Description Logic 2: Reasoning

Leif Harald Karlsen

Autumn 2016

Last time

- As we saw last time, description logics allows us to model knowledge in a natural way.
- Today we will see why we make the restrictions, and what makes exactly these restrictions important.

Contents

Contents

Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

First, we will make the following assumptions:

- We only allow equivalence axioms, $A \equiv D$, where A is atomic and D is not atomic. Each atomic concept should only occur once on a left-hand side.

Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

First, we will make the following assumptions:

- We only allow equivalence axioms, $A \equiv D$, where A is atomic and D is not atomic. Each atomic concept should only occur once on a left-hand side.
- We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where D is defined in terms of A.

Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

First, we will make the following assumptions:

- We only allow equivalence axioms, $A \equiv D$, where A is atomic and D is not atomic. Each atomic concept should only occur once on a left-hand side.
- We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where D is defined in terms of A.
- We only allow ABox axioms on the form $A(c)$ for atomic concepts A (and $R(a, b)$ as usual). (Not really a restriction, as $D(c)$ for complex D can be expressed as $A_D \equiv D$ and $A_D(c)$ for some fresh concept name A_D)

Example ontology in $\mathcal{ALCN}$

TBox:

$$
\begin{aligned}
Animal &\equiv\ \leq 2\ hasParent \sqcap \geq 2\ hasParent \\
Donkey &\equiv Animal \sqcap Stubborn \\
Horse &\equiv Animal \sqcap \neg Stubborn \\
Mule &\equiv Animal \sqcap \exists hasParent.Horse \sqcap \exists hasParent.Donkey
\end{aligned}
$$

Example ontology in $\mathcal{ALCN}$

TBox:

$$
\begin{aligned}
Animal &\equiv\ \leq 2\ hasParent \sqcap \geq 2\ hasParent \\
Donkey &\equiv Animal \sqcap Stubborn \\
Horse &\equiv Animal \sqcap \neg Stubborn \\
Mule &\equiv Animal \sqcap \exists hasParent.Horse \sqcap \exists hasParent.Donkey
\end{aligned}
$$

ABox:

Horse(mary)   Mule(peter)   Horse(hannah)

hasParent(peter, mary)   hasParent(peter, carl)
hasParent(sven, hannah)   hasParent(sven, carl)

Contents

Problems

For concepts C and D:

- **Satisfiability** of concepts (C is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty).

Problems

For concepts C and D:

- **Satisfiability** of concepts (C is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty).
- **Subsumption** of concepts (C is subsumed by D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

Problems

For concepts C and D:

- **Satisfiability** of concepts (C is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty).
- **Subsumption** of concepts (C is subsumed by D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Equivalence** of concepts (C is equivalent to D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

Problems

For concepts C and D:

- **Satisfiability** of concepts (C is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty).
- **Subsumption** of concepts (C is subsumed by D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Equivalence** of concepts (C is equivalent to D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Disjointness** of concepts (C is disjoint from D w.r.t $\mathcal{T}$, if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$).

Problems

For concepts C and D:

- **Satisfiability** of concepts (C is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty).
- **Subsumption** of concepts (C is subsumed by D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Equivalence** of concepts (C is equivalent to D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Disjointness** of concepts (C is disjoint from D w.r.t $\mathcal{T}$, if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$).

For knowledge bases $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$:

- **Instance checking** (whether for some concept C and individual a, $\mathcal{K} \models C(a)$).

Problems

For concepts C and D:

- **Satisfiability** of concepts (C is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty).
- **Subsumption** of concepts (C is subsumed by D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Equivalence** of concepts (C is equivalent to D w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
- **Disjointness** of concepts (C is disjoint from D w.r.t $\mathcal{T}$, if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$).

For knowledge bases $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$:

- **Instance checking** (whether for some concept C and individual a, $\mathcal{K} \models C(a)$).
- **ABox consistency** (whether $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$).

