Development of a Belief Merging Framework for dlvhex

Christoph Redl

May 31, 2010
Outline

1. Motivation
2. Reasons for Incompatibility
3. Task Definition
4. Architecture of the Belief Merging Framework
5. Using the Framework - Hands-on
6. Application Scenario
7. Summary
Motivation

Usage of multiple belief bases

- Usually we have more than one knowledge base
Motivation

Usage of multiple belief bases

- Usually we have more than one knowledge base
- In many applications, knowledge sources are often provided by third-parties
Motivation

Usage of multiple belief bases

- Usually we have more than one knowledge base
- In many applications, knowledge sources are often provided by third-parties
- ... but rarely synchronized
Motivation

Usage of multiple belief bases

- Usually we have more than one knowledge base
- In many applications, knowledge sources are often provided by third-parties
- ... but rarely synchronized

Combining the contents

- We do not want to restrict ourselves to one source
Motivation

Usage of multiple belief bases

- Usually we have more than one knowledge base
- In many applications, knowledge sources are often provided by third-parties
- ... but rarely synchronized

Combining the contents

- We do not want to restrict ourselves to one source
- Naive union can introduce contradictions
Motivation

Usage of multiple belief bases

- Usually we have more than one knowledge base
- In many applications, knowledge sources are often provided by third-parties
- ... but rarely synchronized

Combining the contents

- We do not want to restrict ourselves to one source
- Naive union can introduce contradictions
- Many different merging techniques
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
- AGM postulates: “minimal change”
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
- AGM postulates: “minimal change”

Belief Merging

- Aggregation of sources
- No single formula with absolute priority
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
- AGM postulates: “minimal change”

Belief Merging

- Aggregation of sources
- No single formula with absolute priority
- Variants of AGM postulates are reasonable:
  informally, minimize differences between sources and merged belief base
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
- AGM postulates: “minimal change”

Belief Merging

- Aggregation of sources
- No single formula with absolute priority
- Variants of AGM postulates are reasonable:
  informally, minimize differences between sources and merged belief base

Applications

- judgment aggregation
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
- AGM postulates: “minimal change”

Belief Merging

- *Aggregation* of sources
- No single formula with absolute priority
- Variants of AGM postulates are reasonable:
  informally, minimize differences between sources and merged belief base

Applications

- judgment aggregation
- merging of decision diagrams
Motivation

Belief Revision

- Incorporation of knowledge into existing belief base
- AGM postulates: “minimal change”

Belief Merging

- Aggregation of sources
- No single formula with absolute priority
- Variants of AGM postulates are reasonable:
  informally, minimize differences between sources and merged belief base

Applications

- judgment aggregation
- merging of decision diagrams
- fusion of business databases
Types of Incompatibility

Syntactic Incompatibility

- Sources are written in different formalisms
- Preprocessing step needed
Types of Incompatibility

Syntactic Incompatibility

- Sources are written in different formalisms
- Preprocessing step needed

Examples

- relational databases
- object-orientated databases
- RDF ontologies
- logic programs
Types of Incompatibility

Logic Inconsistencies
Union of data sets leads to contradictions:

\[ \bigcup_i KB_i \models \bot \]
Types of Incompatibility

Logic Inconsistencies
Union of data sets leads to contradictions:

$$\bigcup_i KB_i \models \bot$$

likewise

constraint violation: $$\bigcup_i KB_i \not\models C$$
Types of Incompatibility

Logic Inconsistencies
Union of data sets leads to contradictions:

\[ \bigcup_i KB_i \models \bot \]

likewise

constraint violation: \[ \bigcup_i KB_i \not\models C \]

Example
name is the primary key; a constraint forces the height to be unique for each person.

