mapsto g(a,b), y \mapsto b$ is a solution of $s =^? t$ (a unifier).

Originally introduced in automated deduction

- Basic operation of J.A. Robinson's resolution inference principle [Rob65].
- Important! To compute a most general unifier (mgu).

$f(x,y) =^? f(y,x)$ has many solutions: $x, y \mapsto f(x), x, y \mapsto f(f(x)), \ldots$

$x \mapsto y$ generates all of them, i.e., it is a mgu

Rediscovered in the area of term rewriting systems.

- Knuth-Bendix completion algorithm [KB70]

Equational Unification

Equational Unification

Initial goal: to integrate troublesome axioms (like *commutativity, associativity*) into the unification process.

Equational Unification

Initial goal: to integrate troublesome axioms (like *commutativity, associativity*) into the unification process.

- Changes the nature of the problem:

$$f(a, x) =^? f(b, y) \text{ has no solution w.r.t. "syntactic unification".}$$

Equational Unification

Initial goal: to integrate troublesome axioms (like *commutativity, associativity*) into the unification process.

- Changes the nature of the problem:

$$f(a, x) =^? f(b, y) \text{ has no solution w.r.t. "syntactic unification".}$$

$$\text{But, } x \mapsto b, y \mapsto a \text{ is a solution w.r.t. } \mathrm{C} = \{f(x, y) \approx f(y, x)\}$$

Equational Unification

Initial goal: to integrate troublesome axioms (like *commutativity, associativity*) into the unification process.

- Changes the nature of the problem:

$$f(a, x) =^? f(b, y) \text{ has no solution w.r.t. "syntactic unification".}$$

$$\text{But, } x \mapsto b, y \mapsto a \text{ is a solution w.r.t. } \mathsf{C} = \{f(x, y) \approx f(y, x)\}$$

$$f(a, b) =_{\mathsf{C}} f(b, a)$$

Equational Unification

Initial goal: to integrate troublesome axioms (like *commutativity, associativity*) into the unification process.

- Changes the nature of the problem:

$$f(a,x) =^? f(b,y) \text{ has no solution w.r.t. "syntactic unification".}$$

$$\text{But, } x \mapsto b, y \mapsto a \text{ is a solution w.r.t. } \mathrm{C} = \{f(x,y) \approx f(y,x)\}$$

$$f(a,b) =_{\mathrm{C}} f(b,a)$$

- A little bit more formal/general,

Equational theory. Let E by a set of identities between first-order terms. The equational theory defined by $=_E$ consists of all identities $s = t$ that can be "derived" from E.

E-unification problem. $\Gamma := \{s_1 =^?_E t_1, \ldots, s_n =^?_E t_n\}$. A substitution σ is an E-unifier of Γ if

$$\sigma(s_i) =_E \sigma(t_i), \text{ for all } 1 \leq i \leq n.$$

Equational Unification

Most general unifiers need not exist

Equational Unification

Most general unifiers need not exist

- A C-unification problem with two minimal "non-comparable" unifiers:

$$\Gamma = \{f(x, y) =^?_C f(a, b)\} \qquad x \mapsto a, y \mapsto b \qquad x \mapsto b, y \mapsto a$$

Equational Unification

Most general unifiers need not exist

- A C-unification problem with two minimal "non-comparable" unifiers:

$$\Gamma = \{f(x, y) =^?_C f(a, b)\} \qquad x \mapsto a, y \mapsto b \qquad x \mapsto b, y \mapsto a$$

- Notion of a mgu needs to be extended to that of

  a minimal complete set of unifiers.

Equational Unification

Most general unifiers need not exist

- A C-unification problem with two minimal "non-comparable" unifiers:

$$\Gamma = \{f(x,y) =^?_C f(a,b)\} \qquad x \mapsto a, y \mapsto b \qquad x \mapsto b, y \mapsto a$$

- Notion of a mgu needs to be extended to that of

  a minimal complete set of unifiers.

- Unification type: cardinality of such sets.

Equational Unification

Most general unifiers need not exist

- A C-unification problem with two minimal "non-comparable" unifiers:

$$\Gamma = \{f(x,y) =^?_{\mathsf{C}} f(a,b)\} \qquad x \mapsto a, y \mapsto b \qquad x \mapsto b, y \mapsto a$$

- Notion of a mgu needs to be extended to that of

  a minimal complete set of unifiers.

- Unification type: cardinality of such sets.
  - It can be infinite:
    (associativity) $\mathsf{A} = \{f(x, f(y,z)) \approx f(f(x,y),z)\}$ and $\Gamma = \{f(a,x) =^?_{\mathsf{A}} f(x,a)\}$

Equational Unification

Most general unifiers need not exist

- A C-unification problem with two minimal “non-comparable” unifiers:

$$\Gamma = \{f(x,y) =^?_{\mathrm{C}} f(a,b)\} \qquad x \mapsto a, y \mapsto b \qquad x \mapsto b, y \mapsto a$$

- Notion of a mgu needs to be extended to that of

  a minimal complete set of unifiers.

- Unification type: cardinality of such sets.
  - It can be infinite:
    (associativity) $\mathrm{A} = \{f(x, f(y,z)) \approx f(f(x,y),z)\}$ and $\Gamma = \{f(a,x) =^?_{\mathrm{A}} f(x,a)\}$
  - minimal complete sets of unifiers may not exist (we will later see)

Unification theory

It investigates:

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.
- Computation of E-unifiers (if they exists).

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.
- Computation of E-unifiers (if they exists).
- Unification type of equational theories.

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.
- Computation of E-unifiers (if they exists).
- Unification type of equational theories.

Applications in many areas:

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.
- Computation of E-unifiers (if they exists).
- Unification type of equational theories.

Applications in many areas:

- Databases, Information retrieval, Planning Systems, . . .

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.
- Computation of E-unifiers (if they exists).
- Unification type of equational theories.