Reductions

Theorem

For concepts C and D, we have

(i) C *is subsumed by* $D \Leftrightarrow C \sqcap \neg D$ *is unsatisfiable;*

Reductions

Theorem

For concepts C and D, we have

(i) C *is subsumed by* $D \Leftrightarrow C \sqcap \neg D$ *is unsatisfiable;*

(ii) C *and* D *are equivalent* $\Leftrightarrow$ *both* $(C \sqcap \neg D)$ *and* $(\neg C \sqcap D)$ *are unsatisfiable;*

Reductions

Theorem

For concepts C and D, we have

(i) C *is subsumed by* $D \Leftrightarrow C \sqcap \neg D$ *is unsatisfiable;*

(ii) C *and* D *are equivalent* $\Leftrightarrow$ *both* $(C \sqcap \neg D)$ *and* $(\neg C \sqcap D)$ *are unsatisfiable;*

(iii) C *and* D *are disjoint* $\Leftrightarrow C \sqcap D$ *is unsatisfiable.*

Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

- First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where D' is obtained by recursively replacing every name concept C in D with C' if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:

Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

- First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where D' is obtained by recursively replacing every name concept C in D with C' if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:

$$Donkey \equiv Animal \sqcap Stubborn \rightsquigarrow$$
$$Donkey \equiv \leq 2\, hasParent \sqcap \geq 2\, hasParent \sqcap Stubborn$$

Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

- First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where D' is obtained by recursively replacing every name concept C in D with C' if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:

$$Donkey \equiv Animal \sqcap Stubborn \rightsquigarrow$$
$$Donkey \equiv \leq 2\, hasParent \sqcap \geq 2\, hasParent \sqcap Stubborn$$

- Then we replace every assertion $A(a)$ with $D'(a)$ in $\mathcal{A}$, if $A \equiv D'$ is in $\mathcal{T}'$. E.g.:

$$Donkey(peter) \rightsquigarrow$$
$$(\leq 2\, hasParent \sqcap \geq 2\, hasParent \sqcap Stubborn)(peter)$$

Important results

Theorem

Assume $C \equiv D$ is replaced by $C \equiv D'$ in an expansion of $\langle \mathcal{A}, \mathcal{T} \rangle$ to $\langle \mathcal{A}', \emptyset \rangle$. Then:

(i) C is satisfiable w.r.t. $\mathcal{T} \Leftrightarrow \{D'(x)\}$ is consistent.

Important results

Theorem

Assume $C \equiv D$ is replaced by $C \equiv D'$ in an expansion of $\langle \mathcal{A}, \mathcal{T} \rangle$ to $\langle \mathcal{A}', \emptyset \rangle$. Then:

(i) C is satisfiable w.r.t. $\mathcal{T}$ $\Leftrightarrow$ $\{D'(x)\}$ is consistent.

(ii) $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ $\Leftrightarrow$ $\mathcal{A}'$ is consistent.

Important results

Theorem

Assume $C \equiv D$ *is replaced by* $C \equiv D'$ *in an expansion of* $\langle \mathcal{A}, \mathcal{T} \rangle$ *to* $\langle \mathcal{A}', \emptyset \rangle$*. Then:*

(i) C *is satisfiable w.r.t.* $\mathcal{T} \Leftrightarrow \{D'(x)\}$ *is consistent.*

(ii) $\mathcal{A}$ *is consistent w.r.t.* $\mathcal{T} \Leftrightarrow \mathcal{A}'$ *is consistent.*

(iii) $\langle \mathcal{A}, \mathcal{T} \rangle \vDash C(a) \Leftrightarrow \mathcal{A}' \cup \{\neg D'(a)\}$ *is inconsistent.*

Negated Normal Form

For our reasoning algorithm, we need our concepts in Negated Normal Form (NFF):

$$\neg(C \sqcap D) \leadsto \neg C \sqcup \neg D$$
$$\neg(C \sqcup D) \leadsto \neg C \sqcap \neg D$$
$$\neg \exists R.C \leadsto \forall R.\neg C$$
$$\neg \forall R.C \leadsto \exists R.\neg C$$
$$\neg \leq n\, R \leadsto \geq (n+1)\, R$$
$$\neg \geq n\, R \leadsto \leq (n-1)\, R$$

Contents

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

(ii) While a rule is applicable to an element $\mathcal{A} \in \mathcal{S}_i$:

- Apply rule to $\mathcal{A}$ resulting in $\mathcal{S}_{\mathcal{A}} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$.

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

(ii) While a rule is applicable to an element $\mathcal{A} \in \mathcal{S}_i$:
   - Apply rule to $\mathcal{A}$ resulting in $\mathcal{S}_\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$.
   - Set $\mathcal{S}_{i+1} = (\mathcal{S}_i \setminus \{\mathcal{A}\}) \cup \mathcal{S}_\mathcal{A}$.

