Description Logics

Carlos Areces
carlos.areces@gmail.com

Spring Term 2018
Stanford

What we do Today

- Structuring Knowledge

What we do Today

- Structuring Knowledge
- DLs and their history

What we do Today

- Structuring Knowledge
- DLs and their history
- Applications

What we do Today

- Structuring Knowledge
- DLs and their history
- Applications
- Syntax and Semantics

What we do Today

- Structuring Knowledge
- DLs and their history
- Applications
- Syntax and Semantics
- Constraint Systems

What we do Today

- Structuring Knowledge
- DLs and their history
- Applications
- Syntax and Semantics
- Constraint Systems
- Optimizations

Structuring Knowledge

- Consider the following claims

  Claim 1: *Information about anything abounds.*

Structuring Knowledge

- Consider the following claims

  Claim 1: *Information about anything abounds.*

  For a test, think of any word and run a query on Google. It doesn't matter which word, you will get hundreds of hits.

Structuring Knowledge

- Consider the following claims

  Claim 1: *Information about anything abounds.*

  For a test, think of any word and run a query on Google. It doesn't matter which word, you will get hundreds of hits.

- The problem is

  Claim 2: *Mere information is not interesting.*

Structuring Knowledge

- Consider the following claims

  Claim 1: *Information about anything abounds.*

  For a test, think of any word and run a query on Google. It doesn't matter which word, you will get hundreds of hits.

- The problem is

  Claim 2: *Mere information is not interesting.*

  Only few of the hundreds of hits are actually relevant.

Structuring Knowledge

- In other words,

  Claim 3: *Information should be structured to be useful.*

Structuring Knowledge

- In other words,

  Claim 3: *Information should be structured to be useful.*

  So that we can decide which part of it is important to our problem.

Structuring Knowledge

- In other words,

  Claim 3: *Information should be structured to be useful.*

  So that we can decide which part of it is important to our problem.

- But

  Claim 4: *Classifying information is often a difficult and expensive task.*

Structure, the Key to Knowledge

- Structured representation of knowledge aims to address both conceptual and computational complexity.

Structure, the Key to Knowledge

- Structured representation of knowledge aims to address both conceptual and computational complexity.
  - *Conceptual economy* amounts to building hierarchical structures, where inheritance of attributes through the hierarchy is used to avoid redundancies in the representation.

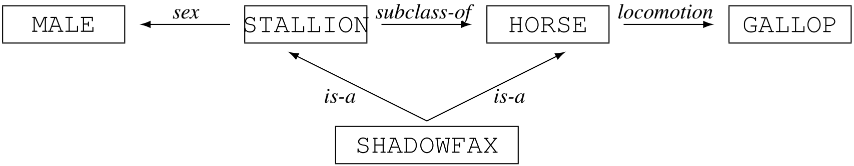
Structure, the Key to Knowledge

- Structured representation of knowledge aims to address both conceptual and computational complexity.
  - *Conceptual economy* amounts to building hierarchical structures, where inheritance of attributes through the hierarchy is used to avoid redundancies in the representation.
  - *Computational economy* refers to the efficiency of reasoning with such structures.

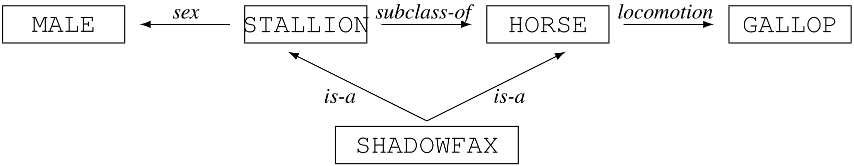
Structure, the Key to Knowledge

- Structured representation of knowledge aims to address both conceptual and computational complexity.
  - *Conceptual economy* amounts to building hierarchical structures, where inheritance of attributes through the hierarchy is used to avoid redundancies in the representation.
  - *Computational economy* refers to the efficiency of reasoning with such structures.
- The idea of developing knowledge representation systems based on a structured representation of knowledge has been pursued for a long time in Artificial Intelligence.

Semantic Networks

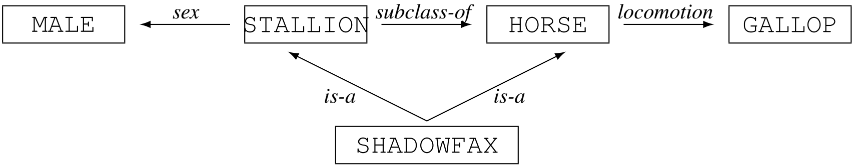


Semantic Networks



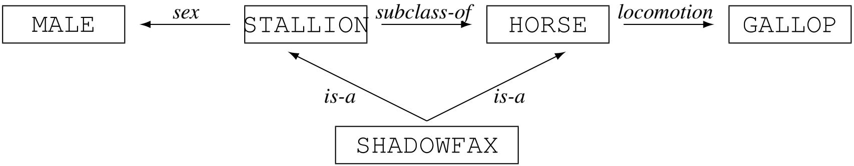
- Woods [1975] defines two main types of arcs:

Semantic Networks



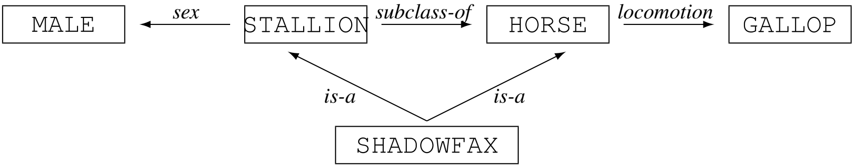
- Woods [1975] defines two main types of arcs:
  - *Intensional.* an arc which contributes to the definition of a concept carries intensional knowledge: *locomotion*

Semantic Networks



- Woods [1975] defines two main types of arcs:
  - *Intensional.* an arc which contributes to the definition of a concept carries intensional knowledge: *locomotion*
  - *Extensional.* an arc which assert membership in a class carries extensional knowledge: *is-a*

Semantic Networks



- Woods [1975] defines two main types of arcs:
  - *Intensional.* an arc which contributes to the definition of a concept carries intensional knowledge: *locomotion*
  - *Extensional.* an arc which assert membership in a class carries extensional knowledge: *is-a*

But things were unclear.

Meaning What?

- The need for a formal semantics was clear. Brachman [1977, 1979] are some of the early references which aimed to address this problem.

Meaning What?

- The need for a formal semantics was clear. Brachman [1977, 1979] are some of the early references which aimed to address this problem.
- **And DLs were born:** Description logics (DLs) are a family of formal languages with a clearly specified semantics, usually in terms of first-order models, together with specialized inference mechanisms to account for knowledge classification.

Meaning What?

- The need for a formal semantics was clear. Brachman [1977, 1979] are some of the early references which aimed to address this problem.
- **And DLs were born:** Description logics (DLs) are a family of formal languages with a clearly specified semantics, usually in terms of first-order models, together with specialized inference mechanisms to account for knowledge classification.
  One of the original aims of the research in this field is to identify fragments of FO able to capture the features needed for representing a particular problem, and which can still allow for the design of efficient reasoning algorithms.

Description Logics

Description Logics

- Description Logics (DL) are formal languages which are specially tailored for knowledge representation.

Description Logics

- Description Logics (DL) are formal languages which are specially tailored for knowledge representation.
- They originate from the Quillian's Semantic Networks and Minsky's Frame paradigm.
- Their main characteristics are:
  - A simple to use language (an extension of the propositional language, without variables);

Description Logics

- Description Logics (DL) are formal languages which are specially tailored for knowledge representation.
- They originate from the Quillian's Semantic Networks and Minsky's Frame paradigm.
- Their main characteristics are:
  - A simple to use language (an extension of the propositional language, without variables);
  - But that includes a notion of quantification (guarded quantification);

Description Logics

- Description Logics (DL) are formal languages which are specially tailored for knowledge representation.
- They originate from the Quillian's Semantic Networks and Minsky's Frame paradigm.
- Their main characteristics are:
  - A simple to use language (an extension of the propositional language, without variables);
  - But that includes a notion of quantification (guarded quantification);
  - With special operators chosen to facilitate the enunciation of definitions;

Description Logics

- Description Logics (DL) are formal languages which are specially tailored for knowledge representation.
- They originate from the Quillian's Semantic Networks and Minsky's Frame paradigm.
- Their main characteristics are:
  - A simple to use language (an extension of the propositional language, without variables);
  - But that includes a notion of quantification (guarded quantification);
  - With special operators chosen to facilitate the enunciation of definitions;
  - With a good balance between expressivity and tractability;

Description Logics

- Description Logics (DL) are formal languages which are specially tailored for knowledge representation.
- They originate from the Quillian's Semantic Networks and Minsky's Frame paradigm.
- Their main characteristics are:
  - A simple to use language (an extension of the propositional language, without variables);
  - But that includes a notion of quantification (guarded quantification);
  - With special operators chosen to facilitate the enunciation of definitions;
  - With a good balance between expressivity and tractability;
  - With highly optimized inference systems.

Description Logics

Description Logics

In a DL we have operators to build definitions using individuals, concepts and roles:

- Individuals are "objects" in a given universe.

Description Logics

In a DL we have operators to build definitions using individuals, concepts and roles:

- Individuals are "objects" in a given universe.
- Concepts correspond to "classes of objects" and will be interpreted as sets in a given universe.

Description Logics

In a DL we have operators to build definitions using individuals, concepts and roles:

- Individuals are "objects" in a given universe.
- Concepts correspond to "classes of objects" and will be interpreted as sets in a given universe.
- Roles correspond to "links between objects" and will be interpreted as binary relations over a given universe.

Description Logics

In a DL we have operators to build definitions using individuals, concepts and roles:

- Individuals are "objects" in a given universe.
- Concepts correspond to "classes of objects" and will be interpreted as sets in a given universe.
- Roles correspond to "links between objects" and will be interpreted as binary relations over a given universe.

Example: The "Happy Father"

$$\mathsf{Concepts} = \{\ \mathsf{Man},\ \mathsf{Woman},\ \mathsf{Happy},\ \mathsf{Rich}\ \}$$

$$\mathsf{Roles} = \{\ \mathsf{has\text{-}children}\ \}$$

$$\mathsf{Individuals} = \{\ \mathsf{carlos}\ \}$$

$$\begin{aligned}\mathsf{HappyFather} \equiv \mathsf{Man}\ &\wedge \exists\, \mathsf{has\text{-}children.Man} \wedge \\ &\exists\, \mathsf{has\text{-}children.Woman} \wedge \\ &\forall\, \mathsf{has\text{-}children.(Happy} \vee \mathsf{Rich)}\end{aligned}$$

$\mathsf{carlos}{:}\neg\mathsf{HappyFather}$

Counterexample!!!

