Modeling Error Explanation

Petr Křemen
petr.kremen@fel.cvut.cz

FEL ČVUT

Our plan

Modeling Error Explanation

Black-box methods

- Algorithms based on CS-trees
- Algorithm based on Reiter's Algorithm
- Algorithm based on Reiter's Algorithm

Modeling Error Explanation

Motivation

- When an inference engine claims inconsistency of an ($\mathcal{ALC}$) theory/unsatisfiability of an ($\mathcal{ALC}$) concept, **what can we do with it ?**
- We can start iterating through all axioms in the theory and look, "what went wrong".
- ... but hardly in case we have **hundred thousand axioms**
- A solution might be to ask the computer to *localize the axioms causing the problem for us.*

Motivation

- When an inference engine claims inconsistency of an ($\mathcal{ALC}$) theory/unsatisfiability of an ($\mathcal{ALC}$) concept, **what can we do with it ?**
- We can start iterating through all axioms in the theory and look, "what went wrong".
- ... but hardly in case we have **hundred thousand axioms**
- A solution might be to ask the computer to *localize the axioms causing the problem for us.*

Motivation

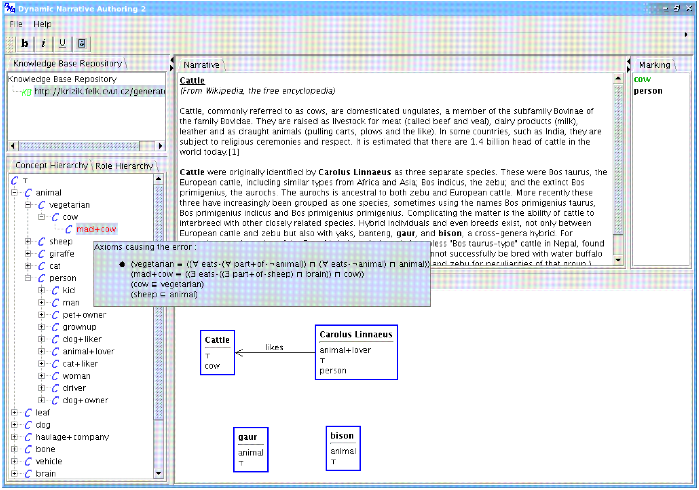
- When an inference engine claims inconsistency of an ($\mathcal{ALC}$) theory/unsatisfiability of an ($\mathcal{ALC}$) concept, **what can we do with it ?**
- We can start iterating through all axioms in the theory and look, "what went wrong".
- ... but hardly in case we have **hundred thousand axioms**
- A solution might be to ask the computer to *localize the axioms causing the problem for us.*

laboratory Gerstner

Motivation

- When an inference engine claims inconsistency of an ($\mathcal{ALC}$) theory/unsatisfiability of an ($\mathcal{ALC}$) concept, **what can we do with it ?**
- We can start iterating through all axioms in the theory and look, "what went wrong".
- ... but hardly in case we have **hundred thousand axioms**
- A solution might be to ask the computer to *localize the axioms causing the problem for us*.

