Declarative Belief Set Merging using Merging Plans

Christoph Redl   Thomas Eiter   Thomas Krennwallner

{redl,eiter,tkren}@kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Knowledge-Based Systems Group

January 24, 2011

Outline

Outline

Motivation

Usage of Multiple Knowledge Bases

- No single point of truth
- Combining knowledge from different sources into a coherent view
- Possibly heterogeneous knowledge bases
- Contents may be contradicting

Motivation

Usage of Multiple Knowledge Bases

- No single point of truth
- Combining knowledge from different sources into a coherent view
- Possibly heterogeneous knowledge bases
- Contents may be contradicting

Examples

- Judgment aggregation (discussed later)
- Merging of decision diagrams

Outline

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)
- *Set of all beliefs*, i.e., all literals: Lit_Σ

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)
- *Set of all beliefs*, i.e., all literals: Lit_Σ
- A belief set is a set $B \subseteq Lit_\Sigma$

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)
- *Set of all beliefs*, i.e., all literals: Lit_Σ
- A belief set is a set $B \subseteq Lit_\Sigma$
- *Set of all belief sets* $\mathcal{A}(\Sigma)$ over Σ: $\mathcal{A}(\Sigma) := 2^{Lit_\Sigma}$

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)
- *Set of all beliefs*, i.e., all literals: Lit_Σ
- A belief set is a set $B \subseteq Lit_\Sigma$
- *Set of all belief sets* $\mathcal{A}(\Sigma)$ over Σ: $\mathcal{A}(\Sigma) := 2^{Lit_\Sigma}$
- A collection of belief sets is a set $\mathcal{B} \subseteq \mathcal{A}(\Sigma)$

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)
- *Set of all beliefs*, i.e., all literals: Lit_Σ
- A belief set is a set $B \subseteq Lit_\Sigma$
- *Set of all belief sets* $\mathcal{A}(\Sigma)$ over Σ: $\mathcal{A}(\Sigma) := 2^{Lit_\Sigma}$
- A collection of belief sets is a set $\mathcal{B} \subseteq \mathcal{A}(\Sigma)$

Definition (Knowledge Bases)

- We abstract from a concrete language for knowledge bases KB

Belief Sets and Knowledge Bases

Definition (Collections of Belief Sets)

- Belief: atomic formula or a negated atomic formula
  Signature $\Sigma = (\Sigma_c, \Sigma_p)$
  (Σ_c ... constant symbols, Σ_p ... predicate symbols)
- *Set of all beliefs*, i.e., all literals: Lit_Σ
- A belief set is a set $B \subseteq Lit_\Sigma$
- *Set of all belief sets* $\mathcal{A}(\Sigma)$ over Σ: $\mathcal{A}(\Sigma) := 2^{Lit_\Sigma}$
- A collection of belief sets is a set $\mathcal{B} \subseteq \mathcal{A}(\Sigma)$

Definition (Knowledge Bases)

- We abstract from a concrete language for knowledge bases KB
- Knowledge bases are identified with assigned collections of belief sets (their "semantics"): $BS(KB) \subseteq \mathcal{A}(\Sigma)$

Belief Sets and Knowledge Bases

Example

$$KB = \{dog(sue) \vee cat(sue), female(sue)\}$$

Associated collections of belief sets depend on the semantics, e.g.,:

Belief Sets and Knowledge Bases

Example

$$KB = \{dog(sue) \vee cat(sue), female(sue)\}$$

Associated collections of belief sets depend on the semantics, e.g.,:

- Minimal Herbrand models:
  $BS(KB) =$
  $\{ \{dog(sue), \neg cat(sue), female(sue)\}, \{\neg dog(sue), cat(sue), female(sue)\} \}$

Belief Sets and Knowledge Bases

Example

$$KB = \{dog(sue) \vee cat(sue), female(sue)\}$$

Associated collections of belief sets depend on the semantics, e.g.,:

- Minimal Herbrand models:
  $BS(KB) =$
  $\{ \{dog(sue), \neg cat(sue), female(sue)\}, \{\neg dog(sue), cat(sue), female(sue)\} \}$
- Classically entailed literals:
  $BS(KB) = \{ \{female(sue)\} \}$

Merging Task

Collection of Knowledge Bases

- Collection of knowledge bases: $KB = KB_1, \ldots, KB_n$
- Associated collections of belief sets: $BS(KB_1), \ldots, BS(KB_n)$
- Task: Integrate them into a single set of belief sets

Merging Task

Collection of Knowledge Bases

- Collection of knowledge bases: $KB = KB_1, \ldots, KB_n$
- Associated collections of belief sets: $BS(KB_1), \ldots, BS(KB_n)$
- Task: Integrate them into a single set of belief sets

Types of Mismatches

- Naive union not always possible
- Mismatches:

Merging Task

Collection of Knowledge Bases

- Collection of knowledge bases: $KB = KB_1, \ldots, KB_n$
- Associated collections of belief sets: $BS(KB_1), \ldots, BS(KB_n)$
- Task: Integrate them into a single set of belief sets

Types of Mismatches

- Naive union not always possible
- Mismatches:
  - language (syntactic) incompatibilities

Merging Task

Collection of Knowledge Bases

- Collection of knowledge bases: $KB = KB_1, \ldots, KB_n$
- Associated collections of belief sets: $BS(KB_1), \ldots, BS(KB_n)$
- Task: Integrate them into a single set of belief sets

Types of Mismatches

- Naive union not always possible
- Mismatches:
  - language (syntactic) incompatibilities
  - logical inconsistencies

Mismatch 1: Language Incompatibilities

The Problem

- Different sources may use different vocabularies
- Syntactically equal beliefs may encode different information (homonyms)
- Syntactically different beliefs may encode the same information (synonyms)

Mismatch 1: Language Incompatibilities

The Problem

- Different sources may use different vocabularies
- Syntactically equal beliefs may encode different information (homonyms)
- Syntactically different beliefs may encode the same information (synonyms)
- Example:
  $P_1 = \{degree(john, \text{“}MSc\text{”}) \leftarrow\}$ vs. $P_2 = \{deg(john, \text{“}Master\ of\ Science\text{”}) \leftarrow\}$

Mismatch 1: Language Incompatibilities

The Problem

- Different sources may use different vocabularies
- Syntactically equal beliefs may encode different information (homonyms)
- Syntactically different beliefs may encode the same information (synonyms)
- Example:
  $P_1 = \{degree(john, \text{``}MSc\text{''}) \leftarrow\}$ vs. $P_2 = \{deg(john, \text{``}Master\ of\ Science\text{''}) \leftarrow\}$

The Solution

- *Common signature*: $\Sigma^C = (\Sigma_c^C, \Sigma_p^C)$

Mismatch 1: Language Incompatibilities

The Problem

- Different sources may use different vocabularies
- Syntactically equal beliefs may encode different information (homonyms)
- Syntactically different beliefs may encode the same information (synonyms)
- Example:
  $P_1 = \{degree(john, \text{“}MSc\text{”}) \leftarrow\}$ vs. $P_2 = \{deg(john, \text{“}Master\ of\ Science\text{”}) \leftarrow\}$

The Solution

- *Common signature*: $\Sigma^C = (\Sigma_c^C, \Sigma_p^C)$
- Convert the collection of belief sets $\mathcal{B}_i = BS(KB_i)$ to a new collection over Σ^C:
  $$\mathcal{B}_i' = \mu_i(\mathcal{B}_i)$$

Mismatch 1: Language Incompatibilities

The Problem

- Different sources may use different vocabularies
- Syntactically equal beliefs may encode different information (homonyms)
- Syntactically different beliefs may encode the same information (synonyms)
- Example:
  $P_1 = \{degree(john, \text{“}MSc\text{”}) \leftarrow\}$ vs. $P_2 = \{deg(john, \text{“}Master\ of\ Science\text{”}) \leftarrow\}$

