Decision-Theoretic Planning with Generalized First Order Decision Diagrams

Saket Joshi[a], Kristian Kersting[b], Roni Khardon[c]

[a]*School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA*
[b]*Knowledge Discovery Department, Fraunhofer IAIS, 53754, Sankt Augustin, Germany*
[c]*Department of Computer Science, Tufts University, Medford, MA 02155, USA*

Abstract

Many tasks in AI require representation and manipulation of complex functions. First order decision diagrams (FODD) are a compact knowledge representation expressing functions over relational structures. They represent numerical functions that, when constrained to the Boolean range, use only existential quantification. Previous work has developed a set of operations for composition and for removing redundancies in FODDs, thus keeping them compact, and showed how to successfully employ FODDs for solving large-scale stochastic planning problems through the formalism of relational Markov decision processes (RMDP). In this paper, we introduce several new ideas enhancing the applicability of FODDs. More specifically, we first introduce Generalized FODDs (GFODD) and composition operations for them, generalizing FODDs to arbitrary quantification. Second, we develop a novel approach for reducing (G)FODDs using model checking. This yields – for the first time – a reduction that maximally reduces the diagram for the FODD case and provides a sound reduction procedure for GFODDs. Finally we show how GFODDs can be used in principle to solve RMDPs with arbitrary quantification, and develop a complete solution for the case where the reward function is specified using an arbitrary number of existential quantifiers followed by an arbitrary number of universal quantifiers.

Keywords:
Knowledge representation, Automated reasoning, First order logic, Model checking, Markov decision process, Dynamic programming, Decision theoretic planning

1. Introduction

The problem of an autonomous agent acting optimally in an environment is central to Artificial Intelligence. There are many variants of this problem. For the case where the stochastic dynamics of the environment are known and the objective can be described by a reward function, Markov decision processes (MDP) have become the standard model [1, 2]. Classical dynamic programming algorithms for solving MDPs [3, 4], however, require explicit state enumeration. This is often impractical as the number of states grows very quickly with the number of domain objects and relations. For example in a domain with predicate $on(X, Y)$, and n objects that can be substituted for X and Y, we have at least n^2 ground propositions and 2^{n^2} potential states. Classical solutions require enumeration of these 2^{n^2} states. In other words, classical dynamic programming solutions to MDPs do not scale to bigger problems because the size of the state space is too large.

One potential solution to this problem is the use of structure in representing state and action spaces. Many problems are naturally described by referring to objects and relations among them. Relational representations naturally factor the state space and they can capture parameterized functions over the state space. The past few years have seen the successes of this approach in the field of Statistical Relational Learning [5] which combines expressive knowledge representation formalisms with statistical approaches to perform probabilistic inference and learning in relational domains. MDPs enhanced with such representations are known as relational or first-order MDPs.

Recently, Boutilier et al. [6] have shown how algorithms for relational MDPs (RMDP) can be used to solve stochastic planning problems. Inspired by this seminal work, several authors have developed different representation schemes and algorithms implementing this idea [7, 8, 9, 10]. In particular, Wang et al. [9] and Joshi and Khardon [11] introduced the First-Order Decision Diagram (FODD) representation, showed how RMDPs can be solved using FODDs, and provided a prototype implementation that performs well on problems from the International Planning Competition. The use of FODDs to date, however, has two main limitations. The first is representation power. FODDs (roughly speaking) represent existential statements but do not allow universal quantification. This excludes some basic planning tasks. For example, a company that has to plan a physical meeting of all employees requires that they are all in a single location thus requiring a quantifier prefix $\exists\forall$ for the goal; the goal can be expressed as "there exists a location such that all employees are in that location". The second is that manipulation algorithms for FODDs require special reductions to ensure that their size is small. Such reductions have been introduced but they are not complete, i.e., they may not yield a small FODD although one exists.

In this article, we show how one can overcome these limitations. Specifically, we make the following three contributions. First, we introduce Generalized FODDs (GFODD), a novel FODD variant that allows for arbitrary quantification as well as more general aggregations of values. Basic algorithms that allow us to perform operations over functions represented by GFODDs are developed. Second, we show how GFODDs can be used to solve RMDPs with arbitrary quantification. Finally, we provide a novel reduction approach based on model checking. This provides the first reduction for FODDs that guarantees that the resulting FODD is "maximally reduced" in a sense which is defined precisely in the technical section. This is a significantly stronger reduction than ones that existed previously for FODDs. In addition we develop model checking reductions for the $\exists^*\forall^*$ quantifier setting of GFODDs, where a finite number of existential quantifiers is followed by a finite number of universal quantifiers. We show that this enables solutions for RMDPs with reward functions given by $\exists^*\forall^*$ statements, where all intermediate constructs in the algorithm are maintained in this form. The new representations and algorithms developed form a significant extension of the scope of the FODD approach to decision-theoretic planning and a significant improvement of our understanding of their reductions.

The new reductions presented in the paper have a relatively high complexity and are not likely to be efficient in practice for large diagrams. However, they provide the basis for easy-to-implement heuristic reductions for FODDs. In recent work [12] we developed such heuristic reductions as well as heuristics for generating the models from problem descriptions. The new reductions provide significant speedup in planning time, over an implementation using theorem proving reductions, while maintaining state-of-the-art performance on problems from the international planning competition. Model checking reductions are therefore important in expanding applicability of FODDs to decision theoretic planning. Practical implementations of reductions for GFODDs will be similarly important for their applicability.

Our results are also closely related to recent work on probabilistic inference with large mod-

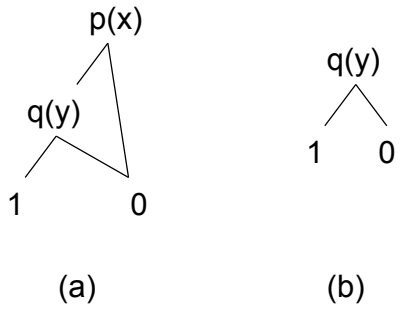

Figure 1: Examples of FODDs. Left going edges represent the branch taken when the predicate is `true` and right edges are the `false` branches.

els. In fact, the relational value iteration algorithm of Boutilier et al. [6] and our implementation of this algorithm using (G)FODDs can be seen to perform some form of lifted inference in probabilistic models. Recently several algorithms that take advantage of model structure in inference have been proposed [13, 14, 15, 16, 17, 18, 19, 20, 21]. Whereas, existing approaches essentially take a single ground model and a single ground question and calculate a numerical solution for the question, our solutions for RMDPs take a family of models and a potentially non-ground question as input, and calculate numerical solutions for all members of the family. Of course the planning models must have some structure to make this possible and this is precisely the structure our algorithms take advantage of.

We proceed as follows. After briefly reviewing FODDs, we present the model checking reduction operator for FODDs in Section 3. Then, in Section 4, we introduce GFODDs and their composition operations. Section 5 extends the model checking reduction operator to GFODDs with the quantifier setting $\exists^*\forall^*$. Finally Section 6 shows the utility of GFODDs for solving RMDPs. To that end we devise a value iteration approach for RMDPs using GFODDs. Note that, since knowledge of RMDPs is not required for the development and algorithms for GFODDs, we have deferred the introduction of RMDPs to Section 6.

2. First-Order Decision Diagrams

This section briefly reviews previous work on FODDs [9]. We use standard terminology from first-order logic [22]. A first-order decision diagram is a labeled directed acyclic graph, where each non-leaf node has exactly 2 outgoing edges labeled `true` and `false`. The non-leaf nodes are labeled by atoms generated from a predetermined signature of predicates, constants and an enumerable set of variables. Leaf nodes have non-negative numeric values. The signature also defines a total order on atoms, and the FODD is ordered with every parent smaller than the child according to that order.

Example 1. *Two examples of FODDs are given in Figure 1; in these and all diagrams in the paper left going edges represent the branch taken when the predicate is `true` and right edges are the `false` branches.*

Thus, a FODD is similar to a formula in first-order logic. Its meaning is similarly defined relative to interpretations of the symbols. An *interpretation* defines a domain of objects, identifies

each constant with an object, and specifies the truth value of each predicate over these objects. In the context of relational MDPs, an interpretation represents a state of the world with the objects and relations among them. Given a FODD and an interpretation, a *valuation* assigns each variable in the FODD to an object in the interpretation. Following Groote and Tveretina [23], the semantics of FODDs are defined as follows. If B is a FODD and I is an interpretation, a valuation ζ that assigns a domain element of I to each variable in B fixes the truth value of every node atom in B under I. The FODD B can then be traversed in order to reach a leaf. The value of the leaf is denoted $Map_B(I, \zeta)$. $Map_B(I)$ is then defined as $max_\zeta Map_B(I)$, that is, an aggregation of $Map_B(I, \zeta)$ over all valuations ζ.

Example 2. *Consider the FODD in Figure 1(a) and the interpretation I with objects a, b and where the only true atoms are $p(a), q(b)$. The valuations $\{x/a, y/a\}$, $\{x/a, y/b\}$, $\{x/b, y/a\}$, and $\{x/b, y/b\}$, will produce the values* 0, 1, 0, 0 *respectively. By the max aggregation semantics, $Map_B(I) = max\{0, 1, 0, 0\} = 1$. Thus, this FODD is equivalent to the formula $\exists x \exists y, p(x) \wedge q(y)$.*

In general, *max* aggregation yields existential quantification when leaves are binary. When using numerical values we can similarly capture value functions for relational MDPs.

The following notation will be used to discuss FODDs and their properties. If e is an edge from node n to node m, then target(e) = m. For node n, the symbols $n_{\downarrow t}$ and $n_{\downarrow f}$ denote the `true` and `false` edges out of n respectively. Furthermore, $l(n)$ denotes the atom associated with node n. Node formulas (NF) and edge formulas (EF) are defined recursively as follows. For a node n labeled $l(n)$ with incoming edges $e_1, \ldots, e_k$, the node formula NF(n) = ($\vee_i$EF(e_i)). The edge formula for the `true` outgoing edge of n is EF($n_{\downarrow t}$) = NF(n) $\wedge$ $l(n)$. The edge formula for the `false` outgoing edge of n is EF($n_{\downarrow f}$) = NF(n) $\wedge \neg l(n)$. These formulas, where all variables are existentially quantified, capture the conditions under which a node or edge are reached. Similarly, if B is a FODD and p is a path from the root to a leaf in B, then the path formula for p, denoted by PF(p) is the conjunction of literals along p. When the variables of p, are existentially quantified, satisfiability of PF(p) under an interpretation I is a necessary and sufficient condition for the path p to be traversed by some valuation under I. If ζ is such a valuation, then we define $Path_B(I, \zeta)$ = p. The leaf reached by path p is denoted as $leaf(p)$.

As seen above FODDs can represent functions over relational structures. These functions can be combined under arithmetic operations, and reduced in order to remove redundancies, in a manner that extends ideas developed for propositional (binary and algebraic) decision diagrams [24, 25]. In particular, Groote and Tveretina [23] introduced four reduction operators (R1 ... R4) and these were augmented with seven more reductions (R5 ... R11) [9, 11]. Intuitively, redundancies in FODDs arise in two different ways. In the first scenario, some edges may never be traversed by any valuation. Reduction operators for such redundancies are called strong reduction operators. The second scenario requires more subtle analysis: there may be parts of the FODD that are traversed under some valuations but because of the max aggregation, the valuations that traverse those parts are never important for determining the map. Operators for such redundancies are called weak reductions operators. Strong reductions preserve $Map_B(I, \zeta)$ for every valuation ζ (thereby preserving $Map_B(I)$) and weak reductions preserve $Map_B(I)$ but not necessarily $Map_B(I, \zeta)$ for every ζ. Using this classification R1-R5 are strong reductions and R6-R11 are weak reductions.

Weak reductions have their basis in the idea that some parts of the FODD dominate the map and therefore parts that are dominated can be removed or replaced by a 0 leaf. However, there are cases when two parts of the FODD dominate each other.

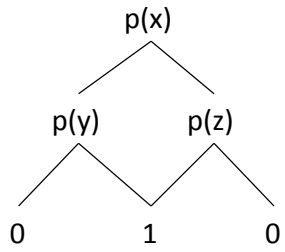

Figure 2: A FODD example illustrating the need for DPOs.

Example 3. *Consider the FODD in Figure 2. This simple FODD contains only 2 paths leading to non-zero leaves:*

1. $p(x), \neg p(y) \rightarrow 1$
2. $\neg p(x), p(z) \rightarrow 1$

Notice that whenever there is a valuation traversing one of the paths, there is another valuation traversing the other and reaching the same leaf. Either of the two edges reaching the 1 *leaf can point to a* 0 *leaf without changing the map. However we cannot allow both the edges to point to a zero leaf as that would change the map of some interpretations.*

To avoid this ambiguity we must specify a total order on the paths, and in this way we can choose which path to remove. A *descending path ordering* (DPO) is constructed specifically for this purpose.

Definition 1. *A descending path ordering (DPO) is an ordered list of all paths from the root to leaves in a FODD, sorted in descending order by the value of the leaf reached by the path. The relative order of paths reaching the same leaf can be set arbitrarily.*

A DPO provides a preference ordering over paths. Paths with different values are naturally ordered by their values and this is incorporated in the DPO. Paths with the same value are ordered according to the (arbitrary) ordering in the DPO where paths with a lower index are preferred to paths with a higher index. This preference is captured in the notion of instrumental paths which is defined next.

Definition 2. *If B is a FODD, and PL is a DPO for B, then a path $p_j \in PL$ is instrumental with respect to PL iff there is an interpretation I such that*

1. *there is a valuation ζ such that $Path_B(I, \zeta) = p_j$, and*
2. *for all valuations η, if $Path_B(I, \eta) = p_k$, then $k \geq j$.*

Paths that are not instrumental can be removed from a diagram without changing the function it computes. The choice of DPO can affect the size of the reduced diagram, but it is not clear at the outset how to best choose a DPO so as to maximally reduce the size of a diagram. This is illustrated and discussed further in the context of the R12 reduction.

Finally, an additional subtlety arises because for RMDP domains we may have some background knowledge about the predicates in the domain specifying some constraints on them. For example, in the blocks world, if block a is clear then $on(x, a)$ is false for all values of x. This fact might help simplify the diagram. We denote such *background knowledge* by $\mathcal{B}$ and allow reductions to rely on such knowledge.

3. R12: The Model Checking Reduction for FODDs

In this section we introduce a new reduction operator R12 (numbered to agree with previous work). The basic intuition behind R12 is to use the semantics of the FODD directly in the reduction process. According to the semantics of FODDs the map is generated by aggregation of values obtained by running all possible valuations through the FODD. Therefore, if we run all possible valuations through the diagram and document the paths taken by the valuations under all possible interpretations, we can identify parts of the diagram that are never important for determining the map. Such parts can then be eliminated to reduce the diagram. Crucially, with some bookkeeping, it is possible to obtain this information without enumerating all possible interpretations and by enumerating all possible valuations over just the variables in the diagram. This is the basic intuition behind R12.

We can avoid enumerating all possible interpretations with the observation that although there can be many interpretations over a set of domain objects, there are only a fixed number of paths in the FODD that a valuation can traverse. For a given valuation ζ, any interpretation can be classified into one of a set of equivalence classes based on the path p that it forces ζ through. All interpretations belonging to an equivalence class have the following in common.