DNA


Dynamic Narrative Authoring 2
File Help
Knowledge Base Repository
Knowledge Base Repository
http://krizik.felk.cvut.cz/generate
Concept Hierarchy
Role Hierarchy
⊤
animal
vegetarian
cow
mad+cow
sheep
giraffe
cat
person
kid
man
pet+owner
grownup
dog+liker
animal+lover
cat+liker
woman
driver
dog+owner
leaf
dog
haulage+company
bone
vehicle
brain
Narrative
Cattle
(From Wikipedia, the free encyclopedia)
Cattle, commonly referred to as cows, are domesticated ungulates, a member of the subfamily Bovinae of the family Bovidae. They are raised as livestock for meat (called beef and veal), dairy products (milk), leather and as draught animals (pulling carts, plows and the like). In some countries, such as India, they are subject to religious ceremonies and respect. It is estimated that there are 1.4 billion head of cattle in the world today.[1]
Cattle were originally identified by Carolus Linnaeus as three separate species. These were Bos taurus, the European cattle, including similar types from Africa and Asia; Bos indicus, the zebu; and the extinct Bos primigenius, the aurochs. The aurochs is ancestral to both zebu and European cattle. More recently these three have increasingly been grouped as one species, sometimes using the names Bos primigenius taurus, Bos primigenius indicus and Bos primigenius primigenius. Complicating the matter is the ability of cattle to interbreed with other closely related species. Hybrid individuals and even breeds exist, not only between European cattle and zebu but also with yaks, banteng, gaur, and bison, a cross-genera hybrid. For
Axioms causing the error :
(vegetarian ≡ ((∀ eats·(∀ part+of·¬animal)) ⊓ (∀ eats·¬animal) ⊓ animal))
(mad+cow ≡ ((∃ eats·((∃ part+of·sheep) ⊓ brain)) ⊓ cow))
(cow ⊑ vegetarian)
(sheep ⊑ animal)
Marking
cow
person
Cattle
⊤
cow
likes
Carolus Linnaeus
animal+lover
⊤
person
gaur
animal
⊤
bison
animal
⊤

MUPS – example

Minimal unsatisfiability preserving subterminology (MUPS) is a minimal set of axioms responsible for concept unsatisfiability.

Example

Consider theory $\mathcal{K}_5 = (\{\alpha_1, \alpha_2, \alpha_3\}, \emptyset)$

$$
\begin{aligned}
\alpha_1 &: \quad \mathit{Person} \sqsubseteq \exists \mathit{hasParent} \cdot (\mathit{Man} \sqcap \mathit{Woman}) \sqcap \forall \mathit{hasParent} \cdot \neg \mathit{Person}, \\
\alpha_2 &: \quad \mathit{Man} \sqsubseteq \neg \mathit{Woman}, \\
\alpha_3 &: \quad \mathit{Man} \sqcup \mathit{Woman} \sqsubseteq \mathit{Person}.
\end{aligned}
$$

Unsatisfiability of *Person* comes independently from two axiom sets (MUPSes), namely $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_3\}$. Check it yourself !

MUPS – example

Minimal unsatisfiability preserving subterminology (MUPS) is a minimal set of axioms responsible for concept unsatisfiability.

Example

Consider theory $\mathcal{K}_5 = (\{\alpha_1, \alpha_2, \alpha_3\}, \emptyset)$

$$\begin{aligned}
\alpha_1 &: \quad Person \sqsubseteq \exists hasParent \cdot (Man \sqcap Woman) \sqcap \forall hasParent \cdot \neg Person, \\
\alpha_2 &: \quad Man \sqsubseteq \neg Woman, \\
\alpha_3 &: \quad Man \sqcup Woman \sqsubseteq Person.
\end{aligned}$$

Unsatisfiability of *Person* comes independently from two axiom sets (MUPSes), namely $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_3\}$. Check it yourself !

MUPS – example

Minimal unsatisfiability preserving subterminology (MUPS) is a minimal set of axioms responsible for concept unsatisfiability.

Example

Consider theory $\mathcal{K}_5 = (\{\alpha_1, \alpha_2, \alpha_3\}, \emptyset)$

$$\begin{aligned}
\alpha_1 &: \quad \mathit{Person} \sqsubseteq \exists \mathit{hasParent} \cdot (\mathit{Man} \sqcap \mathit{Woman}) \sqcap \forall \mathit{hasParent} \cdot \neg \mathit{Person},\\
\alpha_2 &: \quad \mathit{Man} \sqsubseteq \neg \mathit{Woman},\\
\alpha_3 &: \quad \mathit{Man} \sqcup \mathit{Woman} \sqsubseteq \mathit{Person}.
\end{aligned}$$

Unsatisfiability of *Person* comes independently from two axiom sets (MUPSes), namely $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_3\}$. Check it yourself !

laboratory Gerstner

MUPS

Currently two approaches exist for searching all MUPSes for given concept:

black-box methods perform many satisfiability tests using existing inference engine.

