Merging of Biomedical Decision Diagrams

Christoph Redl

May 31, 2010
Outline

1. Motivation
2. Decision Diagrams and Task formally defined
3. Implementation and Usage of the Tool
4. Case Study
5. Summary
Motivation

Decision Diagrams

- Important means for decision making
Motivation

Decision Diagrams

- Important means for decision making
- Intuitively understandable
- Not only for knowledge engineers
Motivation

Decision Diagrams

- Important means for decision making
- Intuitively understandable
- Not only for knowledge engineers
- Common in clinical guidelines
Multiple Diagrams

Applications

- Tumor staging
  e.g., TNM system
  \( \text{Tumor, N} \text{odes, M} \text{etastasis} \)
Multiple Diagrams

Applications

- Tumor staging
  e.g., TNM system
  (Tumor, Nodes, Metastasis)
- Quantification of the severity of a medical case
Multiple Diagrams

Applications

- Tumor staging
  e.g., TNM system
  \((Tumor, Nodes, Metastasis)\)
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
Multiple Diagrams

Applications

- Tumor staging
e.g., TNM system
  (Tumor, Nodes, Metastasis)
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
- Molecular biology: DNA classification
Multiple Diagrams

Applications

- Tumor staging
e.g., TNM system
(Tumor, Nodes, Metastasis)

- Quantification of the severity of a medical case

- Psychology: diagnosis of personality disorders or mental diseases

- Molecular biology: DNA classification

- Applications also in other fields (e.g., economy)
Multiple Diagrams

Applications

- Tumor staging
e.g., TNM system
  (Tumor, Nodes, Metastasis)
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
- Molecular biology: DNA classification
- Applications also in other fields (e.g., economy)

Sometimes we have more than one diagram for a certain purpose
Multiple Diagrams

Applications

- Tumor staging
  e.g., TNM system
  \((\text{Tumor, N}odes, \text{M}etastasis)\)
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
- Molecular biology: DNA classification
- Applications also in other fields (e.g., economy)

Sometimes we have more than one diagram for a certain purpose

Reasons

- different opinions
Multiple Diagrams

Applications

- Tumor staging
  e.g., TNM system
  (Tumor, Nodes, Metastasis)
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
- Molecular biology: DNA classification
- Applications also in other fields (e.g., economy)

Sometimes we have more than one diagram for a certain purpose

Reasons

- different opinions
- randomized machine-learning algorithms
Multiple Diagrams

Applications

- Tumor staging
  e.g., TNM system
  *(Tumor, Nodes, Metastasis)*
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
- Molecular biology: DNA classification
- Applications also in other fields (e.g., economy)

Sometimes we have more than one diagram for a certain purpose

Reasons

- different opinions
- randomized machine-learning algorithms
- several diagrams trained in parallel
Multiple Diagrams

Applications

- Tumor staging
e.g., TNM system
  (Tumor, Nodes, Metastasis)
- Quantification of the severity of a medical case
- Psychology: diagnosis of personality disorders or mental diseases
- Molecular biology: DNA classification
- Applications also in other fields (e.g., economy)

Sometimes we have more than one diagram for a certain purpose

Reasons

- different opinions
- randomized machine-learning algorithms
- several diagrams trained in parallel

Task: Incorporate them into one
Existing Solutions

Decision Procedures

- Leave source diagrams unchanged
- Classification by the following *procedure*:
  1. Classify a case by each diagram
  2. Aggregate the results (e.g., majority voting)
Existing Solutions

Decision Procedures

- Leave source diagrams unchanged
- Classification by the following *procedure*:
  1. Classify a case by each diagram
  2. Aggregate the results (e.g., majority voting)

But

- This does not give us a *single* diagram
- We lose the property of intuitive understandability
Existing Solutions

Decision Procedures

- Leave source diagrams unchanged
- Classification by the following procedure:
  1. Classify a case by each diagram
  2. Aggregate the results (e.g., majority voting)

But

- This does not give us a single diagram
- We lose the property of intuitive understandability
- $\Rightarrow$ Inappropriate for clinical practice
Existing Solutions

Rule-based systems

- Conversion: diagram to production rules
- One rule for each leaf node
Existing Solutions

