Computing Repairs from Active Integrity Constraints

L. Cruz-Filipe[1,2,4] P. Engrácia[1,2] G. Gaspar[3,4] I. Nunes[3,4]

[1]Escola Superior Náutica Infante D. Henrique

[2]Centro de Matemática e Aplicações Fundamentais

[3]Faculty of Sciences, University of Lisbon

[4]Laboratory of Agent Modelling (LabMAg)

LabMAg Seminar
June 25th, 2013

The Problem

Databases typically pose conditions on data ("integrity constraints")...

The Problem

Databases typically pose conditions on data ("integrity constraints")...

...but because of errors sometimes these conditions no longer hold.

The Problem

Databases typically pose conditions on data ("integrity constraints")...

...but because of errors sometimes these conditions no longer hold.

Question

How can we repair a database that no longer satisfies its integrity constraints?

Outline

1. Integrity constraints

Outline

Outline

Outline

1. Integrity constraints
2. Active integrity constraints
3. Founded and justified repairs
4. Future directions

Outline

Outline

A database of family relations (I)

Consider a database with information on family relations.

A database of family relations (I)

Consider a database with information on family relations.

Fact

$$\mathsf{siblingOf}(\mathsf{John}, \mathsf{Mary})$$

A database of family relations (I)

Consider a database with information on family relations.

Fact

siblingOf(John, Mary)

This database should also contain

Missing fact

siblingOf(Mary, John)

A database of family relations (I)

Consider a database with information on family relations.

Fact

$$\mathrm{siblingOf}(\mathrm{John}, \mathrm{Mary})$$

This database should also contain

Missing fact

$$\mathrm{siblingOf}(\mathrm{Mary}, \mathrm{John})$$

Integrity constraint (simple)

$$\forall x \forall y.((\mathrm{siblingOf}(x, y) \land \neg\mathrm{siblingOf}(y, x)) \supset \bot)$$

A database of family relations (II)

Fact

Parent(John)

A database of family relations (II)

Fact

Parent(John)

This database should also contain

Missing fact

fatherOf(John, . . .)

A database of family relations (II)

Fact

Parent(John)

This database should also contain

Missing fact

fatherOf(John, . . .)

(motherOf(John, . . .) would fix this, but would cause other problems)

A database of family relations (II)

Fact

$$\text{Parent}(\text{John})$$

This database should also contain

Missing fact

$$\text{fatherOf}(\text{John}, \ldots)$$

($\text{motherOf}(\text{John}, \ldots)$ would fix this, but would cause other problems)

Integrity constraint (existential)

$$\forall x.(\text{Parent}(x) \supset \exists y.(\text{fatherOf}(x, y) \vee \text{motherOf}(x, y)))$$

A database of family relations (III)

Fact

$$\mathrm{fatherOf}(\mathrm{John}, \mathrm{Paul})$$

A database of family relations (III)

Fact

fatherOf(John, Paul)

Fact

fatherOf(Jack, Paul)

A database of family relations (III)

Fact

fatherOf(John, Paul)

Fact

fatherOf(Jack, Paul)

Something is wrong here. . .

A database of family relations (III)

Fact

$$\text{fatherOf}(\text{John}, \text{Paul})$$

Fact

$$\text{fatherOf}(\text{Jack}, \text{Paul})$$

Something is wrong here. . .

Integrity constraint (cardinality)

$$\forall x \forall y \forall z.((\text{fatherOf}(x, z) \wedge \text{fatherOf}(y, z)) \supset x = y)$$

Sometimes we can fix the problem automatically. . .

Inconsistency

$$\mathsf{siblingOf}(\mathsf{John}, \mathsf{Mary})$$

$$\forall x \forall y.((\mathsf{siblingOf}(x, y) \wedge \neg \mathsf{siblingOf}(y, x)) \supset \bot)$$

Sometimes we can fix the problem automatically. . .

Inconsistency

$$\mathrm{siblingOf}(\mathrm{John}, \mathrm{Mary})$$

$$\forall x \forall y.((\mathrm{siblingOf}(x, y) \wedge \neg\mathrm{siblingOf}(y, x)) \supset \bot)$$

Solution

Add $\mathrm{siblingOf}(\mathrm{Mary}, \mathrm{John})$

Sometimes we can fix the problem automatically. . .

Inconsistency

$$\mathsf{siblingOf}(\mathsf{John}, \mathsf{Mary})$$

$$\forall x \forall y.((\mathsf{siblingOf}(x, y) \wedge \neg \mathsf{siblingOf}(y, x)) \supset \bot)$$

Solution

Add $\mathsf{siblingOf}(\mathsf{Mary}, \mathsf{John})$

. . . but is this so automatic?

Sometimes we can fix the problem automatically. . .

Inconsistency

$$\text{siblingOf}(\text{John}, \text{Mary})$$

$$\forall x \forall y.((\text{siblingOf}(x, y) \wedge \neg\text{siblingOf}(y, x)) \supset \bot)$$

Solution

Add siblingOf(Mary, John)

. . . but is this so automatic?

Another solution

Remove siblingOf(John, Mary)

Sometimes we can fix the problem automatically. . .

Oops.

. . . sometimes not really. . .

