Knowledge Representation

Lecture 5: Practical Reasoning with $\mathcal{EL}$

Patrick Koopmann

November 6, 2023

The story so far:

- Concepts describe sets of individuals
- Ontologies contain axioms about concepts and individuals
- Interpretations and models
- Entailment as basic reasoning task
- Today: How does reasoning with DLs work?

Little Warm-Up Exercise

$$\mathcal{O} = \{ \quad \begin{aligned} \mathit{Alive} &\sqsubseteq \mathit{Animal} \sqcup \mathit{Plant} \\ \mathit{Animal} &\sqsubseteq \exists \mathit{hasParent}.\mathit{Male} \sqcap \exists \mathit{hasParent}.\mathit{Female} \\ \mathit{thomas} &: \mathit{Alive} \\ \mathit{thomas} &: \forall \mathit{hasParent}.\bot \end{aligned} \quad \}$$

What can we say about Thomas?

Flashback: What is Knowledge Representation?

- KR as surrogate
- KR as expression of ontological commitment
- KR as theory of intelligent reasoning
- **KR as medium for efficient computation**
  - automated deduction is useless if it is not practical
  - trade-off between expressivity and reasoning performance
- KR as medium of human expression

Flashback: Reasoning

Reasoning allows us to discover new insights from the knowledge represented in the ontology.

The central reasoning task is entailment:

> $\mathcal{O}$ entails an axiom α ($\mathcal{O} \models \alpha$) if every model of $\mathcal{O}$ is also a model of α.

- We need to consider what all models have in common.
- This is the same as in propositional and first-order logic.

Deciding $\mathcal{O} \models \alpha$

Reasoning looks harder than in propositional logic:

- In propositional logic, we only have a finite number of interpretations to consider
  - interpretations are truth valuations
  - we can go through the interpretations one by one and check

Deciding $\mathcal{O} \models \alpha$

Reasoning looks harder than in propositional logic:

- In propositional logic, we only have a finite number of interpretations to consider
  - interpretations are truth valuations
  - we can go through the interpretations one by one and check
- In description logics, there are infinitely many interpretations!
  - interpretations involve different domain elements
  - their number is arbitrary (up infinitely many)

Deciding $\mathcal{O} \models \alpha$

Reasoning looks harder than in propositional logic:

- In propositional logic, we only have a finite number of interpretations to consider
  - interpretations are truth valuations
  - we can go through the interpretations one by one and check
- In description logics, there are infinitely many interpretations!
  - interpretations involve different domain elements
  - their number is arbitrary (up infinitely many)
- But we are better off than in first-order logic
  - entailment in description logics is decidable
  - entailment for first-order logic is only semi-decidable

No Equivalence Axioms

To keep the following simpler, we assume that our TBoxes contain no equivalence axioms.

If the TBox contains equivalence axioms $C \equiv D$, we can replace each such axiom by the two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

Reasoning in Description Logics

Reasoning with $\mathcal{ALC}$ is truely harder than for propositional logic

- we have to consider different options for different elements
- it may require exponential time in the size of the ontology
- differently to propositional logic ($P = NP$ question), this is certain from a theoretically perspective

Reasoning in Description Logics

Reasoning with $\mathcal{ALC}$ is truely harder than for propositional logic

- we have to consider different options for different elements
- it may require exponential time in the size of the ontology
- differently to propositional logic ($P = NP$ question), this is certain from a theoretically perspective

Before we look at $\mathcal{ALC}$, we look at an easier description logic

A practical fragment of $\mathcal{ALC}$

- $\mathcal{ALC}$ is just one example of a DL
- Different DLs differ in
  - What concept operators they allow
  - What axiom types they allow

A practical fragment of $\mathcal{ALC}$

- $\mathcal{ALC}$ is just one example of a DL
- Different DLs differ in
  - What concept operators they allow
  - What axiom types they allow
- A very important DL is $\mathcal{EL}$, which is a fragment of $\mathcal{ALC}$
  - $\mathcal{EL}$ only allows the concept operators $\top$, $\sqcap$ and $\exists$ $\forall$
  - $\mathcal{EL}$ does allow all axiom types we have seen so far.

A practical fragment of $\mathcal{ALC}$

- $\mathcal{ALC}$ is just one example of a DL
- Different DLs differ in
  - What concept operators they allow
  - What axiom types they allow
- A very important DL is $\mathcal{EL}$, which is a fragment of $\mathcal{ALC}$
  - $\mathcal{EL}$ only allows the concept operators $\top$, $\sqcap$ and $\exists$ $\forall$
  - $\mathcal{EL}$ does allow all axiom types we have seen so far.
- $\mathcal{EL}$ is used predominantly in many large ontologies
  - Very often, most axioms in an ontology are $\mathcal{EL}$ axioms
  - A lot of ontologies are pure $\mathcal{EL}$ ontologies (or in friendly extensions of $\mathcal{EL}$)
  - SNOMED CT, the large medical ontology mentioned in Lecture 3, is one such example

Reasoning in $\mathcal{EL}$

- Some reasoning tasks are not interesting in $\mathcal{EL}$:
  - We do not have $\bot$ or $\neg$