The Solution

- *Common signature*: $\Sigma^C = (\Sigma^C_c, \Sigma^C_p)$
- Convert the collection of belief sets $\mathcal{B}_i = BS(KB_i)$ to a new collection over Σ^C:
  $$\mathcal{B}'_i = \mu_i(\mathcal{B}_i)$$

Formally: A belief set conversion is a function

$$\mu_i : 2^{\mathcal{A}(\Sigma^{KB_i})} \rightarrow 2^{\mathcal{A}(\Sigma^C)}, 1 \leq i \leq n$$

s.t. $\mathcal{B}'_i = \mathcal{B}'_j$ iff they are considered to represent the same information

Mismatch 1: Language Incompatibilities

Example (continued)

$$\mu_1(\mathcal{B}) = \mathcal{B},$$
$$\mu_2(\mathcal{B}) = \{\{degree(X, \text{``}MSc\text{''}) \mid deg(X, \text{``}Master\ of\ Science\text{''}) \in B\} \cup \{degree(X, Y) \mid deg(X, Y) \in B, Y \neq \text{``}Master\ of\ Science\text{''}\} \mid B \in \mathcal{B}\};$$

Mismatch 2: Logical Inconsistencies

Definition (Integrity Constraints)

Application-dependent integrity constraints are abstractly modeled as

$$\mathcal{C} \subseteq 2^{\mathcal{A}(\Sigma^C)},$$

s.t. $\mathcal{B} \subseteq \mathcal{A}(\Sigma^C)$ satisfies the constraints iff $\mathcal{B} \in \mathcal{C}$

Mismatch 2: Logical Inconsistencies

Definition (Integrity Constraints)

Application-dependent integrity constraints are abstractly modeled as

$$\mathcal{C} \subseteq 2^{\mathcal{A}(\Sigma^C)},$$

s.t. $\mathcal{B} \subseteq \mathcal{A}(\Sigma^C)$ satisfies the constraints iff $\mathcal{B} \in \mathcal{C}$

The Problem

- We assume: Each source satisfies the constraints: $\mathcal{B}_i \in \mathcal{C}$ *for all* $1 \leq i \leq n$

Mismatch 2: Logical Inconsistencies

Definition (Integrity Constraints)

Application-dependent integrity constraints are abstractly modeled as

$$\mathcal{C} \subseteq 2^{\mathcal{A}(\Sigma^C)},$$

s.t. $\mathcal{B} \subseteq \mathcal{A}(\Sigma^C)$ satisfies the constraints iff $\mathcal{B} \in \mathcal{C}$

The Problem

- We assume: Each source satisfies the constraints: $\mathcal{B}_i \in \mathcal{C}$ *for all* $1 \leq i \leq n$
- But the union may violate them: $\bigcup_{i=1}^{n} \mathcal{B}_i \notin \mathcal{C}$

Mismatch 2: Logical Inconsistencies

Definition (Integrity Constraints)

Application-dependent integrity constraints are abstractly modeled as

$$\mathcal{C} \subseteq 2^{\mathcal{A}(\Sigma^C)},$$

s.t. $\mathcal{B} \subseteq \mathcal{A}(\Sigma^C)$ satisfies the constraints iff $\mathcal{B} \in \mathcal{C}$

The Problem

- We assume: Each source satisfies the constraints: $\mathcal{B}_i \in \mathcal{C}$ *for all* $1 \leq i \leq n$
- But the union may violate them: $\bigcup_{i=1}^{n} \mathcal{B}_i \notin \mathcal{C}$

The Solution

- We introduce merging operators

$$\circ^{n,m} : \underbrace{\left(2^{\mathcal{A}(\Sigma^C)}\right)^n}_{collections\ of\ belief\ sets} \rightarrow 2^{\mathcal{A}(\Sigma^C)}$$

Mismatch 2: Logical Inconsistencies

Definition (Integrity Constraints)