Inconsistency

$$\mathrm{Parent}(\mathrm{John})$$

$$\forall x.(\mathrm{Parent}(x) \supset \exists y.(\mathrm{fatherOf}(x, y) \vee \mathrm{motherOf}(x, y)))$$

. . . sometimes not really. . .

Inconsistency

$$\mathrm{Parent}(\mathrm{John})$$

$$\forall x.(\mathrm{Parent}(x) \supset \exists y.(\mathrm{fatherOf}(x, y) \vee \mathrm{motherOf}(x, y)))$$

Solution

Add fatherOf(John, ???)

. . . sometimes not really. . .

Inconsistency

$$\text{Parent}(\text{John})$$

$$\forall x.(\text{Parent}(x) \supset \exists y.(\text{fatherOf}(x, y) \vee \text{motherOf}(x, y)))$$

Solution

Add fatherOf(John, ???)

What should we substitute for y?

. . . sometimes not really. . .

Inconsistency

$$\mathsf{Parent}(\mathsf{John})$$

$$\forall x.(\mathsf{Parent}(x) \supset \exists y.(\mathsf{fatherOf}(x, y) \vee \mathsf{motherOf}(x, y)))$$

Solution

Add fatherOf(John, ???)

What should we substitute for y?

Another solution

Remove Parent(John)

...sometimes not really...

Inconsistency

$$\mathsf{Parent}(\mathsf{John})$$

$$\forall x.(\mathsf{Parent}(x) \supset \exists y.(\mathsf{fatherOf}(x, y) \vee \mathsf{motherOf}(x, y)))$$

Solution

Add fatherOf(John, ???)

What should we substitute for y?

Another solution

Remove Parent(John)

Ok, but...

…and computers do not like to make choices.

Inconsistency

$$\mathsf{fatherOf}(\mathsf{John}, \mathsf{Paul})$$

$$\mathsf{fatherOf}(\mathsf{Jack}, \mathsf{Paul})$$

$$\forall x \forall y \forall z.((\mathsf{fatherOf}(x, z) \wedge \mathsf{fatherOf}(y, z)) \supset x = y)$$

. . . and computers do not like to make choices.

Inconsistency

$$\mathsf{fatherOf}(\mathsf{John}, \mathsf{Paul})$$

$$\mathsf{fatherOf}(\mathsf{Jack}, \mathsf{Paul})$$

$$\forall x \forall y \forall z.((\mathsf{fatherOf}(x, z) \wedge \mathsf{fatherOf}(y, z)) \supset x = y)$$

Typical database semantics implies that John $\neq$ Jack.

. . . and computers do not like to make choices.

Inconsistency

$$\mathrm{fatherOf}(\mathrm{John}, \mathrm{Paul})$$

$$\mathrm{fatherOf}(\mathrm{Jack}, \mathrm{Paul})$$

$$\forall x \forall y \forall z.((\mathrm{fatherOf}(x, z) \wedge \mathrm{fatherOf}(y, z)) \supset x = y)$$

Typical database semantics implies that John $\neq$ Jack.

Solution

Remove $\mathrm{fatherOf}(\mathrm{John}, \mathrm{Paul})$

. . . and computers do not like to make choices.

Inconsistency

$$\mathsf{fatherOf}(\mathsf{John}, \mathsf{Paul})$$

$$\mathsf{fatherOf}(\mathsf{Jack}, \mathsf{Paul})$$

$$\forall x \forall y \forall z.((\mathsf{fatherOf}(x, z) \wedge \mathsf{fatherOf}(y, z)) \supset x = y)$$

Typical database semantics implies that John $\neq$ Jack.

Solution

Remove fatherOf(John, Paul)

Another solution

Remove fatherOf(Jack, Paul)

. . . and computers do not like to make choices.

Inconsistency

$$\text{fatherOf}(\text{John}, \text{Paul})$$

$$\text{fatherOf}(\text{Jack}, \text{Paul})$$

$$\forall x \forall y \forall z.((\text{fatherOf}(x, z) \wedge \text{fatherOf}(y, z)) \supset x = y)$$

Typical database semantics implies that John $\neq$ Jack.

Solution

Remove fatherOf(John, Paul)

Another solution

Remove fatherOf(Jack, Paul)

But which?

To make matters worse. . .

To make matters worse. . .

Exercise

Think up a scenario where the “bad” solution is actually the good one and the “good” solution is the bad one.

To make matters worse. . .

Exercise

Think up a scenario where the "bad" solution is actually the good one and the "good" solution is the bad one.

Solution

Imagine information is being removed from the database.

Outline

Active integrity constraints

Motivation

Specify a constraint and propose possible solutions.

Active integrity constraints

Motivation

Specify a constraint and propose possible solutions.

Works both ways:

Active integrity constraints

Motivation

Specify a constraint and propose possible solutions.

Works both ways:

- may express preferences

Active integrity constraints

Motivation

Specify a constraint and propose possible solutions.