1. They force ζ through path p and leaf(p), the leaf reached by path p.
2. They are consistent with PF(p)(ζ).

PF(p)(ζ) is, thus, the most general interpretation that forces ζ through p and can be viewed as a key or identifier for its equivalence class. For the purpose of reduction we are not interested in the interpretations themselves but only in the paths that they force valuations through. Therefore we can restrict our attention to the equivalence classes and avoid enumerating all possible interpretations. In other words, if we collect the abstract interpretation PF(p)(ζ) for every path p that a valuation ζ could possibly take (i.e. every path where PF(p)(ζ) is consistent), along with the corresponding path and leaf reached, we will have all information we need to describe the behavior of ζ under all possible interpretations. The procedure *getBehaviors* described below, does exactly that by simulating the run of a valuation through a FODD. The output of the procedure is a set of $\langle leaf, EL, I\rangle$ 3-tuples, where $leaf$ is the leaf reached by the valuation ζ by traversing the path p (described by the set of edges EL) and I = PF(p)(ζ). Recall that $\mathcal{B}$ denotes the background knowledge on the domain. The procedure is as follows.

Procedure 1. *getBehaviors(valuation ζ, PathFormula PF, EdgeList EL, Node n)*

1. *If n is a leaf, return* $\{\{l(n), EL, PF\}\}$
2. *If* $\mathcal{B} \models PF \rightarrow l(n)(\zeta)$, *then*
   return getBehaviors(ζ, $PF \cup l(n)(\zeta)$, $EL \cup n_{\downarrow t}$, target($n_{\downarrow t}$))
   Else If $\mathcal{B} \models PF \rightarrow \neg l(n)(\zeta)$, *then*
   return getBehaviors(ζ, $PF \cup \neg l(n)(\zeta)$, $EL \cup n_{\downarrow f}$, target($n_{\downarrow f}$))
   Else
   return getBehaviors(ζ, $PF \cup l(n)$, $EL \cup n_{\downarrow t}$, target($n_{\downarrow t}$))
   $\cup$ getBehaviors(ζ, $PF \cup \neg l(n)$, $EL \cup n_{\downarrow f}$, target($n_{\downarrow f}$))

Example 4. *Figure 3 shows an example of the R12 reduction whose details are developed below. For this example we focus on the table in the center of the figure. The table illustrates the result of running the getBehaviors procedure on all possible valuations over the set of domain*

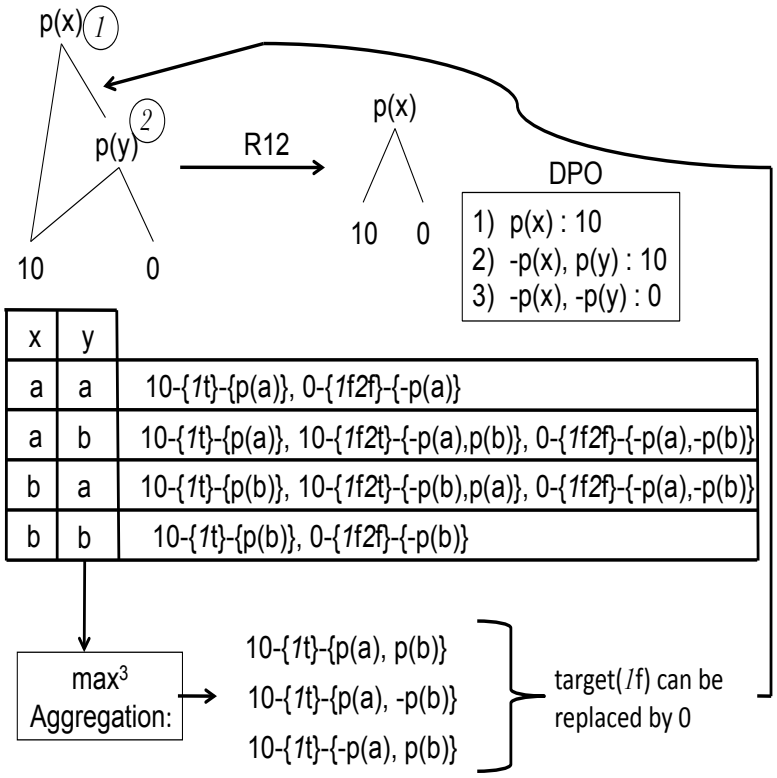

Figure 3: An example of reduction operator R12 for FODDs. Each entry of the form value-{path}-{interpretation} in the table (enclosing angle brackets removed in figure to improve readability) expresses the value obtained by running the valuation of the corresponding row through the diagram under an equivalence class of interpretations. The max^3 aggregation function then calculates the possible aggregates that could be generated under different equivalence classes of interpretations. Since the edge *1f* does not appear in any of the paths in the result of max^3, it is not important toward determining the map and can be removed.

objects $\{a, b\}$ *and the variables x and y appearing in the left FODD. For example, the traversal of valuation* $\{x/a, y/b\}$ *through the FODD has 3 possible eventualities. Either it reaches a* 10 *leaf by traversing path* $\{1t\}$ *(which is short for the path consisting of the true edge of node* 1*), under abstract interpretation* $\{p(a)\}$*, or it reaches a* 10 *leaf by traversing path* $\{1f2t\}$ *(which is short for the path consisting of the false edge of node* 1 *followed by the true edge of node 2), under abstract interpretation* $\{\neg p(a), p(b)\}$ *or (in all other cases) it reaches a* 0 *leaf.*

Note that the different behaviors of a valuation are mutually exclusive because the abstract interpretations associated with these behaviors partition the space of worlds. Any interpretation must be consistent with exactly one of these abstract interpretations and hence must force the behavior corresponding to that abstract interpretation on the valuation.

Thus, as in Figure 3, with the help of the getBehaviors procedure we can tabulate the possible behaviors of all valuations over a set of domain objects. The next step is to generate all possible ways in which an aggregate value can be derived. This can be done without enumerating all interpretations. The table of potential behaviors gives sufficient information to list all possible

ways to aggregate over the set of all valuations, by considering all combinations of behaviors over the set of valuations. Every combination, as long as it is consistent, produces the map as an aggregate value. To facilitate reduction the aggregation has to be augmented so as to expose the valuations and paths that prove to be important for determining the map. Intuitively, paths that were not shown to be important in spite of listing all possible ways to aggregate over the set of all valuations can be removed. To this end, the next section introduces variants of the *max* aggregation function, max^2 and max^3.

3.1. Generalized Aggregation Function and the R12 Reduction

When calculating the map, the max aggregation operation is applied to values obtained by evaluating the FODD under different valuations. As discussed above, for R12, we are interested not just in the aggregate value but also in information that will help us identify which edges are used to determine the map. Toward that, when calculating the maximum, we collect information about the winning path, the valuation that leads to it, and the interpretation (captured by the ground path formula) for which this happens. To enable the such accounting we define three variants of the *max* aggregation operator.

max^1:. The first variant max^1 is the usual aggregation operator that given a set of values $\{v_1, \cdots v_n\}$ returns the aggregate $v = max(\{v_1, \cdots v_n\})$.

max^2:. requires a DPO to calculate its output. The input to max^2 is a set of 3-tuples of the form $\langle v_i, path_i, I_i\rangle$ with the intention that each 3-tuple was produced by getBehaviors on a different valuation ζ_i. The output is a 3-tuple $\langle v_o, path_o, I_o\rangle$ where:

1. $v_o = max^1(\{v_1, v_2 \cdots v_n\})$.
2. $I_o = \bigcup_{i=1}^{n} I_i$.
3. $path_o = path_i$ and $path_i$ has the least index in the DPO among paths with value v_o.

In other words, max^2 takes as input one possible behavior from every valuation (one entry from each row in the valuation table in Figure 3) and aggregates the result, recording the winning path, and the interpretation that induces the corresponding behavior on each valuation.

Example 5. *The example in Figure 3 shows the DPO and the 3 possible aggregation results derived from the table. Each of the 3 results is derived using the max^2 variant. For example, aggregating over*

- $\langle 10, \{1t\}, \{p(a)\}\rangle$ *for* $\{x/a, y/a\}$,
- $\langle 10, \{1t\}, \{p(a)\}\rangle$ *for* $\{x/a, y/b\}$,
- $\langle 10, \{1t\}, \{p(b)\}\rangle$ *for* $\{x/b, y/a\}$,
- $\langle 10, \{1t\}, \{p(b)\}\rangle$ *for* $\{x/b, y/b\}$,

using the max^2 variant gives $\langle 10, \{1t\}, \{p(a), p(b)\}\rangle$ indicating that there is a possible aggregation where the path consisting of the edge $\{1t\}$ is instrumental in determining the map.

The example illustrates that max^2 captures the combined behavior of all valuations on the interpretation I_0 which is part of its output. As motivated above, we would like to capture this information for all possible interpretations. Instead of enumerating interpretations, we generate all possible scenarios by considering all possible ways in which rows in the table produced by getBehaviors can be combined. This is done by max^3.

max^3**:**. requires a DPO to calculate its output. The input to max^3 is a set of sets of 3-tuples, where each set of 3-tuples is associated with a valuation (this corresponds to the entire table from Figure 3), denoted as $T = \{\langle valuation_1 - valueset_1\rangle, \langle valuation_2 - valueset_2\rangle, \cdots \langle valuation_n - valueset_n\rangle\}$. Let T' be the Cartesian product of $\{valueset_\ell\}$ so that $e_i \in T'$ is a set of tuples $\langle value, path, Interpretation\rangle$.

$max^3(T)$ is defined as

$$max^3(T) = \{\langle value_r, path_r, I_r\rangle = max^2(e_i) \mid e_i \in T',\ value_r \geq 0 \text{ and } I_r \text{ is consistent}\}.$$

Thus, $max^3(T)$ is the collection of results of max^2 applied to each element of T' but restricted to the cases where the combined interpretation is consistent and the aggregate value is greater than zero.

Example 6. *The example in Figure 3 shows the result of applying* max^3 *to the elements in the table. There are* $2 \times 3 \times 3 \times 2 = 36$ *possible combinations of valuation behaviors, and hence* 36 *elements in* T' *and corresponding calls to* max^2*. However, only* 3 *of these combinations result in a consistent combined interpretation and positive value. For example, under the given DPO,* $max^2(\{\langle 10, \{1t\}, \{p(a)\}\rangle, \langle 10, \{1t\}, \{p(a)\}\rangle, \langle 10, \{1f2t\}, \{p(a), \neg p(b)\}\rangle, \langle 10, \{1t\}, \{p(b)\}\rangle\}) = \langle 10, \{1t\}, \{p(a), p(b), \neg p(b)\}\rangle$ *is omitted from the result of* $max^3(T)$ *because the combined abstract interpretation is inconsistent. Aggregations resulting in* 0 *value are ignored because* 0 *being the smallest obtainable value, is uninteresting under the max aggregation semantics. Observe that in this example, the path* $\{1t\}$ *is the only instrumental path. Intuitively this implies that the target of any edge not on this path (for instance edge* $1f$*) can be set to* 0 *without changing the map. The resulting FODD is shown on the right.*

Example 7. *Consider the example of Figure 3 but with a DPO that reverses the order of paths 1 and 2. In this case the table produced by getBehaviors is identical, and so is the aggregated value. But the maximizing paths are not the same. The three outputs of* max^3 *are* $\langle 10, \{1t\}, \{p(a), p(b)\}\rangle$, $\langle 10, \{1f2t\}, \{p(a), \neg p(b)\}\rangle$, *and* $\langle 10, \{1f2t\}, \{\neg p(a), p(b)\}\rangle$*. Thus in this case both paths are instrumental and no reduction is achieved. This illustrates that the choice of DPO can be important in reducing a diagram. However, it is not clear how to best choose the DPO. A preference for shorter paths that defaults to lexicographic ordering over equal length paths makes for an easy implementation but may not be the best. Our implementation [11, 12] heuristically alternates this DPO and its reverse in hope of enabling more reductions.*

The reduction is formalized in procedures 2 and 3.

Procedure 2. *R12(FODD B)*

1. *Let PL be a DPO for B.*
2. *Let O be a set of v objects where v is the number of variables in B.*
3. *Let U be the set of all possible valuations of the variables in B over O.*
4. *Let S be the output of Reduction-Aggregation(B, U, PL).*
   That is, $S = \{\langle value_1, path_1, I_1\rangle, \langle value_2, path_2, I_2\rangle, \cdots \langle value_n, path_n, I_n\rangle\}$.
5. *Let* E' *be the set of all edges that appear on any path* $path_i$ *in any 3-tuple in the set* S.
6. *Define* $E = B_E - E'$*, where* B_E *is the set of all edges in B.*
7. *For all edges* $e \in E$*, set target(e) in B to* 0 *to produce FODD* B'.
8. *return* B'.

Procedure 3. *Reduction-Aggregation(FODD B, set of valuations U, DPO PL)*

1. *Let* $Val = \{\}$.
2. *Do for every valuation* $\zeta \in U$.
   (a) *valueset = getBehaviors(*ζ, {}, {}, B_{root}*).*
   (b) *Add the entry* $\langle \zeta - valueset \rangle$ *to Val.*
3. *Let* $T = max^3(Val)$ *under PL.*
4. *return* T.

3.2. Proof of Correctness

This section shows that the R12 procedure removes exactly the right edges on its input FODD. The proof relies on the next lemma which shows that every instrumental path, for any potential interpretation I, is discovered by the procedure. This is shown by arguing that a small portion of I suffices for this purpose and that such a portion is constructed by R12.

Lemma 1. *If a path* p_i *in FODD B is instrumental under PL, and the path reaches a non-zero leaf, then there exists an interpretation* I_o *such that* $\{leaf(p_i), p_i, I_o\}$ *is in the set S calculated in Step 4 of the R12 procedure.*

Proof: If p_i is instrumental under PL then there exist I and ζ such that $Path_B(I, \zeta) = p_i$ and such that for all η, $Path_B(I, \eta) = p_j$ implies $j \geq i$. Let O' be the set of objects in I that participate in ζ. Clearly $1 \leq |O'| \leq |O|$ where O is the set of objects constructed in Step 2 of the algorithm. Let o_1 be an object in O'. Add $|O| - |O'|$ new objects to O' to make the sets O and O' equal in size. Construct interpretation I' by first projecting I to include only the objects in O' and then defining truth values and predicates over the new objects to behave identically to o_1.

Since I' includes the relevant portion of I the valuation ζ traverses p_i under I'. Additionally, if there exists a valuation $\hat{\zeta}$ such that $Path_B(I', \hat{\zeta}) = p_j$ and $j < i$, we can construct valuation $\hat{\zeta}'$ by replacing the new objects in $\hat{\zeta}$ by o_1 so that $Path_B(I, \hat{\zeta}') = p_j$. But this is not possible by the assumption. Therefore we conclude that for all η, $Path_B(I', \eta) = p_j$ implies $j \geq i$.

Let U be the set of all valuations of the variables in B over O'. Let $I_o = \bigcup_{\eta \in U} PF(Path_B(I', \eta))\eta$. That is, I_o includes all the atoms of I' that participate in traversing paths in B for all $\eta \in U$. By construction, the corresponding parts $PF(Path_B(I', \eta))\eta$ will be included in the *valueset* returned by the getBehaviors procedure. Clearly $I_o \subseteq I'$. Therefore if I' is consistent then so is I_o. By the definition of max^3, $S = max^3(Val)$ under PL must contain $\{leaf(p_i), p_i, I_o\}$ when $leaf(p_i)$ is non-zero. ■

The proof of the previous lemma implicitly assumes that the signature does not include equality, whose truth value changes when the objects are reassigned. The lemma and all subsequent discussion can allow for equality by having steps 2-4 of R12 repeated for object set sizes up to v and step 5 take the union of exposed edges. This makes for longer arguments without adding any significant insight and we therefore focus on the simpler version in the paper.

The previous lemma implies that we discover all edges on instrumental paths and this in turn implies that removing other edges does not change the map of the diagram. This intuition is captured in the next lemma and theorem.