(a) | name | height |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Marge</td>
<td>1,78m</td>
</tr>
<tr>
<td>Homer</td>
<td>1,82m</td>
</tr>
<tr>
<td>Bart</td>
<td>1,67m</td>
</tr>
</tbody>
</table>

(b) | name | height |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Marge</td>
<td>1,78m</td>
</tr>
<tr>
<td>Homer</td>
<td>1,82m</td>
</tr>
<tr>
<td>Bart</td>
<td>1,65m</td>
</tr>
</tbody>
</table>
Data Cleanness

- Remain after logic inconsistencies resolved
- Detection requires advanced algorithms: data cleansing
Data Cleanness

- Remain after logic inconsistencies resolved

- Detection requires advanced algorithms: data cleansing

Undesired artefacts concerning

- Differing naming conventions
  e.g., academic degrees, addresses, ...

- Different entries referring to the same real-world object
Data Cleanness

- Remain after logic inconsistencies resolved
- Detection requires advanced algorithms: data cleansing

Undesired artefacts concerning
- Differing naming conventions
  e.g., academic degrees, addresses, ...
- Different entries referring to the same real-world object

Example
Merging of address tables, one with and one without abbreviations
Logic Programs as Belief Bases

Given

\[ \pi = (P_1, \ldots, P_n) \]

vector of belief bases

Given as \emph{logic programs} with answer sets \( AS(P_i) \)
Logic Programs as Belief Bases

Given

\[ \pi = (P_1, \ldots, P_n) \]

vector of belief bases

Given as *logic programs* with answer sets \( AS(P_i) \)

Answer sets of programs are considered to be the stored knowledge
Logic Programs as Belief Bases

Given
\[ \pi = (P_1, \ldots, P_n) \] vector of belief bases

Given as logic programs with answer sets \( AS(P_i) \)

Answer sets of programs are considered to be the stored knowledge

To define
\[ \Sigma^C = (\Sigma^C_c, \Sigma^C_p) \] common signature
\[ \mu = (\mu_1, \ldots, \mu_n) \] vector of mapping functions
\[ \omega = (\circ_1, \ldots, \circ_m) \] merging operators
\[ R \] merging plan
Common Signature and Mappings

Solve the problem of syntactic incompatibility
Common Signature and Mappings

Solve the problem of syntactic incompatibility

Common Signature

- $\Sigma^C = (\Sigma^C_c, \Sigma^C_p)$
- expressive enough to represent any of sources
Common Signature and Mappings

Solve the problem of syntactic incompatibility

Common Signature

- $\Sigma^C = (\Sigma^C_c, \Sigma^C_p)$
- expressive enough to represent any of sources

Mappings

- Let $\mathcal{A} = 2^{\text{Lit}_{\Sigma^C}}$ (set of potential answer sets over $\Sigma^C$)
Common Signature and Mappings

Solve the problem of syntactic incompatibility

Common Signature

- $\Sigma^C = (\Sigma^C_c, \Sigma^C_p)$
- expressive enough to represent any of sources

Mappings

- Let $\mathcal{A} = 2^{\text{Lit}_\Sigma^C}$ (set of potential answer sets over $\Sigma^C$)
- $\mu_i : \{\text{AS}(P_i)\} \rightarrow 2^\mathcal{A}$
Common Signature and Mappings

Solve the problem of syntactic incompatibility

Common Signature

\[ \Sigma^C = (\Sigma^C_c, \Sigma^C_p) \]

expressive enough to represent any of sources

Mappings

Let \( \mathcal{A} = 2^{\text{Lit}_{\Sigma^C}} \) (set of potential answer sets over \( \Sigma^C \))

\[ \mu_i : \{\text{AS}(P_i)\} \rightarrow 2^\mathcal{A} \]

answer sets stay semantically equivalent!
Merging Operators

Resolve logic inconsistencies
(plus: may perform data cleansing tasks)
Merging Operators

Resolve logic inconsistencies
(plus: may perform data cleansing tasks)

\[ \circ_i^{n,m} : 2^A \times \cdots \times 2^A \times D_1 \times \cdots \times D_m \rightarrow 2^A \]

- \( n \) times
- additional parameters
Merging Operators

Resolve logic inconsistencies
(plus: may perform data cleansing tasks)

\[ \circ_i^{n,m} : 2^A \times \cdots \times 2^A \times D_1 \times \cdots \times D_m \rightarrow 2^A \]