Works both ways:

- may express preferences
- may eliminate options

Family relations, revisited

Integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x, y) \wedge \neg\mathsf{siblingOf}(y, x)) \supset \bot)$$

Family relations, revisited

Integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x,y) \wedge \neg\mathsf{siblingOf}(y,x)) \supset \bot)$$

Active integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x,y) \wedge \neg\mathsf{siblingOf}(y,x)) \supset +\mathsf{siblingOf}(y,x))$$

Family relations, revisited

Integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x,y) \wedge \neg\mathsf{siblingOf}(y,x)) \supset \bot)$$

Active integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x,y) \wedge \neg\mathsf{siblingOf}(y,x)) \supset -\mathsf{siblingOf}(x,y))$$

Family relations, revisited

Integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x, y) \wedge \neg \mathsf{siblingOf}(y, x)) \supset \bot)$$

Active integrity constraint

$$\forall x \forall y.((\mathsf{siblingOf}(x, y) \wedge \neg \mathsf{siblingOf}(y, x)) \supset \\ + \mathsf{siblingOf}(y, x) \mid -\mathsf{siblingOf}(x, y))$$

Active integrity constraints

Definition (Caroprese et al., 2011)

An *Active integrity constraint* is a formula of the form

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

where $\{\alpha_1^D, \ldots, \alpha_k^D\} \subseteq \{L_1, \ldots, L_m\}$.

Active integrity constraints

Definition (Caroprese et al., 2011)

An *Active integrity constraint* is a formula of the form

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

where $\{\alpha_1^D, \ldots, \alpha_k^D\} \subseteq \{L_1, \ldots, L_m\}$.

A valid AIC

$$\mathrm{siblingOf}(x, y) \land \neg\mathrm{siblingOf}(y, x) \supset +\mathrm{siblingOf}(y, x)$$

Active integrity constraints

Definition (Caroprese et al., 2011)

An *Active integrity constraint* is a formula of the form

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

where $\{\alpha_1^D, \ldots, \alpha_k^D\} \subseteq \{L_1, \ldots, L_m\}$.

A valid AIC

$$\text{siblingOf}(x, y) \wedge \neg\text{siblingOf}(y, x) \supset +\text{siblingOf}(y, x)$$

An invalid AIC

$$\text{siblingOf}(x, y) \wedge \neg\text{siblingOf}(y, x) \supset -\text{siblingOf}(x, y) \mid +\text{Parent}(x)$$

Intuitive semantics of AICs

A generic AIC

$$L_1, \dots, L_m \supset \alpha_1 \mid \dots \mid \alpha_k$$

Intuitive semantics of AICs

A generic AIC

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

- conjunction on the left ("body")

Intuitive semantics of AICs

A generic AIC

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

- conjunction on the left ("body")
- disjunction on the right ("head")

Intuitive semantics of AICs

A generic AIC

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

- conjunction on the left ("body")
- disjunction on the right ("head")
- semantics of (normal) implication

Intuitive semantics of AICs

A generic AIC

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

- conjunction on the left ("body")
- disjunction on the right ("head")
- semantics of (normal) implication
- holds iff one of the L_is fails (but...)

Intuitive semantics of AICs

A generic AIC

$$L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$$

- conjunction on the left ("body")
- disjunction on the right ("head")
- semantics of (normal) implication
- holds iff one of the L_is fails (but...)
- $\{\alpha_1^D, \ldots, \alpha_k^D\}$ are *updatable* literals

Repairs

Definition

Let I be a database and η be a set of (A)ICs. A *weak repair* for I and η is a consistent set $\mathcal{U}$ of update actions such that:

Repairs

Definition

Let I be a database and η be a set of (A)ICs. A *weak repair* for I and η is a consistent set $\mathcal{U}$ of update actions such that:

- $\mathcal{U}$ consists of essential actions only

Repairs

Definition

Let I be a database and η be a set of (A)ICs. A *weak repair* for I and η is a consistent set $\mathcal{U}$ of update actions such that:

- $\mathcal{U}$ consists of essential actions only
- $\mathcal{I} \circ \mathcal{U} \models \eta$

Repairs

Definition

Let I be a database and η be a set of (A)ICs. A *weak repair* for I and η is a consistent set $\mathcal{U}$ of update actions such that:

- $\mathcal{U}$ consists of essential actions only
- $\mathcal{I} \circ \mathcal{U} \models \eta$

(Sorry for the notation.)

Definition

A *repair* is a weak repair that is minimal w.r.t. inclusion.

Family relations, yet again

Inconsistency

$$\mathrm{siblingOf}(\mathrm{John}, \mathrm{Mary})$$

$$\mathrm{siblingOf}(x, y) \wedge \neg \mathrm{siblingOf}(y, x) \supset +\mathrm{siblingOf}(y, x)$$

Family relations, yet again

A repair

$$+\mathrm{siblingOf}(\mathrm{Mary}, \mathrm{John})$$

Family relations, yet again

A repair

$$+\mathrm{siblingOf}(\mathrm{Mary}, \mathrm{John})$$

Another repair

$$-\mathrm{siblingOf}(\mathrm{John}, \mathrm{Mary})$$

Family relations, yet again

A repair

$$+\mathrm{siblingOf}(\mathrm{Mary}, \mathrm{John})$$

Another repair

$$-\mathrm{siblingOf}(\mathrm{John}, \mathrm{Mary})$$

A weak repair

$$+\mathrm{siblingOf}(\mathrm{Mary}, \mathrm{John}), +\mathrm{Parent}(\mathrm{John})$$

Family relations, yet again

A repair

$$+\mathsf{siblingOf}(\mathsf{Mary}, \mathsf{John})$$

Another repair

$$-\mathsf{siblingOf}(\mathsf{John}, \mathsf{Mary})$$

A weak repair

$$+\mathsf{siblingOf}(\mathsf{Mary}, \mathsf{John}), +\mathsf{Parent}(\mathsf{John})$$

Not a weak repair

$$+\mathsf{siblingOf}(\mathsf{Mary}, \mathsf{John}), -\mathsf{siblingOf}(\mathsf{John}, \mathsf{Mary})$$

Where does the "active" part come in?

