Generalized First Order Decision Diagrams for First Order Markov Decision Processes

Saket Joshi
Tufts University
Medford, MA, USA
sjoshi01@cs.tufts.edu

Kristian Kersting*
Fraunhofer IAIS
Sankt Augustin, Germany
kristian.kersting@iais.fraunhofer.de

Roni Khardon
Tufts University
Medford, MA, USA
roni@cs.tufts.edu

Abstract

First order decision diagrams (FODD) were recently introduced as a compact knowledge representation expressing functions over relational structures. FODDs represent numerical functions that, when constrained to the Boolean range, use only existential quantification. Previous work developed a set of operations over FODDs, showed how they can be used to solve relational Markov decision processes (RMDP) using dynamic programming algorithms, and demonstrated their success in solving stochastic planning problems from the International Planning Competition in the system FODD-Planner. A crucial ingredient of this scheme is a set of operations to remove redundancy in decision diagrams, thus keeping them compact. This paper makes three contributions. First, we introduce Generalized FODDs (GFODD) and combination algorithms for them, generalizing FODDs to arbitrary quantification. Second, we show how GFODDs can be used in principle to solve RMDPs with arbitrary quantification, and develop a particularly promising case where an arbitrary number of existential quantifiers is followed by an arbitrary number of universal quantifiers. Third, we develop a new approach to reduce FODDs and GFODDs using model checking. This yields a reduction that is complete for FODDs and provides a sound reduction procedure for GFODDs.

1 Introduction

Recently, Boutilier *et al.* [2001] have shown how ideas about relational MDPs (RMDP) can be used to solve stochastic planning problems. Several authors have developed different representation schemes and algorithms implementing this idea [Kersting *et al.*, 2004; Hölldobler *et al.*, 2006; Sanner and Boutilier, 2009; Wang *et al.*, 2008]. In particular, [Wang *et al.*, 2008; Joshi and Khardon, 2008] introduced the 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 has two main limitations. The first is representation power, where 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 recall of faulty products requires quantifier prefix $\exists\forall$ for the goal: there exists a depot such that all products are in the depot. The second is that manipulation algorithms for FODDs require special reductions to ensure their size is small. Such reductions have been introduced but they are not complete.

The paper makes three contributions. We introduce Generalized FODDs that allow for arbitrary quantification. We show how they can be used to solve RMDPs with arbitrary quantification. We provide a new approach to reduction based on model checking. This provides a complete reduction for FODDs and a sound reduction to some quantifier settings of GFODDs. This is a significant extension of the scope of the FODD approach to solving stochastic planning problems, and a significant improvement of our understanding of their reductions. Due to space constraints all proofs and some details are omitted from the paper; they are available in the long version of this paper.

Relational MDPs

A Markov decision process (MDP) is a mathematical model of the interaction between an agent and its environment [Puterman, 1994]. Formally a MDP is a 4-tuple $<S, A, T, R>$ defining a set of states S, set of actions A, a transition function T defining the probability $P(s' \mid s, a)$ of getting to state s' from state s 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 expected total 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.

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. Once the optimal value function is known, a policy can be generated by assigning to each state the action that maximizes expected value. Hoey *et al.* [1999] showed that if $R(s)$, $P(s' \mid s, a)$ and $V(s)$ can be represented using algebraic decision dia-

*Supported by the Fraunhofer ATTRACT fellowship STREAM.

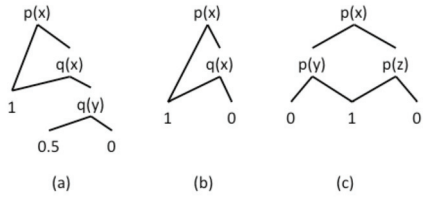

Figure 1: An example FODD

grams (ADDs) [Bahar *et al.*, 1993], then value iteration can be performed entirely using the ADD representation thereby avoiding the need to enumerate the state space. This provides a solution for propositionally factored MDPs but does not handle relational structure. Later Boutilier *et al.* [2001] developed the Symbolic Dynamic Programming (SDP) algorithm in the context of situation calculus. This algorithm provided a framework for dynamic programming solutions of RMDPs that was later employed in several formalisms and systems [Kersting *et al.*, 2004; Hölldobler *et al.*, 2006; Sanner and Boutilier, 2009; Wang *et al.*, 2008; Joshi and Khardon, 2008]. As in the propositional case, each portion of the Bellman equation is captured abstractly so that computation is shared by identical portions. Further details about the algorithm are given in Section 5. Importantly, in this scheme, we need an efficiently manipulable representation assigning values to abstract states.