Example for a merging operator
The union operator \( \circ_U^{2,0} \) is defined as follows:

\[ \circ_U^{2,0} : 2^A \times 2^A \rightarrow 2^A \]

\[ \circ_U^{2,0}(SAS_1, SAS_2) = \{AS_1 \cup AS_2|AS_1 \in SAS_1, AS_2 \in SAS_2, AS_1 \cup AS_2 \neq \bot\} \]

(\( \circ_U^2 \) is binary, no additional parameters)
Merging Plans

A merging plan is **hierarchical** and defines

- the order

- of operators

- to be applied on which belief bases
Merging Plans

A merging plan is **hierarchical** and defines

- the order
- of operators
- to be applied on which belief bases

**The result**

- the set of answer sets delivered by the topmost operator
Example merging plan

\[
\text{Example merging plan}
\]

\[
\begin{align*}
\omega_1(\mu_1(P_1)) & \cup \omega_3(\mu_2(P_2) \cup \mu_3(P_3)) & \cup \omega_5(\mu_4(P_4) \cup \mu_5(P_5)) \\
\end{align*}
\]
Merging Input

common signature

merging plan

operator

mapping

belief bases

merging input

1

1

* 

*
Approach

**Steps**

1. designing a merging language
2. implementing the merging plan compiler
3. implementing external atoms (mergingplugin)
Approach

Steps

1. designing a merging language
2. implementing the merging plan compiler
3. implementing external atoms (mergingplugin)

Diagram:
- Merging input → Merging plan compiler → HEX program
- Merging plan compiler translates merging plan into HEX program
- Program makes use of external atoms
- Answer sets = result of the merging plan
- Merging plugin defines external atoms for:
  - Calling of nested HEX programs
  - Calling of merging operators

Diagram:
- Merging input
- Merging plan compiler
- HEX program
- Merging plugin
- Dlvhex
- \{AS_n\} answer sets
Approach

Steps

1. designing a merging language
2. implementing the merging plan compiler
3. implementing external atoms (mergingplugin)

merging plan compiler

- translates merging plan into HEX program
- program makes use of external atoms
- answer sets = result of the merging plan
Approach

Steps

1. designing a merging language
2. implementing the merging plan compiler
3. implementing external atoms (mergingplugin)

merging plan compiler

- translates merging plan into HEX program
- program makes use of external atoms
- answer sets = result of the merging plan

mergingplugin

defines external atoms for:

- calling of nested HEX programs
- calling of merging operators
Advantages of the Framework

- Rapid prototyping
Advantages of the Framework

- Rapid prototyping

- Routine tasks like information flow management is done automatically
Advantages of the Framework

- Rapid prototyping
- Routine tasks like information flow management is done automatically
- Experiment with different merging plans and operators by parameterizing them
Advantages of the Framework

- Rapid prototyping
- Routine tasks like information flow management is done automatically
- Experiment with different merging plans and operators by parameterizing them
- Develop merging operators once, apply them in many scenarios
Using the Framework

Steps

1. Define your merging task "merging.mp"
2. Run the merging plan compiler (mpcompiler) on this input
3. Execute the result by dlvhex
Using the Framework

Steps

1. Define your merging task “merging.mp”
2. Run the merging plan compiler (mpcompiler) on this input
3. Execute the result by dlvhex
4. (filter the output since the translation of merging plans in HEX programs requires the derivation of some intermediate atoms)
Using the Framework

Steps

1. Define your merging task “merging.mp”
2. Run the merging plan compiler (mpcompiler) on this input
3. Execute the result by dlvhex
4. (filter the output since the translation of merging plans in HEX programs requires the derivation of some intermediate atoms)

Typical call

- Command-line:
  $ mpcompiler merging.mp | dlvhex --filter=a,b,c --
Using the Framework

Steps

1. Define your merging task “merging.mp”
2. Run the merging plan compiler (mpcompiler) on this input
3. Execute the result by dlvhex
4. (filter the output since the translation of merging plans in HEX programs requires the derivation of some intermediate atoms)

Typical call

- Command-line:
  