- ☺ flexible and easily reusable for another (description) logic
- ☹ time consuming

glass-box methods all integrated into an existing reasoning (typically tableau) algorithm.

- ☺ efficient
- ☹ hardly reusable for another (description) logic.

MUPS

Currently two approaches exist for searching all MUPSes for given concept:

black-box methods perform many satisfiability tests using existing inference engine.

- ☺ flexible and easily reusable for another (description) logic
- ☹ time consuming

glass-box methods all integrated into an existing reasoning (typically tableau) algorithm.

- ☺ efficient
- ☹ hardly reusable for another (description) logic.

MUPS

Currently two approaches exist for searching all MUPSes for given concept:

black-box methods perform many satisfiability tests using existing inference engine.

- ☺ flexible and easily reusable for another (description) logic
- ☹ time consuming

glass-box methods all integrated into an existing reasoning (typically tableau) algorithm.

- ☺ efficient
- ☹ hardly reusable for another (description) logic.

Glass-box methods

- For $\mathcal{ALC}$ there exists a complete algorithm with the following idea:
  - tableau algorithm for $\mathcal{ALC}$ is extended in such way that it "remembers which axioms were used during completion graph construction".
  - for each completion graph containing a clash, the axioms that were used during its construction can be transformed into a MUPS.
- Unfortunately, complete glass-box methods do not exist for OWL-DL and OWL2-DL. The same idea (tracking axioms used during completion graph construction) can be used also for these logics, but only as a preprocessing reducing the set of axioms used by a black-box algorithm.

Glass-box methods

- For $\mathcal{ALC}$ there exists a complete algorithm with the following idea:
  - tableau algorithm for $\mathcal{ALC}$ is extended in such way that it "remembers which axioms were used during completion graph construction".
  - for each completion graph containing a clash, the axioms that were used during its construction can be transformed into a MUPS.
- Unfortunately, complete glass-box methods do not exist for OWL-DL and OWL2-DL. The same idea (tracking axioms used during completion graph construction) can be used also for these logics, but only as a preprocessing reducing the set of axioms used by a black-box algorithm.

Glass-box methods

- For $\mathcal{ALC}$ there exists a complete algorithm with the following idea:
  - tableau algorithm for $\mathcal{ALC}$ is extended in such way that it "remembers which axioms were used during completion graph construction".
  - for each completion graph containing a clash, the axioms that were used during its construction can be transformed into a MUPS.
- Unfortunately, complete glass-box methods do not exist for OWL-DL and OWL2-DL. The same idea (tracking axioms used during completion graph construction) can be used also for these logics, but only as a preprocessing reducing the set of axioms used by a black-box algorithm.

Glass-box methods

- For $\mathcal{ALC}$ there exists a complete algorithm with the following idea:
    - tableau algorithm for $\mathcal{ALC}$ is extended in such way that it "remembers which axioms were used during completion graph construction".
    - for each completion graph containing a clash, the axioms that were used during its construction can be transformed into a MUPS.
- Unfortunately, complete glass-box methods do not exist for OWL-DL and OWL2-DL. The same idea (tracking axioms used during completion graph construction) can be used also for these logics, but only as a preprocessing reducing the set of axioms used by a black-box algorithm.

Black-box methods

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

laboratory Gerstner

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

laboratory Gerstner

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

Task formulation

- Let's have *a set of axioms* X of given DL and *reasoner* R for given DL. We want to find MUPSes for :
  1. concept unsatisfiability,
  2. theory (ontology) inconsistency,
  3. arbitrary entailment.
- It can be shown (see [Kal06]) that w.l.o.g. we can deal only with *concept unsatisfiability*.
- **MUPS:** Let's denote $MUPS(C, Y)$ a minimal subset $MUPS(C, Y) \subseteq Y \subseteq X$ causing unsatisfiability of C.
- **Diagnose:** Let's denote $DIAG(C, Y)$ a minimal subset $DIAG(C, Y) \subseteq Y \subseteq X$, such that if $DIAG(C, Y)$ is removed from Y, the concept C becomes satisfiable.

