Incremental Approach to Error Explanations in Ontologies

Inference Support for Semantic Annotations

Petr Křemen

Czech Technical University in Prague, Dept. of Cybernetics

7th International Conference on Knowledge Management

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.
- During the CIPHER project, an urgent need for an *intelligent annotation tool* appeared.

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.
- During the CIPHER project, an urgent need for an *intelligent annotation tool* appeared.
- Such a tool should meet the following requirements :

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.
- During the CIPHER project, an urgent need for an *intelligent annotation tool* appeared.
- Such a tool should meet the following requirements :
  - Adequate graphical language for annotation building and visualization.

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.
- During the CIPHER project, an urgent need for an *intelligent annotation tool* appeared.
- Such a tool should meet the following requirements :
  - Adequate graphical language for annotation building and visualization.
  - Inducing and suggesting new annotations to the user.

Motivation

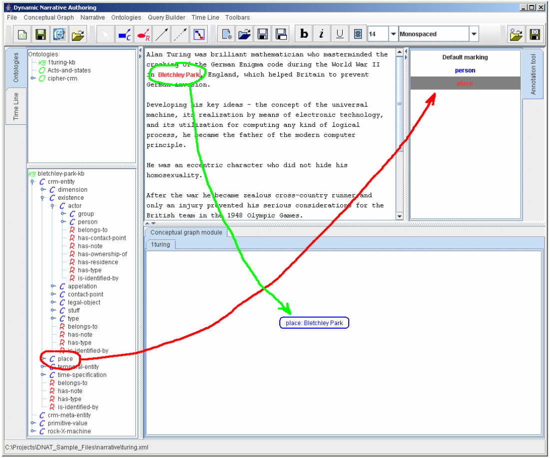
- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.
- During the CIPHER project, an urgent need for an *intelligent annotation tool* appeared.
- Such a tool should meet the following requirements :
  - Adequate graphical language for annotation building and visualization.
  - Inducing and suggesting new annotations to the user.
  - Providing expressive modeling constructs, like n-ary relations.

Motivation

- Efficient searching in large document repositories needs *semantic annotations* that are both
  - expressive enough to allow formulating *interesting queries* and
  - weak enough to allow *efficient reasoning*.
- During the CIPHER project, an urgent need for an *intelligent annotation tool* appeared.
- Such a tool should meet the following requirements :
  - Adequate graphical language for annotation building and visualization.
  - Inducing and suggesting new annotations to the user.
  - Providing expressive modeling constructs, like n-ary relations.
  - **Modeling error explanations** – the problem discussed in this work.

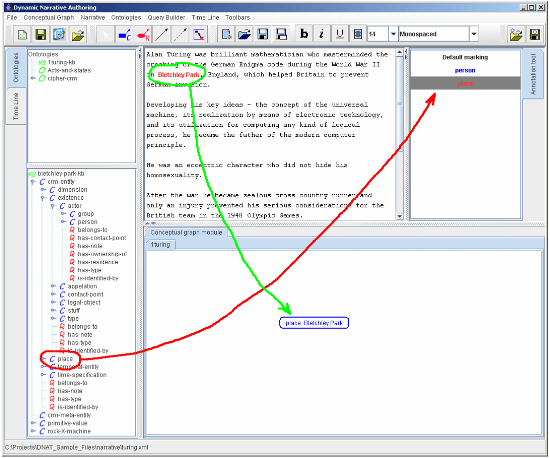
Motivation (2) – Document Annotation Tool

- ☺ frames and conceptual graphs : simple, human understandable.



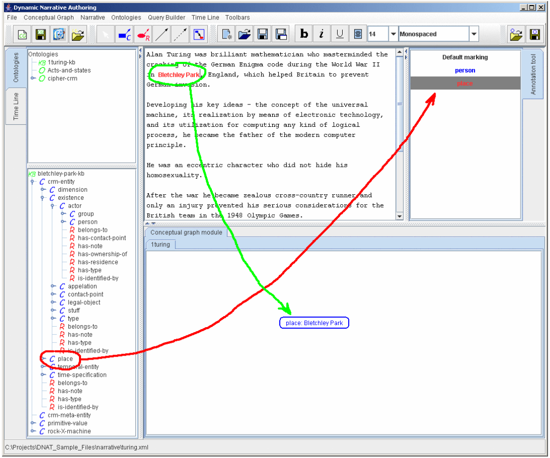
Motivation (2) – Document Annotation Tool

- ☺ frames and conceptual graphs : simple, human understandable.
- ☹ poor inference support



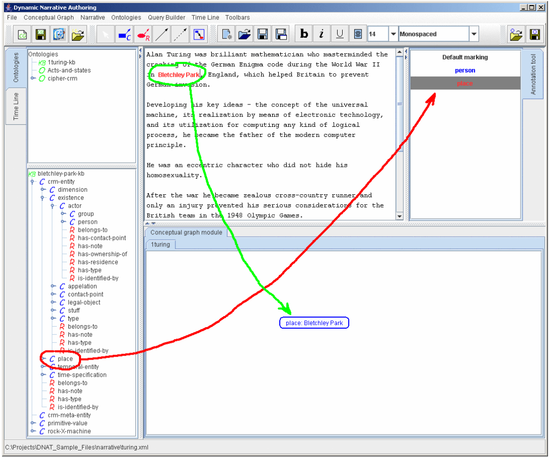
Motivation (2) – Document Annotation Tool

- ☺ frames and conceptual graphs : simple, human understandable.
- ☹ poor inference support
- ☹ no way to detect and localize modeling errors