The notion of (weak) repair ignores the head of the AIC.

Where does the "active" part come in?

The notion of (weak) repair ignores the head of the AIC.

We will come back to that later.

Where does the “active” part come in?

The notion of (weak) repair ignores the head of the AIC.

We will come back to that later.

At this stage, how can we find repairs?

Finding weak repairs is NP-complete

Algorithm

Finding weak repairs is NP-complete

Algorithm

1. Choose a set $\mathcal{U}$ of update actions (based on $\mathcal{I}$)

Finding weak repairs is NP-complete

Algorithm

1. Choose a set $\mathcal{U}$ of update actions (based on $\mathcal{I}$)
2. Compute $\mathcal{I} \circ \mathcal{U}$

Finding weak repairs is NP-complete

Algorithm

1. Choose a set $\mathcal{U}$ of update actions (based on $\mathcal{I}$)
2. Compute $\mathcal{I} \circ \mathcal{U}$
3. Check if all AICs in η hold

Finding weak repairs is NP-complete

Algorithm

1. Choose a set $\mathcal{U}$ of update actions (based on $\mathcal{I}$)
2. Compute $\mathcal{I} \circ \mathcal{U}$
3. Check if all AICs in η hold

Each step can be done in polynomial time on $\mathcal{I}$ and η.

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for* $\mathcal{I}$ *and* η) as follows.

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$
2. For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$, then:

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$
2. For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$, then:
   - for each action L in the body of r, $n' = n \cup \{L^D\}$ is a child of n

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$
2. For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$, then:
   - for each action L in the body of r, $n' = n \cup \{L^D\}$ is a child of n
   - the edge from n to n' is labeled by r

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$
2. For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$, then:
   - for each action L in the body of r, $n' = n \cup \{L^D\}$ is a child of n
   - the edge from n to n' is labeled by r

Lemma

- *This tree is finite.*

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$
2. For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$, then:
    - for each action L in the body of r, $n' = n \cup \{L^D\}$ is a child of n
    - the edge from n to n' is labeled by r

Lemma

- *This tree is finite.*
- *Every consistent leaf is a weak repair for I and η.*

Finding weak repairs is NP-complete, our version

Tree algorithm

Build a tree (the *repair tree for $\mathcal{I}$ and η*) as follows.

1. The root is $\emptyset$
2. For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$, then:
   - for each action L in the body of r, $n' = n \cup \{L^D\}$ is a child of n
   - the edge from n to n' is labeled by r

Lemma

- *This tree is finite.*
- *Every consistent leaf is a weak repair for I and η.*
- *Every repair for I and η corresponds to a leaf in the tree.*

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.
If $\mathcal{U}_i$ is a weak repair, then $\mathcal{U}_i = \mathcal{U}$, otherwise $\mathcal{U}$ is not a repair.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.
If $\mathcal{U}_i$ is a weak repair, then $\mathcal{U}_i = \mathcal{U}$, otherwise $\mathcal{U}$ is not a repair.
Assume that is not the case.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.
If $\mathcal{U}_i$ is a weak repair, then $\mathcal{U}_i = \mathcal{U}$, otherwise $\mathcal{U}$ is not a repair.
Assume that is not the case. Then $\mathcal{I} \circ \mathcal{U}_i \not\models r$ for some rule r.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.
If $\mathcal{U}_i$ is a weak repair, then $\mathcal{U}_i = \mathcal{U}$, otherwise $\mathcal{U}$ is not a repair.
Assume that is not the case. Then $\mathcal{I} \circ \mathcal{U}_i \not\models r$ for some rule r. If the body of r does not contain literals fixed by update actions in $\mathcal{U} \setminus \mathcal{U}_i$, then $\mathcal{I} \circ \mathcal{U} \not\models r$, which is absurd.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.
If $\mathcal{U}_i$ is a weak repair, then $\mathcal{U}_i = \mathcal{U}$, otherwise $\mathcal{U}$ is not a repair.
Assume that is not the case. Then $\mathcal{I} \circ \mathcal{U}_i \not\models r$ for some rule r. If the body of r does not contain literals fixed by update actions in $\mathcal{U} \setminus \mathcal{U}_i$, then $\mathcal{I} \circ \mathcal{U} \not\models r$, which is absurd. Therefore there is a descendant $\mathcal{U}_{i+1}$ of $\mathcal{U}_i$ such that $\mathcal{U}_{i+1} \subseteq \mathcal{U}$.

Repair trees find all repairs

Lemma

Every repair for I and η corresponds to a leaf in the tree.

Proof.