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

(ii) While a rule is applicable to an element $\mathcal{A} \in \mathcal{S}_i$:
   - Apply rule to $\mathcal{A}$ resulting in $\mathcal{S}_\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$.
   - Set $\mathcal{S}_{i+1} = (\mathcal{S}_i \backslash \{\mathcal{A}\}) \cup \mathcal{S}_\mathcal{A}$.

$\sqcap$-rule **Condition:** $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

(ii) While a rule is applicable to an element $\mathcal{A} \in \mathcal{S}_i$:

- Apply rule to $\mathcal{A}$ resulting in $\mathcal{S}_\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$.
- Set $\mathcal{S}_{i+1} = (\mathcal{S}_i \setminus \{\mathcal{A}\}) \cup \mathcal{S}_\mathcal{A}$.

$\sqcap$-rule	**Condition:**	$\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
	Action:	$\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.
$\sqcup$-rule	**Condition:**	$\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
	Action:	$\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$.

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

(ii) While a rule is applicable to an element $\mathcal{A} \in \mathcal{S}_i$:

- Apply rule to $\mathcal{A}$ resulting in $\mathcal{S}_\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$.
- Set $\mathcal{S}_{i+1} = (\mathcal{S}_i \backslash \{\mathcal{A}\}) \cup \mathcal{S}_\mathcal{A}$.

$\sqcap$-rule **Condition:** $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

$\sqcup$-rule **Condition:** $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$.

$\exists$-rule **Condition:** $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no z such that $C(z)$ and $R(x, z)$ in $\mathcal{A}$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x, y)\}$, y fresh.

Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $\mathcal{S}_0 = \{\mathcal{A}_0\}$ (on NNF).

(ii) While a rule is applicable to an element $\mathcal{A} \in \mathcal{S}_i$:

- Apply rule to $\mathcal{A}$ resulting in $\mathcal{S}_\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$.
- Set $\mathcal{S}_{i+1} = (\mathcal{S}_i \backslash \{\mathcal{A}\}) \cup \mathcal{S}_\mathcal{A}$.

$\sqcap$-rule	**Condition:**	$\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
	Action:	$\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.
$\sqcup$-rule	**Condition:**	$\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
	Action:	$\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$.
$\exists$-rule	**Condition:**	$\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no z such that $C(z)$ and $R(x, z)$ in $\mathcal{A}$.
	Action:	$\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x, y)\}$, y fresh.
$\forall$-rule	**Condition:**	$\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x, y)$, but not $C(y)$.
	Action:	$\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

Extension with $\mathcal{N}$

$\geq$-rule **Condition:** $\mathcal{A}$ contains $(\geq n\,R)(x)$, but there are no $z_1, \ldots, z_n$ such that $R(x, z_i)$ $(1 \leq i \leq n)$ and $z_i \neq z_j$ $(1 \leq i < j \leq n)$ are in $\mathcal{A}$.

Action: $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid (1 \leq i < j \leq n)\}$, distinct and fresh $y_1, \ldots, y_n$.

Extension with $\mathcal{N}$

$\geq$-rule **Condition:** $\mathcal{A}$ contains $(\geq n\, R)(x)$, but there are no $z_1, \ldots, z_n$ such that $R(x, z_i)$ $(1 \leq i \leq n)$ and $z_i \neq z_j$ $(1 \leq i < j \leq n)$ are in $\mathcal{A}$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid (1 \leq i < j \leq n)\}$, distinct and fresh $y_1, \ldots, y_n$.

$\leq$-rule **Condition:** $\mathcal{A}$ contains $(\leq n\, R)(x)$ and $R(x, y_1), \ldots, R(x, y_{n+1})$ for distinct names $y_1, \ldots, y_{n+1}$, but not $y_i \neq y_j$ for some $i \neq j$.
Action: $\mathcal{A}_{i,j} = \mathcal{A}[y_i / y_j]$, for each pair y_i, y_j such that $i > j$ and $y_i \neq y_j$ is not in $\mathcal{A}$.