  $\Rightarrow$ We cannot create contradictions

  $\Rightarrow$ every $\mathcal{EL}$ ontology is consistent and coherent

Reasoning in $\mathcal{EL}$

- Some reasoning tasks are not interesting in $\mathcal{EL}$:
  - We do not have $\perp$ or $\neg$
  - $\Rightarrow$ We cannot create contradictions
  - $\Rightarrow$ every $\mathcal{EL}$ ontology is consistent and coherent
- Other reasoning tasks are more interesting:
  - Subsumption: $\mathcal{O} \models C \sqsubseteq D$
  - Instance checking: $\mathcal{O} \models a : C$
  - Classification: Determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Materialization: Determine all $\mathcal{O} \models a : B$ where $a \in \mathbf{I}$ and $B \in \mathbf{C}$

Reasoning in $\mathcal{EL}$

- Some reasoning tasks are not interesting in $\mathcal{EL}$:
  - We do not have $\bot$ or $\neg$
  - $\Rightarrow$ We cannot create contradictions
  - $\Rightarrow$ every $\mathcal{EL}$ ontology is consistent and coherent
- Other reasoning tasks are more interesting:
  - Subsumption: $\mathcal{O} \models C \sqsubseteq D$
  - Instance checking: $\mathcal{O} \models a : C$
  - Classification: Determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Materialization: Determine all $\mathcal{O} \models a : B$ where $a \in \mathbf{I}$ and $B \in \mathbf{C}$
- We first look at an algorithm for subsumption

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$
- We iteratively construct a special model of $\mathcal{O}$ with an element $d \in C^{\mathcal{I}}$
- In this model, d will satisfy all concepts D' for which $\mathcal{O} \models C \sqsubseteq D'$
  - The model is thus called a canonical model

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$
- We iteratively construct a special model of $\mathcal{O}$ with an element $d \in C^{\mathcal{I}}$
- In this model, d will satisfy all concepts D' for which $\mathcal{O} \models C \sqsubseteq D'$
  - The model is thus called a canonical model
- We start with the single element d that should satisfy C

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$
- We iteratively construct a special model of $\mathcal{O}$ with an element $d \in C^{\mathcal{I}}$
- In this model, d will satisfy all concepts D' for which $\mathcal{O} \models C \sqsubseteq D'$
  - The model is thus called a canonical model
- We start with the single element d that should satisfy C
- Throughout the algorithm, elements are marked with concepts they (should) satisfy

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$
- We iteratively construct a special model of $\mathcal{O}$ with an element $d \in C^{\mathcal{I}}$
- In this model, d will satisfy all concepts D' for which $\mathcal{O} \models C \sqsubseteq D'$
  - The model is thus called a canonical model
- We start with the single element d that should satisfy C
- Throughout the algorithm, elements are marked with concepts they (should) satisfy
- Special rules are applied towards satisfying those concepts

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$
- We iteratively construct a special model of $\mathcal{O}$ with an element $d \in C^{\mathcal{I}}$
- In this model, d will satisfy all concepts D' for which $\mathcal{O} \models C \sqsubseteq D'$
  - The model is thus called a canonical model
- We start with the single element d that should satisfy C
- Throughout the algorithm, elements are marked with concepts they (should) satisfy
- Special rules are applied towards satisfying those concepts
- If we eventually assign D to the initial element, then $\mathcal{O} \models C \sqsubseteq D$

The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \\ \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

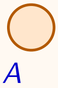
We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \\ \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

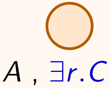


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \qquad \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

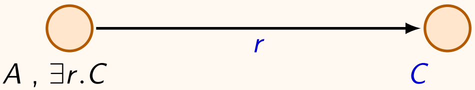


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F,$$
$$\exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

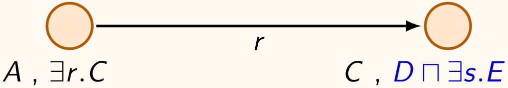


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F,$$
$$\exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

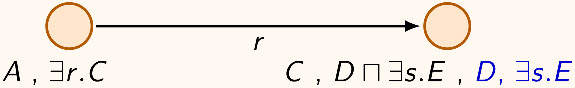


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \\ \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

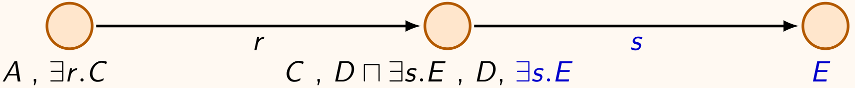


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F,$$
$$\exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

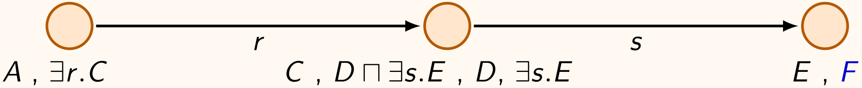


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \\ \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