Motivation (2) – Document Annotation Tool

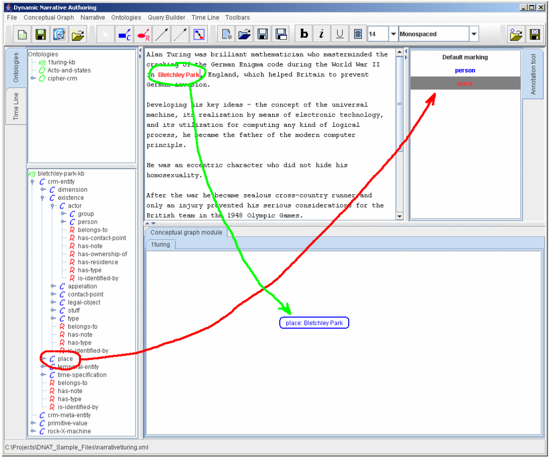
- ☺ frames and conceptual graphs : simple, human understandable.
- ☹ poor inference support
- ☹ no way to detect and localize modeling errors



Motivation (2) – Document Annotation Tool

- ☺ frames and conceptual graphs : simple, human understandable.
- ☹ poor inference support
- ☹ no way to detect and localize modeling errors

! **This lead us to use a description logic based approach.**



Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
    - concept constructs :

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
    - concept constructs :
        - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
        - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
    - concept constructs :
        - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
        - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
        - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).
  - just one role construct : **inverse** – $hasChild^{-}$ – a new role *childOf*.

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).
  - just one role construct : **inverse** – $hasChild^{-}$ – a new role *childOf*.
  - axiom types:

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).
  - just one role construct : **inverse** – $hasChild^{-}$ – a new role *childOf*.
  - axiom types:
    - **subsumption of concepts and roles** – $Man \sqsubseteq Person$, $hasChild \sqsubseteq hasParent$

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild . Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild . Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).
  - just one role construct : **inverse** – $hasChild^{-}$ – a new role *childOf*.
  - axiom types:
    - **subsumption of concepts and roles** – $Man \sqsubseteq Person$, $hasChild \sqsubseteq hasParent$
    - **equivalence, disjointness of concepts** – $Car \equiv Vehicle$, $disjoint(Man, Woman)$

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).
  - just one role construct : **inverse** – $hasChild^-$ – a new role $childOf$.
  - axiom types:
    - **subsumption of concepts and roles** – $Man \sqsubseteq Person$, $hasChild \sqsubseteq hasParent$
    - **equivalence, disjointness of concepts** – $Car \equiv Vehicle$, $disjoint(Man, Woman)$
    - **role transitivity axioms** – $trans(hasDescendant)$

Description Logic $\mathcal{SHIN}$

- We use the $\mathcal{SHIN}$ description logic - being a significant subset of OWL-DL.
- *concepts* (unary predicates), *roles* (binary predicates) and *individuals* (instances) :
  - concept constructs :
    - **boolean constructs** – $\neg, \sqcap, \sqcup$ – with common interpretations (negation, intersection, union)
    - **value restriction** – $\forall hasChild.Woman$ – domain elements, all of its children (if any) are women.
    - **existential restriction** – $\exists hasChild.Man$ – domain elements, having at-least one son.
    - **number restrictions** – $(\geq 7\ hasChild)$ – domain elements, having at least 7 children (similarly for $\leq$).
  - just one role construct : **inverse** – $hasChild^-$ – a new role *childOf*.
  - axiom types:
    - **subsumption of concepts and roles** – $Man \sqsubseteq Person$, $hasChild \sqsubseteq hasParent$
    - **equivalence, disjointness of concepts** – $Car \equiv Vehicle$, $disjoint(Man, Woman)$
    - **role transitivity axioms** – $trans(hasDescendant)$
    - **concept and role assertions** – $school(CVUT)$, $partOf(FEL, CVUT)$

Reasoning in Description Logics

- Basic inference services: *knowledge base consistency*, *concept satisfiability*

Reasoning in Description Logics

- Basic inference services: *knowledge base consistency*, *concept satisfiability*

Reasoning in Description Logics

- Basic inference services: *knowledge base consistency*, *concept satisfiability*

Example (Part of the Mad Cow Ontology)

$$
\begin{array}{rcl}
madCow & \equiv & cow \sqcap \exists eats.(brain \sqcap \exists partOf.sheep) \\
cow & \sqsubseteq & vegetarian \\
vegetarian & \equiv & animal \sqcap \forall eats.\neg animal \sqcap \forall eats.\neg(\exists partOf.animal) \\
animal & \sqsubseteq & \exists eats.\top \\
sheep & \sqsubseteq & animal \\
sheep & \sqsubseteq & \forall eats.grass \\
animal \sqcup \exists partOf.animal & \sqsubseteq & \neg(plant \sqcup \exists partOf.plant)
\end{array}
$$

The concept *madCow* is unsatisfiable (always interpreted as $\emptyset$) due to the red axioms.

Reasoning in Description Logics

- Basic inference services: *knowledge base consistency*, *concept satisfiability*

Example (Part of the Mad Cow Ontology)

$$\begin{array}{rcl}
madCow & \equiv & cow \sqcap \exists eats.(brain \sqcap \exists partOf.sheep) \\
cow & \sqsubseteq & vegetarian \\
vegetarian & \equiv & animal \sqcap \forall eats.\neg animal \sqcap \forall eats.\neg(\exists partOf.animal) \\
animal & \sqsubseteq & \exists eats.\top \\
sheep & \sqsubseteq & animal \\
sheep & \sqsubseteq & \forall eats.grass \\
animal \sqcup \exists partOf.animal & \sqsubseteq & \neg(plant \sqcup \exists partOf.plant)
\end{array}$$

The concept *madCow* is unsatisfiable (always interpreted as $\emptyset$) due to the red axioms.