Application-dependent integrity constraints are abstractly modeled as

$$\mathcal{C} \subseteq 2^{\mathcal{A}(\Sigma^C)},$$

s.t. $\mathcal{B} \subseteq \mathcal{A}(\Sigma^C)$ satisfies the constraints iff $\mathcal{B} \in \mathcal{C}$

The Problem

- We assume: Each source satisfies the constraints: $\mathcal{B}_i \in \mathcal{C} \textit{ for all } 1 \leq i \leq n$
- But the union may violate them: $\bigcup_{i=1}^{n} \mathcal{B}_i \notin \mathcal{C}$

The Solution

- We introduce merging operators
- Maps n collections of belief sets to a new, integrated collection

$$\circ^{n,m} : \underbrace{\left(2^{\mathcal{A}(\Sigma^C)}\right)^n}_{collections\ of\ belief\ sets} \times \underbrace{\mathcal{D}_1 \times \ldots \times \mathcal{D}_m}_{additional\ parameters} \rightarrow 2^{\mathcal{A}(\Sigma^C)}$$

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Integrity constraints: (formally) $\mathcal{C} = \{\mathcal{B} \subseteq \mathcal{A}(\Sigma) \mid \nexists B \in \mathcal{B} : \exists A : \{A, \neg A\} \subseteq B\}$

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Integrity constraints: (formally) $\mathcal{C} = \{\mathcal{B} \subseteq \mathcal{A}(\Sigma) \mid \nexists B \in \mathcal{B} : \exists A : \{A, \neg A\} \subseteq B\}$
Example:

- $\{\{a, b, c\}, \{a, d\}\} \in \mathcal{C}$

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Integrity constraints: (formally) $\mathcal{C} = \{\mathcal{B} \subseteq \mathcal{A}(\Sigma) \mid \nexists B \in \mathcal{B} : \exists A : \{A, \neg A\} \subseteq B\}$
Example:

- $\{\{a, b, c\}, \{a, d\}\} \in \mathcal{C}$
- $\{\{a, b, c\}, \{a, \neg d\}\} \in \mathcal{C}$

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Integrity constraints: (formally) $\mathcal{C} = \{\mathcal{B} \subseteq \mathcal{A}(\Sigma) \mid \nexists B \in \mathcal{B} : \exists A : \{A, \neg A\} \subseteq B\}$
Example:

- $\{\{a, b, c\}, \{a, d\}\} \in \mathcal{C}$
- $\{\{a, b, c\}, \{a, \neg d\}\} \in \mathcal{C}$
- $\{\{a, b, \neg a\}, \{a, \neg d\}\} \notin \mathcal{C}$

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Integrity constraints: (formally) $\mathcal{C} = \{\mathcal{B} \subseteq \mathcal{A}(\Sigma) \mid \nexists B \in \mathcal{B} : \exists A : \{A, \neg A\} \subseteq B\}$
Example:

- $\{\{a, b, c\}, \{a, d\}\} \in \mathcal{C}$
- $\{\{a, b, c\}, \{a, \neg d\}\} \in \mathcal{C}$
- $\{\{a, b, \neg a\}, \{a, \neg d\}\} \notin \mathcal{C}$

Operator definition: (binary, no parameter, i.e., $n = 2, m = 0$)

$$\circ_{\cup}^{2,0}(\mathcal{B}_1, \mathcal{B}_2) = \{B_1 \cup B_2 \mid B_1 \in \mathcal{B}_1, B_2 \in \mathcal{B}_2, \nexists A : \{A, \neg A\} \subseteq (B_1 \cup B_2)\} \ ,$$

Mismatch 2: Logical Inconsistencies

Example: Operator definition

Idea: Union operator preserving consistency under classical semantics

Integrity constraints: (formally) $\mathcal{C} = \{\mathcal{B} \subseteq \mathcal{A}(\Sigma) \mid \nexists B \in \mathcal{B} : \exists A : \{A, \neg A\} \subseteq B\}$
Example:

- $\{\{a, b, c\}, \{a, d\}\} \in \mathcal{C}$
- $\{\{a, b, c\}, \{a, \neg d\}\} \in \mathcal{C}$
- $\{\{a, b, \neg a\}, \{a, \neg d\}\} \notin \mathcal{C}$

Operator definition: (binary, no parameter, i.e., $n = 2, m = 0$)

$$\circ_{\cup}^{2,0}(\mathcal{B}_1, \mathcal{B}_2) = \{B_1 \cup B_2 \mid B_1 \in \mathcal{B}_1, B_2 \in \mathcal{B}_2, \nexists A : \{A, \neg A\} \subseteq (B_1 \cup B_2)\} \quad ,$$

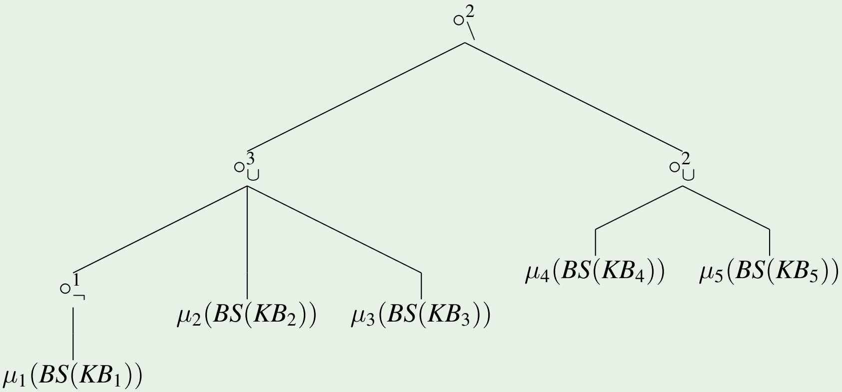
Application:

- $\mathcal{B}_1 = \{\{a, b, c\}, \{\neg a, c\}\}$
- $\mathcal{B}_2 = \{\{\neg a, d\}, \{c, d\}\}$
- $\circ_{\cup}^{2,0}(\mathcal{B}_1, \mathcal{B}_2) = \{ \sout{\{a, b, \neg a, d\}}, \{a, b, c, d\}, \{\neg a, c, d\}, \{\neg a, c, d\}\}$

Merging Plans

Hierarchical arrangement of operators:

Example



Merging Plans

Definition (Merging Plans)

The set $\mathcal{M}_{KB,\Omega}$ of merging plans over knowledge bases $KB = KB_1, \ldots, KB_n$ and a set $\Omega = \{\circ_1, \ldots, \circ_n\}$ of operators is the smallest set such that

(i) each $M \in KB$, called *atomic* merging plan, is in $\mathcal{M}_{KB,\Omega}$;

(ii) if $\circ_i^{n,m} \in \Omega, s_j \in \mathcal{M}_{KB,\Omega}$ and $a_k \in \mathcal{D}_i$ for $1 \leq j \leq n, 1 \leq k \leq m$, then $(\circ_i^{n,m}, s_1, \ldots, s_n, a_1, \ldots, a_m) \in \mathcal{M}_{KB,\Omega}$.

Example (continued)

$$M = (\circ^2_{\setminus},\ (\circ^3_{\cup},\ (\circ^1_{\neg},\ KB_1),\ KB_2,\ KB_3), (\circ^2_{\cup},\ KB_4,\ KB_5)).$$

Merging Task

Definition (Merging Task)

A *merging task* is a quintuple $T = \langle KB, \Sigma^C, \mu, \Omega, M \rangle$

Merging Task

Definition (Merging Task)

A *merging task* is a quintuple $T = \langle KB, \Sigma^C, \mu, \Omega, M\rangle$

Definition (Merging Task Result)