Task formulation (2)

- Let's focus on concept C unsatisfiability. Denote

$$R(C, Y) = \left\{ \begin{array}{ll} \textit{true} & \text{iff} Y \not\models (C \sqsubseteq \bot) \\ \textit{false} & \text{iff} Y \models (C \sqsubseteq \bot)) \end{array} \right\}.$$

- There are many methods (see [dSW03]). We introduce just two of them:
  - Algorithms based on CS-trees.
  - Algorithm for computing a single MUPS [illegible] algorithm [Rei87].

Task formulation (2)

- Let's focus on concept C unsatisfiability. Denote

$$R(C, Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{iff} Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{iff} Y \models (C \sqsubseteq \bot)) \end{array} \right\}.$$

- There are many methods (see [dSW03]). We introduce just two of them:
  - Algorithms based on CS-trees.
  - Algorithm for computing a single MUPS[Kal06] + Reiter algorithm [Rei87].

Task formulation (2)

- Let's focus on concept C unsatisfiability. Denote

$$R(C, Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{iff} Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{iff} Y \models (C \sqsubseteq \bot)) \end{array} \right\}.$$

- There are many methods (see [dSW03]). We introduce just two of them:
  - Algorithms based on CS-trees.
  - Algorithm for computing a single MUPS[Kal06] + Reiter algorithm [Rei87].

Task formulation (2)

- Let's focus on concept C unsatisfiability. Denote

$$R(C,Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{iff} Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{iff} Y \models (C \sqsubseteq \bot)) \end{array} \right\}.$$

- There are many methods (see [dSW03]). We introduce just two of them:
  - Algorithms based on CS-trees.
  - Algorithm for computing a single MUPS[Kal06] + Reiter algorithm [Rei87].

CS-trees

- A naive solution: test for each set of axioms from $\mathcal{T} \cup \mathcal{A}$ for $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, whether the set causes unsatisfiability – minimal sets of this form are MUPSes.
- *Conflict-set trees (CS-trees)* systematize exploration of all these subsets of $\mathcal{T} \cup \mathcal{A}$. The main gist :

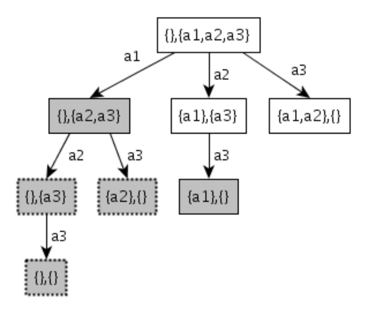
  If we found a set of axioms X that do not cause unsatisfiability of C (i.e. $X \not\models C \sqsubseteq \bot$), then we know (and thus can avoid asking reasoner) that $Y \not\models C \sqsubseteq \bot$ for each $Y \subseteq X$.

- CS-tree is a representation of the state space, where each state s has the form (D, P), where
  - D is a set of axioms that *necessarily has to be part of all MUPSes* found while exploring the subtree of s.
  - P is a set of axioms that *might be part of some MUPSes* found while exploring the subtree of s.

Example

A CS-tree for unsatisfiability of *Person* (abbr. *Pe*, not to be mixed with the set P) in $\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\}$:

$$\underbrace{Pe \sqsubseteq \exists hP \cdot (M \sqcap W) \sqcap \forall hP \cdot \neg Pe}_{\alpha_1}, \quad \underbrace{M \sqsubseteq \neg W}_{\alpha_2}, \quad \underbrace{M \sqcup W \sqsubseteq Pe}_{\alpha_3}.$$



In gray states, the concept *Person* is satsifiable ($R(Pe, D \cup P) = true$). States with a dotted border are pruned by the algorithm.

CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

1. (Inicializace) The root of the tree is an initial state $s_0 = (\emptyset, \mathcal{K})$ – apriori, we don't know any axiom being necessarily in a MUPS ($D_{s_0} = \emptyset$), but potentially all axioms can be there ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Next, we define $Z = (s_0)$ and $R = \emptyset$
2. (Depth First Search) If Z is empty, stop the exploration. Otherwise pop the first element s from Z.
3. (Test) If $R(C, D_s \cup P_s) = \mathit{true}$ then no subset of $D_s \cup P_s$ can cause unsatisfiability – we continue with step 2.
4. (Finding an unsatisfiable set) We add $D_s \cup P_s$ into R and remove from R all $s' \in R$ such that $D_s \cup P_s \subseteq s'$. For $P_s = \alpha_1, \ldots, \alpha_N$ we push to Z a new state $(D_s \cup \{\alpha_1, \ldots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \ldots, \alpha_i\})$ – we continue with step 2.

CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

1. (Inicializace) The root of the tree is an initial state $s_0 = (\emptyset, \mathcal{K})$ – apriori, we don't know any axiom being necessarily in a MUPS ($D_{s_0} = \emptyset$), but potentially all axioms can be there ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Next, we define $Z = (s_0)$ and $R = \emptyset$
2. (Depth First Search) If Z is empty, stop the exploration. Otherwise pop the first element s from Z.
3. (Test) If $R(C, D_s \cup P_s) = true$ then no subset of $D_s \cup P_s$ can cause unsatisfiability – we continue with step 2.
4. (Finding an unsatisfiable set) We add $D_s \cup P_s$ into R and remove from R all $s' \in R$ such that $D_s \cup P_s \subseteq s'$. For $P_s = \alpha_1, \ldots, \alpha_N$ we push to Z a new state $(D_s \cup \{\alpha_1, \ldots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \ldots, \alpha_i\})$ – we continue with step 2.

CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

1. (Inicializace) The root of the tree is an initial state $s_0 = (\emptyset, \mathcal{K})$ – apriori, we don't know any axiom being necessarily in a MUPS ($D_{s_0} = \emptyset$), but potentially all axioms can be there ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Next, we define $Z = (s_0)$ and $R = \emptyset$
2. (Depth First Search) If Z is empty, stop the exploration. Otherwise pop the first element s from Z.
3. (Test) If $R(C, D_s \cup P_s) = \mathit{true}$ then no subset of $D_s \cup P_s$ can cause unsatisfiability – we continue with step 2.
4. (Finding an unsatisfiable set) We add $D_s \cup P_s$ into R and remove from R all $s' \in R$ such that $D_s \cup P_s \subseteq s'$. For $P_s = \alpha_1, \ldots, \alpha_N$ we push to Z a new state $(D_s \cup \{\alpha_1, \ldots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \ldots, \alpha_i\})$ – we continue with step 2.

CS-tree Exploration

The following algorithm is exponential in the number of tableau algorithm runs.

1. (Inicializace) The root of the tree is an initial state $s_0 = (\emptyset, \mathcal{K})$ – apriori, we don't know any axiom being necessarily in a MUPS ($D_{s_0} = \emptyset$), but potentially all axioms can be there ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Next, we define $Z = (s_0)$ and $R = \emptyset$
2. (Depth First Search) If Z is empty, stop the exploration. Otherwise pop the first element s from Z.
3. (Test) If $R(C, D_s \cup P_s) = true$ then no subset of $D_s \cup P_s$ can cause unsatisfiability – we continue with step 2.
4. (Finding an unsatisfiable set) We add $D_s \cup P_s$ into R and remove from R all $s' \in R$ such that $D_s \cup P_s \subseteq s'$. For $P_s = \alpha_1, \ldots, \alpha_N$ we push to Z a new state $(D_s \cup \{\alpha_1, \ldots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \ldots, \alpha_i\})$ – we continue with step 2.

CS-tree Exploration (2)

- Soundness : Step 4 is important – here, we cover all possibilities. It always holds that $D_s \cup P_s$ differs to $D'_s \cup P'_s$ by just one element, where s' is a successor of s.
- Finiteness : Set $D_s \cup P_s$ is finite at the beginning and gets smaller with the tree depth. Furthermore, in step 4 we generate only finite number of states.

CS-tree Exploration (2)

- Soundness : Step 4 is important – here, we cover all possibilities. It always holds that $D_s \cup P_s$ differs to $D'_s \cup P'_s$ by just one element, where s' is a successor of s.
- Finiteness : Set $D_s \cup P_s$ is finite at the beginning and gets smaller with the tree depth. Furthermore, in step 4 we generate only finite number of states.