Lemma 2. *If there exists an instrumental path under PL that contains the edge e in B and the path reaches a non-zero leaf, then e is in the set* E' *calculated in Step 5 of the R12 procedure.*

Proof: If there is an instrumental path $p_i \in PL$ that contains the edge e and reaches a non-zero leaf, then by Lemma 1 there exists an interpretation I_o such that $\{\text{leaf}(p_i), p_i, I_o\} \in S$. By definition of E', $e \in E'$. ■

Theorem 1 (soundness). *For any FODD B, if FODD B' is the output of R12(B), then for all interpretations I, $Map_B(I) = Map_{B'}(I)$.*

Proof: By the definition of R12, the only difference between B and B' is that some edges that pointed to subFODDs in B, point to the 0 leaf in B'. These are the edges in the set E at the end of the R12 procedure. Therefore any valuation crossing these edges achieves a value of 0 in B' but could have achieved a higher value in B under the same interpretation. Valuations not crossing these edges will achieve the same value in B' as they did in B. Therefore for any interpretation I and valuation ζ, $Map_B(I,\zeta) \geq Map_{B'}(I,\zeta)$ and hence $Map_B(I) \geq Map_{B'}(I)$.

Fix any interpretation I and $v = Map_B(I)$. Let ζ be a valuation such that $Map_B(I,\zeta) = v$. If there is more than one ζ that gives value v, we choose one whose path p_j has the least index in PL. By definition, p_j is instrumental and by Lemma 2, either $\text{leaf}(p_j) = 0$ or none of the edges of p_j are removed by R12. In both cases, $Map_{B'}(I,\zeta) = v = Map_B(I)$. By the definition of the max aggregation semantics, $Map_{B'}(I) \geq Map_{B'}(I,\zeta)$ and therefore $Map_{B'}(I) \geq Map_B(I)$. ■

We next show that the reduction achieved by R12 is the best possible with respect to our notions of DPO and instrumental paths.

Theorem 2 (maximum reduction w.r.t. DPO). *If no path crossing edge e and reaching a non-zero leaf in B is instrumental under PL, then R12 removes e.*

Proof: By definition the set of all edges in B is partitioned into sets E and E'. Now, by construction, if $e \in E'$, then there exist a path $p_i \in PL$ and an interpretation I_o such that e is an edge on p_i, $\text{leaf}(p_i)$ is non-zero and $\{\text{leaf}(p_i), p_i, I_o\}$ is in the set S calculated in Step 4 of the R12 procedure. The existence of $\{\text{leaf}(p_i), p_i, I_o\}$ in S implies that under I_o, there is a valuation $\zeta \in U$ such that $Path_B(I_o,\zeta) = p_i$ and for all $\eta \in U$, $Path_B(I_o,\eta) = p_j$ implies $j \geq i$. Therefore p_i is instrumental. Therefore all edges in E' belong to some instrumental path. This implies that e from the statement of the theorem is not in E' and therefore it is removed by R12. ■

3.3. Discussion

The R12 procedure provides a comprehensive reduction operation for FODDs, by guaranteeing maximum reduction w.r.t. a DPO *on its own*. This is in contrast with the fact that all previous published reductions, taken together, do not provide the same guarantee. The main reason is that previous reduction operators rely on theorem proving over *single path formulas* or edge implications. As the following example shows there are cases where such reduction operators fail to reduce a diagram but R12 is successful.

Example 8. *Figure 4 shows an example where R12 succeeds but previous reductions fail. Notice that there are two paths reaching the* 10 *leaf in the left FODD. In this diagram, whenever a valuation reaches the* 1 *leaf there is another valuation that reaches the* 10 *leaf through one of the two paths. However, neither of the path formulas are individually implied by the formula for the path reaching the* 1 *leaf. Similarly neither of the edge formulas for the edges terminating in the* 10 *leaf are implied by the edge formula for the edge terminating in the* 1 *leaf. R12, on the other hand, relies on model checking and is able to reduce the FODD on the left to the FODD on the right.*

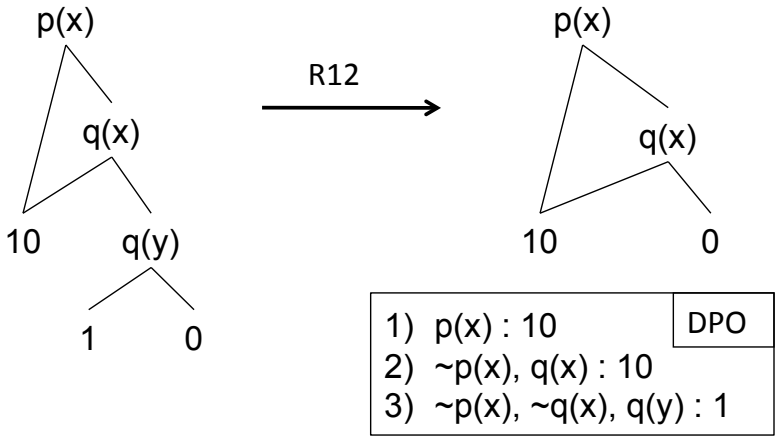

Figure 4: Example where R12 can reduce the diagram but previous reductions fail.

It is important to note, however, that one can in principle define a theorem proving reduction giving the same guarantees.[1] For example, to state that path i is instrumental one can write

$$[\exists x_{p_i} PF(p_i)] \wedge (\wedge_{j<i}[\neg \exists x_{p_j} PF(p_j)]).$$

The path is instrumental if and only if this formula is satisfiable. Thus theorem proving can provide maximum reduction with respect to a DPO in the same way that R12 does. However, the theorem proving may be complex because it involves disjunctive reasoning. In fact, the R10 reduction [11] performs similar reasoning except that it checks the paths $j \leq i$ one at a time in order to make for simpler theorem proving, and therefore does not provide the same guarantees. More importantly, this formulation has a significant disadvantage (shared with R10) in that it enumerates all the paths whose index is smaller than i. The main point in adopting a decision diagram representation over a decision tree, is the fact that a diagram can be exponentially smaller because of repeated sub-trees that are represented only once in a decision diagram. In other words, the number of paths in a diagram can be exponential in its size. In this case, enumerating the paths in a DPO is not practical and the theorem proving formulation will fail. In contrast, R12 does not need to generate the DPO explicitly. Instead the procedure only needs to be able to compare two paths (in max^3) and decide which one is higher in the DPO. As mentioned above this is easy to perform efficiently for suitably chosen DPOs, such as ones preferring shorter path and using lexicographic ordering. Therefore, when the number of paths is large R12 will be superior to the theorem proving formulation.

On the other hand the complexity of R12 is also high in that it involves the enumeration of all possible valuations, and is thus exponential in the number of variables. Therefore, a direct implementation of R12 as specified here will not be practical for FODDs with a large number of variables. In recent work we have introduced heuristic variants of R12 that are more efficient and have shown that they lead to significant speedup over theorem proving reductions [12].

[1] We are grateful to the anonymous reviewer who suggested this.

Finally, R12 is distinguished from previous reductions by the fact that it employs the aggregation function of the FODD itself as its main subroutine. Therefore, one can imagine generalizing it for diagrams containing other aggregation functions. Indeed the next two sections define such generalized diagrams and model checking reductions for them. Corresponding generalized variants of the reductions based on theorem proving are not easy to obtain.

4. Generalized FODDs: Syntax and semantics

The *max* aggregation of FODDs makes them sufficiently expressive to represent many planning problems of interest. However, since the *max* aggregation mirrors existential quantification over the variables of the FODD, many other functions over logical spaces cannot be represented by FODDs. These functions could be represented if the aggregation function was more complex. This idea is captured in the following definition.

Definition 3. *An aggregation function is any function f that takes as input a non-empty set of real values and returns a real value.*

Concrete examples of aggregation functions that are discussed further below include *max*, *min*, *sum*, and *mean*. Other functions like *product*, *variance* and so on are also possible. We will pay special attention to *min* aggregation that allows us to capture universally quantified formulas. In this section and the next, we discuss the properties of generalized FODDs using arbitrary aggregations and the operations that can be performed to manipulate them. We start by a formal definition of Generalized First-Order Decision Diagrams.

Definition 4. *A Generalized First-Order Decision Diagram (GFODD) is a 2-tuple $\langle V, D\rangle$, where (1) V is an ordered list of pairs (v_i, op_{v_i}), where v_i is a variable and op_{v_i} is an aggregation operator,*
(2) The variables v_i are distinct, that is, v_i has exactly one aggregation operator in V,
(3) D is a FODD except that the leaves can be labeled by a special character $\mathcal{D}$ (for discard).

An example of a GFODD is given in Figure 5. The corresponding list V as in the formal specification above is $[(c, \max), (b, \min)]$ but we use the more intuitive alternative notation $\max(c)\min(b)$ or $\max_c \min_b$ where this is clear from the context.

The discard value $\mathcal{D}$ in the definition above allows for some paths in the diagram to provide no value. This can be useful when multiple types of aggregations are used because one does not need to have a "default value" (like the value zero for max aggregation) which does not affect the result. This simplifies the implementation and analysis of one of the reductions presented below.

4.1. Semantics of GFODDs

The semantics for GFODDs follow the same approach of FODDs in that they first calculate the map for all valuations and then aggregate these values. Whereas in FODDs we take a maximum over these values the computation for GFODDs is more complex and follows the aggregation function. To simplify the notation, in the following when $B = \langle V, D\rangle$ and ζ is a valuation we sometimes refer to $Map_D(I, \zeta))$ as $Map_B(I, \zeta)$.

Formally, let $B = \langle V, D\rangle$ be a GFODD where $V = [(v_1, op^1_{v_1}), (v_2, op^2_{v_2}) \cdots (v_n, op^n_{v_n})]$ and let I be an interpretation. The map value $Map_B(I)$ is defined by the following steps:

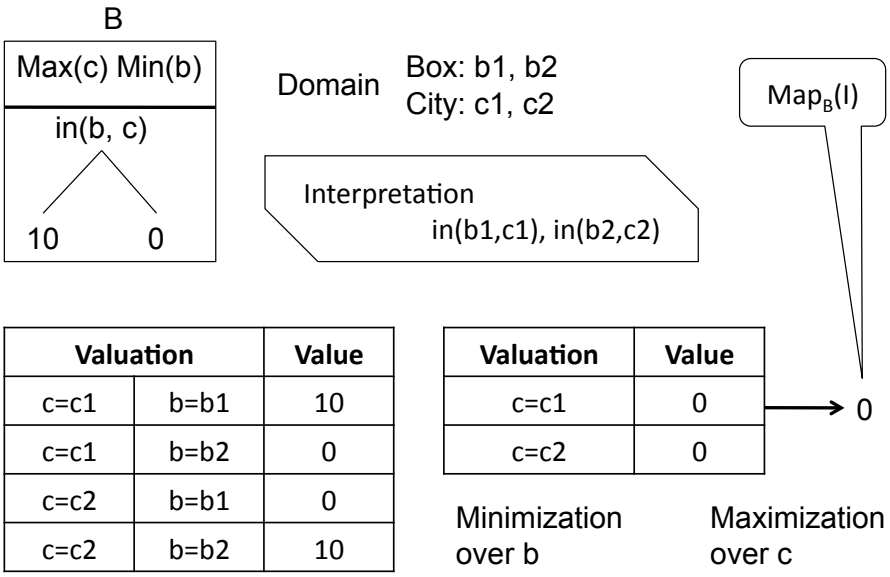

Figure 5: A Generalized FODD Example.

(1) Each valuation ζ, mapping $v_1 \cdots v_n$ to the domain of interpretation I is associated with a value $Map_D(I, \zeta)$. (2) We can now divide up these valuations into blocks. All valuations in a block have the same assignment of values to variables $v_1 \cdots v_{n-1}$ but they differ in the value of the variable v_n. (3) We then "collapse" each block to a single valuation over variables $v_1 \cdots v_{n-1}$ by eliminating the variable v_n and replacing the set of associated values by their aggregate value. If all the values in the block have the value $\mathcal{D}$ then the aggregate value is $\mathcal{D}$. Otherwise, we remove $\mathcal{D}$ from the set of values and apply op^n to the remaining set. This yields a table with the set of all possible valuations defined over the variables $v_1 \cdots v_{n-1}$ each associated with a value (which was obtained by aggregating over the valuations of variable v_n in the block). (4) We repeat the same procedure for variables v_{n-1} to v_1 to produce a final aggregate value. The value of $Map_B(I)$ is this final aggregate value.

The treatment of $\mathcal{D}$ values in step (3) captures the idea of ignoring the corresponding paths when calculating the aggregate value. Thus any $\mathcal{D}$ inputs to an aggregation operator are ignored and if all values are $\mathcal{D}$ this information is passed on to the next level.

Example 9. *The GFODD B in Figure 5 captures the following statement from the logistics domain: There exists a city c such that for all boxes b, box b is in city c. The output of B is* 10 *if all boxes are in one city and* 0 *otherwise.*[2] *In the example GFODD shown,* $V = [(c, \max), (b, \min)]$. *Aggregation is done from right to left, one variable at a time. In the example, the table on the left shows the value of* $Map_B(I, \zeta)$ *for every possible valuation* ζ. $Map_B(I)$ *is calculated by first*

[2]In this example, to keep the GFODD diagram simple, we assume the variables are typed and use only valuations that conform to the types of the variables. Had we used all possible valuations over the set of objects $\{b_1, b_2, c_1, c_2\}$, the diagram would have been more complicated as it would have had to represent the formula $\exists c, \forall b, city(c) \wedge [box(b) \rightarrow in(b, c)]$.

aggregating the values $Map_B(I, \zeta)$ over all assignments for the variable b using the min aggregation. This yields the table in the middle. We then aggregate all of the produced values over all assignments for variable c using the max operation. The resulting value, 0 in this case, is $Map_B(I)$.

In the following we need a notation to refer to the map value and its calculation. The procedure described can be seen to perform aggregation over variables in V by nesting aggregation operators from left (outermost) to right (innermost). i.e.

$$Map_B(I) \quad = \quad op^1_{v_1}\left[op^2_{v_2}\left[\cdots\left[op^n_{v_n}\left[Map_B\left(I,[v_1,v_2,\cdots v_n]\right)\right]\right]\cdots\right]\right].$$

The term in the center, $Map_B(I, [v_1, v_2, \cdots v_n])$, is the value obtained by running a valuation defined by an assignment to the variables $v_1, \cdots v_n$ through B under I. In order to reduce the notational clutter, in the rest of the paper we will drop brackets so that the above equation looks as follows

$$\begin{aligned} Map_B(I) &= op^1_{v_1} op^2_{v_2} \cdots op^n_{v_n}[Map_B(I,[v_1,v_2,\cdots v_n])] \\ &= op^1_{v_1} op^2_{v_2} \cdots op^{n-1}_{v_{n-1}} op^n[c_1^{[v_1\cdots v_{n-1}]} \cdots c_m^{[v_1\cdots v_{n-1}]}] \end{aligned}$$

where each $c_i^{[v_1\cdots v_{n-1}]}$ is a value corresponding to a different object assignment to variable v_n in the block defined by the values assigned to the variables $v_1 \cdots v_{n-1}$.

4.2. Basic Properties of GFODDs

Several observations can be made on GFODDs and their semantics. First, the order of variables in V is important. Changing the order of the variables can obviously change the map of the diagram.

Second, FODDs form a proper subclass of GFODDs where the aggregation operator associated with every variable is *max*. In this case, due to properties of the *max* aggregation, the order of variables in V is not important.

Third, GFODDs with $0/1$ leaves express the same functions as closed, function-free first-order formulas. In particular this can be done by employing the *min* aggregation operator over universally quantified variables and the *max* aggregation operator over existentially quantified variables. To see this consider any GFODD $\langle V, D\rangle$ with $0/1$ leaves and let F be a quantifier-free formula capturing the disjunction of path formulas for paths leading to the 1 leaf. Then interpreting V as quantifiers V, F is a closed first order formula that evaluates to true exactly when $\langle V, D\rangle$ evaluates to true. On the other hand, given a closed first-order formula in prenex normal form V, F where F is in disjunctive normal form, we can build a FODD D by representing each conjunct in F as a FODD directly and then represent their disjunction using the apply procedure of Wang et al. [9]. Now, as above $\langle V, D\rangle$ is equivalent to V, F.