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\mathcal{A}_0 \quad = \quad \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\}$$

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\begin{array}{lcll} \mathcal{A}_0 & = & \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\} & \\ \mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\forall R.(\neg C \sqcup D))(x_0), (\exists R.(C \sqcap D))(x_0)\}\} & \text{by } \sqcap\text{-rule} \end{array}$$

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\begin{array}{rcll}
\mathcal{A}_0 & = & \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\forall R.(\neg C \sqcup D))(x_0), (\exists R.(C \sqcap D))(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} & \text{by } \exists\text{-rule}
\end{array}$$

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\forall R.(\neg C \sqcup D))(x_0), (\exists R.(C \sqcap D))(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_3 & = & \{\mathcal{A}_2^1 \cup \{(\neg C \sqcup D)(y)\}\} & \text{by } \forall\text{-rule}
\end{array}$$

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\forall R.(\neg C \sqcup D))(x_0), (\exists R.(C \sqcap D))(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_3 & = & \{\mathcal{A}_2^1 \cup \{(\neg C \sqcup D)(y)\}\} & \text{by } \forall\text{-rule} \\
\mathcal{A}_4 & = & \{\mathcal{A}_3^1 \cup \{(\neg C)(y)\}, \qquad \mathcal{A}_3^1 \cup \{D(y)\}\} & \text{by } \sqcup\text{-rule}
\end{array}$$

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\begin{array}{rcll}
\mathcal{A}_0 & = & \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\forall R.(\neg C \sqcup D))(x_0), (\exists R.(C \sqcap D))(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_3 & = & \{\mathcal{A}_2^1 \cup \{(\neg C \sqcup D)(y)\}\} & \text{by } \forall\text{-rule} \\
\mathcal{A}_4 & = & \{\mathcal{A}_3^1 \cup \{(\neg C)(y)\}, \quad \mathcal{A}_3^1 \cup \{D(y)\}\} & \text{by } \sqcup\text{-rule} \\
\mathcal{A}_5 & = & \{\mathcal{A}_4^1 \cup \{C(y), D(y)\}, \quad \mathcal{A}_4^2\} & \text{by } \sqcap\text{-rule}
\end{array}$$

Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the k-th set in $\mathcal{A}_n$)

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\forall R.(\neg C \sqcup D))(x_0), (\exists R.(C \sqcap D))(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_3 & = & \{\mathcal{A}_2^1 \cup \{(\neg C \sqcup D)(y)\}\} & \text{by } \forall\text{-rule} \\
\mathcal{A}_4 & = & \{\mathcal{A}_3^1 \cup \{(\neg C)(y)\}, \quad \mathcal{A}_3^1 \cup \{D(y)\}\} & \text{by } \sqcup\text{-rule} \\
\mathcal{A}_5 & = & \{\mathcal{A}_4^1 \cup \{C(y), D(y)\}, \quad \mathcal{A}_4^2\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_6 & = & \{\mathcal{A}_5^1, \quad \mathcal{A}_5^2 \cup \{C(y), D(y)\}\} & \text{by } \sqcap\text{-rule}
\end{array}$$

Now no more rules apply.

Example derivation(2)

We want to check whether the concept

$$\leq 1\ R \sqcap \geq 2\ R$$

is satisfiable.

Example derivation(2)

We want to check whether the concept

$$\leq 1\ R \sqcap \geq 2\ R$$

is satisfiable.

$$\mathcal{A}_0 \quad = \quad \{\{(\leq 1\ R \sqcap \geq 2\ R)(x_0)\}\}$$

Example derivation(2)

We want to check whether the concept

$$\leq 1\ R \sqcap \geq 2\ R$$

is satisfiable.

$$\begin{array}{rcl} \mathcal{A}_0 & = & \{\{(\leq 1\ R \sqcap \geq 2\ R)(x_0)\}\} \\ \mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\geq 2\ R)(x_0)\}\} \qquad \text{by } \sqcap\text{-rule} \end{array}$$

Example derivation(2)

We want to check whether the concept

$$\leq 1\ R \sqcap \geq 2\ R$$

is satisfiable.

$$\begin{array}{lll}
\mathcal{A}_0 & = & \{\{(\leq 1\ R \sqcap \geq 2\ R)(x_0)\}\} \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\geq 2\ R)(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y_1), R(x_0, y_2), y_1 \neq y_2\}\} & \text{by } \geq\text{-rule}
\end{array}$$

Example derivation(2)

We want to check whether the concept

$$\leq 1\ R \sqcap \geq 2\ R$$

is satisfiable.