First Order Decision Diagrams

This section briefly reviews FODDs [Wang *et al.*, 2008] using standard terminology from first order logic (e.g. [Lloyd, 1987]). 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` and is labeled by an atom 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. Three examples of FODDs are given in Figure 1; in these and all diagrams in the paper left going edges represent the `true` branches 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 a truth value of each predicate over these objects. In the context of RMDPs, an interpretation represents a state of the world with the objects and relations among them. The semantics of FODDs is defined as follows [Groote and Tveretina, 2003; Wang *et al.*, 2008]. Given a FODD and an interpretation, a *valuation* assigns each variable in the FODD to an object in the interpretation. If B is a FODD and I is an interpretation, a valuation ζ 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 $max_\zeta Map_B(I, \zeta)$, i.e. an aggregation of $Map_B(I, \zeta)$ over all valuations ζ. For example, consider the FODD in Figure 1(b) and the interpretation I with objects a, b, c and where the only true atoms are $p(a), q(b)$. The valuations $\{x/a\}$, $\{x/b\}$, and $\{x/c\}$ will produce the values 1, 1 and 0, respectively. By the max aggregation semantics, $Map_B(I) = max\{1, 1, 0\} = 1$. Thus, this FODD is equivalent to the formula $\exists x, p(x) \vee q(x)$. In general, max aggregation yields existential quantification when leaves are binary. When using numerical values we can similarly capture value functions for RMDPs.

Akin to ADDs, FODDs can be combined under arithmetic operations, and reduced in order to remove redundancies. Previous work has introduced several reduction operators. Intuitively, redundancies in FODDs arise in two different ways. The first observes that some edges may never be traversed by any valuation. Reduction operators for such redundancies are called strong reduction operators and they preserve $Map_B(I, \zeta)$ for every valuation ζ (thereby preserving $Map_B(I)$). On the other hand, weak reduction operators preserve $Map_B(I)$ but not necessarily $Map_B(I, \zeta)$ for every ζ. Weak reductions allow us to prune the diagrams further. All weak reductions previously introduced rely on notions of implication of reachability between different paths in the diagram, combined with a notion of value domination between the same parts [Wang *et al.*, 2008; Joshi and Khardon, 2008].

Weak reductions offer some subtle difficulties discussed in previous work. One of the issues is illustrated by the example in Figure 1(c). This simple FODD contains only 2 paths leading to non-zero leaves. Notice that whenever there is a valuation traversing one of the paths, there is another valuation traversing the other and reaching the same leaf. Thus either path can be safely removed from the diagram, but at least one must be kept. This suggests that we impose an ordering among paths that will indicate which one is to be preferred in such cases.

Definition 1 *A descending path ordering (DPO) is an ordered list of all paths from the root to a leaf 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 value can be set arbitrarily.*

2 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 directly. The map of a diagram is generated by aggregation of values obtained by running all possible valuations through the FODD. Therefore, if we document the behavior of every possible valuation under every possible interpretation, that is, which path it traverses under which interpretation, we can identify parts of the diagram that are never instrumental in 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. 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 such interpretations are consistent with PF(p)(ζ), where PF(p) denotes the path formula of path p which is the conjunction of literals on the path. Therefore the most general interpretation that forces ζ

through p can be viewed as a key or identifier for its equivalence class. 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 *getValue* does exactly that by simulating the run of a valuation through a FODD. The input to *getValue* is the FODD B to be reduced and a valuation ζ. The output of the procedure is a set of $<leaf, p, I>$ triplets, where $leaf$ is the leaf reached by ζ by traversing path p and I = PF(p)(ζ). The output must contain one triplet corresponding to every path in B such that PF(p)(ζ) is consistent. This can be done by traversing the diagram and, when we reach a node whose truth value has not yet been defined, recursively collecting the paths and values for both possible truth values.