- *Tableau algorithms* prove consistency of given knowledge base by constructing a model for it using a set of inference rules.

Reasoning in Description Logics

- Basic inference services: *knowledge base consistency*, *concept satisfiability*

Example (Part of the Mad Cow Ontology)

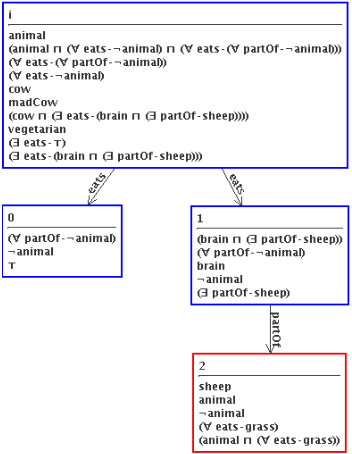
$$
\begin{aligned}
madCow &\equiv cow \sqcap \exists eats.(brain \sqcap \exists partOf.sheep)\\
cow &\sqsubseteq vegetarian\\
vegetarian &\equiv animal \sqcap \forall eats.\neg animal \sqcap \forall eats.\neg(\exists partOf.animal)\\
animal &\sqsubseteq \exists eats.\top\\
sheep &\sqsubseteq animal\\
sheep &\sqsubseteq \forall eats.grass\\
animal \sqcup \exists partOf.animal &\sqsubseteq \neg(plant \sqcup \exists partOf.plant)
\end{aligned}
$$

The concept *madCow* is unsatisfiable (always interpreted as $\emptyset$) due to the red axioms.

- *Tableau algorithms* prove consistency of given knowledge base by constructing a model for it using a set of inference rules.
- These algorithms terminate upon obtaining a clash-free model candidate – a completion graph – on which no more rules are applicable (consistent), or whenever each completion graph contains a clash (inconsistent).

Completion Graph Example

Testing satisfiability of the concept *madCow* a $\mathcal{SHIN}$ tableau reasoner may generate the following completion graph. The generated individual 2 contains a clash – individual 2 belongs to both *animal* and ¬*animal*.



MUPS

Minimal unsatisfiability preserving subterminology (MUPS) is a **minimal set (w.r.t. inclusion) of axioms that are responsible for unsatisfiability of given concept**.

By now, basically two approaches searching for all MUPSes exist:

MUPS

Minimal unsatisfiability preserving subterminology (MUPS) is a **minimal set (w.r.t. inclusion) of axioms that are responsible for unsatisfiability of given concept**.

By now, basically two approaches searching for all MUPSes exist:

black-box methods take an existing reasoning algorithm and perform many satisfiability/consistency checks to obtain MUPSes (Schlobach 2006, Kalyanpur 2006, Reiter 1987).

+ flexible and reusable
- time-intensive

MUPS

Minimal unsatisfiability preserving subterminology (MUPS) is a **minimal set (w.r.t. inclusion) of axioms that are responsible for unsatisfiability of given concept**.

By now, basically two approaches searching for all MUPSes exist:

black-box methods take an existing reasoning algorithm and perform many satisfiability/consistency checks to obtain MUPSes (Schlobach 2006, Kalyanpur 2006, Reiter 1987).

+ flexible and reusable
- time-intensive

glass-box methods are fully integrated into the reasoner (Schlobach 2006, Kalyanpur 2006).

+ more efficient
- poor reusability, **no fully glass box approach exists for even simpler languages than $\mathcal{SHIN}$**

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**
- **to restore given state and to apply given axiom (set).**

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**
- **to restore given state and to apply given axiom (set).**

Two reasoner states corresponding to different axiom orderings are semantically equivalent.

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**
- **to restore given state and to apply given axiom (set).**

Two reasoner states corresponding to different axiom orderings are semantically equivalent.

In our case the **state is** represented by a **set of completion graphs** (see above) generated by the underlying tableau reasoner together with the set of axioms used for its generation.

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**
- **to restore given state and to apply given axiom (set).**

Two reasoner states corresponding to different axiom orderings are semantically equivalent.

In our case the **state is** represented by a **set of completion graphs** (see above) generated by the underlying tableau reasoner together with the set of axioms used for its generation.

We show **two incremental techniques** searching for all MUPSes :

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**
- **to restore given state and to apply given axiom (set).**

Two reasoner states corresponding to different axiom orderings are semantically equivalent.

In our case the **state is** represented by a **set of completion graphs** (see above) generated by the underlying tableau reasoner together with the set of axioms used for its generation.

We show **two incremental techniques** searching for all MUPSes :

- computing single MUPS (*singleMUPSInc*) + Reiter's algorithm

Incremental Methods for Searching all MUPSes

Our techniques do not need full integration with the reasoner, they **need just two operations from the reasoner**:

- **to provide its current state**
- **to restore given state and to apply given axiom (set).**

Two reasoner states corresponding to different axiom orderings are semantically equivalent.

In our case the **state is** represented by a **set of completion graphs** (see above) generated by the underlying tableau reasoner together with the set of axioms used for its generation.

We show **two incremental techniques** searching for all MUPSes :

- computing single MUPS (*singleMUPSInc*) + Reiter's algorithm
- computing directly all MUPSes (*allMUPSesInc2*).