Applications in many areas:

- Databases, Information retrieval, Planning Systems, . . .
- Description Logics: detecting redundancies in ontologies.

Unification theory

It investigates:

- Decidability and complexity of E-unification problems.
- Computation of E-unifiers (if they exists).
- Unification type of equational theories.

Applications in many areas:

- Databases, Information retrieval, Planning Systems, ...
- Description Logics: detecting redundancies in ontologies.
- Modal Logics: special case of recognizability of admissible inference rules.

Description Logics

dl.kr.org

What are Description Logics (DLs)?

"*...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way...*"

What are Description Logics (DLs)?

> "*...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way...*"

Important notions of the domain $\rightarrow$ represented as concept descriptions:

What are Description Logics (DLs)?

"*...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way...*"

Important notions of the domain $\rightarrow$ represented as concept descriptions:

What are Description Logics (DLs)?

"*...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way...*"

Important notions of the domain → represented as concept descriptions:

... a human, that is an athlete,
plays baseball, wears a helmet or a cap,
is not lazy, only owns shiny baseball bats ...

What are Description Logics (DLs)?

"*...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way...*"

Important notions of the domain → represented as concept descriptions:

Atomic properties → Concept names

Human, Athlete, Baseball, Helmet, . . .

... a human, that is an athlete,
plays baseball, wears a helmet or a cap,
is not lazy, only owns shiny baseball bats ...

What are Description Logics (DLs)?

"*...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way...*"

Important notions of the domain → represented as concept descriptions:

... a human, that is an athlete,
plays baseball, wears a helmet or a cap,
is not lazy, only owns shiny baseball bats ...

Atomic properties → Concept names

Human, Athlete, Baseball, Helmet, . . .

Relations → Role names

plays, wears

What are Description Logics (DLs)?

"...a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way..."

Important notions of the domain → represented as concept descriptions:

... a human, that is an athlete, plays baseball, wears a helmet or a cap, is not lazy, only owns shiny baseball bats ...

Atomic properties → Concept names

Human, Athlete, Baseball, Helmet, . . .

Relations → Role names

plays, wears

Concept descriptions: built using the concept/role constructors provided by a DL.

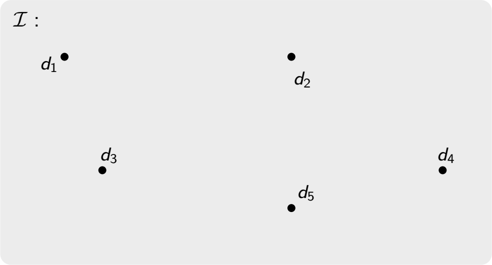
$$\text{Human} \sqcap \text{Athlete} \sqcap \exists \text{wears}.(\text{Helmet} \sqcup \text{Cap}) \sqcap$$
$$\exists \text{plays}.\text{Baseball} \sqcap \neg \text{Lazy} \sqcap \forall \text{owns\_bat}.\text{Shiny}$$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic

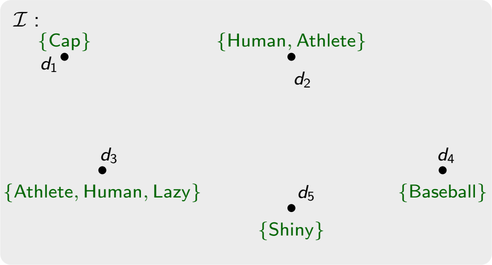
What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



What are Description Logics (DLs)? Semantics

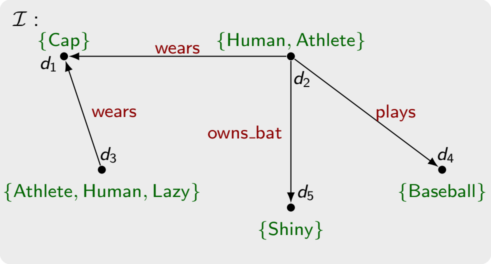
Formal semantics inherited from *first-order* logic



Concept names: unary predicates

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic

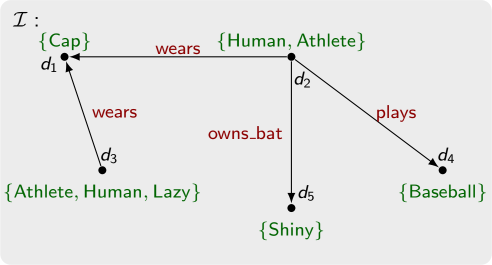


Concept names: unary predicates

Role names: binary predicates

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



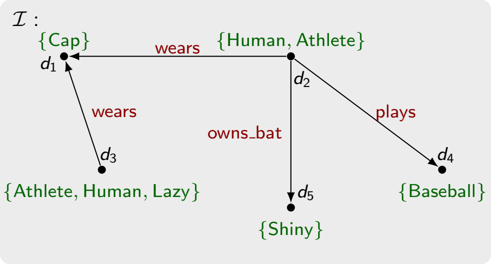
Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

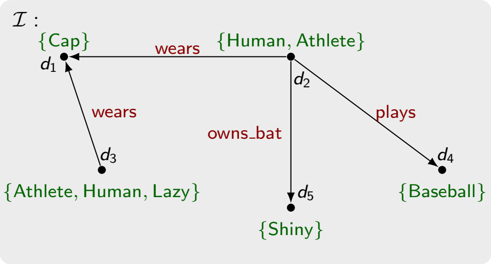
$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

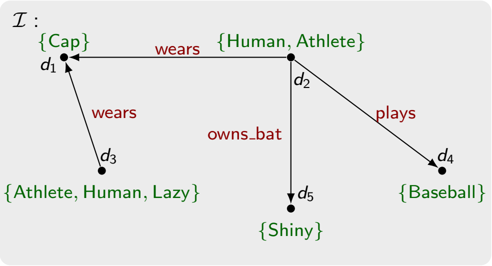
$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$ → $\mathcal{ALC}$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