  $ mpcompiler merging.mp | dlvhex --filter=a,b,c --

- Alternatively:
  
  $ dlvhex --merging --filter=a,b,c merging.mp
Merging plan language: `merging.mp`

[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 plan language: `merging.mp (ctn’d.)`

[merging plan]
{
    operator: setminus;
    {
        operator: union;
        {
            operator: neg;
            {bb1};
        };
        {bb2};
        {bb3};
    };
    {
        operator: union;
        {bb4};
        {bb5};
    };
}
Dalal’s Operator

Definition of implemented version

- Belief bases $K = (AS(P_1), \ldots, AS(P_n))$
  $\mathcal{A} = \text{set of all potential answer sets}$
Dalal’s Operator

Definition of implemented version

- Belief bases $K = (AS(P_1), \ldots, AS(P_n))$
  $\mathcal{A} = \text{set of all potential answer sets}$
- Answer set distance function: $d : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$
Dalal’s Operator

Definition of implemented version

- Belief bases $K = (AS(P_1), \ldots, AS(P_n))$
  $\mathcal{A} =$ set of all potential answer sets
- Answer set distance function: $d : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$
- Answer set to belief base-distances:
  \[
  d(A, P_i) = \min_{J \in AS(P_i)} d(A, J)
  \]
Dalal’s Operator

Definition of implemented version

- Belief bases $K = (\text{AS}(P_1), \ldots, \text{AS}(P_n))$
  $\mathcal{A}$ = set of all potential answer sets
- Answer set distance function: $d : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$
- Answer set to belief base-distances:

\[
d(A, P_i) = \min_{J \in \text{AS}(P_i)} d(A, J)
\]

- Aggregate function: $D : \mathbb{R}^n \rightarrow \mathbb{R}$

\[
D^d(A, K) = D(d(A, P_1), \ldots, d(A, P_n))
\]
Dalal’s Operator

Definition of implemented version

- Belief bases $K = (AS(P_1), \ldots, AS(P_n))$
  $\mathcal{A}$ = set of all potential answer sets
- Answer set distance function: $d : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$
- Answer set to belief base-distances:
  \[
  d(A, P_i) = \min_{J \in AS(P_i)} d(A, J)
  \]
- Aggregate function: $D : \mathbb{R}^n \rightarrow \mathbb{R}$
  \[
  D^d(A, K) = D(d(A, P_1), \ldots, d(A, P_n))
  \]
- $\circ^n(K) = \arg \min_{G \in \mathcal{A} : \text{consistent}} D^d(G, K)$
Fault Diagnosis

Finding an explanation for some observation

Definition
Propositional abduction problem (PAP): $\mathcal{P} = \langle V, H, M, T \rangle$

- $V$ is a finite set of propositional variables
- $H \subseteq V$ is a set of hypothesis
- $M \subseteq V$ is the set of manifestations
- $T$ is a consistent theory
Fault Diagnosis

Finding an explanation for some observation

Definition

Propositional abduction problem (PAP): \( \mathcal{P} = \langle V, H, M, T \rangle \)

- \( V \) is a finite set of propositional variables
- \( H \subseteq V \) is a set of hypothesis
- \( M \subseteq V \) is the set of manifestations
- \( T \) is a consistent theory

\( S \subseteq H \) is a solution iff \( T \cup S \) is consistent and \( T \cup S \models M \)
Fault Diagnosis

Finding an explanation for some observation

Definition

Propositional abduction problem (PAP): $\mathcal{P} = \langle V, H, M, T \rangle$

- $V$ is a finite set of propositional variables
- $H \subseteq V$ is a set of hypothesis
- $M \subseteq V$ is the set of manifestations
- $T$ is a consistent theory

$S \subseteq H$ is a solution iff $T \cup S$ is consistent and $T \cup S \models M$

Challenge: multiple experts with different explanations $S_i$

Task: Finding a group decision $S_G$ s.t.