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \ldots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2, 4\}, \{2, 4, 5\}, \{3, 5\}, 6\}$. The algorithm works as follows :

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \dots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2, 4\}, \{2, 4, 5\}, \{3, 5\}, 6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	[**1**, 2, 3, 4, 5, 6]	$D = []$	e=[1]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \dots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2, 4\}, \{2, 4, 5\}, \{3, 5\}, 6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	[1, **2**, 3, 4, 5, 6]	$D = []$	e=[1,2]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \dots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	[1, 2, **3**, 4, 5, 6]	$D = []$	e=[1,2,3]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \dots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	[1, 2, 3, **4**, 5, 6]	$D = [4]$	e=[1,2,3,4]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \ldots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longleftarrow$	[1, 2, **3**, ~~4, 5, 6~~]	$D = [4]$	e=[4,3]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \ldots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longleftarrow$	[1, **2**, 3, ~~4, 5, 6~~]	$D = [4]$	e=[4,3,2]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \ldots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longleftarrow$	[**1**, 2, 3, ~~4, 5, 6~~]	$D = [4, 1]$	e=[4,3,2,1]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \dots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	[~~1~~, **2**, 3, ~~4, 5, 6~~]	$D = [4,1,2]$	e=[4,1,2]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \ldots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2, 4\}, \{2, 4, 5\}, \{3, 5\}, 6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	$[\cancel{1}, \cancel{2, 3, 4, 5, 6}]$	$D = [4, 1, 2]$	e=[4,1,2]

Algorithm for Single MUPS (*singleMUPSInc*)

- Initial axiom list P and an empty reasoner state e.
- "Feed" the reasoner with axioms one by one until an unsatisfiability is detected.
- When an unsatisfiability is detected, the search direction is changed, current reasoner state e is initialized with D, "overlapping" axioms are pruned (emphasized with strikeout) and the last axiom that caused the unsatisfiability (in bold) is put into the MUPS core D.

Example

Having 6 axioms numbered $1 \ldots 6$, a concept is unsatisfiable due to the MUPSes $\{\{1,2,4\},\{2,4,5\},\{3,5\},6\}$. The algorithm works as follows :

direction	input list P	MUPS core	reasoner state
$\longrightarrow$	[~~1~~, ~~2, 3, 4, 5, 6~~]	$D = [4,1,2]$	e=[4,1,2]

- This algorithm can be used with Reiter's algorithm (Reiter 1987) to compute all MUPSes for given concept.

Algorithm for All MUPSes (*allMUPSesInc2*)

- Improved version of the method (denoted as *allMUPSesInc1*) presented in (De La Banda et al., 2003)

Algorithm for All MUPSes (*allMUPSesInc2*)

- Improved version of the method (denoted as *allMUPSesInc1*) presented in (De La Banda et al., 2003)
- As different orderings of axioms are semantically equivalent, we can **get rid of some redundant calls** to the (expensive) testing procedure.

Algorithm for All MUPSes (*allMUPSesInc2*)

- Improved version of the method (denoted as *allMUPSesInc1*) presented in (De La Banda et al., 2003)
- As different orderings of axioms are semantically equivalent, we can **get rid of some redundant calls** to the (expensive) testing procedure.

cached **is the index in T of the last successful test** ... our improvement

Algorithm for All MUPSes (*allMUPSesInc2*)

- Improved version of the method (denoted as *allMUPSesInc1*) presented in (De La Banda et al., 2003)
- As different orderings of axioms are semantically equivalent, we can **get rid of some redundant calls** to the (expensive) testing procedure.

cached **is the index in T of the last successful test** ... our improvement

D is the MUPS core (we store also the reasoner state corresponding to this set) ... satisfiable set

Algorithm for All MUPSes (*allMUPSesInc2*)

- Improved version of the method (denoted as *allMUPSesInc1*) presented in (De La Banda et al., 2003)
- As different orderings of axioms are semantically equivalent, we can **get rid of some redundant calls** to the (expensive) testing procedure.

cached **is the index in T of the last successful test** ... our improvement

D is the MUPS core (we store also the reasoner state corresponding to this set) ... satisfiable set

P is the axiom list to choose axioms from ... unsatisfiable set

Algorithm for All MUPSes (*allMUPSesInc2*)

- Improved version of the method (denoted as *allMUPSesInc1*) presented in (De La Banda et al., 2003)
- As different orderings of axioms are semantically equivalent, we can **get rid of some redundant calls** to the (expensive) testing procedure.

cached **is the index in T of the last successful test** ... our improvement
: D is the MUPS core (we store also the reasoner state corresponding to this set) ... satisfiable set
: P is the axiom list to choose axioms from ... unsatisfiable set
: T is the list of already explored axioms ... satisfiable set

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$-1, [], [\mathbf{1}, 2, 3], []$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$-1, [], [1, 2, 3], [1]$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$\begin{array}{c} -1, [], [1, 2, 3], [1] \\ \big\downarrow{\scriptstyle [1],\ [1,2]} \\ -1, [], [1, \mathbf{3}], [1] \end{array}$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[d, "{[1],\ [1,2]}"'] \\
-1, [], [1, 3], [1] \arrow[d, "{[1,3]}"'] \\
-1, [], [1], [1]
\end{tikzcd}$$