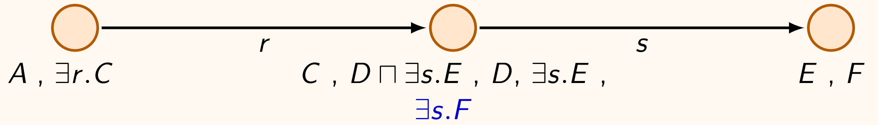


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F,$$
$$\exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

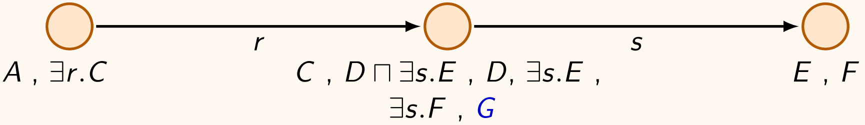


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \qquad \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

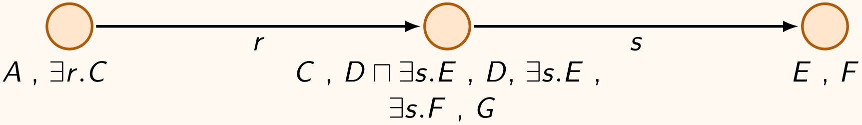


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \qquad \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

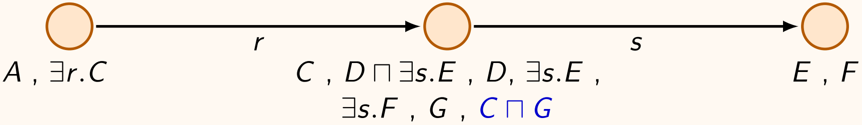


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \qquad \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

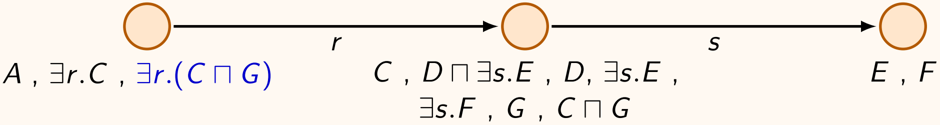


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \qquad \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$

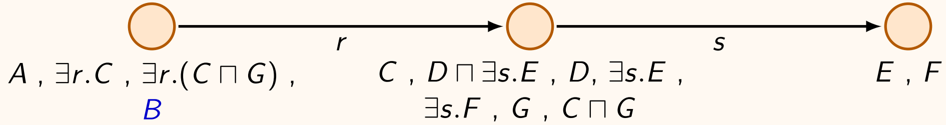


The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \qquad \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$



The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

Yes:

- We added what needs to be in every model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$

The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

Yes:

- We added what needs to be in every model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$
- Only exception: role-successors
  - other models might use different elements as role-successors
  - but for every role-successor we added, there has to be a corresponding role successor in the other models that satisfies at least the same concepts

The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

Yes:

- We added what needs to be in every model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$
- Only exception: role-successors
  - other models might use different elements as role-successors
  - but for every role-successor we added, there has to be a corresponding role successor in the other models that satisfies at least the same concepts

$\Rightarrow$ $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ also on every other model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$

The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

Yes:

- We added what needs to be in every model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$
- Only exception: role-successors
  - other models might use different elements as role-successors
  - but for every role-successor we added, there has to be a corresponding role successor in the other models that satisfies at least the same concepts

$\Rightarrow$ $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ also on every other model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$

- If $A^{\emptyset}$, we also have $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$

The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

Yes:

- We added what needs to be in every model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$
- Only exception: role-successors
  - other models might use different elements as role-successors
  - but for every role-successor we added, there has to be a corresponding role successor in the other models that satisfies at least the same concepts

$\Rightarrow$ $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ also on every other model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$

- If $A^{\emptyset}$, we also have $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$

$\Rightarrow$ $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$ $\square$

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.
- $\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.
- $\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.
- $\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.
- $\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.
- $\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.
- $\exists$-rule 1: If d has $\exists r.C$ assigned, add a new r-successor to d and assign C to it.

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.
- $\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.
- $\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.
- $\exists$-rule 1: If d has $\exists r.C$ assigned, add a new r-successor to d and assign C to it.
- $\exists$-rule 2: If d has an r-successor with C assigned, add $\exists r.C$ to d.

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.
- $\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.
- $\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.
- $\exists$-rule 1: If d has $\exists r.C$ assigned, add a new r-successor to d and assign C to it.
- $\exists$-rule 2: If d has an r-successor with C assigned, add $\exists r.C$ to d.
- $\sqsubseteq$-rule: If d has C assigned and $C \sqsubseteq D \in \mathcal{T}$, then also assign D to d.

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.

$\mathcal{O} = \{ \ A \sqsubseteq B \ \}$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.

$\mathcal{O} = \{\ A \sqsubseteq B\ \}$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.

$\mathcal{O} = \{\ A \sqsubseteq B\ \}$

A , B , $A \sqcap B$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.

$\mathcal{O} = \{ A \sqsubseteq B \}$

A , B , $A \sqcap B$, $(A \sqcap B) \sqcap B$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.

$\mathcal{O} = \{ A \sqsubseteq B \}$

A , B , $A \sqcap B$, $(A \sqcap B) \sqcap B$,
$((A \sqcap B) \sqcap B) \sqcap B$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.