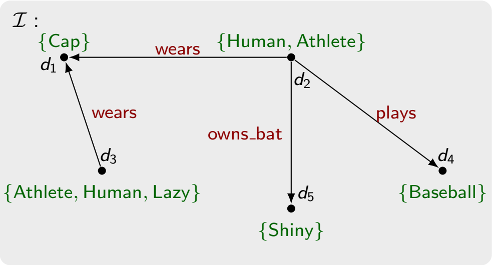
$\leq nr.C$ $\top$ → $\mathcal{ALC}$

Semantics

$(\text{Human} \sqcap \text{Athlete})^{\mathcal{I}} = \{d_2, d_3\}$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$

$\mathcal{ALC}$

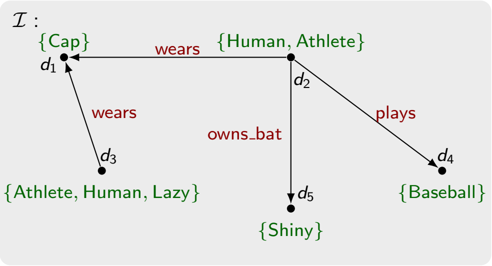
Semantics

$(\text{Human} \sqcap \text{Athlete})^{\mathcal{I}} = \{d_2, d_3\}$

$(\text{Cap} \sqcup \text{Helmet})^{\mathcal{I}} = \{d_1\}$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$ → $\mathcal{ALC}$

Semantics

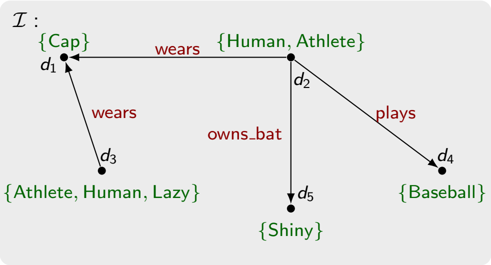
$(\text{Human} \sqcap \text{Athlete})^{\mathcal{I}} = \{d_2, d_3\}$

$(\text{Cap} \sqcup \text{Helmet})^{\mathcal{I}} = \{d_1\}$

$(\neg\text{Lazy})^{\mathcal{I}} = \{d_1, d_2, d_4, d_5\}$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$ → $\mathcal{ALC}$

Semantics

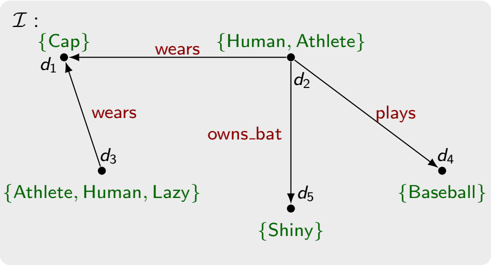
$(\text{Human} \sqcap \text{Athlete})^{\mathcal{I}} = \{d_2, d_3\}$ $(\exists \text{plays.Baseball})^{\mathcal{I}} = \{d_2\}$

$(\text{Cap} \sqcup \text{Helmet})^{\mathcal{I}} = \{d_1\}$

$(\neg\text{Lazy})^{\mathcal{I}} = \{d_1, d_2, d_4, d_5\}$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$ → $\mathcal{ALC}$

Semantics

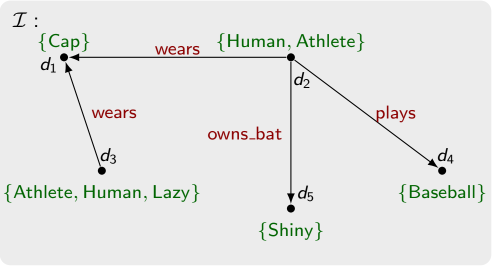
$(\text{Human} \sqcap \text{Athlete})^{\mathcal{I}} = \{d_2, d_3\}$ $(\exists \text{plays.Baseball})^{\mathcal{I}} = \{d_2\}$

$(\text{Cap} \sqcup \text{Helmet})^{\mathcal{I}} = \{d_1\}$ $(\forall \text{owns\_bat.Shiny})^{\mathcal{I}} = \text{dom}(\mathcal{I})$

$(\neg \text{Lazy})^{\mathcal{I}} = \{d_1, d_2, d_4, d_5\}$

What are Description Logics (DLs)? Semantics

Formal semantics inherited from *first-order* logic



Concept names: unary predicates

Role names: binary predicates

Formulas (concept descriptions)

Concept constructors

$\sqcap$ $\forall r.C$

$\neg$ $\exists r.C$ $\sqcup$ $\bot$

$\leq nr.C$ $\top$

$\mathcal{ALC}$

Semantics

$(\text{Human} \sqcap \text{Athlete})^{\mathcal{I}} = \{d_2, d_3\}$ $(\exists \text{plays.Baseball})^{\mathcal{I}} = \{d_2\}$

$(\text{Cap} \sqcup \text{Helmet})^{\mathcal{I}} = \{d_1\}$ $(\forall \text{owns\_bat.Shiny})^{\mathcal{I}} = \text{dom}(\mathcal{I})$

$(\neg\text{Lazy})^{\mathcal{I}} = \{d_1, d_2, d_4, d_5\}$

$\forall r.C \equiv \neg\exists r.\neg C$ $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$ $\bot \equiv \neg\top$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq$ ◯

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq \bigcirc$

Concept inclusions (GCIs)

pitchers are baseball players
and throw fastball

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

$\text{Baseball\_Player} \doteq \bigcirc$

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ $\text{Pitcher} \sqsubseteq \text{Baseball\_Player} \sqcap \exists \text{throws.Fastball}$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq$ ◯

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player $\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq \bigcirc$

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player $\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

$(\text{Baseball\_Player})^{\mathcal{I}} = (\bigcirc)^{\mathcal{I}}$

$(\text{Pitcher})^{\mathcal{I}} \subseteq (\text{Baseball\_Player} \sqcap \exists \text{throws.Fastball})^{\mathcal{I}}$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq$ ◯