- Each node has the form $cached, D, P, T$.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[d, "{[1],\ [1,2]}"'] & \\
-1, [], [1, 3], [1] \arrow[r, "{[3]}"'] \arrow[d, shift right, "{[1,3]}"'] & 0, [3], [\mathbf{1}], [3] \\
-1, [], [1], [1] \arrow[u, shift right] &
\end{tikzcd}$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$
\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[d, "{[1],\ [1,2]}"'] & \\
-1, [], [1, 3], [1] \arrow[r, "{[3]}"'] \arrow[d, shift right, "{[1,3]}"'] & 0, [3], [1], [3] \arrow[d, "\cancel{[1,3]}"'] \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3]
\end{tikzcd}
$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$
\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[d, "{[1],\ [1,2]}"'] & & \\
-1, [], [1, 3], [1] \arrow[r, "{[3]}"'] \arrow[d, shift right, "{[1,3]}"'] & 0, [3], [1], [3] \arrow[r, "\cancel{[1,3]}"'] \arrow[d, shift right, "\cancel{[1,3]}"'] & \text{found } [3,1] \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3] \arrow[u, shift right] &
\end{tikzcd}
$$

- Each node has the form $cached, D, P, T$.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1,2\},\{1,3\}\}$. The algorithm works as follows :

$$\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[d, shift right, "{[1],\ [1,2]}"'] & & \\
-1, [], [1, 3], [1] \arrow[u, shift right] \arrow[d, shift right, "{[1,3]}"'] \arrow[r, shift right, "{[3]}"'] & 0, [3], [1], [3] \arrow[l, shift right] \arrow[d, shift right, "{\cancel{[1,3]}}"'] \arrow[r, shift right, "{\cancel{[1,3]}}"'] & \text{found } [3,1] \arrow[l, shift right] \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3] \arrow[u, shift right] &
\end{tikzcd}$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$
\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[rrr, "{[2]}"'] \arrow[d, shift right, "{[1],\ [1{,}2]}"'] & & & 0, [2], [1, 3], [2] \\
-1, [], [1, 3], [1] \arrow[u, shift right] \arrow[r, shift right, "{[3]}"'] \arrow[d, shift right, "{[1{,}3]}"'] & 0, [3], [1], [3] \arrow[l, shift right] \arrow[r, shift right, "\cancel{[1{,}3]}"'] \arrow[d, shift right, "\cancel{[1{,}3]}"'] & \text{found } [3{,}1] \arrow[l, shift right] & \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3] \arrow[u, shift right] & &
\end{tikzcd}
$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$
\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[rrr, "{[2]}"'] \arrow[d, shift right, "{[1],\ [1,2]}"'] & & & 0, [2], [1, 3], [2] \arrow[d, "\cancel{[1,2]}"'] \\
-1, [], [1, 3], [1] \arrow[u, shift right] \arrow[r, shift right, "{[3]}"'] \arrow[d, shift right, "{[1,3]}"'] & 0, [3], [1], [3] \arrow[l, shift right] \arrow[r, shift right, "\cancel{[1,3]}"'] \arrow[d, shift right, "\cancel{[1,3]}"'] & \text{found } [3,1] \arrow[l, shift right] & -1, [2], [\mathbf{3}], [2] \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3] \arrow[u, shift right] & &
\end{tikzcd}
$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

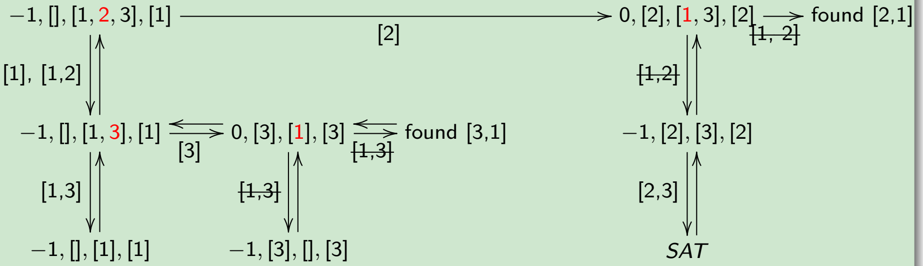
$$
\begin{tikzcd}
-1, [], [1, {\color{red}2}, 3], [1] \arrow[rrr, "{[2]}"'] \arrow[d, shift right, "{[1],\ [1,2]}"'] & & & 0, [2], [{\color{red}1}, 3], [2] \arrow[d, "\cancel{[1,2]}"'] \\
-1, [], [1, {\color{red}3}], [1] \arrow[u, shift right] \arrow[r, shift right, "{[3]}"'] \arrow[d, shift right, "{[1,3]}"'] & 0, [3], [{\color{red}1}], [3] \arrow[l, shift right] \arrow[r, shift right, "\cancel{[1,3]}"'] \arrow[d, shift right, "\cancel{[1,3]}"'] & \text{found } [3,1] \arrow[l, shift right] & -1, [2], [3], [2] \arrow[d, "{[2,3]}"'] \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3] \arrow[u, shift right] & & SAT
\end{tikzcd}
$$

- Each node has the form $cached, D, P, T$.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1,2\},\{1,3\}\}$. The algorithm works as follows :



- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Algorithm for All MUPSes (2) – Example

Example

Having 3 axioms numbered $1, 2, 3$, a concept is unsatisfiable due to the MUPSes $\{\{1, 2\}, \{1, 3\}\}$. The algorithm works as follows :

$$
\begin{tikzcd}
-1, [], [1, 2, 3], [1] \arrow[rrr, shift left, "{[2]}"'] \arrow[d, shift right, "{[1],\ [1,2]}"'] & & & 0, [2], [1, 3], [2] \arrow[lll, shift left] \arrow[r, shift left, "\cancel{[1,\ 2]}"'] \arrow[d, shift right, "\cancel{[1,2]}"'] & \text{found } [2,1] \arrow[l, shift left] \\
-1, [], [1, 3], [1] \arrow[u, shift right] \arrow[r, shift left, "{[3]}"'] \arrow[d, shift right, "{[1,3]}"'] & 0, [3], [1], [3] \arrow[l, shift left] \arrow[r, shift left, "\cancel{[1,3]}"'] \arrow[d, shift right, "\cancel{[1,3]}"'] & \text{found } [3,1] \arrow[l, shift left] & -1, [2], [3], [2] \arrow[u, shift right] \arrow[d, shift right, "{[2,3]}"'] & \\
-1, [], [1], [1] \arrow[u, shift right] & -1, [3], [], [3] \arrow[u, shift right] & & SAT \arrow[u, shift right] &
\end{tikzcd}
$$