Let $\mathcal{U}$ be a repair. If $\mathcal{U}$ is a node in the tree, then $\mathcal{U}$ is a leaf.
We show that we can find a branch $\mathcal{U}_0 = \emptyset, \mathcal{U}_1, \ldots, \mathcal{U}_n$ in the tree such that $\mathcal{U}_i \subseteq \mathcal{U}$ and $\mathcal{U}_n = \mathcal{U}$.
If $\mathcal{U}_i$ is a weak repair, then $\mathcal{U}_i = \mathcal{U}$, otherwise $\mathcal{U}$ is not a repair.
Assume that is not the case. Then $\mathcal{I} \circ \mathcal{U}_i \not\models r$ for some rule r. If the body of r does not contain literals fixed by update actions in $\mathcal{U} \setminus \mathcal{U}_i$, then $\mathcal{I} \circ \mathcal{U} \not\models r$, which is absurd. Therefore there is a descendant $\mathcal{U}_{i+1}$ of $\mathcal{U}_i$ such that $\mathcal{U}_{i+1} \subseteq \mathcal{U}$.
Since $\mathcal{U}$ is finite, this construction must end at $\mathcal{U}$. $\square$

Optimizations

Optimizations

The repair tree can be significantly pruned, e.g. by identifying nodes that correspond to the same set of actions.

Optimizations

The repair tree can be significantly pruned, e.g. by identifying nodes that correspond to the same set of actions.

Inconsistent nodes may also be left out.

Outline

Founded repairs I

The notion of repair ignores the head of the AIC.

Founded repairs I

The notion of repair ignores the head of the AIC.

Definition

An action α is *founded* w.r.t. I, η and $\mathcal{U}$ if there is a rule r such that:

Founded repairs I

The notion of repair ignores the head of the AIC.

Definition

An action α is *founded* w.r.t. I, η and $\mathcal{U}$ if there is a rule r such that:

- α occurs in the head of r

Founded repairs I

The notion of repair ignores the head of the AIC.

Definition

An action α is *founded* w.r.t. I, η and $\mathcal{U}$ if there is a rule r such that:

- α occurs in the head of r
- $\mathcal{I} \circ \mathcal{U}$ satisfies all literals in the body of r except for α^D

Founded repairs I

The notion of repair ignores the head of the AIC.

Definition

An action α is *founded* w.r.t. I, η and $\mathcal{U}$ if there is a rule r such that:

- α occurs in the head of r
- $\mathcal{I} \circ \mathcal{U}$ satisfies all literals in the body of r except for α^D

Definition

A set $\mathcal{U}$ is *founded* w.r.t. I and η if every update action in $\mathcal{U}$ is founded w.r.t. I, η and $\mathcal{U}$.

Founded repairs II

In other words, if $\mathcal{U}$ is founded, then removing an action from $\mathcal{U}$ causes some AIC to be violated.

Founded repairs II

In other words, if $\mathcal{U}$ is founded, then removing an action from $\mathcal{U}$ causes some AIC to be violated.

Definition

A founded (weak) repair is a (weak) repair that is founded.

Founded repairs II

In other words, if $\mathcal{U}$ is founded, then removing an action from $\mathcal{U}$ causes some AIC to be violated.

Definition

A founded (weak) repair is a (weak) repair that is founded.

Catch

A founded repair is *not* a minimal founded weak repair.

Computing founded repairs

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Following the heads of the violated rules, we obtain:

Computing founded repairs

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Following the heads of the violated rules, we obtain:

$$\emptyset$$

Computing founded repairs

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$

$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Following the heads of the violated rules, we obtain:

$$\begin{tikzcd} & \emptyset \arrow[dl, "r_3"'] \\ \{+c\} & \end{tikzcd}$$

Computing founded repairs

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad\qquad r_3 : a, \text{not } c \supset +c$$

$$r_2 : \text{not } a, b \supset -b \qquad\qquad r_4 : b, \text{not } c \supset +c$$

Following the heads of the violated rules, we obtain:

$$\begin{tikzcd}
 & \emptyset \arrow[dl, "r_3"'] \arrow[dr, "r_4"] & \\
\{+c\} & & \{+c\}
\end{tikzcd}$$

Computing founded repairs

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Following the heads of the violated rules, we obtain:

$\{+c\}$ is a founded repair.

Computing founded repairs

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Following the heads of the violated rules, we obtain:

$$\{+c\} \text{ is a founded repair.}$$

But so is $\{-a, -b\}\ldots$

Computing founded repairs

The problematic rules

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$

$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Computing founded repairs

The problematic rules

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$

$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

The problem is that in $\{-a, -b\}$ we have a *circularity of support*.

Computing founded repairs

The problematic rules

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$

$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

The problem is that in $\{-a, -b\}$ we have a *circularity of support*.

With this in mind, the Caroprese et al. introduced *justified repairs*.

Justified repairs

Definition

- $\mathcal{U}$ is *closed* under r if $\mathcal{U}$ contains an action in the head of r whenever $\mathcal{U}$ satisfies the non-updatable literals in r.