Finally, the definition above allows the final aggregate value to be $\mathcal{D}$ in the case where all reachable paths for I yield the value $\mathcal{D}$. To ensure that GFODDs always represent well defined functions we disallow this case.

Definition 5. *A GFODD B is legal iff it obeys the GFODD syntax and for all interpretations I there is a valuation ζ such that $Map_B(I, \zeta) \neq \mathcal{D}$.*

op_c	op^a	safe/unsafe
$\oplus$	*max*	safe
$\oplus$	*min*	safe
$\oplus$	*sum*	unsafe
$\oplus$	*avg*	safe
$\otimes$	*max*	safe
$\otimes$	*min*	safe
$\otimes$	*sum*	safe
$\otimes$	*avg*	safe
max	*max*	safe
max	*min*	safe
max	*sum*	unsafe
max	*avg*	unsafe

Table 1: List of some safe and unsafe pairs for operators.

4.3. Combining GFODDs

So far we have focused on the syntax and semantics of GFODDs that can represent complex functions over relational structures. The utility of such a representation, though, is in performing operations over such functions, for example *max* (taking the maximum), + (addition) and $\times$ (multiplication). We call these operators *combination operators* and provide an algorithm Ex-apply to implement them. Notice that combination operators operate on functions and they are different from aggregation operators that operate on sets of real values. The next definition provides the intended meaning of combination.

Definition 6. *GFODD B is a combination of GFODDs B_1 and B_2 under the binary combination operator op_c iff for all interpretations I, $Map_B(I) = Map_{B_1}(I)\ op_c\ Map_{B_2}(I)$.*

In the above we assume that the functions represented by B_1 and B_2 are independent, i.e., that the variables they aggregate over do not constrain each other. In principle, one could try to define the meaning of combination when a variable appears in both diagrams and aggregated similarly. However, this seems awkward and is not necessary for the calculus of functions we use. Therefore, in the following we assume that the functions being combined do not share variables, that is, their quantifier-free portion is standardized apart.

Aggregation and combination operators can interact, complicating the result of the combination operation. In the following we show that in some cases this does not happen and we can essentially use the algorithm that combines FODDs to combine GFODDs. This is captured by the following condition on combination and aggregation operators:

Definition 7. *A combination operator op_c and an aggregation operator op^a are a safe pair iff op_c distributes over op^a, that is, iff for any set of non-negative values $x_1, x_2, \ldots, x_k$ and any non-negative constant b it holds that*

$$op^a(x_1, x_2, \ldots, x_k)\ op_c\ b = op^a(x_1\ op_c\ b, x_2\ op_c\ b, \ldots, x_k\ op_c\ b)\ .$$

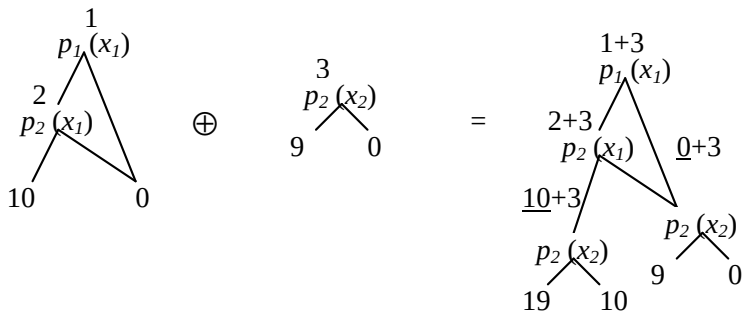

Figure 6: A simple example of adding two FODDs.

Example 10. *The aggregation operator max and combination operator + form a safe pair because for any set* $S = \{c_1 \cdots c_m\}$ *and constant* b, $max\{c_1 \cdots c_m\} + b = max\{c_1 + b, \cdots c_m + b\}$. *The aggregation operator mean and the combination operator max do not form a safe pair. For example* $max\{mean\{1,5,3\},4\} = 4$ *but* $mean\{max\{1,4\}, max\{5,4\}, max\{4,4\}\} = mean\{4,5,4\} = 4.33$.

Table 1 summarizes the safe and unsafe pairs for operators that are of interest to us. We later use the fact that the *max* and *min* aggregation operators are safe with all the combination operators listed. As mentioned above this condition will allow us to use a simple algorithm for combination. The cases that are not safe might still be processed using other algorithms but we leave the details of this for future work.

We next review the details of the procedure apply(B_1,B_2,op) for combining FODDs B_1 and B_2 under operation op [9]. Recall that FODDs use an ordering over the atoms labeling nodes, so that atoms higher in the ordering are always higher in the diagram. Let p and q be the roots of B_1 and B_2 respectively. The apply procedure chooses a new root label (the lower among labels of p, q) and recursively combines the corresponding sub-diagrams, according to the relation between the two labels ($\prec$, $=$, or $\succ$).

Example 11. *Figure 6 illustrates the operation of the apply procedure. In this example, we assume predicate ordering* $p_1 \prec p_2$, *and parameter ordering* $x_1 \prec x_2$. *Non-leaf nodes are annotated with numbers and numerical leaves are underlined for identification during the execution trace. For example, the top level call adds the functions corresponding to nodes 1 and 3. Since* $p_1(x_1)$ *is the smaller label it is picked as the label for the root of the result. Then we must add both left and right child of node 1 to node 3. These calls are performed recursively to yield the diagram on the right.*

The next lemma, by Wang et al. [9], shows that the apply procedure provides the correct map for every valuation:

Lemma 3 ([9]). *Let* $C = apply(A, B, op)$, *then for any* I *and* ζ, $MAP_A(I,\zeta)$ *op* $MAP_B(I,\zeta) = MAP_C(I,\zeta)$.

We next define the combination procedure for GFODDs and prove its correctness.

Definition 8. *Let $B_1 = \langle V_1, D_1 \rangle$ and $B_2 = \langle V_2, D_2 \rangle$ be GFODDs where V_1 and V_2 do not have any variables in common, and let op_c be any combination operator.*

Ex-apply(B_1, B_2, op_c) returns $\langle V, D \rangle$, where

1. *V is the aggregation function obtained by appending V_2 to V_1.*
2. *$D = apply(D_1, D_2, op_c)$.*

To show that this procedure is correct, we start by observing that when combining a diagram B with a constant (a degenerate diagram that has just one leaf node whose value is that constant) one can push the combination operation to the leaves.

Lemma 4. *Let $B = \langle V, D \rangle$ be a GFODD, b a non-negative constant, and op_c a combination operator. If for every aggregation operator op^a in V, (op^a, op_c) is a safe pair, then, for all interpretations I, $Map_B(I)\ op_c\ b = op^1_{v_1} op^2_{v_2} \cdots op^n_{v_n}[Map_B(I, [v_1, v_2 \cdots v_n])\ op_c\ b]$.*

Proof: The proof is by induction on n, the number of operators (and variables) in V. By the semantics of GFODDs,

$$Map_B(I)\ \ op_c\ \ b \quad = \quad op^1_{v_1} \cdots op^n_{v_n}[Map_B(I, [v_1 \cdots v_n])]\ \ op_c\ \ b$$

When $n = 1$, we have

$$\begin{aligned} Map_B(I)\ \ op_c\ \ b \quad &= \quad op^1_{v_1}[Map_B(I, [v_1])]\ \ op_c\ \ b \\ &= \quad op^1_{v_1}[Map_B(I, [v_1])\ \ op_c\ \ b] \end{aligned}$$

because op^1 and op_c form a safe pair. Assume that the statement is true for all V of $n-1$ or fewer aggregation operators. Consider a V with n aggregation operators. We then have,

$$\begin{aligned} Map_B(I)\ \ op_c\ \ b \quad &= \quad op^1_{v_1} \cdots op^n_{v_n}[Map_B(I, [v_1 \cdots v_n])]\ \ op_c\ \ b \\ &= \quad op^1_{v_1}[c_1^{[v_1]} \cdots c_m^{[v_1]}]\ \ op_c\ \ b \\ &= \quad op^1_{v_1}[c_1^{[v_1]}\ \ op_c\ \ b\ \ \cdots\ \ c_m^{[v_1]}\ \ op_c\ \ b] \end{aligned}$$

because op^1 and op_c form a safe pair. Here each $c_i^{[v_1]} = op^2_{v_2} \cdots op^n_{v_n}[Map_B(I, [v_1, v_2 \cdots v_n])]$ for the i^{th} value of the variable v_1. By the inductive hypothesis we know that

$$op^2_{v_2} \cdots op^n_{v_n}[Map_B(I, [v_1, v_2 \cdots v_n])]\ \ op_c\ \ b \quad = \quad op^2_{v_2} \cdots op^n_{v_n}[Map_B(I, [v_1, v_2 \cdots v_n])\ \ op_c\ \ b].$$

Thus,

$$Map_B(I)\ \ op_c\ \ b \quad = \quad op^1_{v_1} op^2_{v_2} \cdots op^n_{v_n}[Map_B(I, [v_1, v_2 \cdots v_n]) op_c\ \ b].$$

■

The next theorem uses the lemma repeatedly with different constants to prove the correctness of Ex-apply.

Theorem 3. *Let $B_1 = \langle V_1, D_1 \rangle$ and $B_2 = \langle V_2, D_2 \rangle$ be GFODDs that do not share any variables and assume that op_c forms a safe pair with all operators in V_1 and V_2. Then $B = \langle V, D \rangle =$ Ex-apply(B_1, B_2, op_c) is a combination of B_1 and B_2 under operator op_c.*

Proof: Let $op^{i,j}$ and $v_{i,j}$ denote the i^{th} operator and variable respectively in V_j. V is a concatenation of V_1 and V_2 by the definition of Ex-apply. Therefore by the definition of the GFODD semantics, for any interpretation I,

$$Map_B(I) = op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} op^{1,2}_{v_{1,2}} \cdots op^{m,2}_{v_{m,2}} [Map_B(I, [v_{1,1} \cdots v_{n,1} v_{1,2} \cdots v_{m,2}])].$$

Since $D = apply(D_1, D_2, op_c)$, by Lemma 3 we have that for all interpretations I and valuations ζ, $Map_D(I, \zeta) = Map_{D_1}(I, \zeta)\ op_c\ Map_{D_2}(I, \zeta)$. In addition, since the variables in V_1 and V_2 are disjoint, we can write any valuation ζ as $\zeta_1\zeta_2$ such that ζ_1 is the sub-valuation of ζ over the variables in V_1 and ζ_2 is the sub-valuation of ζ over the variables in V_2. Thus we can write

$$Map_B(I) = op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} op^{1,2}_{v_{1,2}} \cdots op^{m,2}_{v_{m,2}} [Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c\ \ Map_{B_2}(I, [v_{1,2} \cdots v_{m,2}])].$$

Now the important observation is that since $Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}])$ does not depend on the variables in V_2, when aggregating over the variables in V_2, $Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}])$ can be treated as a constant. Since op_c forms a safe pair with all aggregation operators of V_2, by Lemma 4,

$$\begin{aligned} & Map_B(I) \\ = \ & op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} (Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c\ \ op^{1,2}_{v_{1,2}} \cdots op^{m,2}_{v_{m,2}} (Map_{B_2}(I, [v_{1,2} \cdots v_{m,2}]))) \\ = \ & op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} (Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c\ \ Map_{B_2}(I)). \end{aligned}$$

Similarly when aggregating over variables in V_1, $Map_{B_2}(I)$ can be treated as a constant because it does not depend on the value of any of the variables in V_1. Since op_c forms a safe pair with all the aggregation operators in V_1, by Lemma 4,

$$\begin{aligned} Map_B(I) &= op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} (Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}]))\ \ op_c\ \ Map_{B_2}(I) \\ &= Map_{B_1}(I)\ \ op_c\ \ Map_{B_2}(I) \end{aligned}$$

Thus by definition, B = Ex-apply(B_1, B_2, op_c) is a combination of B_1 and B_2 under the combination operator op_c. ■

The following theorem strengthens this result showing that Ex-apply has some freedom in reordering the aggregation operators while maintaining correctness. This property is useful for our solution of RMDPs.

Theorem 4. *Let $B_1 = \langle V_1, D_1 \rangle$ and $B_2 = \langle V_2, D_2 \rangle$ be GFODDs that do not share any variables and assume that op_c forms a safe pair with all operators in V_1 and V_2. Let $B = \langle V, D \rangle$ = Ex-apply(B_1, B_2, op_c). Let V' be any permutation of V so long as the relative order of operators in V_1 and V_2 remains unchanged, and let $B' = \langle V', D \rangle$. Then for any interpretation I, $Map_B(I) = Map_{B'}(I)$.*

Proof: Let $V_1 = F^1_1 F^1_2 \cdots F^1_k$ and $V_2 = F^2_1 F^2_2 \cdots F^2_k$ so that each F^i_j is a series of zero or more consecutive aggregation operators in V_i. Then $V' = F^1_1 F^2_1 F^1_2 F^2_2 \cdots F^1_k F^2_k$ represents a permutation of V such that the relative order of operators in V_1 and V_2 remains unchanged. By the semantics of GFODDs,

$$Map_{B'}(I) = F^1_1 F^2_1 \cdots F^1_k F^2_k Map_B(I, [v_{1,1} \cdots v_{n,1} v_{1,2} \cdots v_{m,2}])$$

where $v_{i,j,}$ is a variable in B_j. Now, by applying Lemma 3 we get

$$Map_{B'}(I) = F^1_1 F^2_1 \cdots F^1_k F^2_k [Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c\ \ Map_{B_2}(I, [v_{1,2} \cdots v_{m,2}])].$$

Since B_1 and B_2 do not share any variables, and op_c forms a safe pair with all operators in V_1 and V_2, we have the following sequence of equations where in each step we use Lemma 4 and the fact that one of the arguments is a constant with respect to the corresponding block of aggregation operators:

$$\begin{aligned} Map_{B'}(I) &= F_1^1 F_1^2 \cdots F_k^1 [Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}]) \;\; op_c \;\; F_k^2 Map_{B_2}(I, [v_{1,2} \cdots v_{m,2}])] \\ &= F_1^1 F_1^2 \cdots F_{k-1}^2 [F_k^1 Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}]) \;\; op_c \;\; F_k^2 Map_{B_2}(I, [v_{1,2} \cdots v_{m,2}])] \\ &= \cdots \\ &= F_1^1 \cdots F_k^1 Map_{B_1}(I, [v_{1,1} \cdots v_{n,1}]) \;\; op_c \;\; F_1^2 \cdots F_k^2 Map_{B_2}(I, [v_{1,2} \cdots v_{m,2}]). \end{aligned}$$

Finally by Theorem 3, the last term is equal to $Map_B(I)$ implying that $Map_{B'}(I) = Map_B(I)$. ■

5. Model Checking Reductions for GFODDs

The R12 procedure introduced in Section 3 can be extended to operate on GFODDs. In this section we present extensions of R12 for two forms of aggregation functions. The first is a set of diagrams using only *min* aggregation. The second is the set of diagrams with $max^* min^*$ aggregation. In this case the aggregation function consists of a series of zero or more *max* operators followed by a series of zero or more *min* operators. For this case we introduce two variants, $R12_{\mathcal{D}}$ and $R12_0$, with differing computational costs and quality of reduction. We will discuss each of those in turn starting with the R12 procedure for the *min* operator.