Rule-based systems

- Conversion: diagram to production rules
- One rule for each leaf node

But

- Not human-readable
- Overhead
Therefore

New approach
Merge several decision diagrams
- into a standalone one
- without referring to original training data
- without the overhead of translating them into rule-based systems
Decision Diagrams

**Definition**

Let $D$ be a domain and $C$ a set of classes.
Definition
Let $\mathcal{D}$ be a domain and $\mathcal{C}$ a set of classes.
Then a classification function $c$ (classify) has the signature $c : \mathcal{D} \rightarrow \mathcal{C}$
Definition
Let $\mathcal{D}$ be a domain and $\mathcal{C}$ a set of classes. Then a classification function $c$ (classify) has the signature $c : \mathcal{D} \rightarrow \mathcal{C}$ and can also be represented as decision diagram

$$D = \langle V, E, l_c, l_e, \mathcal{M}^Q \rangle,$$

where
$V$ ... set of nodes and
$E \subseteq V \times V$ ... set of directed edges (s.t. $(V, E)$ is acyclic and $D$ has a unique root node)
Decision Diagrams

Definition
Let $\mathcal{D}$ be a domain and $\mathcal{C}$ a set of classes.
Then a classification function $c$ (classify) has the signature $c : \mathcal{D} \to \mathcal{C}$ and can also be represented as decision diagram

$$D = \langle V, E, l_c, l_e, \mathcal{M}^Q \rangle,$$

where
$V$ ... set of nodes and
$E \subseteq V \times V$ ... set of directed edges (s.t. $(V, E)$ is acyclic and $D$ has a unique root node)
$l_c : Leafs(V) \to \mathcal{C}$
Decision Diagrams

Definition

Let $\mathcal{D}$ be a domain and $\mathcal{C}$ a set of classes. Then a \textit{classification function} $c$ (classify) has the signature $c : \mathcal{D} \rightarrow \mathcal{C}$ and can also be represented as decision diagram

$$D = \langle V, E, l_c, l_e, \mathcal{M}^Q \rangle,$$

where

$V$ ... set of nodes and

$E \subseteq V \times V$ ... set of directed edges (s.t. $(V, E)$ is acyclic and $D$ has a unique root node)

$l_c : \text{Leafs}(V) \rightarrow \mathcal{C}$

$l_e : E \rightarrow \Sigma^Q$ where $\Sigma^Q$ is the set of syntactically correct expressions in $Q$
**Definition**

Let $\mathcal{D}$ be a domain and $\mathcal{C}$ a set of classes. Then a *classification function* $c$ (classify) has the signature $c : \mathcal{D} \rightarrow \mathcal{C}$ and can also be represented as decision diagram

$$D = \langle V, E, l_c, l_e, M^Q \rangle,$$

where

$V$ ... set of nodes and

$E \subseteq V \times V$ ... set of directed edges (s.t. $(V, E)$ is acyclic and $D$ has a unique root node)

$l_c : \text{Leafs}(V) \rightarrow \mathcal{C}$

$l_e : E \rightarrow \Sigma^Q$ where $\Sigma^Q$ is the set of syntactically correct expressions in $Q$

$M^Q :$ The meaning function of a query language $Q$ is of kind

$M^Q : \Sigma^Q \rightarrow 2^\mathcal{D}$
Decision Diagrams
Assume that $F$ is a first-order like query language.

Example
Let $D = \{1, 2, \ldots, 15\}, C = \{\text{prime, not prime}\},$
Decision Diagrams

Assume that $F$ is a first-order like query language.

Example

Let $\mathcal{D} = \{1, 2, \ldots, 15\}, \mathcal{C} = \{\text{prime, not prime}\}$, and

$$D_F = \langle \{r, p, n\}, \{(r, p), (r, n)\}, l_c, l_e, \mathcal{M}^F \rangle$$

with

$$l_c(p) = \text{prime}, l_c(n) = \text{not prime}$$
Decision Diagrams

Assume that $F$ is a first-order like query language.