- Each node has the form *cached*, D, P, T.
- The search is performed in the depth first manner. Edges are labeled with the tests that have been performed before the unsatisfiability is found.
- Struck axiom sets represent the tests that are avoided in comparison to *allMUPSesInc1*.

Comparison of Incremental and Black Box Techniques

- Underlying reasoner is our impl. of the $\mathcal{SHIN}$ tableaux algorithm.

[1] http://www.mindswap.org/2005/debugging/ontologies

Comparison of Incremental and Black Box Techniques

- Underlying reasoner is our impl. of the $\mathcal{SHIN}$ tableaux algorithm.
- **Reiter+singleMUPSbb** is a black box approach presented in (Kalyanpur, 2006)

[1] http://www.mindswap.org/2005/debugging/ontologies

Comparison of Incremental and Black Box Techniques

- Underlying reasoner is our impl. of the $\mathcal{SHIN}$ tableaux algorithm.
- **Reiter+singleMUPSbb** is a black box approach presented in (Kalyanpur, 2006)
- **Reiter+singleMUPSinc** is our incremental algorithm for computing a single MUPS. All MUPSes are computed using the Reiter's algorithm.

[1] http://www.mindswap.org/2005/debugging/ontologies

Comparison of Incremental and Black Box Techniques

- Underlying reasoner is our impl. of the $\mathcal{SHIN}$ tableaux algorithm.
- **Reiter+singleMUPSbb** is a black box approach presented in (Kalyanpur, 2006)
- **Reiter+singleMUPSinc** is our incremental algorithm for computing a single MUPS. All MUPSes are computed using the Reiter's algorithm.
- **allMUPSesInc1** is the method presented in (De La Banda, 2006)

[1] http://www.mindswap.org/2005/debugging/ontologies

Comparison of Incremental and Black Box Techniques

- Underlying reasoner is our impl. of the $\mathcal{SHIN}$ tableaux algorithm.
- **Reiter+singleMUPSbb** is a black box approach presented in (Kalyanpur, 2006)
- **Reiter+singleMUPSinc** is our incremental algorithm for computing a single MUPS. All MUPSes are computed using the Reiter's algorithm.
- **allMUPSesInc1** is the method presented in (De La Banda, 2006)
- **allMUPSesInc2** is our modification of *allMUPSesInc1*

[1] http://www.mindswap.org/2005/debugging/ontologies

Comparison of Incremental and Black Box Techniques

- Underlying reasoner is our impl. of the $\mathcal{SHIN}$ tableaux algorithm.
- **Reiter+singleMUPSbb** is a black box approach presented in (Kalyanpur, 2006)
- **Reiter+singleMUPSinc** is our incremental algorithm for computing a single MUPS. All MUPSes are computed using the Reiter's algorithm.
- **allMUPSesInc1** is the method presented in (De La Banda, 2006)
- **allMUPSesInc2** is our modification of *allMUPSesInc1*
- Finding all MUPSes for all concepts in the miniTambis ontology (30/182 unsat. concepts) and the miniEconomy ontology[1](51/338 unsat. concepts):

algorithm	miniTambis (time [ms])	miniEconomy (time [ms])
Reiter + singleMUPSbb	67481	$> 15 min.$
Reiter + singleMUPSinc	**19875**	**19796**
allMUPSesInc1	8655	14110
allMUPSesInc2	**8516**	**13970**

[1] http://www.mindswap.org/2005/debugging/ontologies

Comparison of Incremental Methods w.r.t. Axiom Ordering

- As we need to generate all permutations of the axiom set, we use two small ontologies.

[2] http://protege.stanford.edu/plugins/owl/owl-library/tambis-full

[3] http://www.mindswap.org/2005/debugging/ontologies/madcow.owl

[4] incremental test

Comparison of Incremental Methods w.r.t. Axiom Ordering

- As we need to generate all permutations of the axiom set, we use two small ontologies.
- *TambisP* is the restriction (6 axioms) of the Tambis ontology[2], to the definitions of the unsatisfiable concepts *metal*, *nonmetal* and *metalloid*.

[2] http://protege.stanford.edu/plugins/owl/owl-library/tambis-full
[3] http://www.mindswap.org/2005/debugging/ontologies/madcow.owl
[4] incremental test

Comparison of Incremental Methods w.r.t. Axiom Ordering

- As we need to generate all permutations of the axiom set, we use two small ontologies.
- *TambisP* is the restriction (6 axioms) of the Tambis ontology[2], to the definitions of the unsatisfiable concepts *metal*, *nonmetal* and *metalloid*.
- *madCowP* is the restriction (7 axioms) of the Mad Cow ontology[3] to the axioms causing unsatisfiability of the concept *madCow*.