The *result of a merging task* $T = \langle KB, \Sigma^C, \mu, \Omega, M\rangle$, denoted as $[\![T]\!]$, is

$$[\![T]\!] = \begin{cases} [\mu_i(BS(M))]_{\Sigma_p^C}, & \text{if } M \in KB, \\ [\circ^{n,m}([\![T_1]\!], \ldots, [\![T_n]\!], a_1, \ldots, a_m)]_{\Sigma_p^C}, & \text{if } M = (\circ^{n,m}, s_1, \ldots, s_n, a_1, \ldots, a_m), \end{cases}$$

where $[\mathcal{B}]_{\Sigma_p^C} = \{\{p(a_1, \ldots, a_n) \in BS \mid p = (\neg)p', \, p' \in \Sigma_p^C\} \mid BS \in \mathcal{B}\}$ denotes the projection of $\mathcal{B}$ to the atoms over Σ_p^C, and $T_i = \langle KB, \Sigma^C, \mu, \Omega, s_i\rangle$, $1 \leq i \leq n$.

Intuition

- The result of a merging plan will be defined as the collection of belief sets delivered by the topmost operator

Merging Plans

Example (continued)

$$M = (\circ^2_{\setminus},\ (\circ^3_{\cup},\ (\circ^1_{\neg},\ KB_1),\ KB_2,\ KB_3), (\circ^2_{\cup},\ KB_4,\ KB_5)).$$

Let

$KB_1 = \{a.,\ b.\}, KB_2 = \{x.,\ y.\}, KB_3 = \{\neg a.,\ c.\}, KB_4 = \{a.,\ x.\}, KB_5 = \{c.,\ x.,\ y.\}$

under answer-set semantics ($x.$ is an abbreviation for $x \leftarrow .$)

Merging Plans

Example (continued)

$$M = (\circ^2_{\setminus},\ (\circ^3_{\cup},\ (\circ^1_{\neg},\ KB_1),\ KB_2,\ KB_3), (\circ^2_{\cup},\ KB_4,\ KB_5)).$$

Let

$KB_1 = \{a.,\ b.\}, KB_2 = \{x.,\ y.\}, KB_3 = \{\neg a.,\ c.\}, KB_4 = \{a.,\ x.\}, KB_5 = \{c.,\ x.,\ y.\}$

under answer-set semantics ($x.$ is an abbreviation for $x \leftarrow .$)

Evaluation:

$$[\![\langle \{KB_1, \ldots, KB_5\}, \Sigma^C, \mu_{id}, \Omega, M\rangle]\!] =$$

$$\circ^2_{\setminus}\left(\ [\![(\circ^3_{\cup}, (\circ^1_{\neg},\ KB_1),\ KB_2,\ KB_3)]\!],\ [\![(\circ^2_{\cup}, KB_4, KB_5)]\!]\ \right) =$$

$$\circ^2_{\setminus}\left(\ \circ^3_{\cup}([\![(\circ^1_{\neg},\ KB_1)]\!],\ [\![KB_2]\!],\ [\![KB_3]\!]),\ [\![(\circ^2_{\cup}, KB_4, KB_5)]\!]\ \right) =$$

$$\cdots = \circ^2_{\setminus}(\{\{\neg a, \neg b, c, x, y\}\},\ \{\{a, c, x, y\}\}) = \{\{\neg a, \neg b\}\}.$$

($[\![M]\!]$ is an abbreviation for $[\![\{P_1, \ldots, P_5\}, \Sigma^C, \mu_{id}, \Omega, M]\!]$)

Outline

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

$\Rightarrow$ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

$\Rightarrow$ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Realization of the Components

1. **Knowledge bases**: arbitrary source accessible from dlvhex

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

$\Rightarrow$ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Realization of the Components

1. **Knowledge bases**: arbitrary source accessible from dlvhex
2. **Common signature**:
   A set of predicate symbols, constants are given implicitly

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

$\Rightarrow$ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Realization of the Components

1. **Knowledge bases**: arbitrary source accessible from dlvhex
2. **Common signature**:
   A set of predicate symbols, constants are given implicitly
3. **Belief Set Conversion functions**:
   rules under HEX-semantics; query the source (1) in the body; derive atoms over common signature (2) in the head

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

⇒ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Realization of the Components