Application Areas

Application Areas

- Terminological Knowledge Bases and Ontologies
  - DLs were created exactly for this task
  - Specially useful as a language to define and maintain ontologies

Application Areas

- Terminological Knowledge Bases and Ontologies
  - DLs were created exactly for this task
  - Specially useful as a language to define and maintain ontologies
- Semantic Web
  - To add 'semantic markup' to the information in the web.
  - Such markup would use ontological repositories as a store of common definitions with clear semantics
  - DL inference systems would be used for the development, mantainment and merging of these ontologies, and for the dynamic evolution of resources (e.g. search).

Application Areas

- Terminological Knowledge Bases and Ontologies
  - DLs were created exactly for this task
  - Specially useful as a language to define and maintain ontologies
- Semantic Web
  - To add ‘semantic markup’ to the information in the web.
  - Such markup would use ontological repositories as a store of common definitions with clear semantics
  - DL inference systems would be used for the development, mantainment and merging of these ontologies, and for the dynamic evolution of resources (e.g. search).
- Computational Linguistics
  - Many tasks in computational linguistics require inference and ‘background knowledge’: reference resolution, question/answering.
  - In some cases, the expressive power of DLs is enough and we don’t need to move to FOL.

Terminological KR and Ontologies

- General requirement for medical terminologies

Terminological KR and Ontologies

- General requirement for medical terminologies
- Static lists/taxonomies difficult to build and maintain
  - Need to be very large and highly interconnected
  - Inevitably contain many errors and omissions

Terminological KR and Ontologies

- General requirement for medical terminologies
- Static lists/taxonomies difficult to build and maintain
  - Need to be very large and highly interconnected
  - Inevitably contain many errors and omissions
- Galen project replaced static hierarchy with DL
  - Describe concepts (e.g., spiral fracture of left femur)
  - Use DL classifier to build taxonomy

Terminological KR and Ontologies

- General requirement for medical terminologies
- Static lists/taxonomies difficult to build and maintain
  - Need to be very large and highly interconnected
  - Inevitably contain many errors and omissions
- Galen project replaced static hierarchy with DL
  - Describe concepts (e.g., spiral fracture of left femur)
  - Use DL classifier to build taxonomy
- Needed expressive DL and efficient reasoning
  - Descriptions used transitive roles, inverses, GCIs, etc.
  - Even prototype KB was very large ($\sim$ 3.000 concepts)
  - Existing classifier took $\sim$24 hours to classify KB
  - FaCT system takes $\sim$60 seconds.

The Semantic Web

Most existing Web resources are only understandable by humans

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption
- Semantic Web aims at machine understandability

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption
- Semantic Web aims at machine understandability
  - *Semantic* markup will be added to web resources

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption
- Semantic Web aims at machine understandability
  - *Semantic* markup will be added to web resources
  - Markup will use *Ontologies* for shared understanding

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption
- Semantic Web aims at machine understandability
  - *Semantic* markup will be added to web resources
  - Markup will use *Ontologies* for shared understanding
  - Requirements for DAML ontology language

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption
- Semantic Web aims at machine understandability
  - *Semantic* markup will be added to web resources
  - Markup will use *Ontologies* for shared understanding
  - Requirements for DAML ontology language
  - Should extend existing Web standards (XML, RDF, RDFS)

The Semantic Web

Most existing Web resources are only understandable by humans

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption
- Semantic Web aims at machine understandability
  - *Semantic* markup will be added to web resources
  - Markup will use *Ontologies* for shared understanding
  - Requirements for DAML ontology language
  - Should extend existing Web standards (XML, RDF, RDFS)
  - DAML+Oil

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., ER Diagrams).

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (*n*-ary DL) can capture semantics of many datamodelling methodologies (e.g., ER Diagrams).
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., ER Diagrams).
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners
- DL ABox can also capture semantics of conjunctive queries
  - Hence we can reason about query containment w.r.t. schema

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., ER Diagrams).
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners
- DL ABox can also capture semantics of conjunctive queries
  - Hence we can reason about query containment w.r.t. schema
- DL reasoning can be used to support, e.g.,
  - Schema design and integration
  - Query optimization
  - Interoperability and federation

Short Story of DL

Short Story of DL

- 1st Stage:

  Incomplete Systems (Back, Classic, Loom, ...)
  Based in structural algorithms

Short Story of DL

- 1st Stage:
  - Incomplete Systems (BACK, CLASSIC, LOOM, …)
  - Based in structural algorithms
- 2nd Stage:
  - Development of tableaux algorithms and first complexity results
  - Tableaux based systems for PSpace complete logics (KRIS, CRACK)
  - Research in optimization techniques

Short Story of DL

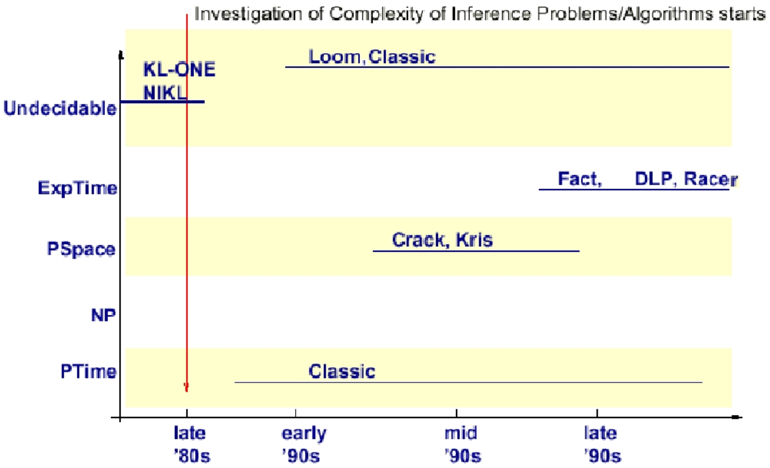
- 1st Stage:
  - Incomplete Systems (BACK, CLASSIC, LOOM, ...)
  - Based in structural algorithms
- 2nd Stage:
  - Development of tableaux algorithms and first complexity results
  - Tableaux based systems for PSpace complete logics (KRIS, CRACK)
  - Research in optimization techniques
- 3rd Stage:
  - Tableaux algorithms for very expressive DLs
  - Tableau based systems with many optimizations for ExpTime Logics (FACT, DLP, RACER, PELLET)
  - Relation with modal logics and fragments of FOL

Short Story of DL

- 1st Stage:
  - Incomplete Systems (BACK, CLASSIC, LOOM, . . .)
  - Based in structural algorithms
- 2nd Stage:
  - Development of tableaux algorithms and first complexity results
  - Tableaux based systems for PSpace complete logics (KRIS, CRACK)
  - Research in optimization techniques
- 3rd Stage:
  - Tableaux algorithms for very expressive DLs
  - Tableau based systems with many optimizations
    for ExpTime Logics (FACT, DLP, RACER, PELLET)
  - Relation with modal logics and fragments of FOL
- 4th Stage:
  - Mature implementations (Commercial!)
  - Applications and tools start to be widely used (e.g., Semantic Web).

System Development

System Development



Terminological Axioms and Assertions

- *Terminological information*: definitions of the basic and derived notions and of the ways they are inter-related. This information is “generic” or “global,” been true in every model of the situation and of every individual in the situation. Or

Terminological Axioms and Assertions

- *Terminological information*: definitions of the basic and derived notions and of the ways they are inter-related. This information is “generic” or “global,” been true in every model of the situation and of every individual in the situation. Or
- *Assertional information*: which records “specific” or “local” information, being true of certain particular individuals in the situation.

Terminological Axioms and Assertions

- *Terminological information*: definitions of the basic and derived notions and of the ways they are inter-related. This information is "generic" or "global," been true in every model of the situation and of every individual in the situation. Or
- *Assertional information*: which records "specific" or "local" information, being true of certain particular individuals in the situation.
- All known information is then modeled as a pair $\langle T, A \rangle$, where T is a set of formulas concerning terminological information (the T-Box) and A is a set of formulas concerning assertional information (the A-Box).

A Layered Language

A Layered Language

- The language is defined in three steps.

A Layered Language

- The language is defined in three steps.
  - Concepts: we construct complex concepts using other concepts (atomics or introduced via definitions) and roles: E.g., $\exists$has-children.Man

A Layered Language

- The language is defined in three steps.
  - Concepts: we construct complex concepts using other concepts (atomics or introduced via definitions) and roles: E.g., $\exists$has-children.Man
  - Definitions: we use concepts to build definitions (or relations between definitions): E.g., HappyFather $\equiv \ldots$

A Layered Language

- The language is defined in three steps.
  - Concepts: we construct complex concepts using other concepts (atomics or introduced via definitions) and roles: E.g., $\exists\mathsf{has\text{-}children.Man}$
  - Definitions: we use concepts to build definitions (or relations between definitions): E.g., $\mathsf{HappyFather} \equiv \ldots$
  - Assertions: assign concepts and roles to particular elements in our model: E.g., $\mathsf{carlos{:}\neg HappyFather}$
- A knowledge base is a pair $\langle D, A\rangle$ with D a set of definitions (the "T-Box") and A a set of assertions (la "A-Box").

A Layered Language

- The language is defined in three steps.
  - Concepts: we construct complex concepts using other concepts (atomics or introduced via definitions) and roles: E.g., $\exists$has-children.Man
  - Definitions: we use concepts to build definitions (or relations between definitions): E.g., HappyFather $\equiv \ldots$
  - Assertions: assign concepts and roles to particular elements in our model: E.g., carlos:$\neg$HappyFather
- A knowledge base is a pair $\langle D, A\rangle$ with D a set of definitions (the "T-Box") and A a set of assertions (la "A-Box").
- A DL inference system takes a knowledge base $\langle D, A\rangle$ and can solve questions like:

$\langle D, A\rangle \models a : C$	"Given $\langle D, A\rangle$, is concept C applicable to a?"
$\langle D, A\rangle \models C \equiv D$	"Given $\langle D, A\rangle$, are concepts C and D quivalent?"