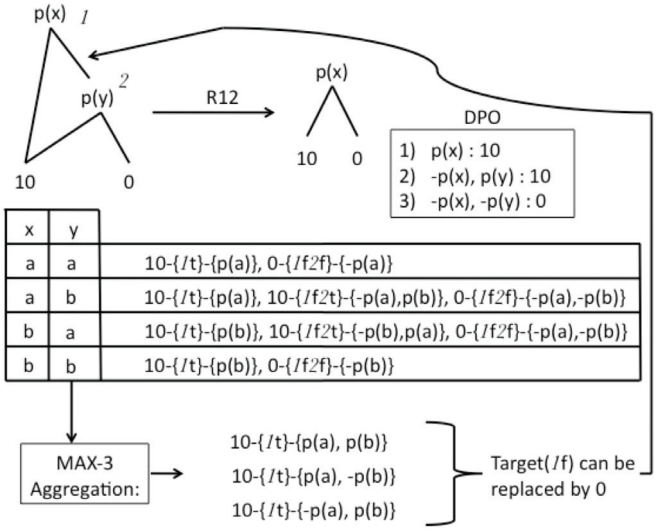

Figure 2: Example of R12

Figure 2 shows an example of the R12 reduction. The reduction is applied to the FODD on the left to reduce it to the FODD on the right. The table illustrates the result of running the *getValue* procedure on all possible valuations over the set of domain 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 *path consisting of the true edge of node* 1), under abstract interpretation $\{p(a)\}$, or it reaches a 10 leaf by traversing path $\{1f2t\}$ under abstract interpretation $\{\neg p(a), p(b)\}$ or otherwise it reaches the 0 leaf via path $\{1f2f\}$.

The next step is to generate all possible ways in which an aggregate value can be derived. Once again we avoid enumerating all interpretations. The table gives sufficient information to list all possible ways to aggregate over the set of all valuations. Just consider all combinations of behaviors over the set of valuations. Every combination (as long as it is consistent) can produce an aggregate value or the map. The aggregation, however, has to be done so as to expose the valuations (and paths) that prove to be instrumental in determining the map. Intuitively, paths that remain unexposed in spite of listing all possible ways to aggregate over the set of all valuations are not instrumental and can be removed. To this end, we introduce variants of the max aggregation function.

Generalized Aggregation Functions: max^2 and max^3

max^2 is defined relative to a fixed DPO, PL. The input is a set of 3-tuples of the form $<v_i, path_i, I_i>$ each corresponding to a valuation ζ_i so that ζ_i traverses path p_i in the FODD under interpretation I_i to reach leaf v_i. The output is a 3-tuple $<v_o, path_o, I_o>$ where $v_o = max_i[v_i]$, $I_o = \bigcup_{i=1} I_i$, and $path_o$ is the path of least index, under the order imposed by PL, with leaf value v_o.

The example in Figure 2 shows the DPO and aggregation results derived from the table. Each of the 3 resultant tuples is derived by collecting one tuple from every row and applying max^2 to the collection. E.g., aggregating over $<10, \{1t\}, \{p(a)\}>$, $<10, \{1t\}, \{p(a)\}>$, $<10, \{1t\}, \{p(b)\}>$, and $<10, \{1t\}, \{p(b)\}>$, using max^2 gives $<10, \{1t\}, \{p(a), p(b)\}>$ indicating that there is a possible aggregation where the path consisting of the edge $\{1t\}$ is instrumental in determining the map.

max^3 just runs max^2 on every possible combination of triplets over the list of valuations and returns the results of those where I_o is consistent and $v_o > 0$. The example in Figure 2 shows the result of applying max^3 to the elements in the table. Only 3 of the $2 \times 3 \times 3 \times 2 = 36$ possible combinations result in a consistent combined interpretation and positive value. Aggregations resulting in 0 value are ignored because 0 is uninteresting under the max aggregation.

