decisiondiagramsplugin - User Guide

Christoph Redl

March 13, 2010

The decisiondiagramsplugin adds support for working with decision diagrams to dlvhex. In consists of a tool for conversion between the dot format and sets of facts as well as several operators for decision diagram modification and merging.

The plugin was developed as part of the master's thesis *Merging of Biomedical Decision Diagrams* [2].

1 File Format Conversions

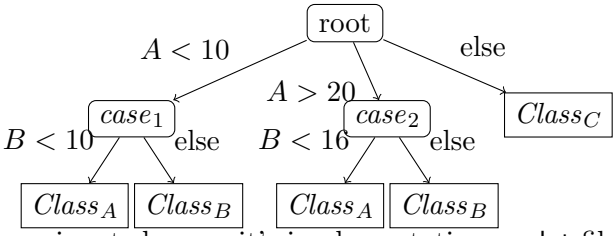
Decision diagrams can be stored in different file formats. While some of them are human-readable, others are better for automatic processing.

1.1 Supported Formats

1.1.1 dot

The dot file format[1] is intuitively readable and thus fit for being used as human readable format for representing decision diagrams. Additionally it is well suited for being visualized using the *dot tools*.

Consider the following decision diagram.



Then the following snippet shows a it's implementation as dot file.

```
digraph G {
        root -> case1 ["A<10"];
        root -> case2 ["A>20"];
        root -> elsecase ["else"];
        root -> case3 ["else"];
```

[1] http://www.graphviz.org

```
        case1 -> case1a ["B<10"];
        case1 -> case1b ["else"];
        case2 -> case2a ["B<16"];
        case2 -> case2b ["else"];
        case1a ["ClassA"];
        case1b ["ClassB"];
        case2a ["ClassA"];
        case2b ["ClassB"];
        case3 ["ClassC"];
}
```

Listing 1: the above decision diagram in dot format

Valid decision diagrams must

- have exactly one root node (which does not need to be explicitly mentioned, but which is implicitly identified by the fact that it has no ingoing edges)
- use only directed edges
- use only edges that are labeled either with

  $$else$$

  or with conditions of form

  $$X \circ Y$$

  where X and Y can be arbitrary strings and $\circ \in \{<, <=, =, >, >=\}$ is an operator
- have leaf nodes that are labeled with arbitrary strings that encode the classification in this node

1.1.2 Sets of Facts

However, dlvhex cannot directly load this format because it's input must be a logic program. Thus a diagram must be represented using predicates.

We define the following predicates:

- *root*(X)
  To define that some constant X is defined as the root node
- *innernode*(X)
  To define some constant X to be an inner node
- *leafnode*(X, Y)
  To define that some constant X is a leaf node with label Y
- *conditionaledge*(X, Y, A, C, B)
  To define that a conditional edge with condition $A \circ B$ (where the operation $\circ$ is given by C) leads from node X to Y

- *elseedge*(X, Y)
  To define that an unconditional edge goes from X to Y

The above diagram can therefore be implemented as follows.

```
root(root).
innernode(case1).
innernode(case2).
leafnode(case3, "ClassC").
leafnode(case1a, "ClassA").
leafnode(case1b, "ClassB").
leafnode(case2a, "ClassA").
leafnode(case2b, "ClassB").
conditionaledge(root, case1, "A", "<", "10").
conditionaledge(root, case2, "A", ">", "20").
elseedge(root, case3).
conditionaledge(case1, case1a, "B", "<", "10").
elseedge(case1, case1b).
conditionaledge(case2, case2a, "B", "<", "16").
elseedge(case2, case2b).
```