Concept Construction: $\mathcal{ALC}$

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.
  - An atomic concept: Man, Woman

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.
  - An atomic concept: Man, Woman
  - Boolean Operators: If C and D are concepts the the following are concepts

$C \wedge D$	the conjunction of C and D	Rich $\wedge$ Handsome

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.
  - An atomic concept: Man, Woman
  - Boolean Operators: If C and D are concepts the the following are concepts

$C \wedge D$	the conjunction of C and D	Rich $\wedge$ Handsome
$C \vee D$	the disjunction of C and D	Rich $\vee$ Handsome

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.
  - An atomic concept: Man, Woman
  - Boolean Operators: If C and D are concepts the the following are concepts

$C \wedge D$	the conjunction of C and D	Rich $\wedge$ Handsome
$C \vee D$	the disjunction of C and D	Rich $\vee$ Handsome
$\neg C$	the negation of C	$\neg$Rich

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.
  - An atomic concept: Man, Woman
  - Boolean Operators: If C and D are concepts the the following are concepts

$C \wedge D$	the conjunction of C and D	Rich $\wedge$ Handsome
$C \vee D$	the disjunction of C and D	Rich $\vee$ Handsome
$\neg C$	the negation of C	$\neg$Rich

  - Relational Operators: if C is a concept and R is a role, the following are concepts

$\forall R.C$	each element acc. through R is in C	$\forall$has-children.Woman

Concept Construction: $\mathcal{ALC}$

- A concept can be
  - $\top$, the trivial concept, of which every element is a member.
  - An atomic concept: Man, Woman
  - Boolean Operators: If C and D are concepts the the following are concepts

$C \wedge D$	the conjunction of C and D	Rich $\wedge$ Handsome
$C \vee D$	the disjunction of C and D	Rich $\vee$ Handsome
$\neg C$	the negation of C	$\neg$Rich

  - Relational Operators: if C is a concept and R is a role, the following are concepts

$\forall R.C$	each element acc. through R is in C	$\forall$has-children.Woman
$\exists R.C$	some element acc. through R is in C	$\exists$has-children.Woman

Building Definitions

Building Definitions

Given two concepts C and D, there are two types of definitions:

- Partial Definitions: $C \sqsubseteq D$. Conditions specified in C are sufficient to qualify elements in C as members of D, but they are not necessary; or vice-versa.

Building Definitions

Given two concepts C and D, there are two types of definitions:

- Partial Definitions: $C \sqsubseteq D$. Conditions specified in C are sufficient to qualify elements in C as members of D, but they are not necessary; or vice-versa.

$$\exists\textsf{has-children}.\textsf{Man} \wedge \exists\textsf{has-children}.\textsf{Woman} \quad \sqsubseteq \quad \textsf{BusyFather} \qquad \text{(suff. condition)}$$

Building Definitions

Given two concepts C and D, there are two types of definitions:

- Partial Definitions: $C \sqsubseteq D$. Conditions specified in C are sufficient to qualify elements in C as members of D, but they are not necessary; or vice-versa.

$$\exists \mathsf{has\text{-}children.Man} \wedge \exists \mathsf{has\text{-}children.Woman} \sqsubseteq \mathsf{BusyFather} \quad \text{(suff. condition)}$$

$$\mathsf{BusyFather} \sqsubseteq \exists \mathsf{has\text{-}children}.\top \quad \text{(nec. condition)}$$

Building Definitions

Given two concepts C and D, there are two types of definitions:

- Partial Definitions: $C \sqsubseteq D$. Conditions specified in C are sufficient to qualify elements in C as members of D, but they are not necessary; or vice-versa.

$$\exists \mathsf{has\text{-}children.Man} \wedge \exists \mathsf{has\text{-}children.Woman} \sqsubseteq \mathsf{BusyFather} \quad \text{(suff. condition)}$$

$$\mathsf{BusyFather} \sqsubseteq \exists \mathsf{has\text{-}children}.\top \quad \text{(nec. condition)}$$

- Total Definitions: $C \equiv D$. Conditions indicated in D are both necessary and sufficient to qualify elements of D as elements of C (and vice-versa). Concepts C and D are equivalent.

Building Definitions

Given two concepts C and D, there are two types of definitions:

- Partial Definitions: $C \sqsubseteq D$. Conditions specified in C are sufficient to qualify elements in C as members of D, but they are not necessary; or vice-versa.

$$\exists\mathsf{has\text{-}children.Man} \wedge \exists\mathsf{has\text{-}children.Woman} \sqsubseteq \mathsf{BusyFather} \quad \text{(suff. condition)}$$

$$\mathsf{BusyFather} \sqsubseteq \exists\mathsf{has\text{-}children}.\top \quad \text{(nec. condition)}$$

- Total Definitions: $C \equiv D$. Conditions indicated in D are both necessary and sufficient to qualify elements of D as elements of C (and vice-versa). Concepts C and D are equivalent.

$$\mathsf{GrandMother} \equiv \mathsf{Woman} \wedge \exists\mathsf{has\text{-}children}.\exists\mathsf{has\text{-}children}.\top$$

  (Equivalent to say both $C \sqsubseteq D$ and $D \sqsubseteq C$)

Building Assertions

Building Assertions

We can "assign assertions" to particular elements in the situation we are describing.

Building Assertions

We can "assign assertions" to particular elements in the situation we are describing.
Given elements a and b, a concept C and a relation R

- Assigning elements to concepts: $a{:}C$. Indicates that C is true of a. I.e., all conditions indicated in C apply to a.

Building Assertions

We can "assign assertions" to particular elements in the situation we are describing.
Given elements a and b, a concept C and a relation R

- Assigning elements to concepts: $a{:}C$. Indicates that C is true of a. I.e., all conditions indicated in C apply to a.

carlos:Argentine

Building Assertions

We can “assign assertions” to particular elements in the situation we are describing.

Given elements a and b, a concept C and a relation R

- Assigning elements to concepts: $a{:}C$. Indicates that C is true of a. I.e., all conditions indicated in C apply to a.

$$\text{carlos:Argentine}$$

$$\text{carlos:(Argentine} \wedge \exists\text{Lives-in.US)}$$

Building Assertions

We can "assign assertions" to particular elements in the situation we are describing.
Given elements a and b, a concept C and a relation R

- Assigning elements to concepts: $a{:}C$. Indicates that C is true of a. I.e., all conditions indicated in C apply to a.

$$\text{carlos:Argentine}$$
$$\text{carlos:(Argentine} \wedge \exists\text{Lives-in.US)}$$

- Assigning elements to relations: $(a, b){:}R$. Indicates that the elements a and b are related via the role R.

Building Assertions

We can "assign assertions" to particular elements in the situation we are describing.
Given elements a and b, a concept C and a relation R

- Assigning elements to concepts: $a{:}C$. Indicates that C is true of a. I.e., all conditions indicated in C apply to a.

$$\text{carlos:Argentine}$$

$$\text{carlos:(Argentine} \land \exists\text{Lives-in.US)}$$

- Assigning elements to relations: $(a, b){:}R$. Indicates that the elements a and b are related via the role R.

$$\text{(carlos,stanford):Lives-in}$$

Semantics: Models

Semantics: Models

Let CON be a set of atomic concepts, ROL a set of roles and IND a set of individuals.

Semantics: Models

Let CON be a set of atomic concepts, ROL a set of roles and IND a set of individuals.

An interpretation or model for $\mathcal{ALC}$ is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that

- $\Delta^{\mathcal{I}}$ is an arbitrary non-empty set

Semantics: Models

Let CON be a set of atomic concepts, ROL a set of roles and IND a set of individuals.

An interpretation or model for $\mathcal{ALC}$ is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that

- $\Delta^{\mathcal{I}}$ is an arbitrary non-empty set
- $\cdot^{\mathcal{I}}$ is an interpretation function for atomic concepts, roles and individuals such that

$$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ for } C \in \mathsf{CON}$$

Semantics: Models

Let CON be a set of atomic concepts, ROL a set of roles and IND a set of individuals.

An interpretation or model for $\mathcal{ALC}$ is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that

- $\Delta^{\mathcal{I}}$ is an arbitrary non-empty set
- $\cdot^{\mathcal{I}}$ is an interpretation function for atomic concepts, roles and individuals such that

$$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ for } C \in \mathsf{CON}$$
$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \text{ for } R \in \mathsf{ROL}$$

Semantics: Models

Let CON be a set of atomic concepts, ROL a set of roles and IND a set of individuals.

An interpretation or model for $\mathcal{ALC}$ is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that

- $\Delta^{\mathcal{I}}$ is an arbitrary non-empty set
- $\cdot^{\mathcal{I}}$ is an interpretation function for atomic concepts, roles and individuals such that

$$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ for } C \in \mathsf{CON}$$
$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \text{ for } R \in \mathsf{ROL}$$
$$a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \text{ for } a \in \mathsf{IND}$$

Semantics: Models

Let CON be a set of atomic concepts, ROL a set of roles and IND a set of individuals.

An interpretation or model for $\mathcal{ALC}$ is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that

- $\Delta^{\mathcal{I}}$ is an arbitrary non-empty set
- $\cdot^{\mathcal{I}}$ is an interpretation function for atomic concepts, roles and individuals such that

$$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ for } C \in \mathsf{CON}$$
$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \text{ for } R \in \mathsf{ROL}$$
$$a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \text{ for } a \in \mathsf{IND}$$

(i.e., a DL model is nothing else than an FO model for the signature $\langle \mathsf{CON} \cup \mathsf{ROL}, \{\}, \mathsf{IND} \rangle$)

Semantics: Concepts

Semantics: Concepts

Given an interpretation $\mathcal{I}$ we can define the interpretation of an arbitrary $\mathcal{ALC}$ concept recursively as

$$
\begin{array}{rcl}
(\neg C)^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cup D^{\mathcal{I}}
\end{array}
$$

Semantics: Concepts

Given an interpretation $\mathcal{I}$ we can define the interpretation of an arbitrary $\mathcal{ALC}$ concept recursively as

$$
\begin{array}{rcl}
(\neg C)^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} & := & \{i \mid \exists j.(i,j) \in R^{\mathcal{I}} \text{ y } j \in C^{\mathcal{I}}\}
\end{array}
$$

Semantics: Concepts

Given an interpretation $\mathcal{I}$ we can define the interpretation of an arbitrary $\mathcal{ALC}$ concept recursively as

$$
\begin{array}{rcl}
(\neg C)^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} & := & \{i \mid \exists j.(i,j) \in R^{\mathcal{I}} \text{ y } j \in C^{\mathcal{I}}\}
\end{array}
$$

$C \sqcup D$ is equivalent to $\neg(\neg C \sqcap \neg D)$ and
$(\forall R.C)$ is equivalent to $\neg(\exists R.\neg C)$.