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\leq 1\ R \sqcap \geq 2\ R)(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\geq 2\ R)(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y_1), R(x_0, y_2), y_1 \neq y_2\}\} & \text{by } \geq\text{-rule}
\end{array}$$

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

$$\mathcal{A}_0 \quad = \quad \{\{(\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D)(x_0)\}\}$$

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

$$\begin{aligned}
\mathcal{A}_0 &= \{\{(\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D)(x_0)\}\} \\
\mathcal{A}_1 &= \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\exists R.C)(x_0), (\exists R.D)(x_0)\}\} \quad \text{by } \sqcap\text{-rule}
\end{aligned}$$

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D)(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\exists R.C)(x_0), (\exists R.D)(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y_1), C(y_1)\}\} & \text{by } \exists\text{-rule}
\end{array}$$

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D)(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\exists R.C)(x_0), (\exists R.D)(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y_1), C(y_1)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_3 & = & \{\mathcal{A}_2^1 \cup \{R(x_0, y_2), D(y_2)\}\} & \text{by } \exists\text{-rule}
\end{array}$$

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

$$\begin{aligned}
\mathcal{A}_0 &= \{\{(\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D)(x_0)\}\} \\
\mathcal{A}_1 &= \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\exists R.C)(x_0), (\exists R.D)(x_0)\}\} && \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 &= \{\mathcal{A}_1^1 \cup \{R(x_0, y_1), C(y_1)\}\} && \text{by } \exists\text{-rule} \\
\mathcal{A}_3 &= \{\mathcal{A}_2^1 \cup \{R(x_0, y_2), D(y_2)\}\} && \text{by } \exists\text{-rule} \\
\mathcal{A}_4 &= \{\mathcal{A}_3^1[y_2/y_1]\} && \text{by } \leq\text{-rule}
\end{aligned}$$

Example derivation(3)

We want to check whether the concept

$$\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D$$

is satisfiable.

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \{\{(\leq 1\ R \sqcap \exists R.C \sqcap \exists R.D)(x_0)\}\} & \\
\mathcal{A}_1 & = & \{\mathcal{A}_0^1 \cup \{(\leq 1\ R)(x_0), (\exists R.C)(x_0), (\exists R.D)(x_0)\}\} & \text{by } \sqcap\text{-rule} \\
\mathcal{A}_2 & = & \{\mathcal{A}_1^1 \cup \{R(x_0, y_1), C(y_1)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_3 & = & \{\mathcal{A}_2^1 \cup \{R(x_0, y_2), D(y_2)\}\} & \text{by } \exists\text{-rule} \\
\mathcal{A}_4 & = & \{\mathcal{A}_3^1[y_2/y_1]\} & \text{by } \leq\text{-rule} \\
 & = & \{\mathcal{A}_2^1 \cup \{R(x_0, y_1), D(y_1)\} &
\end{array}$$

Contents

Definitions

Definition

(i) The algorithm terminates on one ABox $\mathcal{A}$ when no rules are applicable. Such an ABox is then called *complete*.

(ii) A *clash* is an obvious contradiction, that is, $\mathcal{A}$ contains a clash if either:

- $\bot(x) \in \mathcal{A}$, or
- $\{C(x), \neg C(x)\} \subseteq \mathcal{A}$, or
- $\{(\leq n\, R)(x)\} \cup$
  $\{R(x, y_i) \mid 1 \leq i \leq n+1\} \cup$
  $\{y_i \neq y_j \mid (1 \leq i < j \leq n+1)\} \subseteq \mathcal{A}$.

Soundness and completeness

Let $\hat{\mathcal{S}}$ be the set of complete ABoxes resulting from applying the tableau algorithm to $\{\mathcal{A}\}$.

Soundness and completeness

Let $\hat{\mathcal{S}}$ be the set of complete ABoxes resulting from applying the tableau algorithm to $\{\mathcal{A}\}$.

Theorem (Soundness)

If $\mathcal{A}$ has a model, then at least one of the ABoxes of $\hat{\mathcal{S}}$ has a model.

Proof.

Done by induction on the proofs, showing that each rule preserves consistency. □

Soundness and completeness

Let $\hat{\mathcal{S}}$ be the set of complete ABoxes resulting from applying the tableau algorithm to $\{\mathcal{A}\}$.