**Example**

Let $D = \{1, 2, \ldots, 15\}, C = \{\text{prime, not prime}\}$, and

$$D_F = \langle \{r, p, n\}, \{(r, p), (r, n)\}, l_c, l_e, M^F \rangle$$

with

$$l_c(p) = \text{prime}, l_c(n) = \text{not prime}$$

and

$$l_e((r, p)) = f_1 = \#x, y : x > 1 \land y > 1 \land x \cdot y = z$$

$$l_e((r, n)) = f_2 = \exists x, y : x > 1 \land y > 1 \land x \cdot y = z$$
Decision Diagrams

Assume that $F$ is a first-order like query language.

Example
Let $D = \{1, 2, \ldots, 15\}$, $C = \{\text{prime}, \text{not prime}\}$, and

$$D_F = \langle \{r, p, n\}, \{(r, p), (r, n)\}, l_c, l_e, M^F \rangle$$

with

$$l_c(p) = \text{prime}, l_c(n) = \text{not prime}$$

and

$$l_e((r, p)) = f_1 = \exists x, y : x > 1 \land y > 1 \land x \cdot y = z$$
$$l_e((r, n)) = f_2 = \exists x, y : x > 1 \land y > 1 \land x \cdot y = z$$

with the meaning function $M^F(f) = \{ n \in D : I_n(f) = \text{true} \}$ where $I_n$ is the first-order interpretation $I = \{ z \leftarrow n \}$. 
Decision Diagrams

Assume that $F$ is a first-order like query language.

Example

Let $\mathcal{D} = \{1, 2, \ldots, 15\}, \mathcal{C} = \{\text{prime, not prime}\}$, and

$$D_F = \langle \{r, p, n\}, \{(r, p), (r, n)\}, l_c, l_e, \mathcal{M}^F \rangle$$

with

$$l_c(p) = \text{prime}, l_c(n) = \text{not prime}$$

and

$$l_e((r, p)) = f_1 = \#x, y : x > 1 \land y > 1 \land x \cdot y = z$$

$$l_e((r, n)) = f_2 = \exists x, y : x > 1 \land y > 1 \land x \cdot y = z$$

with the meaning function $\mathcal{M}^F(f) = \{n \in \mathcal{D} : I_n(f) = \text{true}\}$ where $I_n$ is the first-order interpretation $I = \{z \leftarrow n\}$.

Example evaluation: We want to classify 7 and start in root node $r$. 
Decision Diagrams

Assume that $F$ is a first-order like query language.

**Example**

Let $D = \{1, 2, \ldots, 15\}$, $C = \{\text{prime, not prime}\}$, and

$$D_F = \langle \{r, p, n\}, \{(r, p), (r, n)\}, l_c, l_e, \mathcal{M}^F \rangle$$

with

$$l_c(p) = \text{prime}, l_c(n) = \text{not prime}$$

and

$$l_e((r, p)) = f_1 = \#x, y : x > 1 \land y > 1 \land x \cdot y = z$$

$$l_e((r, n)) = f_2 = \exists x, y : x > 1 \land y > 1 \land x \cdot y = z$$

with the meaning function $\mathcal{M}^F(f) = \{n \in D : I_n(f) = \text{true}\}$ where $I_n$ is the first-order interpretation $I = \{z \leftarrow n\}$.

**Example evaluation:** We want to classify 7 and start in root node $r$. $7 \in \mathcal{M}^F(f_1)$ but $7 \not\in \mathcal{M}^F(f_2)$, therefore we choose edge $(r, p)$ and end up in node $p$. $l_c(p) = \text{prime}$. 

Task Definition

Definition
The set of all decision diagrams over domain $\mathcal{D}$ and classes $\mathcal{C}$ is denoted as $\mathcal{I}_{\mathcal{D},\mathcal{C}}$. 
Task Definition

Definition
The set of all decision diagrams over domain $\mathcal{D}$ and classes $\mathcal{C}$ is denoted as $\mathcal{B}_{\mathcal{D},\mathcal{C}}$.

Definition
An $n$-ary decision diagram merging operator

$$\circ^n : \underbrace{\mathcal{B}_{\mathcal{D},\mathcal{C}} \times \mathcal{B}_{\mathcal{D},\mathcal{C}} \times \cdots \times \mathcal{B}_{\mathcal{D},\mathcal{C}}}_{\text{n times}} \to \mathcal{B}_{\mathcal{D},\mathcal{C}}$$

maps $n$ input classifiers (over $\mathcal{D}$ and $\mathcal{C}$) onto a new diagram.
Task Definition

Example merging operators

- majority voting
Task Definition

Example merging operators

- majority voting
Task Definition

Example merging operators

- majority voting

- user preferences
Task Definition

Example merging operators

- majority voting

- user preferences
  
  “In doubt, classify it as $X$ rather than $Y$."
  