5.1. R12 for min aggregation

The case of *min* aggregation is obtained as a dual of the *max* aggregation case. However, it is worthwhile considering it explicitly as a building block for the next construction. The notion of instrumental paths here is the dual of the notion of instrumental paths for the *max* aggregation:

Definition 9. *If B is a GFODD with only the min aggregation function, and PL is the DPO for B, then a path $p_j \in PL$ is instrumental with respect to PL iff there is an interpretation I such that*

1. *there is a valuation, ζ, such that $Path_B(I, \zeta) = p_j$, and*
2. *for all valuations η, if $Path_B(I, \eta) = p_k$, then $k \leq j$.*

The generalized aggregation function for the *min* aggregation operator is the same as the one for the *max* operator except that the *max* is replaced by the *min* and no special treatment is given to paths reaching the 0 leaf. We thus have a min^3 generalized aggregation function. Notice that whereas for *max* aggregation we choose the reachable path with smallest index as instrumental (and record it in max^3), for min^3 we pick the reachable path with greatest index as instrumental. The reduction procedure is identical to the case of *max* aggregation except that min^3 is used instead of max^3 and that edges in E have the targets replaced by the discard value $\mathcal{D}$ instead of 0. This is not strictly necessary, as we can replace the target of the edges with a large value (or ∞). But it is useful in preparation for the next construction. A trivial adaptation of the proofs in the previous section yields the corresponding properties for *min* aggregation.

Lemma 5. *If a path p_i in GFODD B is instrumental under PL, then there exists an interpretation I_o such that $\{leaf(p_i), p_i, I_o\} \in S$.*

Lemma 6. *If there exists an instrumental path under PL that contains the edge e in B then* $e \in E'$.

Theorem 5 (soundness). *For any GFODD B using only min aggregation, if GFODD B′ is the output of R12(B), then for all interpretations I,* $Map_B(I) = Map_{B'}(I)$.

Theorem 6 (maximum reduction w.r.t. DPO). *If no path crossing edge e in B is instrumental under PL, then R12 removes e.*

*5.2. Model Checking Reduction for max*min* Aggregation*

This section is concerned with GFODDs employing *max*min** aggregation. The aggregation function consists of a series of zero or more *max* operators followed by a series of zero or more *min* operators. The aggregation function V is therefore split into V^l – the variables aggregated over using the *max* aggregation operator, and V^r – the variables aggregated over using the *min* aggregation operator. Thus, $V = V^l V^r$. We use the superscript l and r (for left and right) to refer to the corresponding blocks of *max* and *min* variables. The set U of all possible valuations of the variables in B can be split into U^l and U^r, the sets of all valuations over the variables in V^l and V^r respectively. Any valuation $\zeta \in U$ can then be written as $\zeta^l\zeta^r$ where $\zeta^l \in U^l$ and $\zeta^r \in U^r$. Thus by the definition of GFODD semantics, for any interpretation I,

$$\begin{aligned} Map_B(I) &= op^1_{v_1} \cdots op^n_{v_n}[Map_B(I, [v_1 \cdots v_n])] \\ &= max_{\zeta^l \in U^l}[min_{\zeta^r \in U^r}[Map_B(I, \zeta^l\zeta^r)]]. \end{aligned}$$

5.2.1. The procedure $R12_{\mathcal{D}}$

Our first reduction operator captures a simple notion of instrumental paths. The intuition is that we can view model evaluation as if performed in blocks. First, for every ζ_l, an assignment of objects to V^l (of max variables), we perform a min competition among all valuations to V^r. Each ζ_l is then associated with a path and value that won the min competition and we perform a max competition among the corresponding values. Therefore, if a path never wins any min competition we may be able to change its value without changing the map of the diagram. The new value must be chosen carefully so that it does not affect any min or max competition on any interpretation, and this requires complex analysis. Instead of choosing such a concrete value we change the value to $\mathcal{D}$. This makes sure that the path will not win any min or max competitions and hence does not change the final value of the diagram.

We proceed with the technical details of this idea. A path is instrumental if it wins a min competition for some interpretation I.

Definition 10. *If B is a GFODD with the max*min* aggregation function, and P is a DPO for B, then a path* $p_i \in P$ *is instrumental iff there is an interpretation I and valuation* $\zeta = \zeta^l\zeta^r$, *where* $\zeta \in U$, $\zeta^l \in U^l$ *and* $\zeta^r \in U^r$, *such that,*

1. $Path_B(I, \zeta) = p_i$,
2. *For every* $\eta^r \in U^r$, *if* $Path_B(I, \zeta^l\eta^r) = p_j$, *then* $j \leq i$ *under P.*

The $R12_{\mathcal{D}}$ procedure for the *max*min** aggregation is identical to the R12 procedure for the *min* aggregation with the following exceptions.

1. Recall that the variables are split into V^l with max aggregation followed by V^r using min aggregation. The set U of valuations is built in the following way. Let O^l be a set of $|V^l|$ objects and O^r a set of $|V^r|$ objects where O^l and O^r are disjoint. Let U^l be the sets of all possible valuations of the variables in V^l over the objects in O^l and let U^r be the set of all possible valuations of the variables in V^r over the objects in the union of O^l and O^r. The set U is then defined as $U = \{\zeta^l\zeta^r \mid \zeta^l \in U^l \text{ and } \zeta^r \in U^r\}$.
   The set of valuations U therefore captures an arbitrary valuation of the variables in V^l to objects in O^l that are not constrained. Similarly the valuation of V^r is not constrained in that it is allows to bind to objects in O^l or to other objects (for which O^r serves as unconstrained objects). The proof below shows that this set is sufficient to expose any instrumental paths.
2. The set S is defined as $S = \bigcup_{\zeta^l}$Reduction-Aggregation(B, U_{ζ^l}, PL), where U_{ζ^l} is the block of valuations corresponding to ζ^l. Thus the set Val in the procedure is divided into blocks, each containing a set of valuations with the same ζ^l. S is the union of the sets generated as a result of applying Reduction-Aggregation using min^3 to each block of Val.

Example 12. *Figure 7 shows a small example of this reduction. The process is similar to the R12 procedure for the max aggregation, except for the generalized aggregation function. A DPO is first established as shown. Sets* $O^l = \{a\}$ *and* $O^r = \{b\}$ *are constructed and the table (Val) is generated by running the getBehaviors procedure on the valuations generated from those. Finally, since Val consists of a single block (since only one variable is associated with the max operator),* $min^3(Val)$ *is evaluated to produce the 5* $\langle$*leaf, path, Interpretation*$\rangle$ *3-tuples as shown. For example combining* $0\text{-}\{1t2f3f\}\text{-}\{p(a), \neg q(a)\}$ *with* $10\text{-}\{1t2f3t4t\}\text{-}\{p(a), \neg q(a), q(b), r(b)\}$ *under* min^3 *we get* $0\text{-}\{1t2f3f\}\text{-}\{p(a), \neg q(a), q(b), r(b)\}$*. The targets of all edges other than the ones present in the paths of the resultant 3-tuples, and concretely the edge* $3t$*, can be replaced by the value* $\mathcal{D}$*.*

The proof of correctness follows the same outline as above but accounts for the extra aggregation operators. We first show that every instrumental path, for any potential interpretation I, is discovered by the procedure.

Lemma 7. *If a path* p_i *in GFODD B employing the* max^*min^* *semantics is instrumental under PL, then there exists an interpretation* I_o *such that* $\{leaf(p_i), p_i, I_o\}$ *is in the set S calculated by the* $R12_{\mathcal{D}}$ *procedure.*

Proof: If p_i is instrumental under PL then there exists an interpretation I over a set of objects O_I and a valuation $\zeta = \zeta^l\zeta^r$ such that $Path_B(I, \zeta) = p_i$ and for every η^r, if $Path_B(I, \zeta^l\eta^r) = p_j$, then $j \leq i$ under PL. Let O'^l be the set of objects that participate in ζ^l and let O'^r be the set of objects that participate in ζ^r but not in ζ^l. Clearly $1 \leq |O'^l| \leq |V^l|$ and $1 \leq |O'^r| \leq |V^r|$. Let $o'^l_1 \in O'^l$ and $o'^r_1 \in O'^r$. Add $|V^l| - |O'^l|$ new objects to O'^l and $|V^r| - |O'^r|$ new objects to O'^r.

Construct interpretation I' by first projecting I to include only the objects in O'^l and O'^r and then defining truth values and predicates over the new objects in O'^l and O'^r to behave identically to o'^l_1 and o'^r_1 respectively. Let O'^l and O'^r be the sets O^l and O^r used in the $R12_{\mathcal{D}}$ procedure to generate the set of valuations U. The set U can be split into blocks so that each valuation $\eta = \eta^l\eta^r$ belonging to U can be assigned to the block corresponding to η^l. Let U_{ζ^l} be the block corresponding to ζ^l.

Since $\zeta \in U_{\zeta^l}$, and I' contains the relevant portion of I, ζ traverses p_i under I'. Additionally if there is a valuation $\eta \in U_{\zeta^l}$ such that $Path_B(I', \eta) = p_j$, and $j > i$ under PL, we could construct

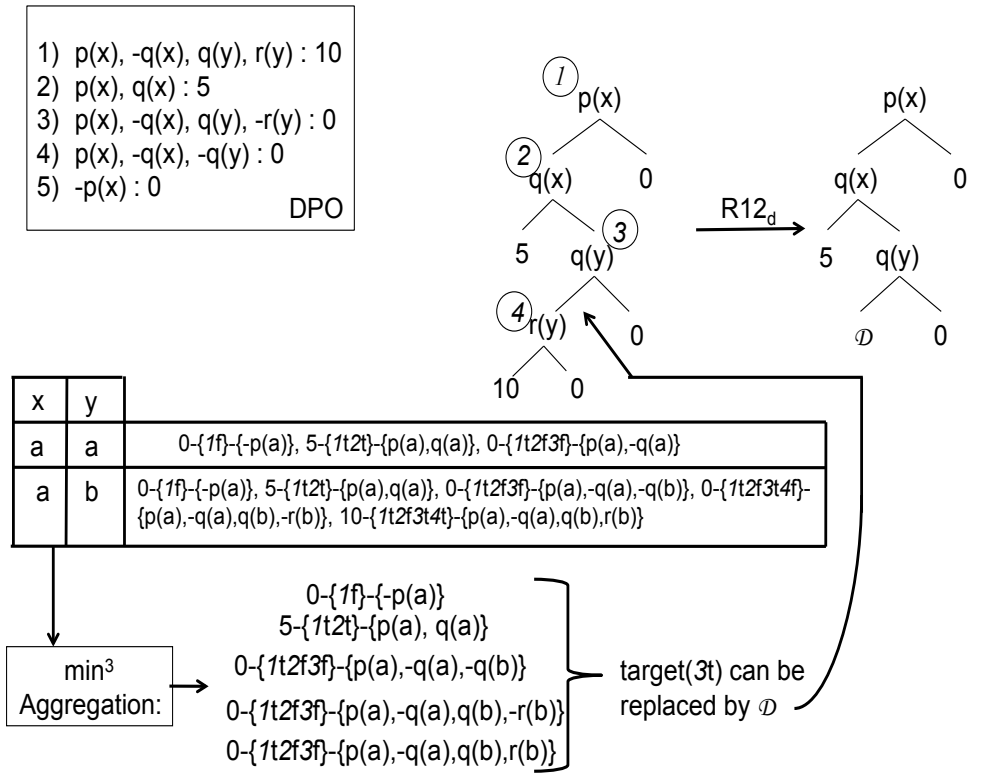

Figure 7: An example of reduction operator $R12_{\mathcal{D}}$ for GFODDs with max^*min^* Aggregation. Each entry of the form value-{path}-{interpretation} in the table (enclosing angle brackets removed in figure to improve readability) expresses the value obtained by running the valuation of the corresponding row through the diagram under an equivalence class of interpretations. The min^3 aggregation function applied to every block (in this case there is just one block with $\zeta^1 = a$ because there is only one variable x associated with the max aggregation operator) then calculates the possible aggregates that could be generated under different equivalence classes of interpretations. Since the edge $3t$ does not appear in any of the paths in the result of min^3, it is not instrumental and can be removed.

another valuation $\hat{\eta} = \eta^l\hat{\eta}^r$ by replacing the new objects in $\hat{\eta}^r$ by o''_1, so that $Path_B(I, \hat{\eta}) = p_j$. However, we know that no such $\hat{\eta}$ exists. Therefore there is no $\eta \in U_{\zeta^l}$ such that $Path_B(I', \eta) = p_j$, and $j > i$ under PL.

Let $I_o = \bigcup_{\eta \in U_{\zeta^l}} PF(Path_B(I', \eta))\eta$. That is, I_o includes all the atoms of I' that participate in traversing paths in B for all valuations in U_{ζ^l}. By construction, the corresponding parts $PF(Path_B(I', \eta))\eta$ will be included in the *valueset* returned by the getBehaviors procedure. Clearly $I_o \subseteq I'$. Therefore if I' is consistent then so is I_o. If Val_{ζ^l} is the block in Val corresponding to the valuations in U_{ζ^l}, then by the definition of min^3, $min^3(Val_{\zeta^l})$ must contain an entry {leaf(p_i), p_i, I_o}. Finally since $min^3(Val_{\zeta^l})$ is a subset of S, S must contain {leaf(p_i), p_i, I_o}. ■

The lemma implies that all edges on instrumental paths are discovered and as a result that replacing the values of other edges with $\mathcal{D}$ does not change the map of the diagram. This intuition is formalized in the next lemma and theorem.

Lemma 8. *If there exists an instrumental path in B under PL that contains the edge e then* $e \in E'$.

Proof: If there is an instrumental path $p_i \in PL$ that contains the edge e, then by Lemma 7 there exists an interpretation I_o such that $\{\text{leaf}(p_i), p_i, I_o\} \in S$. By definition of E', $e \in E'$. ■

Theorem 7 (soundness). *For any GFODD B with max*min* aggregation, if GFODD B′ is the output of* $R12_{\mathcal{D}}(B)$ *then for all interpretations I,* $Map_B(I) = Map_{B'}(I)$.

Proof: By the definition of $R12_{\mathcal{D}}$, the only difference between B and B' is that some edges that pointed to subFODDs in B, point to the discard leaf $\mathcal{D}$ in B'. These are the edges in the set E at the end of the $R12_{\mathcal{D}}$ procedure. Therefore any valuation crossing these edges is discarded from the aggregation function. Valuations not crossing these edges will achieve the same value in B' as they did in B.