$\mathcal{O} = \{ A \sqsubseteq B \}$

A , B , $A \sqcap B$, $(A \sqcap B) \sqcap B$,
$((A \sqcap B) \sqcap B) \sqcap B$, ...

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

$$\mathcal{O} = \{\ A \sqsubseteq \exists r.A\ \}$$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

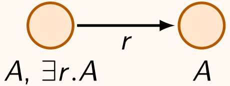
$$\mathcal{O} = \{\ A \sqsubseteq \exists r.A\ \}$$

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

$$\mathcal{O} = \{ \ A \sqsubseteq \exists r.A \ \}$$

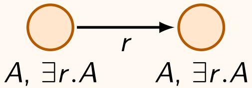


Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

$$\mathcal{O} = \{\ A \sqsubseteq \exists r.A\ \}$$

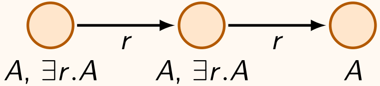


Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

$\mathcal{O} = \{\ A \sqsubseteq \exists r.A\ \}$

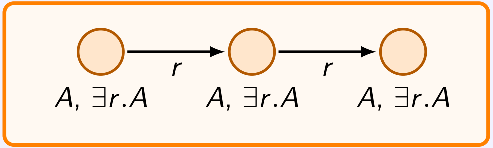


Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

$$\mathcal{O} = \{ A \sqsubseteq \exists r.A \}$$



Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

To have a decision procedure we have to know when to stop.

- How else would we ever know if $\mathcal{O} \not\models C_0 \sqsubseteq D_0$?

Fixing Problem 1: Unbounded Introduction of Concepts

Fixing Problem 1: Unbounded Introduction of Concepts

All inference rules need to be applied with the following side condition:

- If we assign a concept C, then C must occur somewhere in our input
  - in the ontology or in the entailment $C_0 \sqsubseteq D_0$
  - "occur in" includes nested concepts

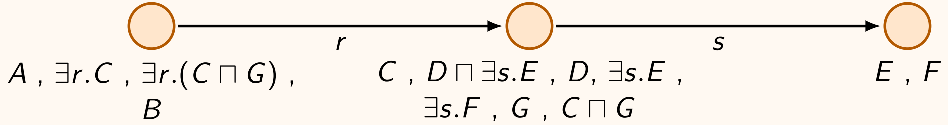
Fixing Problem 1: Unbounded Introduction of Concepts

All inference rules need to be applied with the following side condition:

- If we assign a concept C, then C must occur somewhere in our input
  - in the ontology or in the entailment $C_0 \sqsubseteq D_0$
  - "occur in" includes nested concepts
- Idea:
  - If a concept does not occur at least nested in the TBox, then it will never be needed to trigger the $\sqsubseteq$-rule
  - Other concepts are only relevant if they occur in D_0

Fixing Problem 1: Our Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F, \\ \exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$



With this restriction on the rules, there is actually no more step we can do.

Fixing Problem 2: Unbounded Introduction of Individuals

Fixing Problem 2: Unbounded Introduction of Individuals

- For every individual, we remember the initial concept

Fixing Problem 2: Unbounded Introduction of Individuals

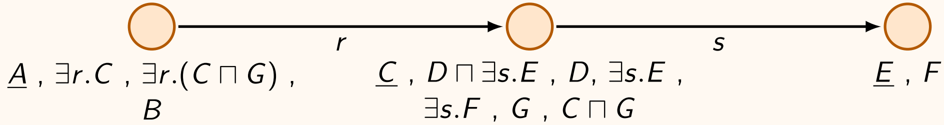
- For every individual, we remember the initial concept
- We modify the ∃-rule 1 as follows:

> ∃-rule 1: If d has $\exists r.C$ assigned:
>
> 1. If there is some e with initial concept $\underline{C}$, make e the r-successor of d
> 2. Otherwise, add a new r-successor to d, and assign to it as initial concept $\underline{C}$

Fixing Problem 2: Our First Example

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F,$$
$$\exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

The outcome of our example remains the same with this modification, only that we now remember the initial concepts:



Fixing Problem 2: The Problematic Example

The other example now stops fairly soon:

$$\mathcal{O} = \{\ A \sqsubseteq \exists r.A\ \}$$

Fixing Problem 2: The Problematic Example

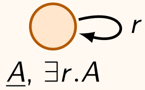
The other example now stops fairly soon:

$\mathcal{O} = \{ \ A \sqsubseteq \exists r.A \ \}$

Fixing Problem 2: The Problematic Example

The other example now stops fairly soon:

$\mathcal{O} = \{ \ A \sqsubseteq \exists r.A \ \}$



The $\mathcal{EL}$-Completion Method

The resulting method is called the $\mathcal{EL}$-Completion Method.

The $\mathcal{EL}$-Completion Method

The resulting method is called the $\mathcal{EL}$-Completion Method.

We will see that it is indeed a decision procedure:

- It is sound, complete and terminating.