To summarize, R12 is defined as follows: we fix a DPO PL, invent as many new objects as the number of variables in B and generate U, the set of all possible valuations of the variables in B over these objects. Then we run *getValue* on each valuation to generate a table as in Figure 2 and run max^3 on it to generate S, the set of resultant triplets. At the end we partition the set of edges in B into 2 sets: E', the set of edges appearing in any path in any triplet in S, and E, the set of edges in B that are not in E'. Intuitively, the edges in E do not belong to any path that determines the map and they can be removed. We say that a path is instrumental if it is the least index path for the DPO reachable for some interpretation. The reduction satisfies the following properties:

Lemma 1 *If there exists an instrumental path p_i under DPO PL that crosses edge e in B and reaches a non-zero leaf, then $\exists$ I_o such that $\{leaf(p_i), p_i, I_o\} \in S$ and therefore $e \in E'$.*

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

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

While this does not provide a normal form for FODDs, i.e. two semantically equivalent diagrams can be reduced but have different syntax, it provides much stronger reduction power compared to previous work. E.g., R12 reduces the

FODD in Figure 1(a) to Figure 1(b). Whenever a valuation reaches the 0.5 leaf there is another valuation traversing one of the two paths reaching the 1 leaf. However, neither of the path (or edge) formulas are individually implied by the formula for the path reaching the 0.5 leaf. Since all previous reductions are based on some notion of single path implication they are inapplicable. R12, on the other hand, is very flexible since it tracks reachability by way of model checking.

3 Generalized FODDs syntax and semantics

A significant extension to expressive power of FODDs can be obtained by changing the aggregation function. E.g., min aggregation leads to universal quantification. Other possible aggregation functions include $\sum$, $mean$, etc. We define Generalized FODDs as follows:

Definition 2 *A Generalized First Order Decision Diagram (GFODD) is a 2-tuple $<V, D>$, where, V, the aggregation function, is an ordered list of distinct variables each associated with its own aggregation operator. D is a FODD except that the leaves can be labeled by a special character d (for discard).*

GFODD semantics

The semantics follow the approach of FODDs except that the aggregation operation is now defined by $V = [(op^1_{v_1}) \cdots (op^n_{v_n})]$. Here every v_i is a variable of the GFODD (ordered v_1 to v_n) and the corresponding op_i is the aggregation operator associated with it. Consider the set of all possible valuations defined over the domain of interpretation I. Each valuation ζ is associated with a value $Map_B(I, \zeta)$. 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. 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 ($Map_B(I, \zeta)$) by their aggregate value produced by applying op^n to the set. Any discard values in the block are removed before applying op^n. If we do this for every block we are left with the set of all possible valuations defined over the variables $v_1 \cdots v_{n-1}$ each associated with a value. We repeat the procedure for variables v_{n-1} to v_1 to produce a final aggregate value, $Map_B(I)$, obtained by nesting aggregation operators with op^n being the innermost.

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

where $[v_1 \cdots v_n]$ is the corresponding valuation. We illustrate this using the example in Figure 3 where GFODD B captures the following statement from the logistics domain: $\exists c \forall b$, box b is in city c. The output of B is 10 if all boxes are in one city and 0 otherwise. Aggregation is done from right to left, one variable at a time. In the example, therefore, first aggregate the values $Map_B(I, \zeta)$ over all assignments of the variable b, using the min operation, producing exactly one value per binding of variable c, and then aggregating all of the produced values over all bindings of variable c using the max operation. 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

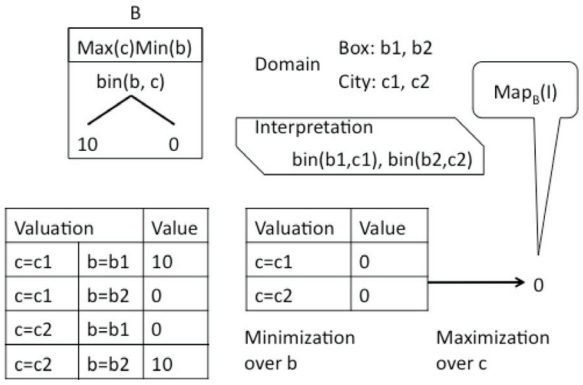

Figure 3: An Generalized FODD Example

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 $\exists c, \forall b, box(b) \rightarrow city(c) \wedge bin(b, c)$.