  (Sometimes one wrong decision is more serious than the other one)
Task Definition

We can make several assumptions about the input diagrams.
Task Definition

We can make several assumptions about the input diagrams.

- It is a tree or a general acyclic graph?
Task Definition

We can make several assumptions about the input diagrams.

- It is a tree or a general acyclic graph?
- What is the maximum node degree?
Task Definition

We can make several assumptions about the input diagrams.

- It is a tree or a general acyclic graph?
- What is the maximum node degree?
- Is the variable ordering along a path fixed?
- etc.
Task Definition

We can make several assumptions about the input diagrams.

- It is a tree or a general acyclic graph?
- What is the maximum node degree?
- Is the variable ordering along a path fixed?
- etc.

Operators can also assert such properties for their result.
Solution

Bad approach: Develop merging operators for the most general task variant, i.e.,
- general decision diagrams
- without any assumptions about node degrees or variable ordering
Solution

**Bad** approach: Develop merging operators for the most general task variant, i.e.,

- general decision diagrams
- without any assumptions about node degrees or variable ordering

This leads to complicated operators
Solution

**Bad** approach: Develop merging operators for the most general task variant, i.e.,
- general decision diagrams
- without any assumptions about node degrees or variable ordering
This leads to complicated operators

**Good** approach: Reduce complicated diagrams to simple ones, i.e.,
Solution

**Bad** approach: Develop merging operators for the most general task variant, i.e.,

- general decision diagrams
- without any assumptions about node degrees or variable ordering

This leads to complicated operators

**Good** approach: Reduce complicated diagrams to simple ones, i.e.,

- first convert acyclic graphs into trees
Solution

Bad approach: Develop merging operators for the most general task variant, i.e.,
- general decision diagrams
- without any assumptions about node degrees or variable ordering
This leads to complicated operators

Good approach: Reduce complicated diagrams to simple ones, i.e.,
- first convert acyclic graphs into trees
- reduce node degrees to make them binary
Solution

**Bad** approach: Develop merging operators for the most general task variant, i.e.,
- general decision diagrams
- without any assumptions about node degrees or variable ordering
This leads to complicated operators

**Good** approach: Reduce complicated diagrams to simple ones, i.e.,
- first convert acyclic graphs into trees
- reduce node degrees to make them binary
- order the nodes
Solution

Therefore we implement two sets of operators
Solution

Therefore we implement two sets of operators

Unary conversion operators

- for preparing and simplifying the input
Solution

Therefore we implement two sets of operators

Unary conversion operators

- for preparing and simplifying the input
- eliminating redundancy from the result
Solution

Therefore we implement two sets of operators

Unary conversion operators

- for preparing and simplifying the input
- eliminating redundancy from the result

Merging operators

- $n$-ary merging operators ($n > 1$) for actual merging task
- they are much simpler if we can make assumptions about the input
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
- Diagrams are given either in a human-readable format like DOT if created manually
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
- Diagrams are given either in a human-readable format like DOT if created manually
- ... or in the file format produced by a machine-learning tool if trained automatically
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
- Diagrams are given either in a human-readable format like DOT if created manually
- ... or in the file format produced by a machine-learning tool if trained automatically
- Operators are written in C++ (user-defined ones can be added)
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
- Diagrams are given either in a human-readable format like DOT if created manually
- ... or in the file format produced by a machine-learning tool if trained automatically
- Operators are written in C++ (user-defined ones can be added)

Decision diagrams are hierarchical: How to handle them with dlvhex?
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
- Diagrams are given either in a human-readable format like DOT if created manually
- ... or in the file format produced by a machine-learning tool if trained automatically
- Operators are written in C++ (user-defined ones can be added)

Decision diagrams are hierarchical: How to handle them with dlvhex?