Another Approach – Reiter's Algorithm

There is an alternative to CS-trees:

1. Find a single (arbitrary) MUPS (*singleMUPS* in the next slides).
2. "remove the source of unsatisfiability provided by MUPS" (Reiter's algorithm in the next slides) from the set of axioms and go explore the remaining axioms in the same manner.

Finding a single $MUPS(C, Y)$ – example

Example

The run of $singleMUPS(Osoba, \mathcal{K}_5)$ introduced next.

Finding a single $MUPS(C, Y)$ – example

Example

The run of $singleMUPS(Osoba, \mathcal{K}_5)$ introduced next.

1.PHASE :

$$\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1\}) = true$$
$$S = \{\alpha_1\}$$

Finding a single $MUPS(C, Y)$ – example

Example

The run of $singleMUPS(Osoba, \mathcal{K}_5)$ introduced next.

1.PHASE :

$$\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1, \alpha_2\}) = \mathit{false}$$
$$S = \{\alpha_1, \alpha_2\}$$

Finding a single $MUPS(C, Y)$ – example

Example

The run of $singleMUPS(Osoba, \mathcal{K}_5)$ introduced next.

1.PHASE :

$$\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1, \alpha_2\}) = false$$
$$S = \{\alpha_1, \alpha_2\}$$

2.PHASE :

$$S = \{\alpha_1, \alpha_2\} \qquad R(C, \{\alpha_1, \alpha_2\} - \{\alpha_1\}) = true$$
$$K = \{\alpha_1\}$$

Finding a single $MUPS(C, Y)$ – example

Example

The run of $singleMUPS(Osoba, \mathcal{K}_5)$ introduced next.

1.PHASE :

$$\mathcal{K}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1, \alpha_2\}) = \mathit{false}$$

$$S = \{\alpha_1, \alpha_2\}$$

2.PHASE :

$$S = \{\alpha_1, \alpha_2\} \qquad R(C, \{\alpha_1, \alpha_2\} - \{\alpha_2\}) = \mathit{true}$$

$$K = \{\alpha_1, \alpha_2\}$$

singleMUPS(C, Y) – finding a single MUPS

The following algorithm is polynomial in the number of tableau algorithm applications – the computational complexity stems from the complexity of tableau algorithm itself.

1 (Initialization) Denote $S = \emptyset$, $K = \emptyset$

2 (Finding superset of MUPS) While $R(C, S) = false$, then $S = S \cup \{\alpha\}$ for some $\alpha \in Y \setminus S$.

3 (Pruning found set) For each $\alpha \in S \setminus K$ evaluate $R(C, S \setminus \{\alpha\})$. If the result is *false*, then $K = K \cup \{\alpha\}$. The resulting K is itself a MUPS.

laboratory Gerstner

singleMUPS(C, Y) – finding a single MUPS

The following algorithm is polynomial in the number of tableau algorithm applications – the computational complexity stems from the complexity of tableau algorithm itself.

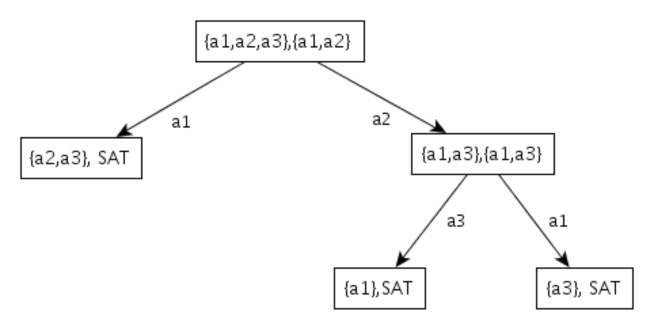
1. (Initialization) Denote $S = \emptyset$, $K = \emptyset$
2. (Finding superset of MUPS) While $R(C, S) = \mathit{false}$, then $S = S \cup \{\alpha\}$ for some $\alpha \in Y \setminus S$.
3. (Pruning found set) For each $\alpha \in S \setminus K$ evaluate $R(C, S \setminus \{\alpha\})$. If the result is *false*, then $K = K \cup \{\alpha\}$. The resulting K is itself a MUPS.