- $S_G$ is a solution to $\mathcal{P}$
- $S_G$ is as similar to $S_i \ \forall i$ as possible
Full Adder

\[ \begin{align*}
&x & y \\
&\text{haAnd1} & \text{haXor1} & \text{faOr1} \\
&\text{haAnd2} & \text{haXor2} \\
&\text{cin} & \text{cout} & \text{s}
\end{align*} \]
Full Adder - Example interpretation

- $x = 1$
- $y = 1$
- $c_{in} = 1$
- $ha1 = 1$
- $haAnd1 = 1$
- $haXor1 = 1$
- $haAnd2 = 1$
- $haXor2 = 1$
- $faOr1 = 1$
- $s = 1$
- $c_{out} = 1$
Full Adder - Malfunctioning
Full Adder

Implemented as logic program “fulladder.dl” (theory)
with observations “fault.obs”
Full Adder

Implemented as logic program “fulladder.dl” (theory) with observations “fault.obs”

Suppose we have 3 experts with different hypotheses

\[ ab(haAnd1).ab(haXor1).ab(haAnd2).ab(haXor2).ab(faOr1). \]
Full Adder

Implemented as logic program “fulladder.dl” (theory) with observations “fault.obs”

Suppose we have 3 experts with different hypotheses

1. $ab(\text{haAnd1}).ab(\text{haXor1}).ab(\text{haAnd2}).ab(\text{haXor2}).ab(\text{faOr1})$.
2. $ab(\text{haAnd1}).ab(\text{haAnd2}).ab(\text{haXor2}).ab(\text{faOr1})$.
   no $ab(\text{haXor1})$!
Full Adder

Implemented as logic program “fulladder.dl” (theory) with observations “fault.obs”

Suppose we have 3 experts with different hypotheses

1. \( ab(haAnd1).ab(haXor1).ab(haAnd2).ab(haXor2).ab(faOr1). \)
2. \( ab(haAnd1).ab(haAnd2).ab(haXor2).ab(faOr1). \)
   no \( ab(haXor1)! \)
3. \( ab(haAnd1).ab(haAnd2).ab(haXor2).ab(faOr1). \)
   no \( ab(haXor2)! \)
Full Adder

Implemented as logic program “fulladder.dl” (theory) with observations “fault.obs”

Suppose we have 3 experts with different hypotheses

1. $ab(haAnd1).ab(haXor1).ab(haAnd2).ab(haXor2).ab(faOr1)$.
2. $ab(haAnd1).ab(haAnd2).ab(haXor2).ab(faOr1)$. no $ab(haXor1)$!
3. $ab(haAnd1).ab(haAnd2).ab(haXor2).ab(faOr1)$. no $ab(haXor2)$!

(Minimal) Solutions

1. $AS(P_J_1) = \{ \{ ab(haXor1) \} , \{ ab(haXor2) \} \}$
2. $AS(P_J_2) = \{ \{ ab(haXor2) \} \}$
3. $AS(P_J_3) = \{ \{ ab(haXor1) \} \}$
Full Adder - Merging individual decisions

Requirements
The group decision must be an explanation and should be similar to the individual’s
Full Adder - Merging individual decisions

Requirements

The group decision must be an explanation and should be similar to the individual’s

Distance function

$I = \text{individual explanation}$
$G = \text{group explanation}$

1. Penalize ignoring of individual beliefs, i.e.,
   
   $a \in I \land a \notin G$
   
   $\neg a \in I \land \neg a \notin G$
Full Adder - Merging individual decisions

Requirements
The group decision must be an explanation and should be similar to the individual’s

Distance function
$I = \text{individual explanation}$
$G = \text{group explanation}$

1. Penalize ignoring of individual beliefs, i.e.,
   $a \in I \land a \notin G$
   $\neg a \in I \land \neg a \notin G$

2. Penalize group beliefs unfounded for an individual, i.e.,
   $a \in G \land a \notin I$
   $\neg a \in G \land \neg a \notin I$
Full Adder - Merging individual decisions

Requirements

The group decision must be an explanation and should be similar to the individual’s

Distance function

$I = \text{individual explanation}$

$G = \text{group explanation}$

1. Penalize ignoring of individual beliefs, i.e.,
   
   \[ a \in I \land a \notin G \]
   \[ \neg a \in I \land \neg a \notin G \]

2. Penalize group beliefs unfounded for an individual, i.e.,
   
   \[ a \in G \land a \notin I \]
   \[ \neg a \in G \land \neg a \notin I \]

3. Penalize both (1) and (2), i.e.,
   
   \[ |I \Delta G| \]
Full Adder

[common signature]
predicate: ab/1;

[belief base]
name: juror1;
dlvargs: "-FRmin fulladder.dl abnormal1.hyp fault.obs";

[belief base] name: juror2; ...
[belief base] name: juror3; ...