Listing 2: the above decision diagram as HEX program

1.1.3 Answer-Sets

A very simple and obvious translation from HEX programs into answer-sets is to to put all the facts simply as atoms into the answer-set. The above diagram can therefore also be implemented as:

```
{root(root),
innernode(case1),
innernode(case2),
leafnode(case3, "ClassC"),
leafnode(case1a, "ClassA"),
leafnode(case1b, "ClassB"),
leafnode(case2a, "ClassA"),
leafnode(case2b, "ClassB"),
conditionaledge(root, case1, "A", "<", "10"),
conditionaledge(root, case2, "A", ">", "20"),
elseedge(root, case3),
conditionaledge(case1, case1a, "B", "<", "10"),
elseedge(case1, case1b),
conditionaledge(case2, case2a, "B", "<", "16"),
elseedge(case2, case2b)}
```

Listing 3: the above decision diagram as answer-set

1.1.4 RapidMiner XML Format

Rapidminer[2] is an open-source data mining tool. It uses a priprietary XML file format to store decision trees. This format is also supported by the

[2] `http://www.rapidminer.com`

tool introduced in 1.2. The details are not relevant for practical work and are skipped therefore. It is only important to know that the import and export functionality for this file format is necessary to process RapidMiner classifiers by the decisiondiagramplugin.

1.2 Conversion

For the conversion between the introduced file formats, the plugin installs a tool called graphconverter. It can be used to translate diagrams in any of the supported file formats into semantically equivalent versions in another format. Assume that the diagram is stored in file "mydiagram.dot". Then the conversion into the according HEX program is done by entering:

graphconverter dot hex <mydiagram.dot >mydiagram.hex

The result is a set of facts that can be loaded by dlvhex. After dlvhex has done it's job, the output will be an answer-set, which is ill-suited for begin read by humans. Thus the plugin also supports conversions in the other direction. Assume that dlvhex' output is stored in file "answerset.as" (using the *silent mode* such that the output contains the *pure* answer-set without any additional information about dlvhex). Then the conversion is done by:

graphconverter as dot <answerset.dot >out_diagram.dot

Between the two converter calls, the diagram is diven as HEX program "mydiagram.hex" that can be processed by dlvhex. Even though one can essentially do anything with this program that is computable, it is strongly intended to be used as part of the input for a revision task.

Note that graphconverter reads from standard input and writes to standard output. The graphconverter expects either one or two parameters. If one parameter is passed, it can be anything of:

- $--toas$
  Converts a dot file into a HEX program.
- $--todot$
  Converts an answer-set into a dot file.
- $--help$
  Displays an online help message.

Note that $--toas$ and $--todot$ are only abbreviations for commonly used conversions. The more general program call passes two parameters, where the first one states the source format and the second one the desired

destination format. Both parameters can be anything from the following list.

Format	parameter name
dot graph	dot
HEX program	hexprogram or hex
answer-set	answerset or as
RapidMiner XML	rmxml or xml

2 Applying Operators

This section presupposes familarity with the *mergingplugin*, especially the usage of revision plans and operators. For a detailled description see the according documentation.

Listing 4 shows a typical revision task definition that uses the operators *unfold* and *tobinarydecisiontree*. Note that the decision diagram encoded as logic program was directly pasted into the mapping rules. This was only done in order to give the program as one self-contained example. In practice, the mapping rules would rather access an external source by the use of external atoms.

```
[common signature]
predicate: root/1;
predicate: innernode/1;
predicate: leafnode/2;
predicate: conditionaledge/5;
predicate: elseedge/2;

[belief base]
name: kb1;
mapping: "
        root(root).
        innernode(root).
        innernode(v1).
        innernode(v2).
        innernode(v3).
        leafnode(leaf1, class1).
        leafnode(leaf2, class2).
        leafnode(leaf3, class3).
        conditionaledge(root, v1, x, \'<\', y).
        conditionaledge(root, v2, z, \'<\', y).
        elseedge(root, v3).
        conditionaledge(v1, leaf1, a, \'<\', y).
        conditionaledge(v1, leaf2, b, \'<\', y).
        elseedge(v1, leaf3).
        conditionaledge(v2, leaf1, aa, \'<\', y).
        conditionaledge(v2, leaf2, bb, \'<\', y).
        elseedge(v2, leaf3).
        conditionaledge(v3, leaf1, aaa, \'<\', y).
        conditionaledge(v3, leaf2, bbb, \'<\', y).
        elseedge(v3, leaf3).
";
```

```
[revision plan]
{
        operator: tobinarydecisiontree;
        {
                operator: unfold;
                {
                        kb1
                };
        };
}
```