Semantics: Definitions and Assertions

Given an interpretation $\mathcal{I}$ we say that

- $\mathcal{I}$ satisfies a partial definition $C \sqsubseteq D$ (total definition $C \equiv D$) iff
$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$

Semantics: Definitions and Assertions

Given an interpretation $\mathcal{I}$ we say that

- $\mathcal{I}$ satisfies a partial definition $C \sqsubseteq D$ (total definition $C \equiv D$) iff
$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies a T-Box T iff it satisfies all definitions (partial or total) in T

Semantics: Definitions and Assertions

Given an interpretation $\mathcal{I}$ we say that

- $\mathcal{I}$ satisfies a partial definition $C \sqsubseteq D$ (total definition $C \equiv D$) iff
  $$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies a T-Box T iff it satisfies all definitions (partial or total) in T
- $\mathcal{I}$ satisfies an assertion $a{:}C$ ($(a, b){:}R$) iff
  $$a^{\mathcal{I}} \in C^{\mathcal{I}} \quad ((a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}})$$

Semantics: Definitions and Assertions

Given an interpretation $\mathcal{I}$ we say that

- $\mathcal{I}$ satisfies a partial definition $C \sqsubseteq D$ (total definition $C \equiv D$) iff
$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies a T-Box T iff it satisfies all definitions (partial or total) in T
- $\mathcal{I}$ satisfies an assertion $a{:}C$ ($(a, b){:}R$) iff
$$a^{\mathcal{I}} \in C^{\mathcal{I}} \quad ((a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies an A-Box A iff it satisfies all assertions in A

Semantics: Definitions and Assertions

Given an interpretation $\mathcal{I}$ we say that

- $\mathcal{I}$ satisfies a partial definition $C \sqsubseteq D$ (total definition $C \equiv D$) iff
$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies a T-Box T iff it satisfies all definitions (partial or total) in T
- $\mathcal{I}$ satisfies an assertion $a{:}C$ ($(a,b){:}R$) iff
$$a^{\mathcal{I}} \in C^{\mathcal{I}} \quad ((a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies an A-Box A iff it satisfies all assertions in A
- $\mathcal{I}$ satisfies a KB $K = \langle T, A \rangle$ iff $\mathcal{I}$ satisfies T and A.

Semantics: Definitions and Assertions

Given an interpretation $\mathcal{I}$ we say that

- $\mathcal{I}$ satisfies a partial definition $C \sqsubseteq D$ (total definition $C \equiv D$) iff
$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies a T-Box T iff it satisfies all definitions (partial or total) in T
- $\mathcal{I}$ satisfies an assertion $a{:}C$ ($(a,b){:}R$) iff
$$a^{\mathcal{I}} \in C^{\mathcal{I}} \quad ((a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}})$$
- $\mathcal{I}$ satisfies an A-Box A iff it satisfies all assertions in A
- $\mathcal{I}$ satisfies a KB $K = \langle T, A \rangle$ iff $\mathcal{I}$ satisfies T and A.

A KB K is consistent (or satisfiable) iff there is an interpretation $\mathcal{I}$ satisfying it.

A Complete Example

A Complete Example

$$
\begin{array}{rcl}
\mathsf{Woman} & \sqsubseteq & \mathsf{Person} \wedge \exists \mathsf{sex.Female} \\
\mathsf{Man} & \sqsubseteq & \mathsf{Person} \wedge \exists \mathsf{sex.Male} \\
\mathsf{FatherOrMother} & \equiv & \mathsf{Person} \wedge \exists \mathsf{has\text{-}children.Person} \\
\mathsf{Mother} & \equiv & \mathsf{Woman} \wedge \mathsf{FatherOrMother} \\
\mathsf{Father} & \equiv & \mathsf{Man} \wedge \mathsf{FatherOrMother}
\end{array}
$$

$$\mathsf{alice}:\mathsf{Mother}$$

$$(\mathsf{alice},\mathsf{betty}):\mathsf{has\text{-}children}$$

$$(\mathsf{alice},\mathsf{carlos}):\mathsf{has\text{-}children}$$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \to d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \geq n\}$
	$\leq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \leq n\}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \geq n\}$
	$\leq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \leq n\}$
one-of ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{d \mid d = a_i^{\mathcal{I}}$ for some $a_i\}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \geq n\}$
	$\leq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \leq n\}$
one-of ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{d \mid d = a_i^{\mathcal{I}}$ for some $a_i\}$
filler ($\mathcal{B}$)	$\exists R.\{a\}$	$\{d \mid d = R^{\mathcal{I}} d a^{\mathcal{I}}\}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \|\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\| \geq n\}$
	$\leq nR$	$\{d_1 \mid \|\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\| \leq n\}$
one-of ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{d \mid d = a_i^{\mathcal{I}}$ for some $a_i\}$
filler ($\mathcal{B}$)	$\exists R.\{a\}$	$\{d \mid d = R^{\mathcal{I}} d a^{\mathcal{I}}\}$
role name	R	$R^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \geq n\}$
	$\leq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \leq n\}$
one-of ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{d \mid d = a_i^{\mathcal{I}}$ for some $a_i\}$
filler ($\mathcal{B}$)	$\exists R.\{a\}$	$\{d \mid d = R^{\mathcal{I}} d a^{\mathcal{I}}\}$
role name	R	$R^{\mathcal{I}}$
role conj. ($\mathcal{R}$)	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \geq n\}$
	$\leq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \leq n\}$
one-of ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{d \mid d = a_i^{\mathcal{I}} \text{ for some } a_i\}$
filler ($\mathcal{B}$)	$\exists R.\{a\}$	$\{d \mid d = R^{\mathcal{I}} d a^{\mathcal{I}}\}$
role name	R	$R^{\mathcal{I}}$
role conj. ($\mathcal{R}$)	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$
inverse roles ($\mathcal{I}$)	R^{-1}	$\{(d_1, d_2) \mid R^{\mathcal{I}}(d_2, d_1)\}$

Syntax and Semantics: Summary

Constructor	Syntax	Semantics
concept name	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
univ. quant.	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \rightarrow d_2 \in C^{\mathcal{I}})\}$
exist. quant. ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}} d_1 d_2 \wedge d_2 \in C^{\mathcal{I}})\}$
number restr. ($\mathcal{N}$)	$\geq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \geq n\}$
	$\leq nR$	$\{d_1 \mid \lvert\{d_2 \mid R^{\mathcal{I}} d_1 d_2\}\rvert \leq n\}$
one-of ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{d \mid d = a_i^{\mathcal{I}}$ for some $a_i\}$
filler ($\mathcal{B}$)	$\exists R.\{a\}$	$\{d \mid d = R^{\mathcal{I}} d a^{\mathcal{I}}\}$
role name	R	$R^{\mathcal{I}}$
role conj. ($\mathcal{R}$)	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$
inverse roles ($\mathcal{I}$)	R^{-1}	$\{(d_1, d_2) \mid R^{\mathcal{I}}(d_2, d_1)\}$

Shadowfax, the Stallion

- Let Σ be a knowledge base $\langle T, A \rangle$ where

Shadowfax, the Stallion

- Let Σ be a knowledge base $\langle T, A \rangle$ where

$$T = \left\{ \begin{array}{rcl} \texttt{HORSES} & \sqsubseteq & \exists\texttt{Locomotion.GALLOP} \\ \texttt{STALLION} & \sqsubseteq & \forall\texttt{Sex.MALE} \\ \texttt{STALLION} & \sqsubseteq & \texttt{HORSE} \\ \texttt{HORSE} \sqcap \forall\texttt{Sex.MALE} & \sqsubseteq & \texttt{STALLION} \end{array} \right\}$$

Shadowfax, the Stallion

- Let Σ be a knowledge base $\langle T, A\rangle$ where

$$T = \left\{ \begin{array}{rcl} \texttt{HORSES} & \sqsubseteq & \exists\texttt{Locomotion.GALLOP} \\ \texttt{STALLION} & \sqsubseteq & \forall\texttt{Sex.MALE} \\ \texttt{STALLION} & \sqsubseteq & \texttt{HORSE} \\ \texttt{HORSE} \sqcap \forall\texttt{Sex.MALE} & \sqsubseteq & \texttt{STALLION} \end{array} \right\}$$

$$A = \{\texttt{shadowfax:STALLION}\}$$

Shadowfax, the Stallion

- Let Σ be a knowledge base $\langle T, A \rangle$ where

$$T = \left\{ \begin{array}{rcl} \texttt{HORSES} & \sqsubseteq & \exists\texttt{Locomotion.GALLOP} \\ \texttt{STALLION} & \sqsubseteq & \forall\texttt{Sex.MALE} \\ \texttt{STALLION} & \sqsubseteq & \texttt{HORSE} \\ \texttt{HORSE} \sqcap \forall\texttt{Sex.MALE} & \sqsubseteq & \texttt{STALLION} \end{array} \right\}$$

$$A = \{\texttt{shadowfax:STALLION}\}$$

- From Σ we can infer further information like, for example, that Shadowfax gallops

$$\Sigma \models \texttt{shadowfax:}\exists\texttt{Locomotion.GALLOPS}.$$

Shadowfax, the Stallion

- Let Σ be a knowledge base $\langle T, A \rangle$ where

$$T = \left\{ \begin{array}{rcl} \texttt{HORSES} & \sqsubseteq & \exists\texttt{Locomotion.GALLOP} \\ \texttt{STALLION} & \sqsubseteq & \forall\texttt{Sex.MALE} \\ \texttt{STALLION} & \sqsubseteq & \texttt{HORSE} \\ \texttt{HORSE} \sqcap \forall\texttt{Sex.MALE} & \sqsubseteq & \texttt{STALLION} \end{array} \right\}$$

$$A = \{\texttt{shadowfax:STALLION}\}$$

- From Σ we can infer further information like, for example, that Shadowfax gallops

$$\Sigma \models \texttt{shadowfax:}\exists\texttt{Locomotion.GALLOPS}.$$

a fact which is not explicit in the knowledge base.