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player$\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

$(\text{Baseball\_Player})^{\mathcal{I}} = (◯)^{\mathcal{I}}$

$(\text{Pitcher})^{\mathcal{I}} \subseteq (\text{Baseball\_Player} \sqcap \exists\text{throws.Fastball})^{\mathcal{I}}$

$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I}$ satisfies all definitions/GCIs in $\mathcal{T}$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

$\text{Baseball\_Player} \doteq \bigcirc$

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ $\text{Pitcher} \sqsubseteq \text{Baseball\_Player} \sqcap \exists\text{throws.Fastball}$

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

$(\text{Baseball\_Player})^{\mathcal{I}} = (\bigcirc)^{\mathcal{I}}$

$(\text{Pitcher})^{\mathcal{I}} \subseteq (\text{Baseball\_Player} \sqcap \exists\text{throws.Fastball})^{\mathcal{I}}$

$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I}$ satisfies all definitions/GCIs in $\mathcal{T}$

Assertional knowledge (knowledge about concrete situations)

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq \bigcirc$

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player$\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

$(\text{Baseball\_Player})^{\mathcal{I}} = (\bigcirc)^{\mathcal{I}}$

$(\text{Pitcher})^{\mathcal{I}} \subseteq (\text{Baseball\_Player} \sqcap \exists\text{throws.Fastball})^{\mathcal{I}}$

$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I}$ satisfies all definitions/GCIs in $\mathcal{T}$

Assertional knowledge (knowledge about concrete situations)

Pitcher(pedro)

Shiny(s) ¬Lazy(omar)

Human(pedro)

owns_bat($omar, s$)

. . .

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq$ ◯

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player $\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

(Baseball_Player)$^{\mathcal{I}}$ = (◯)$^{\mathcal{I}}$

(Pitcher)$^{\mathcal{I}} \subseteq$ (Baseball_Player $\sqcap$ $\exists$throws.Fastball)$^{\mathcal{I}}$

$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I}$ satisfies all definitions/GCIs in $\mathcal{T}$

Assertional knowledge (knowledge about concrete situations)

Pitcher(pedro)

Shiny(s)

$\neg$Lazy(omar)

Human(pedro)

owns_bat($omar, s$)

...

A finite set of assertions is called an ABox $\mathcal{A}$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq$ ◯

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player$\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

$(\text{Baseball\_Player})^{\mathcal{I}} = (◯)^{\mathcal{I}}$

$(\text{Pitcher})^{\mathcal{I}} \subseteq (\text{Baseball\_Player} \sqcap \exists\text{throws.Fastball})^{\mathcal{I}}$

$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I}$ satisfies all definitions/GCIs in $\mathcal{T}$

Assertional knowledge (knowledge about concrete situations)

Pitcher(pedro)

Shiny(s) $\neg$Lazy(omar)

Human(pedro)

owns_bat($omar, s$)

...

A finite set of assertions is called an ABox $\mathcal{A}$

A knowledge base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$

What are Description Logics (DLs)? Representing knowledge

Terminological knowledge (general knowledge about the domain)

Concept definitions

Baseball_Player $\doteq \bigcirc$

Concept inclusions (GCIs)

pitchers are baseball players and throw fastball $\longrightarrow$ Pitcher $\sqsubseteq$ Baseball_Player $\sqcap$ $\exists$throws.Fastball

A finite set of definitions/GCIs is called a TBox $\mathcal{T}$

Semantics

$(\text{Baseball\_Player})^{\mathcal{I}} = (\bigcirc)^{\mathcal{I}}$

$(\text{Pitcher})^{\mathcal{I}} \subseteq (\text{Baseball\_Player} \sqcap \exists\text{throws.Fastball})^{\mathcal{I}}$

$\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I}$ satisfies all definitions/GCIs in $\mathcal{T}$

Assertional knowledge (knowledge about concrete situations)

Pitcher(pedro)

Shiny(s) $\neg$Lazy(omar)

Human(pedro)

owns_bat($omar, s$)

...

A finite set of assertions is called an ABox $\mathcal{A}$

A knowledge base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$

Entailments of $\mathcal{K}$: *Pedro throws FastBall*, ...

Reasoning in DLs

Standard Inferences

Reasoning in DLs

Standard Inferences

- Concept satisfiability.
- Subsumption.

Instance:	Two concepts C, D and a TBox $\mathcal{T}$.
Question:	Does $C^\mathcal{I} \subseteq D^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{T}$?

- Knowledge base consistency, query answering.

Reasoning in DLs

Standard Inferences

- Concept satisfiability.
- Subsumption.

Instance:	Two concepts C, D and a TBox $\mathcal{T}$.
Question:	Does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{T}$?

- Knowledge base consistency, query answering.

Non-Standard Inferences

Reasoning in DLs

Standard Inferences

- Concept satisfiability.
- Subsumption.

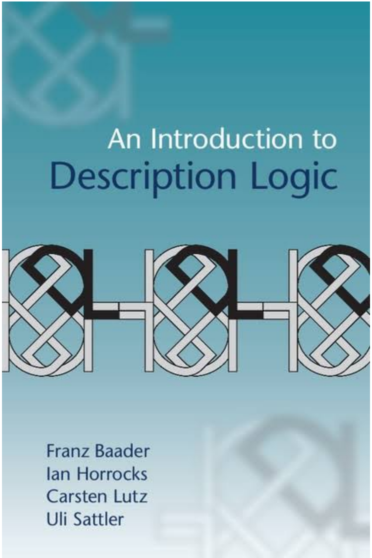
Instance:	Two concepts C, D and a TBox $\mathcal{T}$.
Question:	Does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{T}$?

- Knowledge base consistency, query answering.

Non-Standard Inferences

- Most specific generalizations.
- Least common subsumer.
- Unification.
- ...

More on DLs...