- convert them into a logic program
- use predicates like $\text{innernode}(X)$, $\text{edge}(X, Y, C)$, etc.
Technical Facts

- Implemented as plugin for dlvhex
- Uses and extends the mergingplugin
- Diagrams are given either in a human-readable format like DOT if created manually
- ... or in the file format produced by a machine-learning tool if trained automatically
- Operators are written in C++ (user-defined ones can be added)

Decision diagrams are hierarchical: How to handle them with dlvhex?

- convert them into a logic program
- use predicates like $\text{innernode}(X)$, $\text{edge}(X, Y, C)$, etc.

Therefore the plugin consists of

1. A tool for conversion of decision diagrams
2. Predefined operators
Usage of the Tool

Steps

1. Convert your diagrams into logic programs using graphconverter
Usage of the Tool

Steps

1. Convert your diagrams into logic programs using `graphconverter`.
2. Define your task “diags.mp” which references the diagrams.
Usage of the Tool

Steps

1. Convert your diagrams into logic programs using `graphconverter`

2. Define your task “diags.mp” which references the diagrams

3. Run the merging plan compiler (`mpcompiler`) on this input

Conversion of input diagrams:

```
$ graphconverter dot hex < diagX.dot > diagX.hex
```

Merging:

```
$ dlvhex --merging diags.mp > out.as
```

Conversion of output diagrams:

```
$ graphconverter as dot < out.as > out.dot
```
### Usage of the Tool

#### Steps

1. Convert your diagrams into logic programs using `graphconverter`
2. Define your task “diags.mp” which references the diagrams
3. Run the merging plan compiler (`mpcompiler`) on this input
4. Load the result into `dlvhex`
Usage of the Tool

Steps

1. Convert your diagrams into logic programs using `graphconverter`
2. Define your task “diags.mp” which references the diagrams
3. Run the merging plan compiler (`mpcompiler`) on this input
4. Load the result into `dlvhex`
Usage of the Tool

Steps

1. Convert your diagrams into logic programs using `graphconverter`
2. Define your task “diags.mp” which references the diagrams
3. Run the merging plan compiler `mpcompiler` on this input
4. Load the result into `dlvhex`
5. Convert the result into desired output format
Usage of the Tool

Steps

1. Convert your diagrams into logic programs using `graphconverter`
2. Define your task “diags.mp” which references the diagrams
3. Run the merging plan compiler (`mpcompiler`) on this input
4. Load the result into `dlvhex`
5. Convert the result into desired output format

Typical call

- Conversion of input diagrams $X$:
  $\$ graphconverter dot hex < diagX.dot > diagX.hex$
- Merging:
  $\$ dlvhex --merging diags.mp > out.as$
- Conversion of output diagrams:
  $\$ graphconverter as dot < out.as > out.dot$
Implementation

dot files -> graphconverter dot hex -> set of facts -> merging plan

user

merging plan compiler

dlhex
decisiondiagram plugin

{AS}
answer-set

dot file
Advantages of the Framework

- Framework is considered as an extension to the general belief merging framework
Advantages of the Framework

- Framework is considered as an extension to the general belief merging framework
- Intention is not to provide high-performance throughput, but an easy-to-use rapid prototyping tool
Advantages of the Framework

- Framework is considered as an extension to the general belief merging framework
- Intention is not to provide high-performance throughput, but an easy-to-use rapid prototyping tool
- Different operators can be tried out without reimplementing them
Advantages of the Framework

- Framework is considered as an extension to the general belief merging framework
- Intention is not to provide high-performance throughput, but an easy-to-use rapid prototyping tool
- Different operators can be tried out without reimplementing them
- Input diagrams, training set and merging strategies can be edited quickly without redoing manual remerging
Advantages of the Framework

- Framework is considered as an extension to the general belief merging framework
- Intention is not to provide high-performance throughput, but an easy-to-use rapid prototyping tool
- Different operators can be tried out without reimplementing them
- Input diagrams, training set and merging strategies can be edited quickly without redoing manual remerging
- When best settings have been found, one can consider application-specific reimplementation due to performance reasons
DNA Classification

Application Scenario

- DNA classification
- Given a sequence $S \in \{A, C, G, T\}^*$
- Is it protein-coding or junk DNA? $\{C, N\}$
DNA Classification