The Final $\mathcal{EL}$-Completion Rules

$\top$-rule: Add $\top$ to any individual.

$\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.

$\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.

$\exists$-rule 1: If d has $\exists r.C$ assigned:

1. If there is an element e with initial concept $\underline{C}$ assigned, make e the r-successor of d.
2. Otherwise, add a new r-successor to d, and assign to it as initial concept $\underline{C}$.

$\exists$-rule 2: If d has an r-successor with C assigned, add $\exists r.C$ to d.

$\sqsubseteq$-rule: If d has C assigned and $C \sqsubseteq D \in \mathcal{T}$, then also assign D to d

The $\mathcal{EL}$-Completion Algorithm

Decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$

1. Start with initial element d_0, assign to $\underline{C_0}$ to it as initial concept
2. Set `changed` $:=$ **true**
3. While `changed` $=$ **true** :
   - 3.1 Set `changed` $:=$ **false**
   - 3.2 For every element d in the current interpretation:
     - 3.2.1 Apply all the rules on d in all possible ways so that only concepts from the input get assigned
     - 3.2.2 If a new element was added or a new concept assigned, set `changed` $=$ **true**
4. If D_0 was assigned to d_0, return `YES`, otherwise return `NO`

Concepts from the input: occur, possibly nested, explicitly in $\mathcal{O}$, C_0 or D_0

Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \\ A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B,$$
$$A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$

Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$

d_0

$\underline{A \sqcap \exists r.B}$

Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B,$$
$$A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$

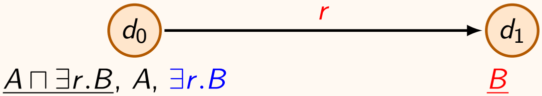
d_0

$\underline{A \sqcap \exists r.B}$, A, $\exists r.B$

Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

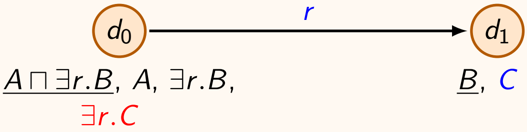
$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$

Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

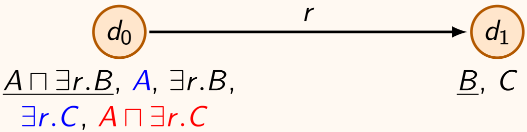
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

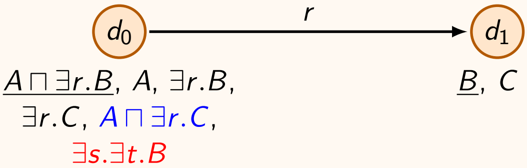
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

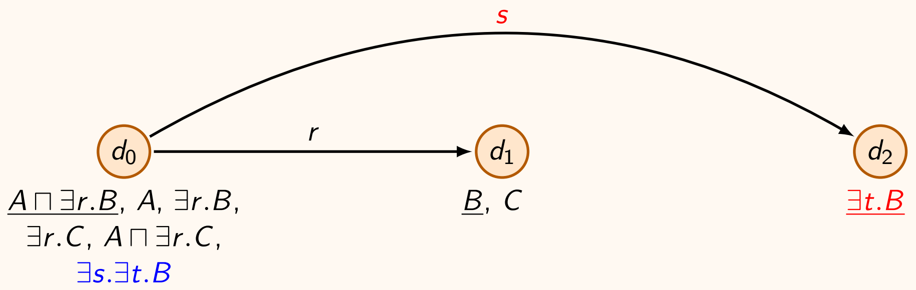
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

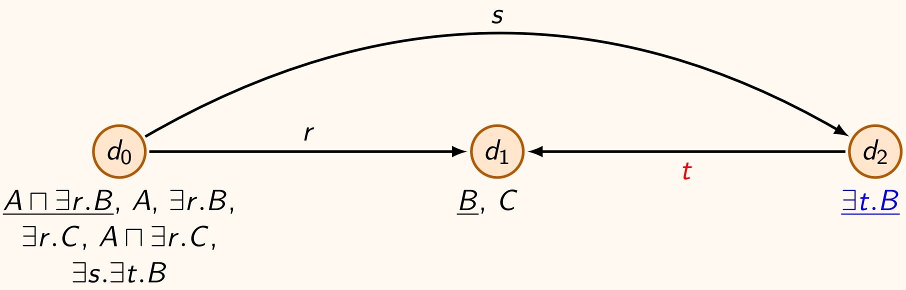
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

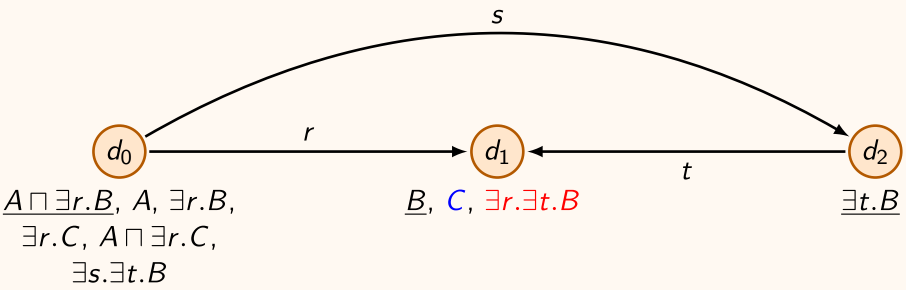
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