An Introduction to
Description Logic
Franz Baader
Ian Horrocks
Carsten Lutz
Uli Sattler

Unification in Description Logics

Detecting redundancies in ontologies

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

$$\text{a) } \exists \mathsf{finding}.(\mathsf{Head\_injury} \sqcap \exists \mathsf{severity}.\mathsf{Severe})$$

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

a) $\exists \text{finding}.(\text{Head\_injury} \sqcap \exists \text{severity}.\text{Severe})$

b) $\exists \text{finding}.(\text{Severe\_injury} \sqcap \exists \text{finding\_site}.\text{Head})$

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

$$\text{a) } \exists \text{finding.}(\text{Head\_injury} \sqcap \exists \text{severity.Severe})$$

not equivalent, but meant to represent the same notion!

$$\text{b) } \exists \text{finding.}(\text{Severe\_injury} \sqcap \exists \text{finding\_site.Head})$$

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

a) $\exists$finding.(Head_injury $\sqcap$ $\exists$severity.Severe)

not equivalent, but meant to represent the same notion!

b) $\exists$finding.(Severe_injury $\sqcap$ $\exists$finding_site.Head)

- Can they be made equivalent?

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

a) $\exists$finding.(Head_injury $\sqcap$ $\exists$severity.Severe)

not equivalent, but meant to represent the same notion!

b) $\exists$finding.(Severe_injury $\sqcap$ $\exists$finding_site.Head)

- Can they be made equivalent?
  1. Select Head_injury and Severe_injury as variables.

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

$$\text{a) } \exists \text{finding}.(\text{Head\_injury} \sqcap \exists \text{severity}.\text{Severe})$$

not equivalent, but meant to represent the same notion!

$$\text{b) } \exists \text{finding}.(\text{Severe\_injury} \sqcap \exists \text{finding\_site}.\text{Head})$$

- Can they be made equivalent?
  1. Select Head_injury and Severe_injury as variables.
  2. Apply the substitution (add definitions to the ontology):

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

$$\text{a) } \exists \text{finding}.(\text{Head\_injury} \sqcap \exists \text{severity}.\text{Severe})$$

not equivalent, but meant to represent the same notion!

$$\text{b) } \exists \text{finding}.(\text{Severe\_injury} \sqcap \exists \text{finding\_site}.\text{Head})$$

- Can they be made equivalent?
  1. Select Head_injury and Severe_injury as variables.
  2. Apply the substitution (add definitions to the ontology):

$$\text{Head\_injury} \mapsto \text{Injury} \sqcap \exists \text{finding\_site}.\text{Head}$$
$$\text{Severe\_injury} \mapsto \text{Injury} \sqcap \exists \text{severity}.\text{Severe}$$

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

$$\text{a) } \exists \text{finding.}(\text{Head\_injury} \sqcap \exists \text{severity.Severe})$$

not equivalent, but meant to represent the same notion!

$$\text{b) } \exists \text{finding.}(\text{Severe\_injury} \sqcap \exists \text{finding\_site.Head})$$

- Can they be made equivalent?
  1. Select Head_injury and Severe_injury as variables.
  2. Apply the substitution (add definitions to the ontology):

$$\text{Head\_injury} \mapsto \text{Injury} \sqcap \exists \text{finding\_site.Head}$$
$$\text{Severe\_injury} \mapsto \text{Injury} \sqcap \exists \text{severity.Severe}$$

Unification of concept descriptions

Detecting redundancies in ontologies

- Two developers of a medical ontology define finding of severe head injury in two different ways:

a) $\exists$finding.(Head_injury $\sqcap$ $\exists$severity.Severe)

not equivalent, but meant to represent the same notion!

b) $\exists$finding.(Severe_injury $\sqcap$ $\exists$finding_site.Head)

- Can they be made equivalent?
  1. Select Head_injury and Severe_injury as variables.
  2. Apply the substitution (add definitions to the ontology):

$$\text{Head\_injury} \mapsto \text{Injury} \sqcap \exists\text{finding\_site.Head}$$
$$\text{Severe\_injury} \mapsto \text{Injury} \sqcap \exists\text{severity.Severe}$$

Unification of concept descriptions

- Semi-automated process: suggests possible candidates to ontology engineers.

Detecting redundancies in ontologies - TBoxes

- Suppose that the second developer uses a different definition, i.e., c) instead of b):

Detecting redundancies in ontologies - TBoxes

- Suppose that the second developer uses a different definition, i.e., c) instead of b):

a) $\exists$finding.(Head_injury $\sqcap$ $\exists$severity.Severe)

c) $\exists$status.Emergency $\sqcap$ $\exists$finding.(Severe_injury $\sqcap$ $\exists$finding_site.Head)

Detecting redundancies in ontologies - TBoxes

- Suppose that the second developer uses a different definition, i.e., c) instead of b):

a) $\exists \text{finding}.(\text{Head\_injury} \sqcap \exists \text{severity}.\text{Severe})$

not unifiable!

c) $\exists \text{status}.\text{Emergency} \sqcap \exists \text{finding}.(\text{Severe\_injury} \sqcap \exists \text{finding\_site}.\text{Head})$

Detecting redundancies in ontologies - TBoxes

- Suppose that the second developer uses a different definition, i.e., c) instead of b):

a) $\exists \text{finding}.(\text{Head\_injury} \sqcap \exists \text{severity}.\text{Severe})$

not unifiable!

c) $\exists \text{status}.\text{Emergency} \sqcap \exists \text{finding}.(\text{Severe\_injury} \sqcap \exists \text{finding\_site}.\text{Head})$

- But they are, in presence of background knowledge (TBox) containing the GCI:

$$\exists \text{finding}.\exists \text{severity}.\text{Severe} \sqsubseteq \exists \text{status}.\text{Emergency}$$

Unification in DLs. Formal definition

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:
  - $\mathsf{N_v}$: concept variables (like Head_injury and Severe_injury).