Reasoning Tasks

Let Σ be a knowledge base, $C_1, C_2 \in \mathsf{CON}(\mathcal{L})$, $R \in \mathsf{ROL}(\mathcal{L})$ and $a, b \in \mathsf{IND}$, we define the following *reasoning tasks*

Reasoning Tasks

Let Σ be a knowledge base, $C_1, C_2 \in \mathsf{CON}(\mathcal{L})$, $R \in \mathsf{ROL}(\mathcal{L})$ and $a, b \in \mathsf{IND}$, we define the following *reasoning tasks*

- *Subsumption*, $\Sigma \models C_1 \sqsubseteq C_2$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.

Reasoning Tasks

Let Σ be a knowledge base, $C_1, C_2 \in \mathsf{CON}(\mathcal{L})$, $R \in \mathsf{ROL}(\mathcal{L})$ and $a, b \in \mathsf{IND}$, we define the following *reasoning tasks*

- *Subsumption*, $\Sigma \models C_1 \sqsubseteq C_2$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- *Instance Checking*, $\Sigma \models a{:}C$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

Reasoning Tasks

Let Σ be a knowledge base, $C_1, C_2 \in \mathsf{CON}(\mathcal{L})$, $R \in \mathsf{ROL}(\mathcal{L})$ and $a, b \in \mathsf{IND}$, we define the following *reasoning tasks*

- *Subsumption*, $\Sigma \models C_1 \sqsubseteq C_2$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- *Instance Checking*, $\Sigma \models a{:}C$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- *Relation Checking*, $\sigma \models (a, b){:}R$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}})$.

Reasoning Tasks

Let Σ be a knowledge base, $C_1, C_2 \in \mathsf{CON}(\mathcal{L})$, $R \in \mathsf{ROL}(\mathcal{L})$ and $a, b \in \mathsf{IND}$, we define the following *reasoning tasks*

- *Subsumption*, $\Sigma \models C_1 \sqsubseteq C_2$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- *Instance Checking*, $\Sigma \models a{:}C$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- *Relation Checking*, $\sigma \models (a, b){:}R$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}})$.
- *Concept Consistency*, $\Sigma \not\models C \doteq \perp$. Check whether for some interpretation $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C^{\mathcal{I}} \neq \{\}$.

Reasoning Tasks

Let Σ be a knowledge base, $C_1, C_2 \in \mathsf{CON}(\mathcal{L})$, $R \in \mathsf{ROL}(\mathcal{L})$ and $a, b \in \mathsf{IND}$, we define the following *reasoning tasks*

- *Subsumption*, $\Sigma \models C_1 \sqsubseteq C_2$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- *Instance Checking*, $\Sigma \models a{:}C$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- *Relation Checking*, $\sigma \models (a, b){:}R$. Check whether for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}})$.
- *Concept Consistency*, $\Sigma \not\models C \doteq \bot$. Check whether for some interpretation $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$ we have $C^{\mathcal{I}} \neq \{\}$.
- *Knowledge Base Consistency*, $\Sigma \not\models \bot$.
  Check whether there exists $\mathcal{I}$ such that $\mathcal{I} \models \Sigma$.

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C
  - The model is represented by a tree T

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C
  - The model is represented by a tree T
    - Nodes in T correspond to individuals in the model.
    - Nodes are labeled with sets of subconcepts of C.
    - Edges are labeled with role names in C.

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C
  - The model is represented by a tree T
    - Nodes in T correspond to individuals in the model.
    - Nodes are labeled with sets of subconcepts of C.
    - Edges are labeled with role names in C.
  - Start from the root node labeled $\{C\}$.

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C
  - The model is represented by a tree T
    - Nodes in T correspond to individuals in the model.
    - Nodes are labeled with sets of subconcepts of C.
    - Edges are labeled with role names in C.
  - Start from the root node labeled $\{C\}$.
  - Apply expansion rules to node labels until
    - Expansion completed (the tree represents a valid model)

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C
  - The model is represented by a tree T
    - Nodes in T correspond to individuals in the model.
    - Nodes are labeled with sets of subconcepts of C.
    - Edges are labeled with role names in C.
  - Start from the root node labeled $\{C\}$.
  - Apply expansion rules to node labels until
    - Expansion completed (the tree represents a valid model)
    - Contradictions prove there is no model

Subsumption and Satisfiability

- Subsumption is transformed into satisfiability
- A Tableaux algorithm (which in DL terms is often called a constraint system) is used to test satisfiability.
  - Try to build a model of a concept C
  - The model is represented by a tree T
    - Nodes in T correspond to individuals in the model.
    - Nodes are labeled with sets of subconcepts of C.
    - Edges are labeled with role names in C.
  - Start from the root node labeled $\{C\}$.
  - Apply expansion rules to node labels until
    - Expansion completed (the tree represents a valid model)
    - Contradictions prove there is no model
  - Many rules uses non-deterministic expansions $\Rightarrow$ search

Constraint Systems

- Let VAR be disjoint from IND, a *constraint* is a formula

$$s{:}C \mid (s,t){:}R \mid s \dot{\neq} t \mid \forall x.x{:}C,$$

where $s, t \in \mathsf{IND} \cup \mathsf{VAR}$, $C \in \mathsf{CON}(\mathcal{L})$ and $R \in \mathsf{ROL}(\mathcal{L})$.

Constraint Systems

- Let VAR be disjoint from IND, a *constraint* is a formula

$$s{:}C \mid (s,t){:}R \mid s \dot{\neq} t \mid \forall x.x{:}C,$$

  where $s, t \in \mathsf{IND} \cup \mathsf{VAR}$, $C \in \mathsf{CON}(\mathcal{L})$ and $R \in \mathsf{ROL}(\mathcal{L})$.
- Let $\mathcal{I}$ be an interpretation, an *$\mathcal{I}$-assignment* is a function α that maps every variable in VAR to an element of $\Delta^{\mathcal{I}}$ and every individual a in IND to $a^{\mathcal{I}}$.

Satisfaction of Constraints

- We use pairs $\langle \mathcal{I}, \alpha \rangle$ to define *satisfaction of constraints*. Let $s^{\langle \mathcal{I}, \alpha \rangle}$ be $s^{\mathcal{I}}$ if $s \in \mathsf{IND}$ and $\alpha(s)$ if $s \in \mathsf{VAR}$,

Satisfaction of Constraints

- We use pairs $\langle \mathcal{I}, \alpha \rangle$ to define *satisfaction of constraints*. Let $s^{\langle \mathcal{I}, \alpha \rangle}$ be $s^{\mathcal{I}}$ if $s \in \mathsf{IND}$ and $\alpha(s)$ if $s \in \mathsf{VAR}$,

  - $\langle \mathcal{I}, \alpha \rangle \models s{:}C$ iff $s^{\langle \mathcal{I}, \alpha \rangle} \in C^{\mathcal{I}}$,

Satisfaction of Constraints

- We use pairs $\langle \mathcal{I}, \alpha \rangle$ to define *satisfaction of constraints*. Let $s^{\langle \mathcal{I}, \alpha \rangle}$ be $s^{\mathcal{I}}$ if $s \in \mathsf{IND}$ and $\alpha(s)$ if $s \in \mathsf{VAR}$,
  - $\langle \mathcal{I}, \alpha \rangle \models s{:}C$ iff $s^{\langle \mathcal{I}, \alpha \rangle} \in C^{\mathcal{I}}$,
  - $\langle \mathcal{I}, \alpha \rangle \models (s, t){:}R$ iff $R^{\mathcal{I}}(s^{\langle \mathcal{I}, \alpha \rangle}, t^{\langle \mathcal{I}, \alpha \rangle})$,

Satisfaction of Constraints

- We use pairs $\langle \mathcal{I}, \alpha \rangle$ to define *satisfaction of constraints*. Let $s^{\langle \mathcal{I}, \alpha \rangle}$ be $s^{\mathcal{I}}$ if $s \in \mathsf{IND}$ and $\alpha(s)$ if $s \in \mathsf{VAR}$,

  - $\langle \mathcal{I}, \alpha \rangle \models s{:}C$ iff $s^{\langle \mathcal{I}, \alpha \rangle} \in C^{\mathcal{I}}$,
  - $\langle \mathcal{I}, \alpha \rangle \models (s, t){:}R$ iff $R^{\mathcal{I}}(s^{\langle \mathcal{I}, \alpha \rangle}, t^{\langle \mathcal{I}, \alpha \rangle})$,
  - $\langle \mathcal{I}, \alpha \rangle \models s \dot{\neq} t$ iff $s^{\langle \mathcal{I}, \alpha \rangle} \neq t^{\langle \mathcal{I}, \alpha \rangle}$,

Satisfaction of Constraints

- We use pairs $\langle \mathcal{I}, \alpha \rangle$ to define *satisfaction of constraints*. Let $s^{\langle \mathcal{I}, \alpha \rangle}$ be $s^{\mathcal{I}}$ if $s \in \mathsf{IND}$ and $\alpha(s)$ if $s \in \mathsf{VAR}$,
  - $\langle \mathcal{I}, \alpha \rangle \models s{:}C$ iff $s^{\langle \mathcal{I}, \alpha \rangle} \in C^{\mathcal{I}}$,
  - $\langle \mathcal{I}, \alpha \rangle \models (s, t){:}R$ iff $R^{\mathcal{I}}(s^{\langle \mathcal{I}, \alpha \rangle}, t^{\langle \mathcal{I}, \alpha \rangle})$,
  - $\langle \mathcal{I}, \alpha \rangle \models s \dot{\neq} t$ iff $s^{\langle \mathcal{I}, \alpha \rangle} \neq t^{\langle \mathcal{I}, \alpha \rangle}$,
  - $\langle \mathcal{I}, \alpha \rangle \models \forall x.x : C$ iff $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$.

Constraint Systems

- A *constraint system* CS is a finite, non-empty set of constraints. A pair $\langle \mathcal{I}, \alpha \rangle$ *satisfies* CS if $\langle \mathcal{I}, \alpha \rangle$ satisfies every constraint in CS, in which case we say that CS is *satisfiable*.