The following algorithm is polynomial in the number of tableau algorithm applications – the computational complexity stems from the complexity of tableau algorithm itself.

1. (Initialization) Denote $S = \emptyset$, $K = \emptyset$
2. (Finding superset of MUPS) While $R(C, S) = \mathit{false}$, then $S = S \cup \{\alpha\}$ for some $\alpha \in Y \setminus S$.
3. (Pruning found set) For each $\alpha \in S \setminus K$ evaluate $R(C, S \setminus \{\alpha\})$. If the result is *false*, then $K = K \cup \{\alpha\}$. The resulting K is itself a MUPS.

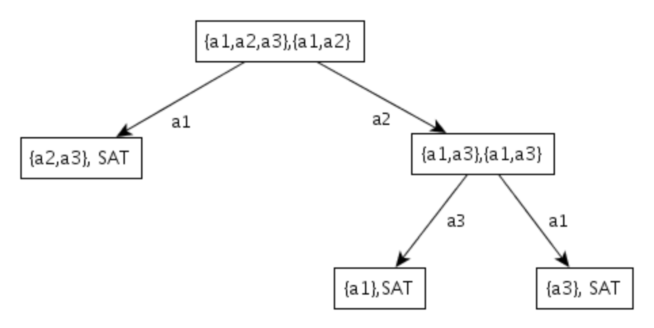
Finding all MUPSes – Reiter Algorithm, example

Example (continued)



The algorithm ends up with two MUPSes $\{\alpha_1, \alpha_2\}$ a $\{\alpha_1, \alpha_3\}$. "For free" we got diagnoses $\{\alpha_1\}$ a $\{\alpha_2, \alpha_3\}$.

Example (continued)



The algorithm ends up with two MUPSes $\{\alpha_1, \alpha_2\}$ a $\{\alpha_1, \alpha_3\}$. "For free" we got diagnoses $\{\alpha_1\}$ a $\{\alpha_2, \alpha_3\}$.

Finding all MUPSes – Reiter Algorithm

- Reiter algorithm runs *singleMUPS*(C, Y) multiple times to construct so called "Hitting Set Tree", nodes of which are pairs $(\mathcal{K}_i, M_i)$, where $\mathcal{K}_i$ lacks some axioms comparing to $\mathcal{K}$ and $M_i = singleMUPS(C, \mathcal{K}_i)$, or $M_i =$ "SAT", if C is satisfiable w.r.t. $\mathcal{K}_i$.
- Paths from the root to leaves build up *diagnoses* (i.e. minimal sets of axioms, each of which removed from $\mathcal{K}$ causes satisfiability of C).
- Number of *singleMUPS*(C, Y) calls is at most exponential w.r.t. the initial axioms count. Why ?

Finding all MUPSes – Reiter Algorithm

- Reiter algorithm runs *singleMUPS*(C, Y) multiple times to construct so called "Hitting Set Tree", nodes of which are pairs $(\mathcal{K}_i, M_i)$, where $\mathcal{K}_i$ lacks some axioms comparing to $\mathcal{K}$ and $M_i = singleMUPS(C, \mathcal{K}_i)$, or $M_i =$ "SAT'', if C is satisfiable w.r.t. $\mathcal{K}_i$.
- Paths from the root to leaves build up *diagnoses* (i.e. minimal sets of axioms, each of which removed from $\mathcal{K}$ causes satisfiability of C).
- Number of *singleMUPS*(C, Y) calls is at most exponential w.r.t. the initial axioms count. Why ?