[2] http://protege.stanford.edu/plugins/owl/owl-library/tambis-full
[3] http://www.mindswap.org/2005/debugging/ontologies/madcow.owl
[4] incremental test

Comparison of Incremental Methods w.r.t. Axiom Ordering

- As we need to generate all permutations of the axiom set, we use two small ontologies.
- *TambisP* is the restriction (6 axioms) of the Tambis ontology[2], to the definitions of the unsatisfiable concepts *metal*, *nonmetal* and *metalloid*.
- *madCowP* is the restriction (7 axioms) of the Mad Cow ontology[3] to the axioms causing unsatisfiability of the concept *madCow*.

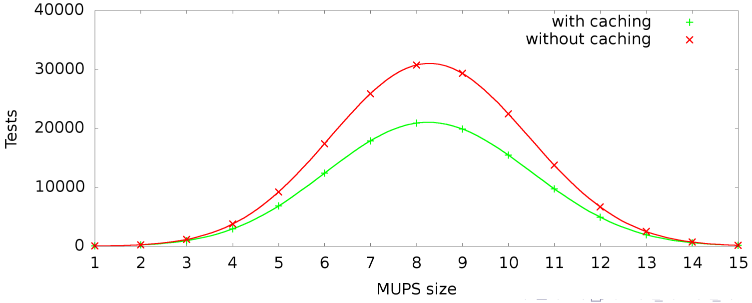
tambisP	# of IT[4]	avg # of IT	var of IT
Reiter + singleMUPSinc	**268362**	**124.29**	**206.81**
allMUPSesInc1	75696	35.04	36.44
allMUPSesInc2	**61590**	**28.51**	**16.76**

madCowP	# of IT	avg # of IT	var of IT
Reiter + singleMUPSinc	**277200**	**55.00**	**8.00**
allMUPSesInc1	131040	26.00	0.00
allMUPSesInc2	**119520**	**23.04**	**0.50**

[2] http://protege.stanford.edu/plugins/owl/owl-library/tambis-full
[3] http://www.mindswap.org/2005/debugging/ontologies/madcow.owl
[4] incremental test

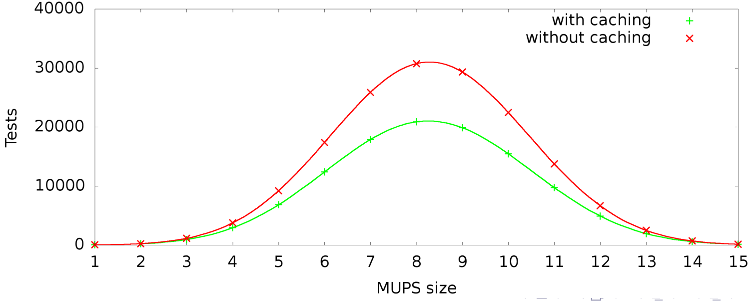
Evaluation of Caching

- For which knowledge bases is the caching gain significant ?



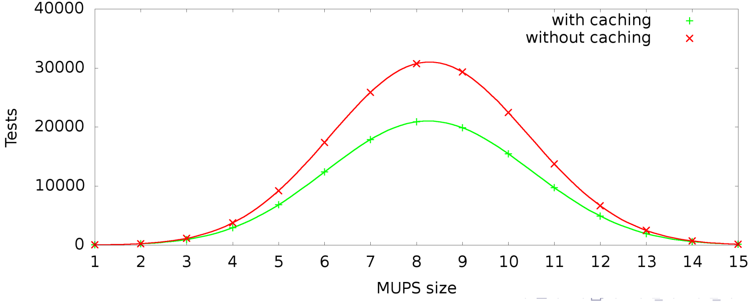
Evaluation of Caching

- For which knowledge bases is the caching gain significant ?
- Let's have an artificial KB containing n (depicted for $n = 15$) axioms.



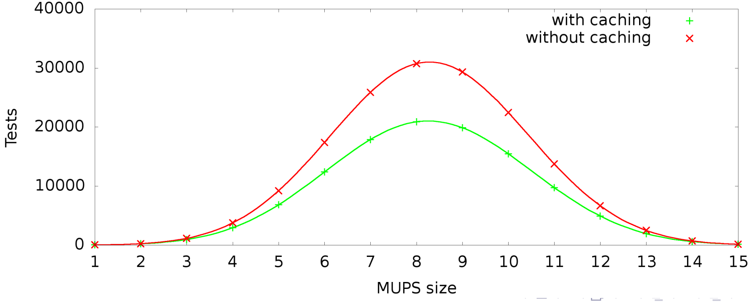
Evaluation of Caching

- For which knowledge bases is the caching gain significant ?
- Let's have an artificial KB containing n (depicted for $n = 15$) axioms.
- For each $1 \leq k \leq n$ (the x-axis) the MUPS set contains all axiom combinations of size k (horizontal axis).