Constraint Systems

- A *constraint system* CS is a finite, non-empty set of constraints. A pair $\langle \mathcal{I}, \alpha \rangle$ *satisfies* CS if $\langle \mathcal{I}, \alpha \rangle$ satisfies every constraint in CS, in which case we say that CS is *satisfiable*.
- A knowledge base $\Sigma = \langle T, A \rangle$ in a language with the $\mathcal{C}$ and $\mathcal{U}$ constructors can be easily translated into a constraint system CS_Σ by taking

$$\mathsf{CS}_\Sigma = A \cup \{\forall x.x{:}\neg C \sqcup D \mid C \sqsubseteq D \in T\}.$$

Constraint Systems

- A *constraint system* CS is a finite, non-empty set of constraints. A pair $\langle \mathcal{I}, \alpha \rangle$ *satisfies* CS if $\langle \mathcal{I}, \alpha \rangle$ satisfies every constraint in CS, in which case we say that CS is *satisfiable*.
- A knowledge base $\Sigma = \langle T, A \rangle$ in a language with the $\mathcal{C}$ and $\mathcal{U}$ constructors can be easily translated into a constraint system CS_Σ by taking

$$\mathsf{CS}_\Sigma = A \cup \{\forall x.x{:}\neg C \sqcup D \mid C \sqsubseteq D \in T\}.$$

- And given a knowledge base Σ, and an assertion $a{:}C$

$$\Sigma \models a{:}C \text{ iff } \mathsf{CS}_\Sigma \cup \{a{:}\neg C\} \text{ is unsatisfiable.}$$

Constraint Rules

- $\mathsf{CS} \rightarrow_{\sqcap} \{s{:}C_1, s{:}C_2\} \cup \mathsf{CS}$,
  if $s{:}C_1 \sqcap C_2 \in \mathsf{CS}$ and $\{s{:}C_1, s{:}C_2\} \not\subseteq \mathsf{CS}$;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\sqcap} \{s{:}C_1, s{:}C_2\} \cup \mathsf{CS}$,
  if $s{:}C_1 \sqcap C_2 \in \mathsf{CS}$ and $\{s{:}C_1, s{:}C_2\} \not\subseteq \mathsf{CS}$;
- $\mathsf{CS} \rightarrow_{\sqcup} \{s{:}D\} \cup \mathsf{CS}$
  if $s{:}C_1 \sqcup C_2 \in \mathsf{CS}$, $\{s{:}C_1, s{:}C_2\} \cap \mathsf{CS} = \emptyset$,
  and $D = C_1$ or $D = C_2$;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\sqcap} \{s{:}C_1, s{:}C_2\} \cup \mathsf{CS}$,
  if $s{:}C_1 \sqcap C_2 \in \mathsf{CS}$ and $\{s{:}C_1, s{:}C_2\} \not\subseteq \mathsf{CS}$;
- $\mathsf{CS} \rightarrow_{\sqcup} \{s{:}D\} \cup \mathsf{CS}$
  if $s{:}C_1 \sqcup C_2 \in \mathsf{CS}$, $\{s{:}C_1, s{:}C_2\} \cap \mathsf{CS} = \emptyset$,
  and $D = C_1$ or $D = C_2$;
- $\mathsf{CS} \rightarrow_{\exists} \{(s,y){:}R, y{:}C\} \cup \mathsf{CS}$
  if $s{:}\exists R.C \in \mathsf{CS}$, there is no t such that t
  is a direct R-successor of s in CS
  and $t{:}C$ is in CS, and y is a new variable;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\sqcap} \{s{:}C_1, s{:}C_2\} \cup \mathsf{CS}$,
  if $s{:}C_1 \sqcap C_2 \in \mathsf{CS}$ and $\{s{:}C_1, s{:}C_2\} \not\subseteq \mathsf{CS}$;
- $\mathsf{CS} \rightarrow_{\sqcup} \{s{:}D\} \cup \mathsf{CS}$
  if $s{:}C_1 \sqcup C_2 \in \mathsf{CS}$, $\{s{:}C_1, s{:}C_2\} \cap \mathsf{CS} = \emptyset$,
  and $D = C_1$ or $D = C_2$;
- $\mathsf{CS} \rightarrow_{\exists} \{(s, y){:}R, y{:}C\} \cup \mathsf{CS}$
  if $s{:}\exists R.C \in \mathsf{CS}$, there is no t such that t
  is a direct R-successor of s in CS
  and $t{:}C$ is in CS, and y is a new variable;
- $\mathsf{CS} \rightarrow_{\forall} \{t{:}C\} \cup \mathsf{CS}$
  if $s{:}\forall R.C$ is in CS, t is a filler of R for s,
  and $t{:}C$ is not in CS;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\geq} \{(s, y_i){:}R, y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n\} \cup \mathsf{CS}$
  if $s{:}(\geq n\ R) \in \mathsf{CS}$, there do not exist n
  pairwise separated fillers of R for s in CS,
  and $y_1, \ldots, y_n$ are new variables;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\geq} \{(s, y_i){:}R, y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n\} \cup \mathsf{CS}$
  if $s{:}(\geq n\ R) \in \mathsf{CS}$, there do not exist n
  pairwise separated fillers of R for s in CS,
  and $y_1, \ldots, y_n$ are new variables;
- $\mathsf{CS} \rightarrow_{\leq} \mathsf{CS}[t/z]$
  if $s{:}(\leq n\ R)$ is in CS, s has more than n fillers of R for s
  and t, z are two fillers of R for s that are not separated;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\geq} \{(s, y_i){:}R, y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n\} \cup \mathsf{CS}$
  if $s{:}(\geq n\ R) \in \mathsf{CS}$, there do not exist n
  pairwise separated fillers of R for s in CS,
  and $y_1, \ldots, y_n$ are new variables;
- $\mathsf{CS} \rightarrow_{\leq} \mathsf{CS}[t/z]$
  if $s{:}(\leq n\ R)$ is in CS, s has more than n fillers of R for s
  and t, z are two fillers of R for s that are not separated;
- $\mathsf{CS} \rightarrow_{\mathcal{O}} \mathsf{CS}[x/a_i]$, if $x{:}\{a_1, \ldots, a_n\} \in \mathsf{CS}$ and $1 \leq i \leq n$.

Constraint Rules

- $\mathsf{CS} \rightarrow_{\geq} \{(s, y_i){:}R, y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n\} \cup \mathsf{CS}$
  if $s{:}(\geq n\ R) \in \mathsf{CS}$, there do not exist n
  pairwise separated fillers of R for s in CS,
  and $y_1, \ldots, y_n$ are new variables;
- $\mathsf{CS} \rightarrow_{\leq} \mathsf{CS}[t/z]$
  if $s{:}(\leq n\ R)$ is in CS, s has more than n fillers of R for s
  and t, z are two fillers of R for s that are not separated;
- $\mathsf{CS} \rightarrow_{\mathcal{O}} \mathsf{CS}[x/a_i]$, if $x{:}\{a_1, \ldots, a_n\} \in \mathsf{CS}$ and $1 \leq i \leq n$.
- $\mathsf{CS} \rightarrow_{\mathcal{B}} \{(s, a){:}R\} \cup \mathsf{CS}$
  if $s{:}\exists R.\{a\}$ is in CS and $(s, a){:}R$ is not in CS;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\mathcal{R}} \{(s,t){:}R_1, (s,t){:}R_2\} \cup \mathsf{CS}$
  if $(s,t){:}R_1 \sqcap R_2$ is in CS, and
  either $(s,t){:}R_1$ or $(s,t){:}R_2$ is not in CS;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\mathcal{R}} \{(s,t){:}R_1, (s,t){:}R_2\} \cup \mathsf{CS}$
  if $(s,t){:}R_1 \sqcap R_2$ is in CS, and
  either $(s,t){:}R_1$ or $(s,t){:}R_2$ is not in CS;
- $\mathsf{CS} \rightarrow_{-1} \{(t,s){:}R\} \cup \mathsf{CS}$
  if $(s,t){:}R^{-1}$ is in CS and $(t,s){:}R$ is not in CS;

Constraint Rules

- $\mathsf{CS} \rightarrow_{\mathcal{R}} \{(s,t){:}R_1, (s,t){:}R_2\} \cup \mathsf{CS}$
  if $(s,t){:}R_1 \sqcap R_2$ is in CS, and
  either $(s,t){:}R_1$ or $(s,t){:}R_2$ is not in CS;
- $\mathsf{CS} \rightarrow_{-1} \{(t,s){:}R\} \cup \mathsf{CS}$
  if $(s,t){:}R^{-1}$ is in CS and $(t,s){:}R$ is not in CS;
- $\mathsf{CS} \rightarrow_{\forall x} \{s{:}C\} \cup \mathsf{CS}$
  if $\forall x.x{:}C$ is in CS, s appears in CS, and $s{:}C$ is not in CS.

Soundness and Completeness

Theorem: Let C be an $\mathcal{ALCN}$ concept and T obtained by applying the constraint expansion rules to C. Then

1. The rule application terminates,

Soundness and Completeness

Theorem: Let C be an $\mathcal{ALCN}$ concept and T obtained by applying the constraint expansion rules to C. Then
1. The rule application terminates,
2. If T is consistent and $\rightarrow$ is applicable to T, then $\rightarrow$ yields a consistent T',

Soundness and Completeness

Theorem: Let C be an $\mathcal{ALCN}$ concept and T obtained by applying the constraint expansion rules to C. Then

1. The rule application terminates,
2. If T is consistent and $\rightarrow$ is applicable to T, then $\rightarrow$ yields a consistent T',
3. If T contains a clash, then T has no model, and

Soundness and Completeness

Theorem: Let C be an $\mathcal{ALCN}$ concept and T obtained by applying the constraint expansion rules to C. Then
1. The rule application terminates,
2. If T is consistent and $\rightarrow$ is applicable to T, then $\rightarrow$ yields a consistent T',
3. If T contains a clash, then T has no model, and
4. If no more rules apply to T, then T defines a (canonical) model for C.

Soundness and Completeness

Theorem: Let C be an $\mathcal{ALCN}$ concept and T obtained by applying the constraint expansion rules to C. Then
1. The rule application terminates,
2. If T is consistent and $\rightarrow$ is applicable to T, then $\rightarrow$ yields a consistent T',
3. If T contains a clash, then T has no model, and
4. If no more rules apply to T, then T defines a (canonical) model for C.

Corollary: The tableau algorithm is a PSPACE decision procedure for consistency (and subsumption) of $\mathcal{ALCN}$ concepts. And $\mathcal{ALCN}$ has the tree property.