1. **Knowledge bases**: arbitrary source accessible from dlvhex
2. **Common signature**:
   A set of predicate symbols, constants are given implicitly
3. **Belief Set Conversion functions**:
   rules under HEX-semantics; query the source (1) in the body; derive atoms over common signature (2) in the head
4. **Merging operators**: C++ functions in plugin libaries

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

$\Rightarrow$ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Realization of the Components

1. **Knowledge bases**: arbitrary source accessible from dlvhex
2. **Common signature**:
   A set of predicate symbols, constants are given implicitly
3. **Belief Set Conversion functions**:
   rules under HEX-semantics; query the source (1) in the body; derive atoms over common signature (2) in the head
4. **Merging operators**: C++ functions in plugin libaries
5. **Merging Plan**: Plain text with hierarchical structure

Towards Automated Evaluation

Goal

- Define merging task formally
- Compute its result automatically

$\Rightarrow$ MELD System - **ME**rging **L**ibrary for **D**LVHEX

Realization of the Components

1. **Knowledge bases**: arbitrary source accessible from dlvhex
2. **Common signature**:
   A set of predicate symbols, constants are given implicitly
3. **Belief Set Conversion functions**:
   rules under HEX-semantics; query the source (1) in the body; derive atoms over common signature (2) in the head
4. **Merging operators**: C++ functions in plugin libaries
5. **Merging Plan**: Plain text with hierarchical structure

Merging Task Language

Example: `merging.mt`

```
[common signature]
    predicate: a/0;
    predicate: b/0;
    predicate: c/0;
    predicate: p/1;
    predicate: q/3;
[belief base]
    name:bb1;
    mapping: "some_rule.";        % query external source here
    mapping: "q(X, Y, Z) :- &rdf[...](X, Y, Z).";
[belief base]
    name:bb2;
    source: "some_program.hex";     % or within this program
...
```

Merging Task Language

Example: `merging.mt` (ctn'd)

```
[merging plan]
{   operator: setminus;
      {
         operator: union;
            {
               operator: neg;
                  {bb1};
            };
            {bb2};
            {bb3};
      };
      {
      operator: union;
         {bb4};
         {bb5};
      };
}
```

Advantages of the Framework

Support for Prototyping Applications

- Reuse merging operators once

Advantages of the Framework

Support for Prototyping Applications

- Reuse merging operators once
- Rapid prototyping of applications

Advantages of the Framework

Support for Prototyping Applications

- Reuse merging operators once
- Rapid prototyping of applications
- Quick restructuring of merging plans, exchange of operators, parameter modification

Advantages of the Framework

Support for Prototyping Applications

- Reuse merging operators once
- Rapid prototyping of applications
- Quick restructuring of merging plans, exchange of operators, parameter modification
- Automatic recomputation of result

Advantages of the Framework

Support for Prototyping Applications

- Reuse merging operators once
- Rapid prototyping of applications
- Quick restructuring of merging plans, exchange of operators, parameter modification
- Automatic recomputation of result
- Experimenting with different merging plans

Outline

Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

- Idea: Integrated collection of belief sets should be similar to the sources

Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

- Idea: Integrated collection of belief sets should be similar to the sources
- Distance function: Compare collections of belief sets, e.g., Hamming distance

Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

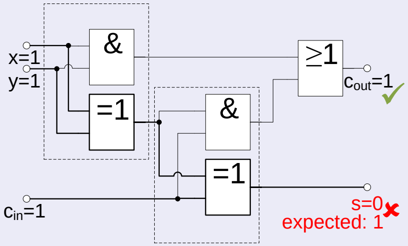
- Idea: Integrated collection of belief sets should be similar to the sources
- Distance function: Compare collections of belief sets, e.g., Hamming distance
- Operator: Merged collection has minimal distance to sources

Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

- Idea: Integrated collection of belief sets should be similar to the sources
- Distance function: Compare collections of belief sets, e.g., Hamming distance
- Operator: Merged collection has minimal distance to sources