Theorem (Soundness)

If $\mathcal{A}$ has a model, then at least one of the ABoxes of $\hat{\mathcal{S}}$ has a model.

Proof.

Done by induction on the proofs, showing that each rule preserves consistency. □

Theorem (Completeness)

If at least one of the ABoxes, $\hat{\mathcal{A}}$ of $\hat{\mathcal{S}}$ is clash free, then at $\mathcal{A}$ has a model.

Proof.

Done by constructing a model for $\mathcal{A}$ from $\hat{\mathcal{A}}$. □

Assumptions

- We only allow equivalence axioms, $A \equiv D$, where A is atomic and D is not atomic. Each atomic concept should only occur once on a left-hand side.
- We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where D is defined in terms of A.
- We only allow ABox axioms on the form $A(c)$ for atomic concepts A (and $R(a, b)$ as usual). (Not really a restriction, as $D(c)$ for complex D can be expressed as $A_D \equiv D$ and $A_D(c)$ for some fresh concept name A_D)

Assumptions

- We only allow equivalence axioms, $A \equiv D$, where A is atomic and D is not atomic. Each atomic concept should only occur once on a left-hand side.
- We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where D is defined in terms of A.
- We only allow ABox axioms on the form $A(c)$ for atomic concepts A (and $R(a, b)$ as usual). (Not really a restriction, as $D(c)$ for complex D can be expressed as $A_D \equiv D$ and $A_D(c)$ for some fresh concept name A_D)
- **New:** We only allow ABoxes on the form $\{C(x_0)\}$ as input to the tableau algorithm.

Terminaton on single concept

Theorem (Termination)

If C_0 is an $\mathcal{ALCN}$-concept, then the tableau algorithm terminates on $\{\{C_0(x_0)\}\}$, that is, there cannot be an infinite sequence of rule applications

$$\{\{C_0(x_0)\}\} \to \mathcal{S}_1 \to \mathcal{S}_2 \to \dots$$

Proof-sketch of termination

To prove termination, we do the following:

- We first define a function f mapping each state $\mathcal{S}$ in the proof (each set of ABoxes) to a set Q for which there is a strict well-ordering $<$.

Proof-sketch of termination

To prove termination, we do the following:

- We first define a function f mapping each state $\mathcal{S}$ in the proof (each set of ABoxes) to a set Q for which there is a strict well-ordering $<$.
- Then, we prove that if $\mathcal{S}'$ is the result of applying a rule to a state $\mathcal{S}$, then $f(\mathcal{S}') < f(\mathcal{S})$.

Proof-sketch of termination

To prove termination, we do the following:

- We first define a function f mapping each state $\mathcal{S}$ in the proof (each set of ABoxes) to a set Q for which there is a strict well-ordering $<$.
- Then, we prove that if $\mathcal{S}'$ is the result of applying a rule to a state $\mathcal{S}$, then $f(\mathcal{S}') < f(\mathcal{S})$.
- The result then follows from the fact that any strictly decreasing sequence in a well-ordered set is finite.

Proof-sketch of termination

Lemma

Let $\mathcal{A}$ be an ABox contained in $\mathcal{S}_i$ for some $i \geq 1$.

- *For every individual $x \neq x_0$ occuring in $\mathcal{A}$, there is a unique sequence $R_1, \ldots, R_l$ ($l \geq 1$) of role names and a unique sequence $x_1, \ldots, x_{l-1}$ of individual names such that $\{R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_l(x_{l-1}, x)\} \subseteq \mathcal{A}$. In this case, we say that x occurs on level l in $\mathcal{A}$.*

Proof-sketch of termination

Lemma

Let $\mathcal{A}$ be an ABox contained in $\mathcal{S}_i$ for some $i \geq 1$.

- *For every individual $x \neq x_0$ occuring in $\mathcal{A}$, there is a unique sequence $R_1, \ldots, R_l$ ($l \geq 1$) of role names and a unique sequence $x_1, \ldots, x_{l-1}$ of individual names such that $\{R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_l(x_{l-1}, x)\} \subseteq \mathcal{A}$. In this case, we say that x occurs on level l in $\mathcal{A}$.*
- *If $C(x) \in \mathcal{A}$ for an individual x on level l, then the maximal role depth of C (i.e. the maximal nesting of constructors involving roles) is bounded by the maximal role depth of C_0 minus l. Consequently, the level of any individual in $\mathcal{A}$ is bounded by the maximal role depth of C_0.*

Proof-sketch of termination

Lemma (Cont.)