Fix any interpretation I over any set O_I of objects. Let U be the set of all valuations of the variables in B over O_I. Each valuation $\eta \in U$ can be expressed as $\eta = \eta^l\eta^r$ such that $\eta^l \in U^l$ and $\eta^r \in U^r$. $Map_B(I)$ can then be expressed as

$$Map_B(I) \quad = \quad max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[Map_B(I, \eta^l\eta^r)].$$

Now for any $\eta^l \in U^l$, let p_i be a path such that there exists a valuation $\eta^r \in U^r$, $Path_B(I, \eta^l\eta^r) = p_i$ and for all $\iota^r \in U^r$, $Path_B(I, \eta^l\iota^r) = p_j$ implies that $j \leq i$ under the same DPO employed in the $R12_{\mathcal{D}}$ reduction procedure. By definition p_i is instrumental and hence by Lemma 8 none of the edges on p_i are affected by $R12_{\mathcal{D}}$. Therefore $Map_B(I, \eta^l\eta^r) = Map_{B'}(I, \eta^l\eta^r)$. We therefore conclude that for the block of η^l at least one real value (the minimizing one) exists, and other values may be replaced with $\mathcal{D}$ which is ignored by the aggregation function. Therefore,

$$min_{\eta^r \in U^r}[Map_B(I, \eta^l\eta^r)] \quad = \quad min_{\eta^r \in U^r}[Map_{B'}(I, \eta^l\eta^r)].$$

Since this is true for every $\eta^l \in U^l$, it is also true for the aggregation, that is

$$max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[Map_B(I, \eta^l\eta^r)] \quad = \quad max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[Map_{B'}(I, \eta^l\eta^r)].$$

Therefore $Map_B(I) = Map_{B'}(I)$. ■

5.2.2. The Procedure $R12_0$

The introduction of the discard value in the leaves makes handling and interpretation of diagrams awkward. In this section we show that at some additional computational cost this can be avoided. With some extra bookkeeping, a variant of the R12 procedure can avoid replacing edge targets with the discard value $\mathcal{D}$, and in the process, potentially remove more redundancies from a *max*min** GFODD. To motivate the new procedure, consider again what happens during evaluation of interpretation I on GFODD B. As observed above, each block b of valuations corresponding to a ζ^l is collapsed under *min* aggregation. Let P_b denote the set of paths in B traversed by the valuations in b and ordered by the given DPO. We view this procedure as a competition among the paths in P_b. The winner of this competition is the path of highest index in P_b. Denote this path by p_b. The *min* competition applied to all blocks creates a "super block" $\hat{b}$ of all the winners, each corresponding to a ζ^l. Finally all the ζ^ls are collapsed under the *max* aggregation. This process can, in turn, be viewed as a *max* competition among the paths in $P_{\hat{b}}$. The winner of this competition is the path with the least index in $P_{\hat{b}}$. Obviously this path also

wins the *min* competition inside its own block. Note that the block winning the max competition is not uniquely determined because there can be more than one block with the same path winning the min competition. We call any such block a *max* block, refer to *max* blocks generically as b^*, and refer to the unique winning path as p_{b^*}. Then, $Map_B(I) = \text{leaf}(p_{b^*})$.

We use the notation introduced in this discussion in the rest of this section. In particular we have: ζ_l a valuation to the max variables, its block b, the set of paths P_b and the path winning the min competition p_b. In addition we have each max block b^* with the the corresponding P_{b^*} and the unique winning path p_{b^*}. All these implicitly depend on the interpretation I, but we suppress I from the notation because it will always be clear from the context.

Using this analysis we observe the following:

1. If the value of the leaf reached by any path in a *max* block is *reduced* to a value at least as large as $\text{leaf}(p_{b^*})$, the map remains unchanged. This is because the *min* competition on the *max* block will still produce the same result. Additionally, since we are only reducing the values of other paths, the values of winners of other min competitions can only be reduced and therefore p_{b^*} will still win the max competition.
2. If the value of the leaf reached by any path in any block b other than the *max* blocks is reduced to 0, $\text{leaf}(p_{b^*})$ will still win the *max* competition and the map will be preserved.

The above observations suggest that we can reduce a GFODD in the following way,

1. Preserve the targets of all edges in all paths winning the final max competition under any interpretation. We call these *instrumental* edges.
2. Identify edges on paths in B that appear in the *max* blocks under any possible interpretation I. We call these *block* edges. For each block edge e, replace target(e) by a value that is (1) at least as large as $\text{leaf}(p_{b^*})$ under I and (2) no larger than the smallest leaf reachable by traversing e. Notice that (1) means that p_{b^*} wins the *min* competition of max blocks and (2) makes sure we never add value to any path.
3. Replace the targets of all other edges by 0.

In the remainder of this section, we develop these ideas more formally, describe the $R12_0$ reduction procedure and prove its correctness. The input to the procedure is a GFODD $B = \langle V, D\rangle$ and a DPO for B. The output is a reduced GFODD B'. We first redefine the generalized aggregation functions min^3 and max^3 to capture the bookkeeping needed for block edges.

min^3**:**. as before the input Val to min^3 is a set of sets of 3-tuples $\langle value, path, interpretation\rangle$, where each set of 3-tuples is associated with a valuation. The output is a set of all possible 4-tuples $\langle v_o, p_o, E_o, I_o\rangle$ generated as follows:

1. Let $X = \{\langle v_1, p_1, I_1\rangle, \cdots, \langle v_{|Val|}, p_{|Val|}, I_{|Val|}\rangle\}$ be a set constructed by picking one 3-tuple from the set corresponding to each valuation $\zeta \in Val$.
2. $v_o = min\{v_1, \cdots, v_{|Val|}\}$.
3. p_o is the path of highest index under DPO PL that appears in a 3-tuple in X and such that $\text{leaf}(p_o) = v_o$.
4. E_o is the set of all the edges appearing in all the paths in all of the 3-tuples in X except the edges in p_o.
5. $I_o = \bigcup_i I_i$ where $\langle v_i, p_i, I_i\rangle \in X$.
6. I_o is consistent.

Thus min^3 is exactly as before except that we also collect the set E_o. Notice that if p_0 happens to be instrumental then E_0 identifies the edges that act as block edges in this case. If p_0 is not instrumental then E_0 is not of interest. Next, max^3 is adapted to take as input the set of outputs of min^3 (each run for a different ζ_l) and identify in its output the instrumental path and winning blocks and blocks edges for each I_0 generated.

max^3**:**. the input Val to max^3 is a set of sets of 4-tuples $\langle value,path,EdgeList,interpretation\rangle$ where each set of 4-tuples is associated with a valuation. The output is a set of all possible 4-tuples $\langle v_o,p_o,E_o,I_o\rangle$ generated as follows.

1. Let $X = \{\langle v_1,p_1,E_1,I_1\rangle, \cdots, \langle v_{|Val|},p_{|Val|},E_{|Val|},I_{|Val|}\rangle\}$ be a set constructed by picking one 4-tuple from the set corresponding to each valuation $\zeta \in Val$.
2. $v_o = max\{v_1, \cdots, v_{|Val|}\}$.
3. p_o is the path of least index under DPO PL that appears in a 4-tuple in X such that leaf(p_o) $= v_o$.
4. E_o is a set E_i such that $p_o = p_i$ and $v_o = v_i$; here if there is more than one i satisfying the condition then each such E_i is given in a separate output tuple.
5. $I_o = \bigcup_i I_i$ where $\langle v_i,p_i,E_i,I_i\rangle \in X$.
6. I_o is consistent.

Thus max^3 is exactly as before except that we also process the sets E_i and produce the set E_o. max^3 picks the E_i that corresponds to the winning path p_i from its input. If there is more than one block with the same winning path then each of them produces an output tuple. Therefore, in the output of max^3, I_0 is a consistent interpretation whose instrumental path is p_0 and where some of its block edges are listed in E_0.

The $R12_0$ procedure is as follows.

1. Recall that the variables are split into V^l with max aggregation followed by V^r using min aggregation. The set U of valuations is built in the following way. Let O^l be a set of $|V^l|$ objects and O^r a set of $(|V^l|^{|V^l|} + 1)|V^r|$ objects where O^l and O^r are disjoint. Let U^l be the sets of all possible valuations of the variables in V^l over the objects in O^l and let U^r be the set of all possible valuations of the variables in V^r over the objects in the union of O^l and O^r. The set U is then defined as $U = \{\zeta^l\zeta^r \mid \zeta^l \in U^l \text{ and } \zeta^r \in U^r\}$.
   As in the previous reduction the set U is constructed to allow for a sufficiently rich set of valuations. Here we allow for an arbitrary valuation to V^l using objects in O^l. Next we consider every fixed valuation to V^l and the block of valuations to V^r that extends it. We allow each of the $|V^l|^{|V^l|}$ blocks to use a fresh set of $|V^r|$ objects (or any of the other objects). In this way the winner of the *min* competition in each block is not constrained by valuations in other blocks. Finally, we must allow a path of block edges to be unconstrained by other bindings in the block. We therefore add another set of $|V^r|$ objects. As the proof below shows this allows us to expose all instrumental paths and all block edges in the diagram.
2. For every edge we maintain 3 variables. low(e) and high(e) are bounds on its value and InstrEdge(e) is a flag. These are initialized as follows. For all edges e in B, set low(e) $= -1$, high(e) $= l_e$, where l_e is the value of the smallest leaf reachable through e in B, and InstrEdge(e) $= 0$.
3. Run the $maxmin^3$ procedure as follows.
   (a) Divide Val into $|U^l|$ blocks of valuations each block corresponding to a valuation $\zeta^l \in U^l$. Let X be the set of these blocks.

(b) Let $Y = \{\langle \zeta^l, \text{Reduction-Aggregation(B,b,PL)}\rangle \mid \zeta^l \in U^l$ and $b \in X$ is the block corresponding to $\zeta^l\}$, where Reduction-Aggregation uses the newly defined min^3.
(c) Let $S = max^3(Y)$.
(d) For every 4-tuple $\langle v_o, p_o, E_o, I_o\rangle \in S$, do
   i. For every edge $e \in p_o$, set InstrEdge(e) = 1.
   ii. For every edge $e \in E_o$, set low(e) to max{low(e), v_o}.

4. Finally the target of every edge e is replaced as follows:
   (a) If InstrEdge(e) = 1, do not replace.
   (b) If InstrEdge(e) = 0 and low(e) $\neq -1$ (that is, e is a block edge) and high(e) $\geq$ low(e), then replace target(e) by any suitable value v, such that low(e) $\leq v \leq$ high(e).
   (c) If InstrEdge(e) = 0 and low(e) = -1 (that is, e is not a block edge) then replace target(e) by 0.

Figure 8 shows an example of the $R12_0$ reduction where several of the steps in the algorithm are illustrated.

In the remainder of this section we provide a proof of soundness for $R12_0$. To that end we first define idealized properties of a reduction procedure in the style of $R12_0$. We then show that if a reduction has these properties then it is sound, and that $R12_0$ indeed has these properties. This allows us to break the argument into two independent portions and in this way simplifies the proof.

Definition 11. *An edge e in a GFODD B is instrumental iff* $e \in p_{b^*}$ *under some interpretation.*

Definition 12. *An edge e in a GFODD B is a block edge if it is not instrumental and* $e \in path \in P_{b^*}$ *for some max block* b^* *under some interpretation.*

Definition 13. *For any block edge e, CannotExceed(e) is the value of the smallest leaf reachable through e and CannotLag(e) is the value of the largest value of leaf(*p_{b^*}*) over all possible interpretations, when a path containing e appears in a max block.*

Definition 14. *A reduction procedure R that reduces a given GFODD B to produce GFODD B′ is block-safe if it conforms to the following rules.*

1. *R identifies all instrumental edges in B and for each such identified edge e, R maintains target(e).*
2. *R identifies all block edges in B and for each such identified edge e, R replaces target(e) by any leaf value v such that CannotLag(e)* $\leq v \leq$ *CannotExceed(e).*
3. *For each edge e that is not identified by R as an instrumental or block edge, R replaces target(e) by* 0.

Thus our idealized reduction is *block-safe*; the next theorem shows that any such procedure is sound.

Theorem 8. *If reduction procedure R is block-safe and* $B' = R(B)$*, then for every interpretation I,* $Map_B(I) = Map_{B'}(I)$.

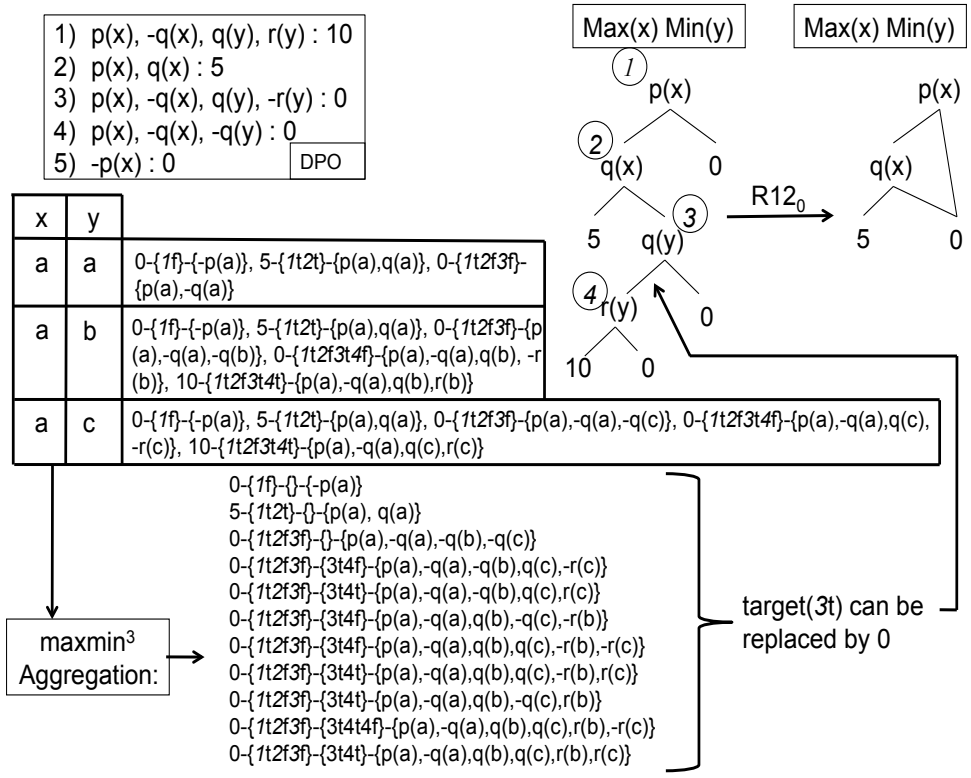

Figure 8: An example of the reduction $R12_0$. The initial diagram is the same as in Example 12 and Figure 7. We have $|V^l| = |V^r| = 1$ and hence $|O^l| = 1$ and $|O^r| = (|V^l|^{|V^l|} + 1)|V^r| = 2$ and therefore y is allowed to bind to the 3 objects in $O^l \cup O^r$. Each entry of the form value-{path}-{interpretation} in the table (enclosing angle brackets removed in figure to improve readability) expresses the value obtained by running the valuation of the corresponding row through the diagram under an equivalence class of interpretations. The *maxmin*^{3} aggregation function then calculates the possible aggregates that could be generated under different equivalence classes of interpretations. Since we have only one block, we only need to run the extended *min*^{3} aggregation on this example. The result is shown below the table. For example the entries 0-{$1t2f3f$}-{$p(a), \neg q(a)$}, 10-{$1t2f3t4t$}-{$p(a), \neg q(a), q(b), r(b)$} and 10-{$1t2f3t4t$}-{$p(a), \neg q(a), q(c), r(c)$}, give the last row in the result. Overall, the edges *3t*,*4t* and *4f* are identified as a block edges. For edge *3t*, InstrEdge(*3t*) = 0 because no winner of the max block contains edge *3t*. high(*3t*) = 0 because the smallest leaf reachable by traversing *3t* is 0. The *maxmin*^{3} procedure sets low(*3t*) to 0 because the highest leaf reached by any path defeating the paths containing *3t* in the max block is 0. Thus target(*3t*) can be set to 0 without violating the constraint low(*3t*) $\leq$ target(*3t*) $\leq$ high(*3t*). Setting the target of *3t* to 0 reduces the diagram. Note that in this example all edges shown are block edges because there is only one block - the max block. All the edges appearing in the result of *maxmin*^{3} are instrumental edges and their targets are preserved by the reduction procedure.