Finding all MUPSes – Reiter Algorithm

- Reiter algorithm runs $\mathit{singleMUPS}(C, Y)$ multiple times to construct so called "Hitting Set Tree", nodes of which are pairs $(\mathcal{K}_i, M_i)$, where $\mathcal{K}_i$ lacks some axioms comparing to $\mathcal{K}$ and $M_i = \mathit{singleMUPS}(C, \mathcal{K}_i)$, or $M_i = \text{``}SAT''$, if C is satisfiable w.r.t. $\mathcal{K}_i$.
- Paths from the root to leaves build up *diagnoses* (i.e. minimal sets of axioms, each of which removed from $\mathcal{K}$ causes satisfiability of C).
- Number of $\mathit{singleMUPS}(C, Y)$ calls is at most exponential w.r.t. the initial axioms count. Why ?

Finding all MUPSes – Reiter Algorithm (2)

1 (Initialization) Find single MUPS for C in $\mathcal{K}$, and construct the root $s_0 = (\mathcal{K}, singleMUPS(C, \mathcal{K}))$ of the hitting set tree. Next, set $Z = (s_0)$.

2 (Depth First Search) If Z is empty, STOP.

3 (Test) Otherwise pop an element from Z and denote it as $s_i = (\mathcal{K}_i, M_i)$. If $M_i =$ "SAT", then go to step 2.

4 (Decomposition) For each $\alpha \in M_i$ insert into Z a new node $(\mathcal{K}_i \setminus \{\alpha\}, singleMUPS(\mathcal{K}_i \setminus \{\alpha\}, C))$. Go to step 2.

Finding all MUPSes – Reiter Algorithm (2)

1. (Initialization) Find single MUPS for C in $\mathcal{K}$, and construct the root $s_0 = (\mathcal{K}, singleMUPS(C, \mathcal{K}))$ of the hitting set tree. Next, set $Z = (s_0)$.
2. (Depth First Search) If Z is empty, STOP.
3. (Test) Otherwise pop an element from Z and denote it as $s_i = (\mathcal{K}_i, M_i)$. If $M_i = \text{"}SAT\text{"}$, then go to step 2.
4. (Decomposition) For each $\alpha \in M_i$ insert into Z a new node $(\mathcal{K}_i \setminus \{\alpha\}, singleMUPS(\mathcal{K}_i \setminus \{\alpha\}, C))$. Go to step 2.

Finding all MUPSes – Reiter Algorithm (2)

1. (Initialization) Find single MUPS for C in $\mathcal{K}$, and construct the root $s_0 = (\mathcal{K}, singleMUPS(C, \mathcal{K}))$ of the hitting set tree. Next, set $Z = (s_0)$.
2. (Depth First Search) If Z is empty, STOP.
3. (Test) Otherwise pop an element from Z and denote it as $s_i = (\mathcal{K}_i, M_i)$. If $M_i =$ "SAT", then go to step 2.
4. (Decomposition) For each $\alpha \in M_i$ insert into Z a new node $(\mathcal{K}_i \setminus \{\alpha\}, singleMUPS(\mathcal{K}_i \setminus \{\alpha\}, C))$. Go to step 2.

Finding all MUPSes – Reiter Algorithm (2)

1. (Initialization) Find single MUPS for C in $\mathcal{K}$, and construct the root $s_0 = (\mathcal{K}, singleMUPS(C, \mathcal{K}))$ of the hitting set tree. Next, set $Z = (s_0)$.
2. (Depth First Search) If Z is empty, STOP.
3. (Test) Otherwise pop an element from Z and denote it as $s_i = (\mathcal{K}_i, M_i)$. If $M_i =$ "SAT", then go to step 2.
4. (Decomposition) For each $\alpha \in M_i$ insert into Z a new node $(\mathcal{K}_i \setminus \{\alpha\}, singleMUPS(\mathcal{K}_i \setminus \{\alpha\}, C))$. Go to step 2.

Modeling Error Explanation – Summary

- finding MUPSes is the most common way for explaining modeling errors.
- black-box vs. glass box methods. Other methods involve e.g. incremental methods [dSW03].
- the goal is to find MUPSes (and diagnoses) – what to do in order to solve a modeling problem (unsatisfiability,inconsistency).
- above mentioned methods are quite universal – they can be used for many other problems that are not related with description logics.