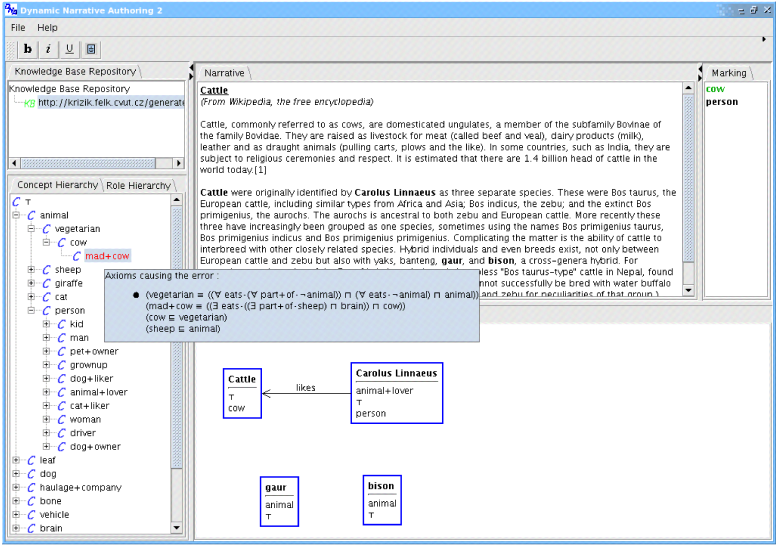


Evaluation of Caching

- For which knowledge bases is the caching gain significant ?
- Let's have an artificial KB containing n (depicted for $n = 15$) axioms.
- For each $1 \leq k \leq n$ (the x-axis) the MUPS set contains all axiom combinations of size k (horizontal axis).
- **The caching is most efficient (about 30% gain) when the MUPSes are approximately medium-sized w.r.t. the number of axioms.**



Annotation Tool Prototype



Summary

- Two novel promising **incremental algorithms** for error explanations were proposed and implemented.**Comparing to the black box methods they proved to be more efficient and robust to axiom ordering**.

Summary

- Two novel promising **incremental algorithms** for error explanations were proposed and implemented.**Comparing to the black box methods they proved to be more efficient and robust to axiom ordering**.
- The "single MUPS" versions are less efficient than the "all MUPS" versions, but **"single MUPS" versions provide error explanations for free** (using Reiter's algorithm), while "all MUPS" versions provide only conflict sets.

Summary

- Two novel promising **incremental algorithms** for error explanations were proposed and implemented.**Comparing to the black box methods they proved to be more efficient and robust to axiom ordering**.
- The "single MUPS" versions are less efficient than the "all MUPS" versions, but **"single MUPS" versions provide error explanations for free** (using Reiter's algorithm), while "all MUPS" versions provide only conflict sets.
- An inference services based on the proposed algorithms was **implemented in the new annotation tool prototype**, providing the user with explanations for given concept unsatisfiability.

Future Work

- Optimizations of the presented explanation techniques :

Future Work

- Optimizations of the presented explanation techniques :
  - Incomplete glass-box approach as the preprocessing step for the incremental methods discussed above.

Future Work

- Optimizations of the presented explanation techniques :
  - Incomplete glass-box approach as the preprocessing step for the incremental methods discussed above.
  - Partitioning of the axiom set.

Future Work

- Optimizations of the presented explanation techniques :
  - Incomplete glass-box approach as the preprocessing step for the incremental methods discussed above.
  - Partitioning of the axiom set.
- Inference support for advanced modeling constructs, like n-ary relations. SoA reasoning techniques irreversibly reify them to concepts. This approach has two disadvantages :

Future Work

- Optimizations of the presented explanation techniques :
  - Incomplete glass-box approach as the preprocessing step for the incremental methods discussed above.
  - Partitioning of the axiom set.
- Inference support for advanced modeling constructs, like n-ary relations. SoA reasoning techniques irreversibly reify them to concepts. This approach has two disadvantages :
  - The axiomatization of reification generates lots of new axioms into the knowledge base.

Future Work

- Optimizations of the presented explanation techniques :
  - Incomplete glass-box approach as the preprocessing step for the incremental methods discussed above.
  - Partitioning of the axiom set.
- Inference support for advanced modeling constructs, like n-ary relations. SoA reasoning techniques irreversibly reify them to concepts. This approach has two disadvantages :
  - The axiomatization of reification generates lots of new axioms into the knowledge base.
  - Error explanations contain artificial concepts and roles – poor interpretability.

Future Work

- Optimizations of the presented explanation techniques :
  - Incomplete glass-box approach as the preprocessing step for the incremental methods discussed above.
  - Partitioning of the axiom set.
- Inference support for advanced modeling constructs, like n-ary relations. SoA reasoning techniques irreversibly reify them to concepts. This approach has two disadvantages :
  - The axiomatization of reification generates lots of new axioms into the knowledge base.
  - Error explanations contain artificial concepts and roles – poor interpretability.

Example

$reign(Person, Time, Location)$ generates a concept *Reign*, and three new properties `reignPerson`, `reignTime` and `reignLocation`. New axioms are generated that state these properties to be *functional*, to have corresponding domain and range and to allow only syntactically valid n-ary relations.

Future Work (2)

- How to solve the above mentioned problems ?

Future Work (2)

- How to solve the above mentioned problems ?
  - To develop a compact description logic based language that allows for n-ary relations and that is suitable for semantic annotations.

Future Work (2)

- How to solve the above mentioned problems ?
  - To develop a compact description logic based language that allows for n-ary relations and that is suitable for semantic annotations.
  - To develop a tableau reasoning algorithm that will operate directly with n-ary relations.

Future Work (2)

- How to solve the above mentioned problems ?
    - To develop a compact description logic based language that allows for n-ary relations and that is suitable for semantic annotations.
    - To develop a tableau reasoning algorithm that will operate directly with n-ary relations.
    - To develop error explanation techniques for this n-ary description logic.

Resources

- Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny.
  Finding All Minimal Unsatisfiable Subsets.
  In *PPDP'03*, 2003.
- Aditya Kalyanpur.
  Debugging and Repair of OWL Ontologies.
  PhD thesis, University of Maryland, 2006.
- Raymond Reiter.
  A Theory of Diagnosis from First Principles.
  Artificial Intelligence, 32(1):57–96, April 1987.
- Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen.
  Debugging Incoherent Terminologies.
  Technical report, Vrije Universiteit Amsterdam, 2006.