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:
  - $\mathsf{N_v}$: concept variables (like Head_injury and Severe_injury).
  - $\mathsf{N_c}$: concept constants (like Severe, Head, Emergency).

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:
  - $\mathsf{N_v}$: concept variables (like Head_injury and Severe_injury).
  - $\mathsf{N_c}$: concept constants (like Severe, Head, Emergency).

- A substitution σ is a mapping of the form:

$$\sigma : \mathsf{N_v} \mapsto \text{the set of all } \mathcal{L} \text{ concept descriptions.}$$

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:
  - $\mathsf{N_v}$: concept variables (like Head_injury and Severe_injury).
  - $\mathsf{N_c}$: concept constants (like Severe, Head, Emergency).

- A substitution σ is a mapping of the form:

$$\sigma : \mathsf{N_v} \mapsto \text{the set of all } \mathcal{L} \text{ concept descriptions.}$$

  σ is extended to arbitrary concepts inductively

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:
  - $\mathsf{N_v}$: concept variables (like Head_injury and Severe_injury).
  - $\mathsf{N_c}$: concept constants (like Severe, Head, Emergency).

- A substitution σ is a mapping of the form:

$$\sigma : \mathsf{N_v} \mapsto \text{the set of all } \mathcal{L} \text{ concept descriptions.}$$

  σ is extended to arbitrary concepts inductively (in $\mathcal{ALC}$):

$$\sigma(\top) := \top \qquad \sigma(A) := A, \text{ for all } A \in \mathsf{N_c}$$
$$\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D) \qquad \sigma(C \sqcup D) := \sigma(C) \sqcup \sigma(D)$$
$$\sigma(\exists r.C) := \exists r.\sigma(C) \qquad \sigma(\forall r.C) := \forall r.\sigma(C)$$

Unification in DLs. Formal definition

Let $\mathcal{L}$ be some description logic.

- The set $\mathsf{N_C}$ of concept names is partitioned into two sets:
  - $\mathsf{N_v}$: concept variables (like Head_injury and Severe_injury).
  - $\mathsf{N_c}$: concept constants (like Severe, Head, Emergency).

- A substitution σ is a mapping of the form:

$$\sigma : \mathsf{N_v} \mapsto \text{the set of all } \mathcal{L} \text{ concept descriptions.}$$

  σ is extended to arbitrary concepts inductively (in $\mathcal{ALC}$):

$$\sigma(\top) := \top \qquad \sigma(A) := A, \text{ for all } A \in \mathsf{N_c}$$
$$\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D) \qquad \sigma(C \sqcup D) := \sigma(C) \sqcup \sigma(D)$$
$$\sigma(\exists r.C) := \exists r.\sigma(C) \qquad \sigma(\forall r.C) := \forall r.\sigma(C)$$

Definition 1 ($\mathcal{L}$-unification)

An $\mathcal{L}$-unification problem is of the form:

$$\Gamma := \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}.$$

A substitution σ is a unifier of Γ if

$$\sigma(C_i) \equiv \sigma(D_i), \text{ for all } 1 \leq i \leq n.$$

Unification in DLs. Formal definition - TBoxes

Restricted to ground TBoxes: a general TBox $\mathcal{T}$ is ground if it contains no variables.

Unification in DLs. Formal definition - TBoxes

Restricted to ground TBoxes: a general TBox $\mathcal{T}$ is ground if it contains no variables.

Definition 2 ($\mathcal{L}$-unification w.r.t. a general TBox)

Let $\mathcal{T}$ be a general TBox that is ground. An $\mathcal{L}$-unification problem w.r.t. $\mathcal{T}$ is of the form:

$$\Gamma := \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}.$$

A substitution σ is a unifier of Γ w.r.t. $\mathcal{T}$ if

$$\sigma(C_i) \equiv_{\mathcal{T}} \sigma(D_i), \text{ for all } 1 \leq i \leq n.$$

Definition 1 corresponds to the special case where $\mathcal{T} = \emptyset$.

Unification in DLs. Formal definition - TBoxes

Restricted to ground TBoxes: a general TBox $\mathcal{T}$ is ground if it contains no variables.

Definition 2 ($\mathcal{L}$-unification w.r.t. a general TBox)

Let $\mathcal{T}$ be a general TBox that is ground. An $\mathcal{L}$-unification problem w.r.t. $\mathcal{T}$ is of the form:

$$\Gamma := \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}.$$

A substitution σ is a unifier of Γ w.r.t. $\mathcal{T}$ if

$$\sigma(C_i) \equiv_{\mathcal{T}} \sigma(D_i), \text{ for all } 1 \leq i \leq n.$$

Definition 1 corresponds to the special case where $\mathcal{T} = \emptyset$.

The decision problem

$\mathcal{L}$-Unification Decision Problem

Instance:	A ground general TBox $\mathcal{T}$ and an $\mathcal{L}$-unification problem Γ.
Question:	Is there a unifier σ of Γ w.r.t. $\mathcal{T}$?

Unification in DLs. Additional notions from unification theory

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

$\mathcal{M}$ is called minimal, **iff** $\mathcal{M}$ also satisfies:

- if $\sigma, \theta \in \mathcal{M}$, then $\sigma \preceq \theta$ implies $\sigma = \theta$.

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \text{ **iff** exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

$\mathcal{M}$ is called minimal, **iff** $\mathcal{M}$ also satisfies:

- if $\sigma, \theta \in \mathcal{M}$, then $\sigma \preceq \theta$ implies $\sigma = \theta$.

- Unification type

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

$\mathcal{M}$ is called minimal, **iff** $\mathcal{M}$ also satisfies:

- if $\sigma, \theta \in \mathcal{M}$, then $\sigma \preceq \theta$ implies $\sigma = \theta$.

- Unification type

An $\mathcal{L}$-unification problem Γ has type

- unitary **iff** it has a minimal complete $\mathcal{M}$ of size 1.