[merging plan]
{
operator: dalal; aggregate: "sum";
penalize: "ignoring";
constraints: "fulladder.dl"; constraints: "fault.obs";
{juror1}; {juror2}; {juror3};
}
Full Adder - Group Decision

Individual explanations

1. \( AS(P_{J_1}) = \{\{ab(\text{haXor1})\}, \{ab(\text{haXor2})\}\} \)
2. \( AS(P_{J_2}) = \{\{ab(\text{haXor2})\}\} \)
3. \( AS(P_{J_3}) = \{\{ab(\text{haXor1})\}\} \)

Possible group explanations

1. \( E_1 = \{ab(\text{haXor1}), ab(\text{haXor2})\} \)
2. \( E_2 = \{ab(\text{haXor1})\} \)
3. \( E_3 = \{ab(\text{haXor2})\} \)
Full Adder - Group Decision

Individual explanations

1. \( AS(P_J_1) = \{\{ab(haXor1)\}, \{ab(haXor2)\}\} \)
2. \( AS(P_J_2) = \{\{ab(haXor2)\}\} \)
3. \( AS(P_J_3) = \{\{ab(haXor1)\}\} \)

Possible group explanations

1. \( E_1 = \{ab(haXor1), ab(haXor2)\} \)
2. \( E_2 = \{ab(haXor1)\} \)
3. \( E_3 = \{ab(haXor2)\} \)

Distances to Individuals
Penalizing ignoring only

<table>
<thead>
<tr>
<th></th>
<th>( AS(P_J_1) )</th>
<th>( AS(P_J_2) )</th>
<th>( AS(P_J_3) )</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( E_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( E_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Full Adder - Group Decision

Individual explanations

1. \( AS(P_{J_1}) = \{\{ab(haXor1)\}, \{ab(haXor2)\}\} \)
2. \( AS(P_{J_2}) = \{\{ab(haXor2)\}\} \)
3. \( AS(P_{J_3}) = \{\{ab(haXor1)\}\} \)

Possible group explanations

1. \( E_1 = \{ab(haXor1), ab(haXor2)\} \)
2. \( E_2 = \{ab(haXor1)\} \)
3. \( E_3 = \{ab(haXor2)\} \)

Distances to Individuals
Penalizing ignoring and unfounded group beliefs (\(|I\Delta G|\))

<table>
<thead>
<tr>
<th></th>
<th>( AS(P_{J_1}) )</th>
<th>( AS(P_{J_2}) )</th>
<th>( AS(P_{J_3}) )</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>( E_2 )</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2  ⇐</td>
</tr>
<tr>
<td>( E_3 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2  ⇐</td>
</tr>
</tbody>
</table>
Summary

- Task: Merging of several belief bases
Summary

- **Task**: Merging of several belief bases
- **Approach**: Merging language
  - with **user-defined** merging operators
- **merging input** $\Rightarrow$ **compiler** $\Rightarrow$ **dlvhex** $\Rightarrow$ **answer sets**
Summary

- Task: Merging of several belief bases
- Approach: Merging language
  - with user-defined merging operators
- merging input $\Rightarrow$ compiler $\Rightarrow$ dlvhex $\Rightarrow$ answer sets

Advantages

- Develop merging operators only once or select one of the preinstalled ones (like Dalal)
- No need for manual re-merging after each change of the setting
- Try out several operators and evaluate which behaves best
- No routine tasks (like information flow between sources)
- User can focus on development and optimization of merging procedures in narrower sense!