Justified repairs

Definition

- $\mathcal{U}$ is *closed* under r if $\mathcal{U}$ contains an action in the head of r whenever $\mathcal{U}$ satisfies the non-updatable literals in r.
- An action is a *no-effect action* w.r.t. $\mathcal{I}$ and $\mathcal{U}$ if it does not change $\mathcal{I}$ or $\mathcal{I} \circ \mathcal{U}$.

Justified repairs

Definition

- $\mathcal{U}$ is *closed* under r if $\mathcal{U}$ contains an action in the head of r whenever $\mathcal{U}$ satisfies the non-updatable literals in r.
- An action is a *no-effect action* w.r.t. $\mathcal{I}$ and $\mathcal{U}$ if it does not change $\mathcal{I}$ or $\mathcal{I} \circ \mathcal{U}$.
- $\mathcal{U}$ is a justified action set w.r.t. $\mathcal{I}$ and η if $\mathcal{U}$ is a minimal set of update actions containing all no-effect actions w.r.t. $\mathcal{I}$ and $\mathcal{U}$ and closed under η.

Justified repairs

Definition

- $\mathcal{U}$ is *closed* under r if $\mathcal{U}$ contains an action in the head of r whenever $\mathcal{U}$ satisfies the non-updatable literals in r.
- An action is a *no-effect action* w.r.t. $\mathcal{I}$ and $\mathcal{U}$ if it does not change $\mathcal{I}$ or $\mathcal{I} \circ \mathcal{U}$.
- $\mathcal{U}$ is a justified action set w.r.t. $\mathcal{I}$ and η if $\mathcal{U}$ is a minimal set of update actions containing all no-effect actions w.r.t. $\mathcal{I}$ and $\mathcal{U}$ and closed under η.
- $\mathcal{U}$ is a justified weak repair for $\mathcal{I}$ and η if $\mathcal{U} \cup \mathrm{ne}(\mathcal{I},\mathcal{U})$ is a justified action set.

Justified repairs

Definition

- $\mathcal{U}$ is *closed* under r if $\mathcal{U}$ contains an action in the head of r whenever $\mathcal{U}$ satisfies the non-updatable literals in r.
- An action is a *no-effect action* w.r.t. $\mathcal{I}$ and $\mathcal{U}$ if it does not change $\mathcal{I}$ or $\mathcal{I} \circ \mathcal{U}$.
- $\mathcal{U}$ is a justified action set w.r.t. $\mathcal{I}$ and η if $\mathcal{U}$ is a minimal set of update actions containing all no-effect actions w.r.t. $\mathcal{I}$ and $\mathcal{U}$ and closed under η.
- $\mathcal{U}$ is a justified weak repair for $\mathcal{I}$ and η if $\mathcal{U} \cup \mathrm{ne}(\mathcal{I}, \mathcal{U})$ is a justified action set.

Why?

Not even intuitive. . .

Eliminates some stuff. . .

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$

$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Not even intuitive...

Eliminates some stuff...

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Now $\{-a, -b\}$ is a founded repair that is not justified.

Not even intuitive...

Eliminates some stuff...

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Now $\{-a, -b\}$ is a founded repair that is not justified.

...but actually too much.

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Not even intuitive...

Eliminates some stuff...

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, \text{not } b \supset -a \qquad r_3 : a, \text{not } c \supset +c$$
$$r_2 : \text{not } a, b \supset -b \qquad r_4 : b, \text{not } c \supset +c$$

Now $\{-a, -b\}$ is a founded repair that is not justified.

...but actually too much.

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Again $\{-a, -b\}$ is a founded repair that is not justified.

Justified repair trees I

We can adapt the repair tree as follows.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.
- For each node n and rule r, if $\mathcal{I} \circ \mathcal{U}_n \not\models r$, then each α in the head of r yields a descendant n' of n with:

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.
- For each node n and rule r, if $\mathcal{I} \circ \mathcal{U}_n \not\models r$, then each α in the head of r yields a descendant n' of n with:
  - $\mathcal{U}_{n'} = \mathcal{U}_n \cup \{\alpha\}$;

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.
- For each node n and rule r, if $\mathcal{I} \circ \mathcal{U}_n \not\models r$, then each α in the head of r yields a descendant n' of n with:
  - $\mathcal{U}_{n'} = \mathcal{U}_n \cup \{\alpha\}$;
  - $\mathcal{J}_{n'} = (\mathcal{J}_n \cup \{\mathrm{nup}(r)\}) \setminus \mathcal{U}_n$.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.
- For each node n and rule r, if $\mathcal{I} \circ \mathcal{U}_n \not\models r$, then each α in the head of r yields a descendant n' of n with:
  - $\mathcal{U}_{n'} = \mathcal{U}_n \cup \{\alpha\}$;
  - $\mathcal{J}_{n'} = (\mathcal{J}_n \cup \{\text{nup}(r)\}) \setminus \mathcal{U}_n$.
  - If $\mathcal{U}_{n'}$ is inconsistent, then n' is removed.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.
- For each node n and rule r, if $\mathcal{I} \circ \mathcal{U}_n \not\models r$, then each α in the head of r yields a descendant n' of n with:
  - $\mathcal{U}_{n'} = \mathcal{U}_n \cup \{\alpha\}$;
  - $\mathcal{J}_{n'} = (\mathcal{J}_n \cup \{\mathrm{nup}(r)\}) \setminus \mathcal{U}_n$.
  - If $\mathcal{U}_{n'}$ is inconsistent, then n' is removed.
  - If $\mathcal{U}_{n'} \cap (\mathcal{J}_{n'})^D \neq \emptyset$, then n' is removed.