Proof: Fix any interpretation I over any set O_I of objects. Let U be the set of all valuations of the variables in B over O_I. Let $\zeta = \zeta^l\zeta^r \in U$ be a valuation traversing p_{b^*} in B. $Map_B(I)$ can then be expressed as

$$\begin{aligned} Map_B(I) &= max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[Map_B(I, \eta^l\eta^r)] \\ &= max[min_{\zeta^r \in U^r}[Map_B(I, \zeta^l\zeta^r)], max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[Map_B(I, \eta^l\eta^r)]]]. \end{aligned}$$

Since the definition of *block-safe* guarantees that the target of every edge e is not replaced by a value greater than CannotExceed(e), target(e) only decreases in value. Therefore, for any valuation $\eta \in U$, leaf($Path_B(I,\eta)$) $\geq$ leaf($Path_{B'}(I,\eta)$). Therefore we have,

$$max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[Map_{B'}(I,\eta^l\eta^r)]] \leq max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[Map_B(I,\eta^l\eta^r)]].$$

Additionally, the definition of *block-safe* guarantees that all instrumental edges are preserved and that the value reached by the block edges is never reduced below leaf(p_{b^*}). Therefore, ζ reaches leaf(p_{b^*}) in both B and B'. No other valuation in any *max* block b^* reaches a value less than leaf(p_{b^*}) when evaluated on B'. Thus,

$$\begin{aligned} min_{\zeta^r \in U^r}[Map_{B'}(I,\zeta^l\zeta^r)] &= min_{\zeta^r \in U^r}[Map_B(I,\zeta^l\zeta^r)] \\ &= leaf(p_{b^*}). \end{aligned}$$

Finally,

$$\begin{aligned} Map_{B'}(I) &= max[min_{\zeta^r \in U^r}[Map_{B'}(I,\zeta^l\zeta^r)], max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[Map_{B'}(I,\eta^l\eta^r)]]] \\ &= min_{\zeta^r \in U^r}[Map_{B'}(I,\zeta^l\zeta^r)] \\ &= leaf(p_{b^*}) \\ &= Map_B(I). \end{aligned}$$

■

Therefore, to prove soundness of $R12_0$, we can focus on showing that it is block-safe as we do in the next theorem.

It is clear from the construction that $R12_0$ identifies some instrumental edges and some block edges. The difficulty is in showing that it identifies *all* such edges over an infinite set of interpretations some of which have infinite domains. The following proof shows that each such edge is discovered by one of the finite combinations in our procedure. Note that even if two edges are the block edges of the same p_{b^*}, they do not need to be discovered at the same time or using the same I_o in our procedure. Instead it is sufficient that each is discovered and marked as a block edge at some point in the algorithm. This is the approach taken in the next proof showing that every instrumental edge (on p_{b^*} below) and block edge (on p_j below) are appropriately accounted for by $R12_0$.

Theorem 9. *$R12_0$ is block-safe.*

Proof: Line 4 in the $R12_0$ procedure enumerates the treatment of different edges in B. Accordingly to prove the theorem we need to show that:

1. If an edge e in B is instrumental under some interpretation I, then $R12_0$ sets InstrEdge(e) = 1.
2. If an edge e is a block edge under some interpretation I, then $R12_0$ sets the value low(e) $\geq$ CannotLag(e).
3. If an edge e is a block edge under some interpretation I, then $R12_0$ sets the value high(e) $\leq$ CannotExceed(e).

Of the above, 3 is true by the definition of $R12_0$ because high(e) is initialized to the correct value and is never changed. We next show that the procedure correctly identifies every instrumental edge and every block edge, and sets the correct bound for block edges.

Consider any interpretation I. Let $\zeta = \zeta^l\zeta^r$ be a valuation traversing $p_{b^*} = p_i$ in B under I. Therefore ζ^l identifies a *max* block and we refer to this block as b^* below. Let $\eta = \zeta^l\eta^r$ be any other valuation in the *max* block b^* that does not win the min competition and let $Path_B(I,\eta) = p_j$. Therefore, p_i is instrumental, and the edges in p_j are potentially block edges (this holds unless they are instrumental for some other I) and the lower bound for these edges must be $\geq leaf(p_i)$.

Let O'^l be the set of objects that participate in ζ^l and define the set $O'^r = \{\, o \notin O'^l \mid o$ participates in η^r or in ι^r, where ι^l contains only the objects from O'^l and $\iota^l\iota^r$ wins the *min* competition in the block of $\iota^l\}$. By construction $|O'^l| \leq |V^l|$ and $|O'^r| \leq (|V^l|^{|V^l|} + 1)|V^r|$. Let $o'^l_1 \in O'^l$ and $o'^r_1 \in O'^r$. Add $|V^l| - |O'^l|$ new objects to O'^l and $(|V^l|^{|V^l|} + 1)|V^r| - |O'^r|$ new objects to O'^r.

Construct interpretation I' by first projecting I to include only the objects in O'^l and O'^r and then defining truth values and predicates over the new objects added to O'^l and O'^r to behave identically to o'^l_1 and o'^r_1 respectively. Let O'^l and O'^r be the sets O^l and O^r used in the $R12_0$ procedure to generate the set of valuations U.

Since I' contains the relevant portion of I, $Path_B(I',\zeta) = p_i$ and $Path_B(I',\eta) = p_j$. In addition, p_i is the winner of the min competition in the block b^* under I'. To see this, note that if there exists valuation $\zeta^l\iota^r \in U$ such that $Path_B(I',\zeta^l\iota^r) = p_k$ and $k > i$ under PL, then we could construct another valuation $\zeta^l\hat{\iota^r}$ by replacing the new objects in ι^r by o'^r_1 so that $Path_B(I,\zeta^l\hat{\iota^r}) = p_k$. However, we know that there is no such $\zeta^l\hat{\iota^r}$. An identical argument proves that if b is a block in U corresponding to ι^l, then p_b defined relative to I is the winner of the *min* competition in b under I'.

So far we have shown that the winners of all min competitions in I for blocks in U are maintained in I' without direct reference to our algorithm. We next focus on $R12_0$ showing that the appropriate paths are discovered.

Let $I_{\iota^l\iota^r} = PF(Path_B(I',\iota^l\iota^r))\iota^l\iota^r$ be the set of atoms on the path $p_{\iota^l\iota^r}$ in B traversed by some valuation $\iota^l\iota^r$ under I'. By construction, a 3-tuple $\langle \text{leaf}(p_{\iota^l\iota^r}), p_{\iota^l\iota^r}, I_{\iota^l\iota^r}\rangle$ appears in the output of the getBehaviors procedure, when run on $\iota^l\iota^r$. Therefore, by the definition of Reduction-Aggregation and min^3, the set Y generated in Step 3b of $R12_0$ must contain an entry $\langle \text{leaf}(p_b), p_b, E_b, I_b\rangle$, where $I_b = \bigcup_{\iota^r \in U^r} PF(Path_B(I',\iota^l\iota^r))\iota^l\iota^r$. Similarly the set produced by applying min^3 to the *max* block b^* must contain an entry $\langle \text{leaf}(p_{b^*}), p_{b^*}, E_{b^*}, I_{b^*}\rangle$, where $I_{b^*} = \bigcup_{\iota^r \in U^r} PF(Path_B(I',\zeta^l\iota^r))\zeta^l\iota^r$. In addition by the same argument, E_{b^*} must contain all the edges in p_j.

Now, by the definition of max^3, the set S built in Step 3c of $R12_0$ must contain an entry $\langle \text{leaf}(p_{b^*}), p_{b^*}, E_{b^*}, I_o\rangle$ where $I_o = \bigcup_{\iota \in U} PF(Path_B(I',\iota))\iota$ is consistent because it is a subset of I'.

Therefore $e \in p_{b^*}$ is marked instrumental by $R12_0$. Every edge $e \in p_j$ is marked with low(e) $\geq$ leaf(p_{b^*}). Since the choice of I, p_{b^*} and p_j was arbitrary in the above argument, this holds for all block edges, implying that low(e) $\geq$ CannotLag(e). Thus $R12_0$ is *block-safe*. ■

Corollary 1 (soundness). *For any GFODD B with $max^* min^*$ aggregation, if GFODD B' is the output of $R12_0(B)$ then for all interpretations I, $Map_B(I) = Map_{B'}(I)$.*

6. An Application of GFODDs for Value Iteration in Relational MDPs

So far we have described a general theory of GFODDs. This included the syntax and semantics of GFODDs, combination procedures and reduction procedures for GFODDs. In this section we show how GFODDs can be used to solve Relational MDPs.

6.1. Relational Markov Decision Processes

A Markov decision process (MDP) is a mathematical model of decision making in a dynamic environment [1, 2]. Formally a MDP is a 4-tuple $\langle S, A, T, R \rangle$ defining a set of states S, set of actions A, a transition function T defining the probability $P(s_j \mid s_i, a)$ of getting to state s_j from state s_i on taking action a, and an immediate reward function $R(s)$. The objective of solving a MDP is to generate a policy that maximizes the agent's total, expected, discounted, reward. Intuitively, the expected utility or value of a state is equal to the reward obtained in the state plus the discounted value of the state reached by the best action in the state. This is captured by the Bellman equation as $V(s) = Max_a[R(s) + \gamma\Sigma_{s'}P(s'|s,a)V(s')]$. The value iteration algorithm is a dynamic programming algorithm that treats the Bellman equation as an update rule and iteratively updates the value of every state until convergence. The value iteration update is $V^{n+1}(s) \leftarrow Max_a[R(s) + \gamma\Sigma_{s'}P(s'|s,a)V^n(s')]$. Once the optimal value function is known, a policy can be generated by assigning to each state the action that maximizes expected value.

Several approaches have been introduced to take advantage of factored state spaces where a state is described by specifying values of a set of propositions [26, 27, 28]. In particular Hoey et al. [29] showed that if $R(s)$, $P(s' \mid s, a)$ and $V(s)$ can be represented using algebraic decision diagrams (ADD) [24, 25], then value iteration can be performed entirely using the ADD representation avoiding the need to enumerate the state space. This improved the scalability of classical solutions to MDPs by replacing the enumeration of states implicit in the equation above with ADDs, a compact feature based representation, thereby taking advantage of the structure in the problem. However, further structure in the domain can be exploited and more general solutions can be found by viewing the world as consisting of objects with relations among them. MDPs represented in this way are known as Relational MDPs. Addressing Relational MDPs, Boutilier et al. [6] developed the Symbolic Dynamic Programming (SDP) algorithm in the context of situation calculus. This algorithm provided a framework for dynamic programming solutions to Relational MDPs that was later employed in several formalisms and systems [7, 8, 10, 9]. One of the important ideas in SDP was to represent stochastic actions as deterministic alternatives under nature's control. This helps simplify the probabilistic reasoning required because goal regression over deterministic action alternatives can be decoupled from the probabilities of action effects. This separation is necessary when transition functions are represented as relational schema. Using these ideas, a RMDP is specified by

1. A set of world predicates. Each literal, formed by instantiating a predicate using objects from the domain, can be either `true` or `false` in a given state. For example in the box-world domain, world literals are of the form box-in-city(*box*, *city*), box-on-truck(*box*, *truck*), truck-in-city(*truck*, *city*) etc.
2. A set of action predicates. Each action literal, formed by instantiating an action predicate using objects from the domain, defines a concrete action. For example in the box-world domain, action literals are of the form load-box-on-to-truck-in-city(*box*, *truck*, *city*), unload-box-from-truck-in-city(*box*, *truck*, *city*), drive-truck(*truck*, *source-city*, *dest-city*), etc.
3. A state transition function that provides an abstract description of the probabilistic move from one state to another. For example, using a STRIPS-like notation, the transition defined by the action load-box-on-to-truck-in-city can be described as
   Action: load-box-on-to-truck-in-city(*box*, *truck*, *city*):
   Preconditions: box-in-city(*box*, *city*), truck-in-city(*truck*, *city*)

outcome 1**:** *probability* 0.8, box-on-truck($box, truck$), ¬box-in-city($box, city$)
outcome 2**:** *probability* 0.2, nothing changes.
If the preconditions of the action, box-in-city($box, city$) and truck-in-city($truck, city$), are satisfied then with probability 0.8, the action will generate the effect box-on-truck(box, $truck$) and ¬box-in-city($box, city$). The state remains unchanged with probability 0.2. As this example illustrates, the effects of actions in RMDPs are often correlated and cannot be considered to occur independently of one another. Therefore, a scheme that captures such correlations compactly is useful in this context.

4. An abstract reward function describing conditions under which rewards are obtained. For example in the boxworld domain, the reward function is [$\forall box \forall city$, destination($box, city$) $\rightarrow$ box-in-city($box, city$)] constructed so as to capture the goal of transporting all boxes from their source cities to their respective destination cities.

An interesting fact to notice about RMDPs is that the state space in the underlying MDP is not fully specified because the set of objects in the domain is left out. When fixing the domain of objects the specification induces a concrete MDP. Thus a RMDP represents a family of concrete MDPs.

The above RMDP can be described using various schema languages. Wang et al. [9] describe the RMDP by representing the reward function and the domain dynamics using FODDs. Domain dynamics are described by Truth Value Diagrams (TVD), and diagrams capturing probabilistic action choice. A TVD is a FODD describing, for each deterministic alternative of each probabilistic action and for each world predicate, the conditions under which the corresponding world literal is `true` when the action is executed and that action alternative occurs. Figure 9 shows an example of a TVD for the parameterized world predicate $p(U, V)$ under the deterministic action $A(x^*, y^*)$ in a hypothetical planning domain. In addition, for each deterministic action variant $A_j(\vec{x})$, the diagram $prob(A_j(\vec{x}))$ provides the probability that $A_j(\vec{x})$ is chosen when $A(\vec{x})$ is executed.

6.2. The VI-GFODD Algorithm

In this section we show that the FODD based value iteration (VI) algorithm can be generalized to handle cases where the reward function is described by a GFODD with $max^* min^*$ aggregation. We start by describing the *VI-GFODD* algorithm. A subsequent discussion shows why *VI-GFODD* produces the correct result at each step. The algorithm is as follows:

1. **Regression:** The n step-to-go value function V_n is regressed over every deterministic variant $A_j(\vec{x})$ of every action $A(\vec{x})$ to produce $Regr(V_n, A_j(\vec{x}))$ by replacing each node in V^{n-1} by its corresponding Truth Value Diagram (TVD) without changing the aggregation function.
2. **Add Action Variants:** The Q-function $Q_{V_n}^{A(\vec{x})} = R \oplus [\gamma \otimes \oplus_j (prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x})))]$ for each action $A(\vec{x})$ is generated by combining regressed diagrams using Ex-apply.
3. **Object Maximization:** Maximize over the action parameters of $Q_{V_n}^{A(\vec{x})}$ to produce $Q_{V_n}^{A}$ for each action $A(\vec{x})$, thus obtaining the value achievable by the best ground instantiation of $A(\vec{x})$. This step is implemented by converting action parameters in $Q_{V_n}^{A(\vec{x})}$ to variables each associated with the *max* aggregation operator, and appending these operators to the head of the aggregation function.
4. **Maximize over Actions:** The $n + 1$ step-to-go value function $V_{n+1} = \max_A Q_{V_n}^{A}$, is generated by combining the diagrams using Ex-apply.