– *If* $C(x) \in \mathcal{A}$ *then* C *is a subdescription of* C_0*. Consequently, the number of different concept assertions on* x *is bounded by the size of* C_0*.*

Proof-sketch of termination

Lemma (Cont.)

- *If $C(x) \in \mathcal{A}$ then C is a subdescription of C_0. Consequently, the number of different concept assertions on x is bounded by the size of C_0.*
- *The number of different role successors of x in $\mathcal{A}$ (i.e. individuals y such that $R(x, y) \in \mathcal{A}$ for a role name R) is bounded by the sum of of the numbers occuring in the at-least restrictions in C_0 pluss the number of different existential restrictions in C_0.*

Finite tree model property

The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;

Finite tree model property

The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
- the depth of the tree is linearly bounded by the size of C_0;

Finite tree model property

The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
- the depth of the tree is linearly bounded by the size of C_0;
- the branching factor of the tree is bounded by the sum of the numbers occuring in the at-least restrictions pluss the number of different existential restrictions in C_0.

Finite tree model property

The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
- the depth of the tree is linearly bounded by the size of C_0;
- the branching factor of the tree is bounded by the sum of the numbers occuring in the at-least restrictions pluss the number of different existential restrictions in C_0.
- This means that $\mathcal{ALCN}$ enjoys the *finite tree model property*, that is, any satisfiable concept C_0 is satisfiable in a finite interpretation that has the shape of a tree whose root belongs to C_0.

Contents

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox?

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$\mathcal{A}_0 \quad = \quad \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\}$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$
\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\}
\end{aligned}
$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a, x_0), A(x_0)\}
\end{aligned}$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a,x_0), A(x_0)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R.A)(x_0)\}
\end{aligned}$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a,x_0), A(x_0)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R.A)(x_0)\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{R(x_0,x_1), A(x_1)\}
\end{aligned}$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$
\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a,x_0), A(x_0)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R.A)(x_0)\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{R(x_0,x_1), A(x_1)\} \\
\mathcal{A}_4 &= \mathcal{A}_1 \cup \{A(a), R(a,x_1), A(x_1)\}
\end{aligned}
$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$
\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a,x_0), A(x_0)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R.A)(x_0)\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{R(x_0,x_1), A(x_1)\} \\
\mathcal{A}_4 &= \mathcal{A}_1 \cup \{A(a), R(a,x_1), A(x_1)\} \\
&\vdots \\
\mathcal{A}_i &= \mathcal{A}_1 \cup \{A(a), R(a,x_j), A(x_j)\}
\end{aligned}
$$

Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$
\begin{aligned}
\mathcal{A}_0 &= \{R(a,a), (\leq 1\, R)(a), (\forall R.\exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a,x_0), A(x_0)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R.A)(x_0)\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{R(x_0,x_1), A(x_1)\} \\
\mathcal{A}_4 &= \mathcal{A}_1 \cup \{A(a), R(a,x_1), A(x_1)\} \\
&\vdots \\
\mathcal{A}_i &= \mathcal{A}_1 \cup \{A(a), R(a,x_j), A(x_j)\}
\end{aligned}
$$

However, if we restrict the $\exists$-rule and the $\geq$-rule to only be applicable when no other rules are, then we can guarantee termination also for general $\mathcal{ALCN}$-ABoxes.

Other extensions

Atomic Inclusions

- If we allow (acyclic) Aboxes with inclusions of the form $A \sqsubseteq C$ where A is a base name, then we can just make a fresh concept A_{new} and extend the inclusion to a definition, by replacing it with $A \equiv C \sqcap A_{new}$.

Other extensions

Atomic Inclusions

- If we allow (acyclic) Aboxes with inclusions of the form $A \sqsubseteq C$ where A is a base name, then we can just make a fresh concept A_{new} and extend the inclusion to a definition, by replacing it with $A \equiv C \sqcap A_{new}$.

  E.g.

$$Donkey \sqsubseteq Animal \sqcap Stubborn$$
$$\downarrow$$
$$Donkey \equiv Animal \sqcap Stubborn \sqcap Donkey_{new}$$

Other extensions

General Inclusions

- If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts C and D, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$$

  where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.