Combining GFODDs

Our Value Iteration algorithm requires operations max, $+$ and $\times$ over functions defined by GFODDs. We perform these by GFODD combination.

Definition 3 *GFODD B is a combination of GFODDs B_1 and B_2 under combination operator op_c iff $\forall$ interpretations I, $Map_B(I) = Map_{B_1}(I)\ op_c\ Map_{B_2}(I)$.*

Definition 4 *Combination operator op_c and aggregation operator op^a are a safe pair iff for any non-negative values $x_1, \cdots, x_k$ and non-negative constant b, $op^a(x_1, \cdots, x_k)\ op_c\ b = op^a(x_1\ op_c\ b, \cdots, x_k\ op_c\ b)$.*

E.g., aggregation operator max and combination operator $+$ are a safe pair because for any set $S = \{c_1 \cdots c_m\}$ and constant b, $max\{c_1 \cdots c_m\} + b = max\{c_1 + b, \cdots c_m + b\}$. Aggregation operator $mean$ and combination operator max are not a safe pair.

FODDs can be combined with the *apply* operation [Wang *et al.*, 2008]. *Apply* produces a combination of 2 FODDs under combination operator op_c by choosing the smaller root (according to the FODD predicate order) to be the root of the resultant FODD and then recursing on the subdiagrams. When the computation reaches the leaves, the result is op_c applied to the leaf values. for GFODDs, if either leaf value is d, so is the result. The following Theorem shows that GFODDs can be combined leaving some freedom in the order of aggregation. We call the resulting procedure *Ex-apply*.

Theorem 3 *Given GFODDs $B_1 = <V_1, D_1>$ and $B_2 = <V_2, D_2>$, $V_1 \cap V_2 = \phi$, and combination operator op_c, if all operators in $V_1 \cup V_2$, form a safe pair with op_c, $D = apply(D_1, D_2, op_c)$ and V is obtained by combining V_1 and V_2 in a way that preserves the order of variables within V_1 and within V_2, then $B = <V, D>$ is a combination of B_1 and B_2.*

4 R12 for Max*Min* Aggregation

The R12 reduction can be extended to GFODDs with min aggregation by defining generalized aggregation function min^3. min^3 is just a dual of max^3 except that no special treatment

is given to paths reaching the 0 leaf and in the reduction procedure, targets of edges in E are replaced by d instead of 0. Since these edges are not instrumental in determining the map and d values are just discarded during aggregation, correctness is preserved.

We next extend R12 to GFODDs with max^*min^* aggregation. In this case the aggregation function V consists of a series of zero or more max operators followed by a series of zero or more min operators. The corresponding case in first order logic has the quantifier prefix $\exists^*\forall^*$ and is decidable. V can be written as V^lV^r, where V^l and V^r contain all max and min aggregated variables respectively. The set U of all possible valuations of the variables in diagram B can be split into U^l and U^r, the sets of all valuations over the variables in V^l and V^r. Thus, for all $\zeta \in U$, $\zeta = \zeta^l\zeta^r$ where $\zeta^l \in U^l$ and $\zeta^r \in U^r$. By the definition of GFODD semantics, for any interpretation I,

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

Consider evaluating B on some interpretation I. U can be viewed as divided into blocks, each corresponding to one valuation ζ^l. During aggregation each block is collapsed under the min aggregation and the aggregate values of all blocks are collapsed under the max aggregation. Intuitively, as long as R12 preserves all paths reaching the smallest leaf (under a DPO) in every block under every interpretation, the map will also be preserved. Other paths can be removed. This does not reduce all edges possible, but it is easy to track. R12 for max^*min^* aggregation is identical to the R12 procedure for min aggregation with the following exceptions.