Justified repair trees I

We can adapt the repair tree as follows.

Modified tree algorithm

- Each node n is a pair of sets of repair actions $\mathcal{U}_n, \mathcal{J}_n$.
- The root of the tree is $\emptyset, \emptyset$.
- The edges are labeled as before.
- For each node n and rule r, if $\mathcal{I} \circ \mathcal{U}_n \not\models r$, then each α in the head of r yields a descendant n' of n with:
  - $\mathcal{U}_{n'} = \mathcal{U}_n \cup \{\alpha\}$;
  - $\mathcal{J}_{n'} = (\mathcal{J}_n \cup \{\mathrm{nup}(r)\}) \setminus \mathcal{U}_n$.
  - If $\mathcal{U}_{n'}$ is inconsistent, then n' is removed.
  - If $\mathcal{U}_{n'} \cap (\mathcal{J}_{n'})^D \neq \emptyset$, then n' is removed.

Motivation

Keep track of the non-updatable atoms in the rules that were used.

Justified repair trees II

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Justified repair trees II

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Corresponding justified repair tree:

Justified repair trees II

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Corresponding justified repair tree:

$$\emptyset, \emptyset$$

Justified repair trees II

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Corresponding justified repair tree:

$$\begin{array}{c} \emptyset, \emptyset \\ \downarrow r_1 \\ \{-a\}, \{+b\} \end{array}$$

Justified repair trees II

Example

Take $\mathcal{I} = \{a, b\}$ and

$$r_1 : a, b \supset -a \qquad r_2 : a, \text{not } b \supset -a \qquad r_3 : \text{not } a, b \supset -b$$

Corresponding justified repair tree:

$$\begin{array}{c} \emptyset, \emptyset \\ \big\downarrow r_1 \\ \{-a\}, \{+b\} \\ \big\downarrow r_3 \\ \{-a, \underline{-b}\}, \{\underline{+b}\} \\ \times \end{array}$$

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Catch

Deciding whether there is a justified repair for $\mathcal{I}$ and η is Σ^2_P-complete.

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Catch

Deciding whether there is a justified repair for $\mathcal{I}$ and η is Σ^2_P-complete (NP-complete in the presence of an NP-complete oracle).

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Catch

Deciding whether there is a justified repair for $\mathcal{I}$ and η is Σ_P^2-complete. Therefore sometimes there must be nodes in the tree that do not correspond to justified (weak) repairs.

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Catch

Deciding whether there is a justified repair for $\mathcal{I}$ and η is Σ_P^2-complete. Therefore sometimes there must† be nodes in the tree that do not correspond to justified (weak) repairs.

† at least if you believe that NP $\neq \Sigma_P^2$.

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Catch

Deciding whether there is a justified repair for $\mathcal{I}$ and η is Σ_P^2-complete. Therefore sometimes there must† be nodes in the tree that do not correspond to justified (weak) repairs.

† at least if you believe that NP $\neq \Sigma_P^2$. (We do.)

Justified repair trees III

Theorem

Let $\mathcal{U}$ be a justified repair for $\mathcal{I}$ and η. Then there is a leaf n in the justified repair tree for $\mathcal{I}$ and η such that $\mathcal{U}_n = \mathcal{U}$.

Catch

Deciding whether there is a justified repair for $\mathcal{I}$ and η is Σ_P^2-complete. Therefore sometimes there must† be nodes in the tree that do not correspond to justified (weak) repairs.

† at least if you believe that NP $\neq \Sigma_P^2$. (We do.)
Checking that a repair is justified is again NP-complete, so our algorithm is (asymptotically) optimal.

Normalized AICs

Definition

An AIC is *normalized* if its head only contains one action.

Normalized AICs

Definition

An AIC is *normalized* if its head only contains one action.

Lemma (Caroprese et al.)

If η is a set of normalized AICs, then existence of justified repairs for $\mathcal{I}$ and η is NP-complete.

Normalized AICs

Definition

An AIC is *normalized* if its head only contains one action.

Lemma (Caroprese et al.)

If η is a set of normalized AICs, then existence of justified repairs for $\mathcal{I}$ and η is NP-complete.

Theorem

If η is a set of normalized AICs, then every leaf of the justified repair tree for $\mathcal{I}$ and η corresponds to a justified repair for $\mathcal{I}$ and η.

Normalized AICs

Definition

An AIC is *normalized* if its head only contains one action.

Lemma (Caroprese et al.)

If η is a set of normalized AICs, then existence of justified repairs for $\mathcal{I}$ and η is NP-complete.

Theorem

If η is a set of normalized AICs, then every leaf of the justified repair tree for $\mathcal{I}$ and η corresponds to a justified repair for $\mathcal{I}$ and η.

We also believe that our algorithm is nicer than the one given by Caroprese et al.

Normalized AICs

Definition

An AIC is *normalized* if its head only contains one action.

Lemma (Caroprese et al.)