Soundness and Completeness II

Proof of Lemma:

1. (Termination) The algorithm "increasingly" constructs a tree whose

   depth is linear in $|C|$: quantifier depth decreases from node to succs.

   breadth is linear in $|C|$ (even if number in NRs are coded in binary)

Soundness and Completeness II

Proof of Lemma:

1. (Termination) The algorithm "increasingly" constructs a tree whose
   depth is linear in $|C|$: quantifier depth decreases from node to succs.
   breadth is linear in $|C|$ (even if number in NRs are coded in binary)
2. (Local Consistency) Easy to prove (by definition of the semantics) that if $\mathcal{I}$ is a model of T and $\rightarrow$ can be applied to T to obtain T', then $\mathcal{I}'$ is a model of T'.

Soundness and Completeness II

Proof of Lemma:

1. (Termination) The algorithm "increasingly" constructs a tree whose
   depth is linear in $|C|$: quantifier depth decreases from node to succs.
   breadth is linear in $|C|$ (even if number in NRs are coded in binary)
2. (Local Consistency) Easy to prove (by definition of the semantics) that if $\mathcal{I}$ is a model of T and $\rightarrow$ can be applied to T to obtain T', then $\mathcal{I}'$ is a model of T'.

3 Obvious: T with a clash has no model – recall definition of a clash

$$\begin{array}{rcl} \{A, \neg A\} & \subseteq & \mathcal{L}(x)\text{or} \\ \{(\geq mR), (\leq nR)\} & \subseteq & (x) \text{ for } n < m \end{array}$$

Soundness and Completeness III

4. (Canonical model) A “complete” tree T defines a (tree) pre-model $\mathcal{I}$:

Soundness and Completeness III

4. (Canonical model) A "complete" tree T defines a (tree) pre-model $\mathcal{I}$:
   nodes correspond to elements of $\Delta^{\mathcal{I}}$

Soundness and Completeness III

4. (Canonical model) A “complete” tree T defines a (tree) pre-model $\mathcal{I}$:
   nodes correspond to elements of $\Delta^{\mathcal{I}}$
   edges define role-relationship

Soundness and Completeness III

4. (Canonical model) A “complete” tree T defines a (tree) pre-model $\mathcal{I}$:
   nodes correspond to elements of $\Delta^{\mathcal{I}}$
   edges define role-relationship
   $x \in A^{\mathcal{I}}$ iff $A \in \mathcal{L}(x)$ for concept names A.

Soundness and Completeness III

4. (Canonical model) A “complete” tree T defines a (tree) pre-model $\mathcal{I}$:
   nodes correspond to elements of $\Delta^{\mathcal{I}}$
   edges define role-relationship
   $x \in A^{\mathcal{I}}$ iff $A \in \mathcal{L}(x)$ for concept names A.

- Check that $C \in \mathcal{L}(x)$ implies $x \in C^{\mathcal{I}}$ — if C is not a **number restriction**.

Soundness and Completeness III

4. (Canonical model) A “complete” tree T defines a (tree) pre-model $\mathcal{I}$:

   nodes correspond to elements of $\Delta^{\mathcal{I}}$

   edges define role-relationship

   $x \in A^{\mathcal{I}}$ iff $A \in \mathcal{L}(x)$ for concept names A.

- Check that $C \in \mathcal{L}(x)$ implies $x \in C^{\mathcal{I}}$ — if C is not a **number restriction**.
- For NRs, if $(\geq nR) \in \mathcal{L}(x)$ and x has less than n R-successors, **copy** some R-successors (including sub-trees) to obtain n R-successors.

Soundness and Completeness III

4. (Canonical model) A "complete" tree T defines a (tree) pre-model $\mathcal{I}$:
   nodes correspond to elements of $\Delta^{\mathcal{I}}$
   edges define role-relationship
   $x \in A^{\mathcal{I}}$ iff $A \in \mathcal{L}(x)$ for concept names A.

- Check that $C \in \mathcal{L}(x)$ implies $x \in C^{\mathcal{I}}$ — if C is not a **number restriction**.
- For NRs, if $(\geq nR) \in \mathcal{L}(x)$ and x has less than n R-successors, **copy** some R-successors (including sub-trees) to obtain n R-successors.
- $\rightsquigarrow$ canonical tree model for input concept.

PSPACE

- To make the tableau algorithm run in PSPACE, start by recalling Savitch: PSPACE = NPSPACE

PSPACE

- To make the tableau algorithm run in PSPACE, start by recalling Savitch: PSPACE = NPSPACE
- Observe that branches are independent from each other

PSPACE

- To make the tableau algorithm run in PSPACE, start by recalling Savitch: PSPACE = NPSPACE
- Observe that branches are independent from each other
- Observe that each node (label) requires linear space only

PSPACE

- To make the tableau algorithm run in PSPACE, start by recalling Savitch: PSPACE = NPSPACE
- Observe that branches are independent from each other
- Observe that each node (label) requires linear space only
- recall that path are of lenght $\leq |C|$
  $\rightsquigarrow$ each path can be stored in $\mathcal{O}(|C^2|)$

PSPACE

- To make the tableau algorithm run in PSPACE, start by recalling Savitch: PSPACE = NPSPACE
- Observe that branches are independent from each other
- Observe that each node (label) requires linear space only
- recall that path are of lenght $\leq |C|$
  $\rightsquigarrow$ each path can be stored in $\mathcal{O}(|C^2|)$
- construct/search the tree **depth first**

PSPACE

- To make the tableau algorithm run in PSPACE, start by recalling Savitch: PSPACE = NPSPACE
- Observe that branches are independent from each other
- Observe that each node (label) requires linear space only
- recall that path are of lenght $\leq |C|$
  $\rightsquigarrow$ each path can be stored in $\mathcal{O}(|C^2|)$
- construct/search the tree **depth first**
- re-use space from already constructed branches.

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $(\geq nR\ C)$ and $(\leq nR\ C)$

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $(\geq nR\ C)$ and $(\leq nR\ C)$
- $\mathcal{ALC}$ with inverse roles (e.g. `has-child`$^{-}$)

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $(\geq nR\ C)$ and $(\leq nR\ C)$
- $\mathcal{ALC}$ with inverse roles (e.g. $\texttt{has-child}^{-}$)
- $\mathcal{ALC}$ with role conjunction $\exists(R \sqcap S).C$ and $\forall(R \sqcap S).C$.

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $(\geq nR\ C)$ and $(\leq nR\ C)$
- $\mathcal{ALC}$ with inverse roles (e.g. $\texttt{has-child}^{-}$)
- $\mathcal{ALC}$ with role conjunction $\exists(R \sqcap S).C$ and $\forall(R \sqcap S).C$.
- TBoxes with acyclic concept definitions $A \doteq C$:

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $(\geq nR\ C)$ and $(\leq nR\ C)$
- $\mathcal{ALC}$ with inverse roles (e.g. $\texttt{has-child}^{-}$)
- $\mathcal{ALC}$ with role conjunction $\exists(R \sqcap S).C$ and $\forall(R \sqcap S).C$.
- TBoxes with acyclic concept definitions $A \doteq C$:

  unfolding (macro expansion) is easy, but suboptimal: may yield exponential blow-up

Extensions

This tableau algorithm can be modified to a PSPACE decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $(\geq nR\ C)$ and $(\leq nR\ C)$
- $\mathcal{ALC}$ with inverse roles (e.g. $\texttt{has-child}^{-}$)
- $\mathcal{ALC}$ with role conjunction $\exists(R \sqcap S).C$ and $\forall(R \sqcap S).C$.
- TBoxes with acyclic concept definitions $A \doteq C$:

unfolding	(macro expansion) is easy, but suboptimal: may yield exponential blow-up
lazy unfolding	(unfolding on demand) is optimal, consistency is PSPACE decidable

Extensions

Language extensions that require more elaborate techniques include

Extensions

Language extensions that require more elaborate techniques include

- TBoxes with general axioms $C \sqsubseteq D$:

Extensions

Language extensions that require more elaborate techniques include

- TBoxes with general axioms $C \sqsubseteq D$:
    - each node must be labelled with $\neg C \sqcup D$
    - quantifier depth no longer decreases
    - $\leadsto$ termination not guaranteed

Extensions

Language extensions that require more elaborate techniques include

- TBoxes with general axioms $C \sqsubseteq D$:
  - each node must be labelled with $\neg C \sqcup D$
  - quantifier depth no longer decreases
  - $\rightsquigarrow$ termination not guaranteed
- Transitive closure of roles

Extensions

Language extensions that require more elaborate techniques include

- TBoxes with general axioms $C \sqsubseteq D$:
  - each node must be labelled with $\neg C \sqcup D$
  - quantifier depth no longer decreases
  - $\leadsto$ termination not guaranteed
- Transitive closure of roles
  - node labels $(\forall R^*.C)$ yields C in all R^n-successor labels.
  - quantifier depth no longer decreases
  - $\leadsto$ termination not guaranteed

Extensions

Language extensions that require more elaborate techniques include

- TBoxes with general axioms $C \sqsubseteq D$:
  - each node must be labelled with $\neg C \sqcup D$
  - quantifier depth no longer decreases
  - $\leadsto$ termination not guaranteed
- Transitive closure of roles
  - node labels $(\forall R^*.C)$ yields C in all R^n-successor labels.
  - quantifier depth no longer decreases
  - $\leadsto$ termination not guaranteed

Use **blocking** (cycle detection) to ensure termination
(but the right blocking to not destroy soundness and completeness)

Non-Termination

- As already mentioned, for **ALC** with **general axioms**, the basic algorithm is **non-terminating**.

Non-Termination

- As already mentioned, for **ALC** with **general axioms**, the basic algorithm is **non-terminating**.
- E.g, if $\texttt{human} \sqsubseteq \exists\texttt{hasmother.human} \in T$, then $\neg\texttt{human} \sqcup \exists\texttt{hasmother.human}$ has to be added to every node.