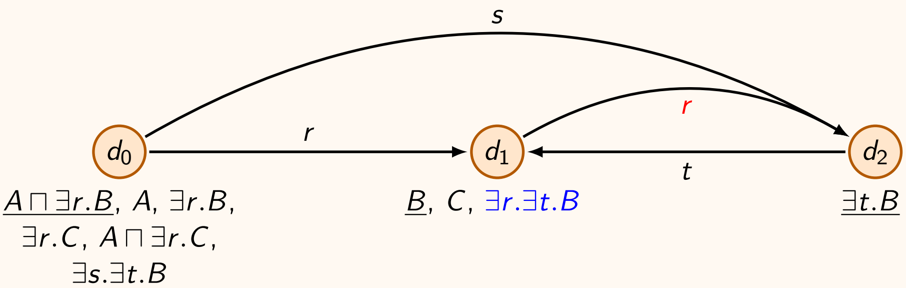
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

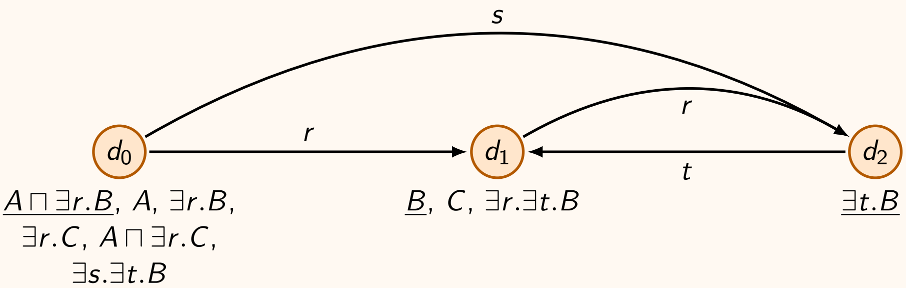
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

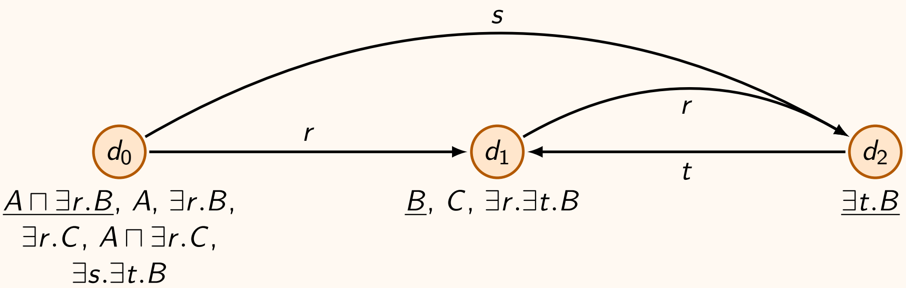
We want to decide whether $\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A$



Example

$$\mathcal{O} = \mathcal{T} = \{ \quad B \sqsubseteq C, \qquad C \sqsubseteq \exists r.\exists t.B, \qquad A \sqcap \exists r.C \sqsubseteq \exists s.\exists t.B \quad \}$$

We want to decide whether $\cancel{\mathcal{O} \models A \sqcap \exists r.B \sqsubseteq \exists t.A}$ NO



Termination

Lemma: The algorithm stops after at most n^2 steps, where n is the number of concepts in the input

Termination

Lemma: The algorithm stops after at most n^2 steps, where n is the number of concepts in the input

Proof:

- We add at most one element for each concept in the input: at most n elements.
  - Because we never introduce two elements with the same initial concept.

Termination

Lemma: The algorithm stops after at most n^2 steps, where n is the number of concepts in the input

Proof:

- We add at most one element for each concept in the input: at most n elements.
  - Because we never introduce two elements with the same initial concept.
- For each element, we only add concepts occurring in the input: at most n concepts per element.

Termination

Lemma: The algorithm stops after at most n^2 steps, where n is the number of concepts in the input

Proof:

- We add at most one element for each concept in the input: at most n elements.
  - Because we never introduce two elements with the same initial concept.
- For each element, we only add concepts occurring in the input: at most n concepts per element.
- In each step, we either add an element or assign a new concept to an existing element: at most $n \times n$ steps. □

Termination

Lemma: The algorithm stops after at most n^2 steps, where n is the number of concepts in the input

Proof:

- We add at most one element for each concept in the input: at most n elements.
  - Because we never introduce two elements with the same initial concept.
- For each element, we only add concepts occurring in the input: at most n concepts per element.
- In each step, we either add an element or assign a new concept to an existing element: at most $n \times n$ steps. □

Note: This is better than for propositional logic!

Completeness

Lemma: If the algorithm returns `NO`, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Completeness

Lemma: If the algorithm returns `NO`, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.

Completeness

Lemma: If the algorithm returns `NO`, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.