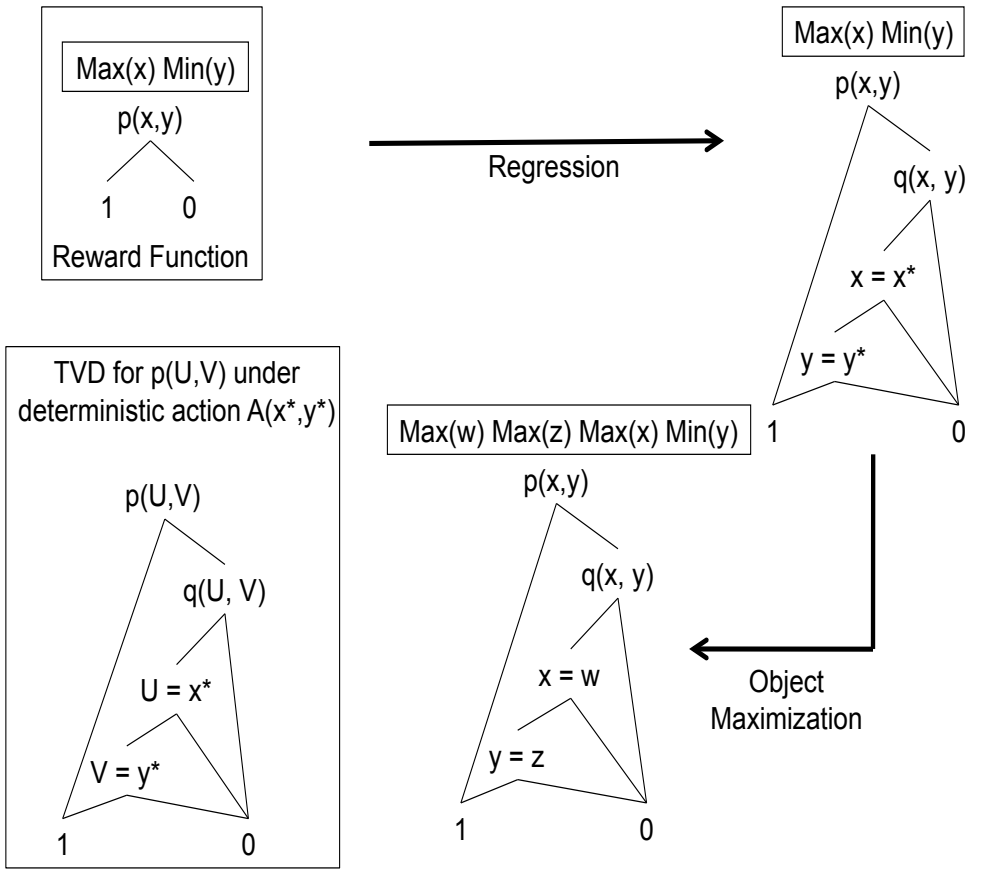

Figure 9: Example of Regression and Object Maximization in the VI-GFODD algorithm. This domain contains a single deterministic action. Therefore steps 2 and 4 of the algorithm are not needed. The reward is 1 if $[\exists x, \forall y, p(x,y)]$ and it is 0 otherwise. The reward function is regressed over the deterministic action $A(x^*, y^*)$. The action is defined such that $p(U,V)$ is true after the action if it was true before or if $q(U,V)$ was true before and the action performed was $A(U,V)$. Regression replaces every node in the value function with the corresponding TVD and object maximization replaces the action parameters with quantified variables.

Example 13. *Figure 9 shows an example of the VI algorithm using GFODDs for a simple domain. This domain contains a single deterministic action. Therefore we do not need to multiply by* $prob(A_j(\vec{x}))$ *and to sum over the variants* A_j *in Step 2 of the algorithm and similarly Step 4 is not needed. In this example we completely skip Step 2 and focus on the other two steps in the algorithm. The reward is* 1 *if* $[\exists x, \forall y, p(x,y)]$ *holds and is* 0 *otherwise. The reward function is regressed over the deterministic action* $A(x^*, y^*)$*, which is defined such that* $p(x,y)$ *is true after the action if it was true before or if* $q(x,y)$ *was true before and the action performed was* $A(x,y)$*. Since the action can make at most one* $p(x,y)$ *true at a time, intuitively, the regressed diagram should capture the union of the following conditions for returning a value of* 1*.*

1. *there exists x, such that for all y,* $p(x,y)$ *holds.*
2. *there exists x, such that for all but one y,* $p(x,y)$ *is true and for that y,* $q(x,y)$ *is true.*

Figure 9 shows the diagram after being regressed and object maximized. The final diagram is correct because it returns a 1 *iff one of above situations occur. If* $[\exists x, \forall y, p(x,y)]$ *is true, then*

all valuations in the blocks with that value of x and fixed values for w and z will reach the 1 *leaf directly from the root. Evaluating Min*(y) *will collapse these blocks to partial valuations with a* 1 *value. Now since the rest of the aggregation is maximization, the* 1 *value will be returned as the map. If there exists x, such that for all but one y,* $p(x, y)$ *is true and for that y,* $q(x, y)$ *is true, then all valuations in the blocks with that value of x, the other values of y and fixed values for w and z reach a* 1 *leaf directly through the root. The valuation in the block with the one value of y would traverse right from the root but would still reach the* 1 *leaf depending on the condition* $w = x$ *and* $z = y$*. Note that there will be exactly one block where this valuation will reach the* 1 *leaf. Evaluating Min*(y) *would collapse that block into a valuation with value* 1*. Since the rest of the aggregation is maximization, the* 1 *value will be returned as the map. When neither of the conditions is true, there will be at least one valuation in every block that reaches a* 0 *leaf. Hence evaluating Min*(y) *would collapse every block to a valuation with a* 0 *value.*

For Value Iteration to work correctly with GFODDs, all the steps of the algorithm listed above must be correct. Regression by block replacement is correct regardless of the aggregation function. Recall that a TVD for a predicate under deterministic action $A_j(\vec{x})$ describes conditions under which the predicate is `true` after $A_j(\vec{x})$ is executed. Wang et al. [9] impose the constraint that TVDs cannot include free variables. Using this constraint the diagrams before and after regression have exactly the same variables. Wang et al. [9] show that regression is correct for any valuation.

Lemma 9. *[9] Fix any concrete instantiation of the state space. Let s denote a state resulting from executing an action* $A(\vec{x})$ *in state* $\hat{s}$*.*

If V_n *is the n step to go value function, BR-regress*$(V_n, A(\vec{x}))$ *is the result of regressing* V_n *over the deterministic action* $A(\vec{x})$*, and* ζ *is any valuation to the variables of* V_n *(and thus also the variables of BR-regress*$(V_n, A(\vec{x}))$*), then* $MAP_{V_n}(s, \zeta) = MAP_{\textit{BR-regress}(V_n, A(\vec{x}))}(\hat{s}, \zeta)$*.*

The lemma shows that the corresponding map values are the same for any valuation ζ. Therefore, the aggregation of the values is the same for any aggregation function, and any V_n.

The third step, Object Maximization, is correct because converting action parameters in $Q_{V_n}^{A(\vec{x})}$ to variables each associated with the *max* aggregation operator, and appending these operators to the head of the aggregation function of $Q_{V_n}^{A}$, implies that the map of $Q_{V_n}^{A}$ under any interpretation will now be the map of $Q_{V_n}^{A(\vec{x})}$ maximized over all possible values of the action parameters, as required. Steps 2 and 4 are correct by Theorem 4 showing the correctness of Ex-apply. Since value iteration requires combining diagrams under the $\oplus$, $\otimes$ and the *max* operators, only GFODDs with aggregation operators that are safe with the combination operators $\oplus$, $\otimes$ and *max* may be used. Thus aggregation operators *max* and *min* can be used. To extend the algorithm to use other aggregation operators (like *sum* and *mean*) one needs to develop appropriate combination algorithms but the rest of the algorithm remains the same.

Thus we have a correct value iteration algorithm for GFODDs with *max* and *min* aggregations. In addition, Theorem 4 guarantees that if we start with a reward function GFODD with an aggregation of the form $max^* min^*$, then throughout value iteration all GFODDs produced can be made to have an aggregation function of the same form. With the R12 reductions for this case, we have a sound procedure that can help keep the diagrams compact over the value iteration process. We have therefore shown:

Theorem 10. *For any Relational MDP where the aggregation function of the reward function diagram contains only operators that are safe with the combination operators* $+$, $\times$ *and max, the algorithm VI-GFODD produces the correct value function at every iteration.*

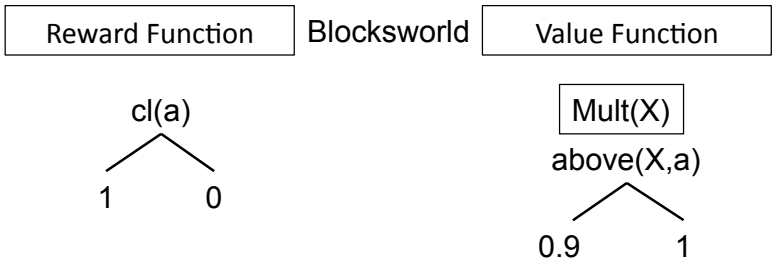

Figure 10: Reward and value function for the goal $cl(a)$ in the blocksworld domain.

Corollary 2. *For any Relational MDP where the reward function has a $max^* min^*$ aggregation, VI-GFODD produces the correct value function at every iteration, all intermediate results and the final result use $max^* min^*$ aggregation, and the R12 procedure can be used to reduce the diagrams throughout the algorithm.*

7. Conclusions and Future Work

This paper significantly extends the representation power of first-order decision diagrams and our algorithmic understanding of their reductions. We show how Generalized FODDs allow for arbitrary aggregation functions, thereby facilitating representation of complex functions, and how basic operations on them can be performed. In particular we can naturally capture and manipulate logical formulas with existential and universal quantifiers using max and min aggregation. In addition we show that first-order value iteration can be supported in a more expressive setting when the MDP is represented by GFODDs. This new formulation can naturally handle universal goals that were handled heuristically by previous implementations of first-order value iteration [10, 11].

Additionally, GFODDs might prove useful in addressing issues related to problems where the lifted value function is infinite in size. For instance, Kersting et al. [7] showed an example in the blocksworld domain where the goal is to make a particular block, a, clear (denoted $cl(a)$) and the value function is infinite in size because there could be any number of blocks on top of a. However, the value function can be represented compactly using GFODDs in conjunction with a more descriptive predicate, *above*, as shown in Figure 10. In the figure, $above(X, a)$ is true for any block X that is part of a tower stacked on top of block a, aggregation over X is performed by the multiplication operator and the discount factor is 0.9. Thus the multiplicative aggregation implicitly captures the number of steps to the goal. Although the existence of a compact value function does not imply an efficient algorithm to produce it, at least in this particular case we know that the problem is not inherently that of representation.

The other main contribution in the paper is the idea and analysis of model checking reductions. The same basic idea provides model checking reduction operators for both FODDs and a useful subset of GFODDs. In the former case, we prove the reduction to be, in some technical sense, maximal. The maximum reduction guarantee for FODDs falls short of providing a normal form because it relies on a DPO to define which parts of a diagram may be reduced when there are mutual implication relations. Therefore the same semantic function may have different minimal representations. However, the guarantee is much stronger than those of previous reductions.

Wang et al. [9] discuss normal form for FODDs. Examples of FODDs given there, using a simple decidable fragment, show that for normal form we may need some syntactic manipulation of diagrams. Therefore going beyond the guarantee given in this paper may be hard or expensive to compute. Nevertheless, there is a potential for exploring this and the possibility of efficient reductions for other interesting subsets of GFODDs in future work.

This work also suggests a new approach for practical implementations of FODDs. The model checking reductions of this paper require enumeration of substitutions which has high complexity. A promising idea is to use a sample of interpretations, judicially chosen, and reduce the diagrams relative to these interpretations. We refer the reader to [12] for recent work providing a validation of this idea in the context of RMDPs where the implementation shows a significant speedup over theorem proving reductions while maintaining performance in terms of solving planning problems using FODDs. It would be interesting to develop extensions of these heuristics that support efficient reductions for GFODDs. Such an approach will allow for the very expressive setting of GFODDs to be handled efficiently through the heuristic approximation embedded in the model checking reductions.

Finally it would be interesting to investigate the utility of GFODDs in other applications, like lifted inference and Statistical Relational Learning, that can benefit from expressive function representations.

Acknowledgments

Kristian Kersting was supported by the Fraunhofer ATTRACT fellowship STREAM. Saket Joshi and Roni Khardon were partly supported by the NSF grants IIS 0936687 and IIS 0964457, and Saket Joshi was additionally supported by a Computing Innovation Postdoctoral Fellowship.

References

[1] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley, 1994.
[2] S. Russel, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall Series in Artificial Intelligence, 2002.
[3] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.
[4] R. Howard, Dynamic Programming and Markov Processes, MIT Press, 1960.
[5] L. Getoor, B. Tasker, An Introduction to Statistical Relational Learning, MIT Press, 2007.
[6] C. Boutilier, R. Reiter, B. Price, Symbolic dynamic programming for first-order MDPs, in: Proceedings of the International Joint Conference of Artificial Intelligence (2001), pp. 690–700.
[7] K. Kersting, M. van Otterlo, L. De Raedt, Bellman goes relational, in: Proceedings of the International Conference on Machine Learning (2004), pp. 465–472.
[8] S. Hölldobler, E. Karabaev, O. Skvortsova, FluCaP: a heuristic search planner for first-order MDPs, Journal of Artificial Intelligence Research 27 (2006) 419–439.
[9] C. Wang, S. Joshi, R. Khardon, First-Order decision diagrams for relational MDPs, Journal of Artificial Intelligence Research 31 (2008) 431–472.
[10] S. Sanner, C. Boutilier, Practical solution techniques for first-order MDPs, Artificial Intelligence 173 (2009) 748–788.
[11] S. Joshi, R. Khardon, Probabilistic relational planning with first-order decision diagrams, Journal of Artificial Intelligence Research 41 (2011) 231–266.
[12] S. Joshi, K. Kersting, R. Khardon, Self-Taught decision theoretic planning with first-order decision diagrams, in: Proceedings of the International Conference on Automated Planning and Scheduling (2010), pp. 89–96.
[13] D. Poole, First-Order probabilistic inference, in: Proceedings of the International Joint Conference of Artificial Intelligence (2003), pp. 985–991.
[14] R. Braz, E. Amir, D. Roth, Lifted first-order probabilistic inference, in: Proceedings of the International Joint Conference of Artificial Intelligence (2005), pp. 1319–1325.

[15] A. Jaimovich, O. Meshi, N. Friedman, Template-based inference in symmetric relational Markov random fields, in: Proceedings of Uncertainty in Artificial Intelligence (2007), pp. 191–199.
[16] B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, L. Kaelbling, Lifted probabilistic inference with counting formulas, in: Proceedings of the National Conference on Artificial Intelligence (2008), pp. 1062–1068.
[17] P. Singla, P. Domingos, Lifted first-order belief propagation, in: Proceedings of the National Conference on Artificial Intelligence (2008), pp. 1094–1099.
[18] P. Sen, A. Deshpande, L. Getoor, Exploiting shared correlations in probabilistic databases, in: Proceedings of the International Conference on Very Large Data Bases (2008), pp. 809–820.
[19] P. Sen, A. Deshpande, L. Getoor, Bisimulation-based approximate lifted inference, in: Proceedings of Uncertainty in Artificial Intelligence (2009), pp. 496–505.
[20] K. Kersting, B. Ahmadi, S. Natarajan, Counting belief propagation, in: Proceedings of Uncertainty in Artificial Intelligence (2009), pp. 277–284.
[21] J. Kisynski, D. Poole, Lifted aggregation in directed first-order probabilistic models, in: Proceedings of the International Joint Conference of Artificial Intelligence (2009), pp. 1922–1929.
[22] J. Lloyd, Foundations of Logic Programming, Springer Verlag, 1987. Second Edition.
[23] J. Groote, O. Tveretina, Binary decision diagrams for first-order predicate logic, Journal of Logic and Algebraic Programming 57 (2003) 1–22.
[24] R. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers C-35 (1986) 677–691.
[25] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic decision diagrams and their applications, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (1993), pp. 188–191.
[26] C. Boutilier, R. Dearden, M. Goldszmidt, Stochastic dynamic programming with factored representations, Artificial Intelligence 121 (2000) 49–107.
[27] M. Kearns, D. Koller, Efficient reinforcement learning in factored MDPs, in: Proceedings of the International Joint Conference of Artificial Intelligence (1999), pp. 740–747.
[28] C. Guestrin, D. Koller, R. Parr, S. Venkataraman, Efficient solution algorithms for factored MDPs, Journal of Artificial Intelligence Research 19 (2003) 399–468.
[29] J. Hoey, R. St-Aubin, A. Hu, C. Boutilier, SPUDD: Stochastic planning using decision diagrams, in: Proceedings of Uncertainty in Artificial Intelligence (1999), pp. 279–288.