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

$\mathcal{M}$ is called minimal, **iff** $\mathcal{M}$ also satisfies:

- if $\sigma, \theta \in \mathcal{M}$, then $\sigma \preceq \theta$ implies $\sigma = \theta$.

- Unification type

An $\mathcal{L}$-unification problem Γ has type

- unitary **iff** it has a minimal complete $\mathcal{M}$ of size 1.
- finitary **iff** it has a finite minimal complete $\mathcal{M}$.

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

$\mathcal{M}$ is called minimal, **iff** $\mathcal{M}$ also satisfies:

- if $\sigma, \theta \in \mathcal{M}$, then $\sigma \preceq \theta$ implies $\sigma = \theta$.

- Unification type

An $\mathcal{L}$-unification problem Γ has type

- unitary **iff** it has a minimal complete $\mathcal{M}$ of size 1.
- finitary **iff** it has a finite minimal complete $\mathcal{M}$.
- finitary **iff** it has an infinite minimal complete $\mathcal{M}$.

Unification in DLs. Additional notions from unification theory

- Comparing unifiers. Instantiation pre-order $\preceq$.

Let θ, σ be two unifiers of an $\mathcal{L}$-unification problem Γ. We define,

$$\theta \preceq \sigma \textbf{ iff } \text{exists } \lambda \text{ s.t. } \sigma(X) \equiv \lambda(\theta(X)) \text{ for all } X \in \Gamma.$$

- Minimal complete set of unifiers.

A set of substitutions $\mathcal{M}$ is a complete set of unifiers of Γ **iff**

- $\sigma \in \mathcal{M}$ implies σ is a unifier of Γ.
- if σ is a unifier of Γ, then $\theta \preceq \sigma$ for some $\theta \in \mathcal{M}$.

$\mathcal{M}$ is called minimal, **iff** $\mathcal{M}$ also satisfies:

- if $\sigma, \theta \in \mathcal{M}$, then $\sigma \preceq \theta$ implies $\sigma = \theta$.

- Unification type

An $\mathcal{L}$-unification problem Γ has type

- unitary **iff** it has a minimal complete $\mathcal{M}$ of size 1.
- finitary **iff** it has a finite minimal complete $\mathcal{M}$.
- finitary **iff** it has an infinite minimal complete $\mathcal{M}$.
- zero **iff** it does not have a minimal complete $\mathcal{M}$.

Additional motivation $\rightarrow$ Unification modulo an equational theory

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

  concept var. → variable symbols
  concept const. → free constants

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

concept var. → variable symbols	concept constr. → function symbols
concept const. → free constants	$\Sigma = \{\wedge^2, {h_{r_1}}^1, \ldots, {h_{r_n}}^1, \mathbb{1}\}$

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

concept var. → variable symbols
concept const. → free constants

concept constr. → function symbols
$\Sigma = \{\wedge^2, {h_{r_1}}^1, \ldots, {h_{r_n}}^1, \mathbb{1}\}$

concept descriptions over $\mathsf{N_R} = \{r_1, \ldots, r_n\}$ ⟶ terms over Σ

Additional motivation $\rightarrow$ Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

concept var. $\rightarrow$ variable symbols
concept const. $\rightarrow$ free constants

concept constr. $\rightarrow$ function symbols
$\Sigma = \{\wedge^2, {h_{r_1}}^1, \ldots, {h_{r_n}}^1, \mathbb{1}\}$

concept descriptions over $\mathsf{N_R} = \{r_1, \ldots, r_n\}$ $\longrightarrow$ terms over Σ

$$A \sqcap \forall r_1.\top \sqcap \forall r_2.(X \sqcap B) \longrightarrow a \wedge h_{r_1}(\mathbb{1}) \wedge h_{r_2}(x \wedge b)$$

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

concept var. → variable symbols
concept const. → free constants

concept constr. → function symbols
$\Sigma = \{\wedge^2, {h_{r_1}}^1, \ldots, {h_{r_n}}^1, \mathbb{1}\}$

concept descriptions over $\mathsf{N_R} = \{r_1, \ldots, r_n\}$ ⟶ terms over Σ

$$A \sqcap \forall r_1.\top \sqcap \forall r_2.(X \sqcap B) \longrightarrow a \wedge h_{r_1}(\mathbb{1}) \wedge h_{r_2}(x \wedge b)$$

- Equational theory ACUIh → axiomatizes equivalence in $\mathcal{FL}_0$

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

concept var. → variable symbols
concept const. → free constants

concept constr. → function symbols
$\Sigma = \{\wedge^2, {h_{r_1}}^1, \ldots, {h_{r_n}}^1, \mathbb{1}\}$

concept descriptions over $\mathsf{N_R} = \{r_1, \ldots, r_n\}$ ⟶ terms over Σ

$A \sqcap \forall r_1.\top \sqcap \forall r_2.(X \sqcap B) \longrightarrow a \wedge h_{r_1}(\mathbb{1}) \wedge h_{r_2}(x \wedge b)$

- Equational theory ACUIh → axiomatizes equivalence in $\mathcal{FL}_0$

$\sqcap$ is
commutative → $x \wedge y \approx y \wedge x$,
associative → $(x \wedge y) \wedge z \approx x \wedge (y \wedge z)$,
idempotent → $x \wedge x \approx x$,
has $\top$ as unit → $x \wedge \mathbb{1} = x$.

Additional motivation → Unification modulo an equational theory

- For many DLs, $\equiv$ can be axiomatized using finitely many equational axioms.
- Unification in such a DL can be viewed as unification modulo the corresponding equational theory.