If η is a set of normalized AICs, then existence of justified repairs for $\mathcal{I}$ and η is NP-complete.

Theorem

If η is a set of normalized AICs, then every leaf of the justified repair tree for $\mathcal{I}$ and η corresponds to a justified repair for $\mathcal{I}$ and η.

We also believe that our algorithm is nicer than the one given by Caroprese et al. It was developed following the intuitive semantics of AICs.

Outline

Optimizing the algorithms

Optimizing the algorithms

The complexity of these algorithms depends mainly on the size of η.

Optimizing the algorithms

The complexity of these algorithms depends mainly on the size of η.

Splitting η in independent components can help parallelizing the search for repairs.

Optimizing the algorithms

The complexity of these algorithms depends mainly on the size of η.

Splitting η in independent components can help parallelizing the search for repairs.

If one is interested in finding only one (justified) repair, search techniques can speed up the process.

Optimizing the algorithms

The complexity of these algorithms depends mainly on the size of η.

Splitting η in independent components can help parallelizing the search for repairs.

If one is interested in finding only one (justified) repair, search techniques can speed up the process.

Next step

Implementing, testing and improving these algorithms

Outside the database world

Outside the database world

We are interested in more general knowledge bases.

Outside the database world

We are interested in more general knowledge bases, e.g. description logic knowledge bases; OWL ontologies.

Outside the database world

We are interested in more general knowledge bases, e.g. description logic knowledge bases; OWL ontologies.

Definition

A *generalized AIC* over some logic $\mathcal{L}$ is a rule $\varphi \supset \alpha$ where:

Outside the database world

We are interested in more general knowledge bases, e.g. description logic knowledge bases; OWL ontologies.

Definition

A *generalized AIC* over some logic $\mathcal{L}$ is a rule $\varphi \supset \alpha$ where:

- φ is a decidable condition over $\mathcal{L}$

Outside the database world

We are interested in more general knowledge bases, e.g. description logic knowledge bases; OWL ontologies.

Definition

A *generalized AIC* over some logic $\mathcal{L}$ is a rule $\varphi \supset \alpha$ where:

- φ is a decidable condition over $\mathcal{L}$
- α is an action "fixing" φ (i.e. executing α guarantees that φ does not hold)

Outside the database world

We are interested in more general knowledge bases, e.g. description logic knowledge bases; OWL ontologies.

Definition

A *generalized AIC* over some logic $\mathcal{L}$ is a rule $\varphi \supset \alpha$ where:

- φ is a decidable condition over $\mathcal{L}$
- α is an action "fixing" φ (i.e. executing α guarantees that φ does not hold)

This is no longer a purely syntactical notion.

Outside the database world

We are interested in more general knowledge bases, e.g. description logic knowledge bases; OWL ontologies.

Definition

A *generalized AIC* over some logic $\mathcal{L}$ is a rule $\varphi \supset \alpha$ where:

- φ is a decidable condition over $\mathcal{L}$
- α is an action "fixing" φ (i.e. executing α guarantees that φ does not hold)

This is no longer a purely syntactical notion.

Justified repairs make no sense in this setting. . .

Well-founded repair trees I

Definition

In the database/AIC setting, the *well-founded repair tree* for $\mathcal{I}$ and η is built like the justified repair tree, but omitting the sets $\mathcal{J}_n$.

Well-founded repair trees I

Definition

In the database/AIC setting, the *well-founded repair tree* for $\mathcal{I}$ and η is built like the justified repair tree, but omitting the sets $\mathcal{J}_n$.

Definition (Equivalent)

In the database/AIC setting, the *well-founded repair tree* for $\mathcal{I}$ and η is built like the repair tree, but using only the actions in the head of rules.

Well-founded repair trees I

Definition

In the database/AIC setting, the *well-founded repair tree* for $\mathcal{I}$ and η is built like the justified repair tree, but omitting the sets $\mathcal{J}_n$.

Definition (Equivalent)

In the database/AIC setting, the *well-founded repair tree* for $\mathcal{I}$ and η is built like the repair tree, but using only the actions in the head of rules.

New notion of repair

This tree computes all justified repairs and some founded (weak) repairs, but eliminates true circularity of support.

Well-founded repair trees II

Well-founded repair trees generalize nicely to outside the database world.

Well-founded repair trees II

Well-founded repair trees generalize nicely to outside the database world.

Next steps

1. Formalize the notion of generalized AIC and study it in particular settings.

Well-founded repair trees II

Well-founded repair trees generalize nicely to outside the database world.

Next steps

1. Formalize the notion of generalized AIC and study it in particular settings.
2. Characterize the (weak) repairs computed by the well-founded repair tree in this new setting.

Outline

What we achieved. . .

What we achieved. . .

- Operational semantics for active integrity constraints

What we achieved. . .

- Operational semantics for active integrity constraints
- Tree algorithms for repairs and justified repairs

What we achieved. . .

- Operational semantics for active integrity constraints
- Tree algorithms for repairs and justified repairs
- More fine-grained distinction between founded and justified repairs

. . . and what we still hope to do

. . . and what we still hope to do

- Optimization of the tree algorithms, through parallelization

. . . and what we still hope to do

- Optimization of the tree algorithms, through parallelization
- Generalizations outside the database world

Thank you.