Other extensions

General Inclusions

– If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts C and D, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$$

where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.

– Thus, for any individual x in the ABox, we can just add

$$((\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))(x)$$

whenever they are introduced.

Other extensions

General Inclusions

- If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts C and D, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$$

  where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.
- Thus, for any individual x in the ABox, we can just add

$$((\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))(x)$$

  whenever they are introduced.
- However, we now lose termination, for instance for the knowledge base with ABox $\{\top(x_0)\}$ and TBox $\{\top \sqsubseteq \exists R.\top\}$.

Other extensions

General Inclusions

- If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts C and D, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$$

  where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.
- Thus, for any individual x in the ABox, we can just add

$$((\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))(x)$$

  whenever they are introduced.
- However, we now lose termination, for instance for the knowledge base with ABox $\{\top(x_0)\}$ and TBox $\{\top \sqsubseteq \exists R.\top\}$.
- This can be fixed with *blocking*.

Other extensions

Inclusions: Blocking

Definition

We will say that a variable y is an *ancestor* of a variable x if there exists some R where either $R(y, x)$, or there exists some variable z where z is an ancestor of x and $R(y, z)$.

Other extensions

Inclusions: Blocking

Definition

We will say that a variable y is an *ancestor* of a variable x if there exists some R where either $R(y, x)$, or there exists some variable z where z is an ancestor of x and $R(y, z)$.

Definition

We say that an application of a $\exists$-rule or a $\geq$-rule to a variable x is *directly blocked* by a variable y if

$$\{D|D(x) \in \mathcal{A}\} \subseteq \{D'|D'(y) \in \mathcal{A}\}$$

and y is an *ancestor* of x.

Other extensions

Inclusions: Blocking

Definition

We will say that a variable y is an *ancestor* of a variable x if there exists some R where either $R(y, x)$, or there exists some variable z where z is an ancestor of x and $R(y, z)$.

Definition

We say that an application of a $\exists$-rule or a $\geq$-rule to a variable x is *directly blocked* by a variable y if

$$\{D|D(x) \in \mathcal{A}\} \subseteq \{D'|D'(y) \in \mathcal{A}\}$$

and y is an *ancestor* of x.

Definition

We say that an application of a $\exists$-rule or a $\geq$-rule to a variable x is *blocked* if it is direcly blocked, or if an ancestor of x is directly blocked.

Contents

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Proof (Sketch).

In $\textsc{PSpace}$*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is PSPACE*-complete for acyclic TBoxes.*

Proof (Sketch).

In PSPACE*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Proof (Sketch).

In $\textsc{PSpace}$*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Proof (Sketch).

In $\textsc{PSpace}$*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Proof (Sketch).

In $\textsc{PSpace}$*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Generated successors can be treated sepparately, so we only need to store one path of the tree.

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Proof (Sketch).

In $\textsc{PSpace}$*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Generated successors can be treated sepparately, so we only need to store one path of the tree. Furthermore, we do not need to generate all n individuals for every $(\geq n\, R)(x)$.

Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\textsc{PSpace}$*-complete for acyclic TBoxes.*

Proof (Sketch).

In $\textsc{PSpace}$*:* If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Generated successors can be treated sepparately, so we only need to store one path of the tree. Furthermore, we do not need to generate all n individuals for every $(\geq n\, R)(x)$.

$\textsc{PSpace}$*-hard:* Can be reduced to validity of Quantified Boolean Formula. □

More complexity results

DL	Combined complexity	Data complexity
$\mathcal{ALC}$	ExpTime-complete	NP-complete
$\mathcal{ALCN}$	ExpTime-complete	NP-complete
$\mathcal{SHIQ}$	ExpTime-complete	NP-complete
$\mathcal{SHOIN}(D)$	NExpTime-complete	NP-hard
$\mathcal{SROIQ}$	N2ExpTime-complete	NP-hard
$\mathcal{EL}$	P-complete	P-complete
$\mathcal{RL}$	P-complete	P-complete
DL_{Lite}	In P	In LOGSpace

More info

For more info see:

- F. Baader and W. Nutt's chapter *Basic Description Logics* from *The Description Logic Handbook*.
- P. Hitzler, M. Krötzsch, and S. Rudolph's book *Foundations of Semantic Web Technologies*.

Thanks for listening!