Application Scenario

- DNA classification
- Given a sequence $S \in \{A, C, G, T\}^*$
- Is it protein-coding or junk DNA? $\{C, N\}$

Approach

- Compute a 20-dimensional feature vector
- Features are statistically motivated
  (incorporating biological knowledge)
DNA Classification

Application Scenario

- DNA classification
- Given a sequence \( S \in \{A, C, G, T\}^* \)
- Is it protein-coding or junk DNA? \( \{C, N\} \)

Approach

- Compute a 20-dimensional feature vector
- Features are statistically motivated
  (incorporating biological knowledge)

Examples

- Triplet \( ATG \) more frequent in coding sequences
DNA Classification

Application Scenario

- DNA classification
- Given a sequence $S \in \{A, C, G, T\}^*$
- Is it protein-coding or junk DNA? $\{C, N\}$

Approach

- Compute a 20-dimensional feature vector
- Features are statistically motivated (incorporating biological knowledge)

Examples

- Triplet $ATG$ more frequent in coding sequences
- In coding sequences, the base at first codon position is frequently a purine base
DNA Classification

The role of the merging framework

- Train *multiple* decision trees for the task
- Merge them afterwards
DNA Classification

The role of the merging framework

- Train *multiple* decision trees for the task
- Merge them afterwards

Advantages

- Better accuracy by using different algorithms
- Parallel computing
- Try out different combinations of training algorithms; no need for manual remerging after each changes
DNA Classification

Suppose we have 3 trees trained with different algorithms and training sets

Each has accuracy \( \approx 50\% \)
DNA Classification

Merging operator in use (Stephen Salzberg)

1. Leaf nodes do not only store classification but also frequency distribution.
DNA Classification

Merging operator in use (Stephen Salzberg)

1. Leaf nodes do not only store classification but also frequency distribution.
2. Insert second tree into each leaf of the first one (iterate for more than two trees).
DNA Classification

Merging operator in use (Stephen Salzberg)

1. Leaf nodes store classification and frequency distribution.
2. Insert the second tree into each leaf of the first one (iterate for more than two trees).
3. Weight the trees according to the size of the training set used during training.
DNA Classification

Merging operator in use (Stephen Salzberg)

1. Leafs do not only store classification but also frequency distribution
2. Insert second tree into each leaf of the first one (iterate for more than two trees)
3. Weight trees according to size of training set used during training
4. Recompute classification for each (new) leaf according to new distribution
DNA Classification

After merging

Accuracy $\approx 65\%$
Task: Incorporate several decision diagrams into one
Summary

- **Task**: Incorporate several decision diagrams into one
- **Approach**:
  - implement a plugin that allows decision diagram handling with dlvhex
  - provide predefined operators
Summary

- **Task:** Incorporate several decision diagrams into one
- **Approach:**
  - implement a plugin that allows decision diagram handling with dlvhex
  - provide predefined operators

**input diagrams** $\Rightarrow$ **graphconverter** $\Rightarrow$
$\Rightarrow$ **dlvhex** $\Rightarrow$ **answer set** $\Rightarrow$
$\Rightarrow$ **graphconverter** $\Rightarrow$ **output diagram**
Summary

- Task: Incorporate several decision diagrams into one
- Approach:
  - implement a plugin that allows decision diagram handling with dlvhex
  - provide predefined operators

input diagrams $\Rightarrow$ graphconverter $\Rightarrow$ dlvhex $\Rightarrow$ answer set $\Rightarrow$
$\Rightarrow$ graphconverter $\Rightarrow$ output diagram

Advantages

- Easy-to-use rapid prototyping tool
- No manual incorporation of diagrams
- Allows changes of the scenario without redoing routine tasks
Summary

- Task: Incorporate several decision diagrams into one
- Approach:
  - implement a plugin that allows decision diagram handling with dlvhex
  - provide predefined operators
- input diagrams $\Rightarrow$ graphconverter $\Rightarrow$ dlvhex $\Rightarrow$ answer set $\Rightarrow$ graphconverter $\Rightarrow$ output diagram

Advantages

- Easy-to-use rapid prototyping tool
- No manual incorporation of diagrams
- Allows changes of the scenario without redoing routine tasks

$\Rightarrow$ dd-plugin = mergingplugin + dd processing with dlvhex + dd operators