Non-Termination

- As already mentioned, for **ALC** with **general axioms**, the basic algorithm is **non-terminating**.
- E.g, if $\texttt{human} \sqsubseteq \exists\texttt{hasmother.human} \in T$, then $\neg\texttt{human} \sqcup \exists\texttt{hasmother.human}$ has to be added to every node.

w $\mathcal{L}(w) = \{\texttt{human}, \neg\texttt{human} \sqcup \exists\texttt{hasmother.human}, \exists\texttt{hasmother.human}\}$

$\downarrow$ hasmother

x $\mathcal{L}(w) = \{\texttt{human}, \neg\texttt{human} \sqcup \exists\texttt{hasmother.human}, \exists\texttt{hasmother.human}\}$

$\downarrow$ hasmother

y $\mathcal{L}(w) = \{\texttt{human}, \neg\texttt{human} \sqcup \exists\texttt{hasmother.human}, \exists\texttt{hasmother.human}\}$

$\downarrow$ hasmother

Blocking

- When creating a new node, check ancestors for equal (superset) label

Blocking

- When creating a new node, check ancestors for equal (superset) label
- If such a node is found, new node is **blocked**

Blocking

- When creating a new node, check ancestors for equal (superset) label
- If such a node is found, new node is **blocked**

w $\quad \mathcal{L}(w) = \{\texttt{human}, \neg\texttt{human} \sqcup \exists\texttt{hasmother.human}, \exists\texttt{hasmother.human}\}$

$\downarrow$ `hasmother`

x $\quad \mathcal{L}(w) = \{\texttt{human}, \neg\texttt{human} \sqcup \exists\texttt{hasmother.human}\}$

Implementing DL Systems

If we implement these constraint systems naively we will soon run into problems

Implementing DL Systems

If we implement these constraint systems naively we will soon run into problems

- Space usage
  - The storage required for the tableaux data-structures can grow
  - But this is rarely a serious problem in practice.

Implementing DL Systems

If we implement these constraint systems naively we will soon run into problems

- Space usage
  - The storage required for the tableaux data-structures can grow
  - But this is rarely a serious problem in practice.
- Time usage
  - The search required due to non-deterministic expansion can overcome the prover.
  - This is a *serious* problem in practice.

Implementing DL Systems

If we implement these constraint systems naively we will soon run into problems

- Space usage
  - The storage required for the tableaux data-structures can grow
  - But this is rarely a serious problem in practice.
- Time usage
  - The search required due to non-deterministic expansion can overcome the prover.
  - This is a *serious* problem in practice.
  - Which is mitigated by:
    - Careful choice of algorithms.

Implementing DL Systems

If we implement these constraint systems naively we will soon run into problems

- Space usage
  - The storage required for the tableaux data-structures can grow
  - But this is rarely a serious problem in practice.
- Time usage
  - The search required due to non-deterministic expansion can overcome the prover.
  - This is a *serious* problem in practice.
  - Which is mitigated by:
    - Careful choice of algorithms.
    - Highly optimized implementations

Highly Optimized Implementations

Optimizations can be performed at two levels

Highly Optimized Implementations

Optimizations can be performed at two levels

- When computing the *classification* (partial ordering) of concepts.

Highly Optimized Implementations

Optimizations can be performed at two levels

- When computing the *classification* (partial ordering) of concepts.
  - Objective is to minimize number of subsumption tests

Highly Optimized Implementations

Optimizations can be performed at two levels

- When computing the *classification* (partial ordering) of concepts.
  - Objective is to minimize number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use *enhanced traversal* that exploits information from previous tests.

Highly Optimized Implementations

Optimizations can be performed at two levels

- When computing the *classification* (partial ordering) of concepts.
  - Objective is to minimize number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use *enhanced traversal* that exploits information from previous tests.
  - Also use structural information from the KB
    - E.g., when selecting the order in which to classify concepts.

Highly Optimized Implementations

Optimizations can be performed at two levels

- When computing the *classification* (partial ordering) of concepts.
  - Objective is to minimize number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use *enhanced traversal* that exploits information from previous tests.
  - Also use structural information from the KB
    - E.g., when selecting the order in which to classify concepts.
  - Computing *subsumption* between concepts

Highly Optimized Implementations

Optimizations can be performed at two levels

- When computing the *classification* (partial ordering) of concepts.
  - Objective is to minimize number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use *enhanced traversal* that exploits information from previous tests.
  - Also use structural information from the KB
    - E.g., when selecting the order in which to classify concepts.
  - Computing *subsumption* between concepts
    - Objective is to minimize cost of single subsumption tests
    - Small number of hard tests can dominate classification time
    - Latest DL research has mainly addressed this problem.

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- Algorithmic optimizations

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- Algorithmic optimizations
  - Main aim is to reduce search space due to non-determinism

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- Algorithmic optimizations
  - Main aim is to reduce search space due to non-determinism
  - Integral part of implementations

Optimizing Subsumption Testing

Optimizations techniques broadly fall into two categories

- Pre-processing optimizations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- Algorithmic optimizations
  - Main aim is to reduce search space due to non-determinism
  - Integral part of implementations
  - But often generally applicable to search based algorithms

Pre-Processing Optimizations

Useful techniques include

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into "definition" axioms

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into "definition" axioms
  - Definition axioms efficiently dealt with by lazy expansion

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into “definition” axioms
  - Definition axioms efficiently dealt with by lazy expansion
- Avoidance of potentially costly reasoning whenever possible

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into “definition” axioms
  - Definition axioms efficiently dealt with by lazy expansion
- Avoidance of potentially costly reasoning whenever possible
  - Normalization can discover “obvious” (un) satisfiability

Pre-Processing Optimizations

Useful techniques include

- Normalization and simplification of concepts
  - Refinement of technique first used in the $\mathcal{KRIS}$ system
  - Lexically normalize and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into "definition" axioms
  - Definition axioms efficiently dealt with by lazy expansion
- Avoidance of potentially costly reasoning whenever possible
  - Normalization can discover "obvious" (un) satisfiability
  - Structural analysis can discover "obvious" subsumption

Algorithmic Optimizations

Useful techniques include

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- Caching

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- Caching
  - Cache partial models
  - Cache satisfiability status (of labels)

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- Caching
  - Cache partial models
  - Cache satisfiability status (of labels)
- Heuristic ordering of propositional and modal expansions

Algorithmic Optimizations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- Caching
  - Cache partial models
  - Cache satisfiability status (of labels)
- Heuristic ordering of propositional and modal expansions
  - Minimize/maximize constrainedness (e.g., MOMS)
  - Maximize backtracking (e.g., oldest first)

Normalization and Simplification

- Normalize concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$.
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$.

Normalization and Simplification

- Normalize concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$.
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$.
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$

Normalization and Simplification

- Normalize concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$.
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$.
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$
- Lazily unfold concepts in tableaux algorithm
  - Use name/pointers to refer to complex concepts
  - Only add structure as required by the progress of the algorithm
  - Detect clashes between lexically equivalent concepts

Absorption I

- Reasoning w.r.t. sets of GCI axioms can be very costly

Absorption I

- Reasoning w.r.t. sets of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \cup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes, search space is already 2^100
  - GALEN KB contains **thousands** of axioms.

Absorption I

- Reasoning w.r.t. sets of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \cup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes, search space is already 2^100
  - GALEN KB contains **thousands** of axioms.
- Reasoning w.r.t. primitive definition axioms is relatively efficient

Absorption I

- Reasoning w.r.t. sets of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \cup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes, search space is already 2^100
  - GALEN KB contains **thousands** of axioms.
- Reasoning w.r.t. primitive definition axioms is relatively efficient
  - For $CN \sqsubseteq D$, add D **only** to node labels containing CN
  - for $CN \sqsupseteq D$, add $\neg D$ **only** to node labels containing $\neg CN$
  - We can expand definitions lazily
    - Only add definitions **after** other local (propositional) expansions.
    - Only add definitions one step at a time.

Absorption II

- Transform GCIs into primitive definitions, e.g.:
  - $CN \sqcap C \sqsubset D \longrightarrow CN \subseteq D \sqcup \neg C$
  - $CN \sqcup C \sqsupset D \longrightarrow CN \supseteq D \sqcap \neg C$

Absorption II

- Transform GCIs into primitive definitions, e.g.:
  - $CN \sqcap C \sqsubset D \longrightarrow CN \subseteq D \sqcup \neg C$
  - $CN \sqcup C \sqsupset D \longrightarrow CN \supseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.:
  - $CN \sqsubseteq A, CN \sqsubseteq D \sqcup \neg C \longrightarrow CN \sqsubseteq A \sqcap (D \sqcup \neg C)$
  - $CN \sqsupseteq A, CN \sqsupseteq D \sqcap \neg C \longrightarrow CN \sqsupseteq A \sqcup (D \sqcap \neg C)$

Absorption II

- Transform GCIs into primitive definitions, e.g.:
  - $CN \sqcap C \sqsubset D \longrightarrow CN \subseteq D \sqcup \neg C$
  - $CN \sqcup C \sqsupset D \longrightarrow CN \supseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.:
  - $CN \sqsubseteq A, CN \sqsubseteq D \sqcup \neg C \longrightarrow CN \sqsubseteq A \sqcap (D \sqcup \neg C)$
  - $CN \sqsupseteq A, CN \sqsupseteq D \sqcap \neg C \longrightarrow CN \sqsupseteq A \sqcup (D \sqcap \neg C)$
- Use lazy expansion techniques with primitive definitions
  - Disjunctions only added to "relevant" node labels

Absorption II

- Transform GCIs into primitive definitions, e.g.:
  - $CN \sqcap C \sqsubset D \longrightarrow CN \subseteq D \sqcup \neg C$
  - $CN \sqcup C \sqsupset D \longrightarrow CN \supseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.:
  - $CN \sqsubseteq A, CN \sqsubseteq D \sqcup \neg C \longrightarrow CN \sqsubseteq A \sqcap (D \sqcup \neg C)$
  - $CN \sqsupseteq A, CN \sqsupseteq D \sqcap \neg C \longrightarrow CN \sqsupseteq A \sqcup (D \sqcap \neg C)$
- Use lazy expansion techniques with primitive definitions
  - Disjunctions only added to “relevant” node labels
- Performance improvements often exceptional
  - At least **four orders of magnitude** with GALEN

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points **without exploring** alternative branches

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points **without exploring** alternative branches
  - Effect is to prune away part of the search space

Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points **without exploring** alternative branches
  - Effect is to prune away part of the search space
  - Performance improvements again very good.

Backjumping

E.g if $\exists R.\neg A \sqcap \forall R(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcap D_n) \in \mathcal{L}(x)$

Caching

- Cache the satisfiability status of a node label

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq$ D if $C \sqcap \neg D$ is satisfiable

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq$ D if $C \sqcap \neg D$ is satisfiable
  - $C \sqcap \neg D$ satisfiable if models of C and $\neg D$ can be merged.

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialized, look in cache
    - use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq$ D if $C \sqcap \neg D$ is satisfiable
  - $C \sqcap \neg D$ satisfiable if models of C and $\neg D$ can be merged.
  - If not, continue with standard subsumption test.