Completeness

Lemma: If the algorithm returns $\texttt{NO}$, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

Completeness

Lemma: If the algorithm returns `NO`, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

  $\Rightarrow$ $\mathcal{I} \models C \sqsubseteq D$ for every $C \sqsubseteq D \in \mathcal{T}$.

Completeness

Lemma: If the algorithm returns NO, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

  $\Rightarrow$ $\mathcal{I} \models C \sqsubseteq D$ for every $C \sqsubseteq D \in \mathcal{T}$.
- We have $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin D_0^{\mathcal{I}}$.
  - by the previous observation

Completeness

Lemma: If the algorithm returns NO, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

  $\Rightarrow$ $\mathcal{I} \models C \sqsubseteq D$ for every $C \sqsubseteq D \in \mathcal{T}$.
- We have $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin D_0^{\mathcal{I}}$.
  - by the previous observation

$\Rightarrow$ $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models C_0 \sqsubseteq D_0$,

Completeness

Lemma: If the algorithm returns `NO`, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

  $\Rightarrow$ $\mathcal{I} \models C \sqsubseteq D$ for every $C \sqsubseteq D \in \mathcal{T}$.
- We have $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin D_0^{\mathcal{I}}$.
  - by the previous observation

$\Rightarrow$ $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models C_0 \sqsubseteq D_0$, but what about the ABox?

Completeness

Lemma: If the algorithm returns NO, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

  $\Rightarrow$ $\mathcal{I} \models C \sqsubseteq D$ for every $C \sqsubseteq D \in \mathcal{T}$.
- We have $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin D_0^{\mathcal{I}}$.
  - by the previous observation

$\Rightarrow$ $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models C_0 \sqsubseteq D_0$, but what about the ABox?

- The ABox is not relevant: we can extend any model of $\mathcal{O}$ by adding the stuff in $\mathcal{I}$! $\square$

Soundness

Lemma: If the algorithm returns `YES`, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Soundness

Lemma: If the algorithm returns `YES`, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns `YES`.

Soundness

Lemma: If the algorithm returns YES, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns YES.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.

Soundness

Lemma: If the algorithm returns $\texttt{YES}$, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns $\texttt{YES}$.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.

Soundness

Lemma: If the algorithm returns $\texttt{YES}$, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns $\texttt{YES}$.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d'_0 \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.
  - Only difference: $\exists$-rule 1 now uses a corresponding, existing role successor in $\mathcal{I}'$.

Soundness

Lemma: If the algorithm returns $\mathtt{YES}$, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns $\mathtt{YES}$.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.
  - Only difference: $\exists$-rule 1 now uses a corresponding, existing role successor in $\mathcal{I}'$.
- One can show that, whenever the algorithm would assign C to d, then also $d \in C^{\mathcal{I}'}$.

Soundness

Lemma: If the algorithm returns YES, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns YES.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.
  - Only difference: $\exists$-rule 1 now uses a corresponding, existing role successor in $\mathcal{I}'$.
- One can show that, whenever the algorithm would assign C to d, then also $d \in C^{\mathcal{I}'}$.
- The algorithm assigns D_0 to d_0', and therefore $d_0' \in D_0^{\mathcal{I}'}$.

Soundness

Lemma: If the algorithm returns $\texttt{YES}$, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns $\texttt{YES}$.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.
  - Only difference: $\exists$-rule 1 now uses a corresponding, existing role successor in $\mathcal{I}'$.
- One can show that, whenever the algorithm would assign C to d, then also $d \in C^{\mathcal{I}'}$.
- The algorithm assigns D_0 to d_0', and therefore $d_0' \in D_0^{\mathcal{I}'}$.
- $\mathcal{I}'$ was an arbitrary model, and d_0' an arbitrary element of C_0', so the same would happen for all models and all elements in C_0'.

Soundness

Lemma: If the algorithm returns YES, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns YES.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.
  - Only difference: $\exists$-rule 1 now uses a corresponding, existing role successor in $\mathcal{I}'$.
- One can show that, whenever the algorithm would assign C to d, then also $d \in C^{\mathcal{I}'}$.
- The algorithm assigns D_0 to d_0', and therefore $d_0' \in D_0^{\mathcal{I}'}$.
- $\mathcal{I}'$ was an arbitrary model, and d_0' an arbitrary element of C_0', so the same would happen for all models and all elements in C_0'.
- Hence, $\mathcal{O} \models C \sqsubseteq D$. □

The Completion Algorithm is a Decision Procedure

These lemmas together give us the following theorem:

Theorem: The $\mathcal{EL}$ Completion Algorithm is a decision procedure for $\mathcal{EL}$ concept subsumption from $\mathcal{EL}$ ontologies.

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

- Instance checking: determine whether $\mathcal{O} \models a : C$
  - Same procedure, but start with the ABox

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

- Instance checking: determine whether $\mathcal{O} \models a : C$
  - Same procedure, but start with the ABox
- Classification: determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Start with a node for every concept name

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

- Instance checking: determine whether $\mathcal{O} \models a : C$
  - Same procedure, but start with the ABox
- Classification: determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Start with a node for every concept name
- Materialization: determine all $\mathcal{O} \models a : A$ where $a \in \mathbf{I}$ and $A \in \mathbf{C}$
  - Same algorithm as for instance checking.