Listing 4: demonstration of the plugin usage

The decisiondiagramplugin ships with several special merging and modification operators for decision diagrams. They expect the belief bases to be sets of facts which were generated out of decision diagrams using the graphconverter (see 1.2). The output will again be a set of facts that encodes a diagram, which can be back-converted into a dot file. Intermediate results are sets of answer-sets.

The following subsections describe the operators that are included in the plugin. Note that this is just a very quick and informal description that should enable the user to explore the capabilities or adapt operators according to individual needs. For a more detailled and formal description, see the cited thesis.

2.1 Modification Operators

2.1.1 unfold

The input can be any number of general decision diagrams. The output will contain the same number of diagrams, where each of them has (independently) been converted into a tree. This is done by duplication of subtrees if necessary.

2.1.2 tobinarydecisiondiagram

The operator is unary, i.e. it works on a single belief base. It expects the input to encode a diagram that is a *tree*, i.e. it contains no sharing of subnodes.

Then the output is again a tree where each node has at most two successors (*binary* tree). This is done by introduction of intermediate nodes.

2.1.3 orderbinarydecisiontree

The operator is again unary. It expects it's input to be a *binary* decision *tree*. It's output will a semantically equivalent binary decision tree, where on each path from the root to a leaf node, the variables are only queried in lexical ordering.

2.1.4 simplify

The input can be any number of decision diagrams. Each of them is (independently) simplified by some algorithms that will leave the semantics of the diagram unchanged. That is, only the structure of the diagram will be modified, which makes them more readable.

2.2 Merging Operators

2.2.1 userpreferences

This operator is n-ary, i.e. arbitrary many diagrams can be passed. Additionally it expects arbitrary many key-value pairs as parameters, where the keys are ignored and the values are of form:

$$\mathsf{X} >> \mathsf{Y} \text{ or } \mathsf{X} > \mathsf{n} > \mathsf{Y}$$

where X and Y are the names of class labels (as used in leaf nodes) and n is an integer ≥ 0. A rule of form $\mathsf{X} >> \mathsf{Y}$ expresses that "in doubt, X is preferred to Y", whereas $\mathsf{X} > \mathsf{n} > \mathsf{Y}$ states "X is preferred to Y if there are at least n more input diagrams that vote for X than for Y".

The output is a diagram where each domain element is classified according to this rules. Note that the rules are evaluated in top-down manner. That is, the result of of a prior rule can be overwritten by a later (applicable!) rule.

2.2.2 majorityvoting

The input can be any number of (general) decision diagrams. The output is a diagram where each domain element is automatically classified by each of the inputs. Then the final class label is determined by majority decision. In case that this does not lead to a unique result, the input diagram with the least index forces it's decision.

2.2.3 avg

The operator expects exactly two *ordered binary trees* as input parameter. The result will again be an ordered binary tree of the following form.

For each node from the root to the leafs, if one of the input trees contains condition $X \circ c_1$ and the other one $X \circ c_2$, the resulting tree will contain $X \circ \frac{c_1+c_2}{2}$, i.e. the mean of the comparison value is computed. In case that one of the inputs contains $X \circ c_1$ and the other one $Y \circ c_2$, the result will contain $X \circ c_1$ at this position (since X is lexically smaller than Y), and the second diagram is incorporated recursivly in both subtrees.

In case of contradicting leaf nodes or incompatible comparison operators (e.g. $X < c_1$ and $X > c_2$), the result is "unknown".

3 Conclusion

For a more detailed and formal description of the operators see [2]. For an introduction to the belief merging framework see [1].

References

[1] Christoph Redl. Development of a belief merging framewerk for dlvhex. 2010.

[2] Christoph Redl. Merging of decision diagrams. 2010.