Example (DL $\mathcal{FL}_0$, only $\top, \sqcap, \forall r.C$)

- Concept descriptions vs. terms:

concept var. → variable symbols
concept const. → free constants

concept constr. → function symbols
$\Sigma = \{\wedge^2, {h_{r_1}}^1, \ldots, {h_{r_n}}^1, \mathbb{1}\}$

concept descriptions over $\mathsf{N_R} = \{r_1, \ldots, r_n\}$ ⟶ terms over Σ

$$A \sqcap \forall r_1.\top \sqcap \forall r_2.(X \sqcap B) \longrightarrow a \wedge h_{r_1}(\mathbb{1}) \wedge h_{r_2}(x \wedge b)$$

- Equational theory ACUIh → axiomatizes equivalence in $\mathcal{FL}_0$

$\sqcap$ is
commutative → $x \wedge y \approx y \wedge x$,
associative → $(x \wedge y) \wedge z \approx x \wedge (y \wedge z)$,
idempotent → $x \wedge x \approx x$,
has $\top$ as unit → $x \wedge \mathbb{1} = x$.

$\forall r_i$ satisfies
$\forall r_i.\top \equiv \top$ → $h_{r_i}(\mathbb{1}) = \mathbb{1}$
$\forall r_i(C \sqcap D) \equiv \forall r_i.C \sqcap \forall r_i.D$
↓
$h_{r_i}(x \wedge y) \approx h_{r_i}(x) \wedge h_{r_i}(y)$

Additional motivation → Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Additional motivation → Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Results can be transferred

Additional motivation → Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Results can be transferred

$C \equiv^? D$ has a unifier **iff** $\tau(C)$ and $\tau(D)$ are unifiable.

Additional motivation → Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Results can be transferred

$C \equiv^? D$ has a unifier **iff** $\tau(C)$ and $\tau(D)$ are unifiable.

$\Rightarrow$

Unification in ACUIh is ExpTime-complete.

Additional motivation → Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Results can be transferred

$C \equiv^? D$ has a unifier **iff** $\tau(C)$ and $\tau(D)$ are unifiable.

$\Rightarrow$

Unification in ACUIh is ExpTime-complete.

$\mathcal{FL}_0$ has unification type zero (ACUIh has unification type zero [Baa93].)

Additional motivation $\rightarrow$ Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Results can be transferred

$C \equiv^? D$ has a unifier **iff** $\tau(C)$ and $\tau(D)$ are unifiable.

$\Rightarrow$

Unification in ACUIh is ExpTime-complete.

$\mathcal{FL}_0$ has unification type zero (ACUIh has unification type zero [Baa93].)

Unification w.r.t. a TBox $\rightarrow$ unification w.r.t. an additional set of ground identities:

$$\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\} \rightarrow G_{\mathcal{T}} = \bigcup_{i=1}^{n} \{\tau(C_i) \wedge \tau(D_i) \approx \tau(D_i)\}.$$

Additional motivation → Unification modulo an equational theory

Lemma 3 [BN01]

Let C and D be two $\mathcal{FL}_0$ concept descriptions. Then,

$$C \equiv D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh}} \tau(D).$$

Results can be transferred

$C \equiv^? D$ has a unifier **iff** $\tau(C)$ and $\tau(D)$ are unifiable.

$\Rightarrow$

Unification in ACUIh is ExpTime-complete.

$\mathcal{FL}_0$ has unification type zero (ACUIh has unification type zero [Baa93].)

Unification w.r.t. a TBox → unification w.r.t. an additional set of ground identities:

$$\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\} \rightarrow G_{\mathcal{T}} = \bigcup_{i=1}^{n} \{\tau(C_i) \wedge \tau(D_i) \approx \tau(D_i)\}.$$

$$C \equiv_{\mathcal{T}} D \textbf{ iff } \tau(C) \approx_{\mathsf{ACUIh} \cup G_{\mathcal{T}}} \tau(D).$$

Research on unification in DLs

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

- First investigated for $\mathcal{FL}_0$: ExpTime-complete w.r.t. the empty TBox.

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

- First investigated for $\mathcal{FL}_0$: ExpTime-complete w.r.t. the empty TBox.
- In $\mathcal{EL}$, the problem is easier: NP-complete.

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

- First investigated for $\mathcal{FL}_0$: ExpTime-complete w.r.t. the empty TBox.
- In $\mathcal{EL}$, the problem is easier: NP-complete.
- Only a few results exist for unification w.r.t. arbitrary TBoxes.

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

- First investigated for $\mathcal{FL}_0$: ExpTime-complete w.r.t. the empty TBox.
- In $\mathcal{EL}$, the problem is easier: NP-complete.
- Only a few results exist for unification w.r.t. arbitrary TBoxes.

Boolean DLs

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

- First investigated for $\mathcal{FL}_0$: ExpTime-complete w.r.t. the empty TBox.
- In $\mathcal{EL}$, the problem is easier: NP-complete.
- Only a few results exist for unification w.r.t. arbitrary TBoxes.

Boolean DLs

- Important open problem: unification in the nomal modal logic K (syntactic variant of $\mathcal{ALC}$).

Research on unification in DLs

Mostly concentrated in sub-Boolean fragments of $\mathcal{ALC}$

- First investigated for $\mathcal{FL}_0$: ExpTime-complete w.r.t. the empty TBox.
- In $\mathcal{EL}$, the problem is easier: NP-complete.
- Only a few results exist for unification w.r.t. arbitrary TBoxes.

Boolean DLs

- Important open problem: unification in the nomal modal logic K (syntactic variant of $\mathcal{ALC}$).
- Undecidability results for very expressive DLs: transferred from research in Modal Logics.

References I

Franz Baader.
Unification in commutative theories, hilbert's basis theorem, and gröbner bases.
J. ACM, 40(3):477–503, 1993.

Franz Baader and Paliath Narendran.
Unification of Concept Terms in Description Logics.
J. Symb. Comput., 31(3):277–305, 2001.

DONALD E. KNUTH and PETER B. BENDIX.
Simple word problems in universal algebras.
In JOHN LEECH, editor, *Computational Problems in Abstract Algebra*, pages 263 – 297. Pergamon, 1970.

John Alan Robinson.
A machine-oriented logic based on the resolution principle.
J. ACM, 12(1):23–41, 1965.