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

- Instance checking: determine whether $\mathcal{O} \models a : C$
  - Same procedure, but start with the ABox
- Classification: determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Start with a node for every concept name
- Materialization: determine all $\mathcal{O} \models a : A$ where $a \in \mathbf{I}$ and $A \in \mathbf{C}$
  - Same algorithm as for instance checking.
  - *(Note: role assertions cannot be inferred from other axioms in $\mathcal{EL}$)*

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

- Instance checking: determine whether $\mathcal{O} \models a : C$
  - Same procedure, but start with the ABox
- Classification: determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Start with a node for every concept name
- Materialization: determine all $\mathcal{O} \models a : A$ where $a \in \mathbf{I}$ and $A \in \mathbf{C}$
  - Same algorithm as for instance checking.
  - *(Note: role assertions cannot be inferred from other axioms in $\mathcal{EL}$)*

Note: Algorithms always takes at most n^2 steps!

- Better complexity as propositional logic.
- Modern $\mathcal{EL}$ reasoners like ELK process 10,000s of axioms in seconds.

Beyond $\mathcal{EL}$?

There are some extensions to $\mathcal{EL}$ that still allow reasoning in polynomial time, even if we allow $\perp$.

Beyond $\mathcal{EL}$?

There are some extensions to $\mathcal{EL}$ that still allow reasoning in polynomial time, even if we allow $\bot$.

However, any other concept constructor we have seen until now makes reasoning even harder than for propositional logic:

- disjunction: $\sqcup$
- negation: $\neg$
- value restrictions: $\forall$

Beyond $\mathcal{EL}$?

There are some extensions to $\mathcal{EL}$ that still allow reasoning in polynomial time, even if we allow $\bot$.

However, any other concept constructor we have seen until now makes reasoning even harder than for propositional logic:

- disjunction: $\sqcup$
- negation: $\neg$
- value restrictions: $\forall$

Allowing any of these three constructs makes reasing $\textsc{ExpTime}$-hard, which means reasoning may require a number of steps that is exponential in the size of the ontology (indepedently of whether $P = NP$).

Challenges with $\mathcal{ALC}$

First challenge: Disjunction $C \sqcup D$

- For instances of $C \sqcup D$, we do not know whether we need to satisfy C or D .

$\Rightarrow$ Case distinction required

$\Rightarrow$ There is no canonical model as for $\mathcal{EL}$

Challenges with $\mathcal{ALC}$

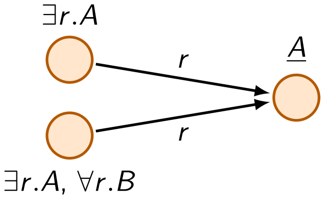
Second challenge: Value restrictions $\forall r.C$

- We need a rule like the following:

> $\forall$-rule: If d as $\forall r.C$ assigned and e is an r-succesor of d, then assign C to e

$\Rightarrow$ Additional concepts come from predecessors of a node.

$\Rightarrow$ We cannot reuse individuals as before.



Challenges with $\mathcal{ALC}$

Third challenge: Negation $\neg C$

- Concepts can be unsatisfiable $\rightarrow$ model construction can fail
  - cannot have A and $\neg A$ at the same time

Challenges with $\mathcal{ALC}$

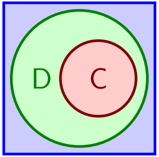
Third challenge: Negation $\neg C$

- Concepts can be unsatisfiable $\rightarrow$ model construction can fail
  - cannot have A and $\neg A$ at the same time
- Nested expressions
  - What do we do with $\neg(A \sqcap \neg(B \sqcup C))$?

Challenges with $\mathcal{ALC}$

Third challenge: Negation $\neg C$

- Concepts can be unsatisfiable $\rightarrow$ model construction can fail
  - cannot have A and $\neg A$ at the same time
- Nested expressions
  - What do we do with $\neg(A \sqcap \neg(B \sqcup C))$?
- Different ways to express the same thing:
  - $C \sqsubseteq D$
  - $\neg D \sqsubseteq \neg C$
  - $C \sqcap \neg D \sqsubseteq \bot$
  - $\top \sqsubseteq \neg C \sqcup D$



Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

Given an ontology $\mathcal{O}$ and a concept C, C is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of C and $\mathcal{O}$).

Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

> Given an ontology $\mathcal{O}$ and a concept C, C is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of C and $\mathcal{O}$).

- For $\mathcal{EL}$, this wouldn't have made sense, since every concept is satisfiable

Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

> Given an ontology $\mathcal{O}$ and a concept C, C is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of C and $\mathcal{O}$).

- For $\mathcal{EL}$, this wouldn't have made sense, since every concept is satisfiable
- In $\mathcal{ALC}$, we can reduce many problems to it:
  - To decide $\mathcal{O} \models C \sqsubseteq D$, we check whether $C \sqcap \neg D$ is unsatisfiable
  - To decide consistency of $\mathcal{O}$, we check whether $\top$ is satisfiable