Fault diagnosis

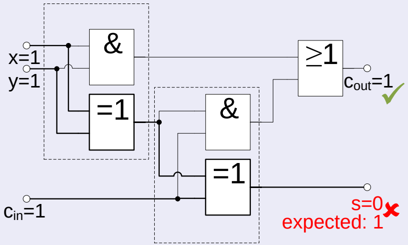


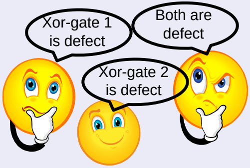
Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

- Idea: Integrated collection of belief sets should be similar to the sources
- Distance function: Compare collections of belief sets, e.g., Hamming distance
- Operator: Merged collection has minimal distance to sources

Fault diagnosis





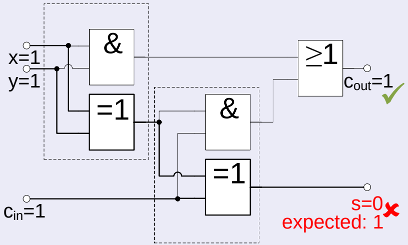
Goal: Find a group decision

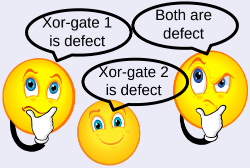
Judgment Aggregation

Distance-based Merging Operators [Gabbay et al., 2009]

- Idea: Integrated collection of belief sets should be similar to the sources
- Distance function: Compare collections of belief sets, e.g., Hamming distance
- Operator: Merged collection has minimal distance to sources

Fault diagnosis





Goal: Find a group decision s.t.

- it is still be an explanations
- it is *similar* to individual opinions

Outline

Conclusion

- Approach for merging of several collections of belief sets:
  1. Belief set conversion functions
  2. hierarchical merging plans with merging operators

Conclusion

- Approach for merging of several collections of belief sets:
  1. Belief set conversion functions
  2. hierarchical merging plans with merging operators
- Prototype implementation: MELD as a plugin for dlvhex

Conclusion

- Approach for merging of several collections of belief sets:
  1. Belief set conversion functions
  2. hierarchical merging plans with merging operators
- Prototype implementation: MELD as a plugin for dlvhex
- **Applications:** Judgment Aggregation, Merging of Decision Diagrams, ...

Conclusion

- Approach for merging of several collections of belief sets:
  1. Belief set conversion functions
  2. hierarchical merging plans with merging operators
- Prototype implementation: MELD as a plugin for dlvhex
- **Applications:** Judgment Aggregation, Merging of Decision Diagrams, ...
- **URL:** `http://www.kr.tuwien.ac.at/research/dlvhex/meld.html`

Conclusion

- Approach for merging of several collections of belief sets:
  1. Belief set conversion functions
  2. hierarchical merging plans with merging operators
- Prototype implementation: MELD as a plugin for dlvhex
- **Applications:** Judgment Aggregation, Merging of Decision Diagrams, ...
- **URL:** `http://www.kr.tuwien.ac.at/research/dlvhex/meld.html`

Advantages

- Reusing of operators
- Evaluating different operators empirically
- Automatic recomputation of result
- Release user from routine tasks

References

Dov M. Gabbay, Odinaldo Rodrigues, Gabriella Pigozzi
Connections between Belief Revision, Belief Merging and Social Choice
In: Journal of Logic and Computation **19**(3) (2009)

Konieczny, S., Pérez, R.P.:
On the logic of merging.
In: KR'98. (1998) 488–498

Redl, C.:
Development of a belief merging framework for dlvhex.
Master's thesis, Vienna University of Technology (June 2010)
http://www.ub.tuwien.ac.at/dipl/2010/AC07808210.pdf

Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.:
dlvhex: A system for integrating multiple semantics in an answer-set programming framework.
In: WLP'06. (2006) 206–210

Salzberg, S., Delcher, A.L., Fasman, K.H., Henderson, J.:
A decision tree system for finding genes in DNA.
Journal of Computational Biology **5**(4) (1998) 667–680