(1) The set U of all possible valuations is generated as follows. O^l and O^r are disjoint sets of respectively $|V^l|$ and $|V^r|$ newly invented objects. U^l and U^r are the sets of all possible valuations of variables in V^l and V^r over O^l and $O^l \cup O^r$ respectively. $U = \{\zeta^l\zeta^r \mid \zeta^l \in U^l \text{ and } \zeta^r \in U^r\}$.

(2) The generalized aggregation function $maxmin^3$ returns all the triplets generated by applying min^3 to each block.

Figure 4 shows an example of this reduction. Here V^l = Max(x) and V^r = Min(y) making $|V^l| = |V^r| = |O^l| = |O^r| = 1$. Therefore we invent $O^l = \{a\}$ and $O^r = \{b\}$. Notice that the table built by the procedure consists of a single block (since only one variable is associated with the max and so there is only one ζ^l) but in general this is repeated for every block. The targets of all edges other than the ones present in the paths of the resultant triplets (shown below the table) can be replaced by the value d. This reduction satisfies the following properties.

Lemma 2 *If there exists a path p_i reaching the smallest leaf (under the DPO PL) in some block under some interpretation and p_i crosses e in B then $\exists$ I_o such that $\{leaf(p_i), p_i, I_o\} \in S$ and thus $e \in E'$.*

Theorem 4 (soundness) *For GFODD B with the max^*min^* aggregation, if $B' = R12(B)$, then $\forall$ interpretations I, $Map_B(I) = Map_{B'}(I)$.*

A closer examination of the discussion above shows that we can make more precise constraints on the values of edges

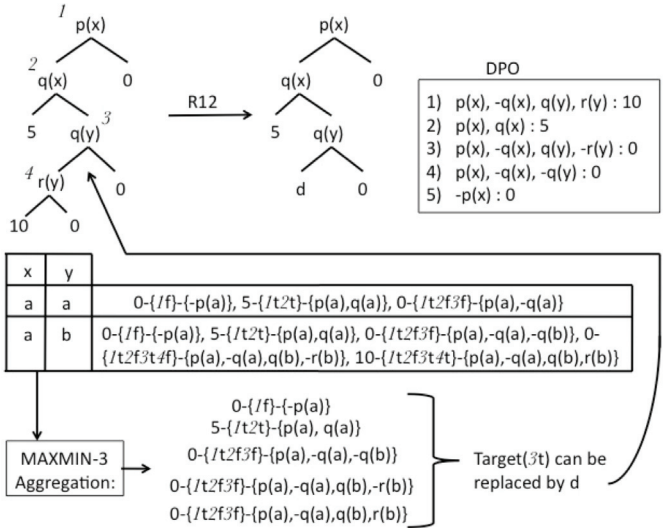

Figure 4: Example of R12 for max^*min^* aggregation

participating in the evaluation of diagram B on interpretation I. In particular, the winning path must be preserved, the value of edges participating in the same block (using the same ζ_l) can go down but they must not be smaller than the final value, and edges in other blocks can be reduced to zero. Thus tighter bookkeeping may allow us to prune the diagram much more than the version of R12 given above. We leave this to be investigated in future work.

5 Value Iteration with GFODDs

Value Iteration with GFODDs is an instance of the SDP algorithm [Boutilier *et al.*, 2001]. We restrict attention to cases where the reward function is represented by a GFODD with max^*min^* aggregation function. The following discussion shows why our algorithm *VI-GFODD* produces the correct result for each of the following 4 steps of SDP.

Regression: The $n-1$ step-to-go value function V^{n-1} is regressed over every deterministic variant a_i^j of every action a_i to produce $RegV_i^j$ by replacing each node in V^{n-1} by its corresponding Truth Value Diagram (TVD) without changing the aggregation function. A TVD for a predicate under deterministic action a_i^j describes conditions under which the predicate becomes true after a_i^j is executed. Wang *et al.* [2008] impose the constraint that TVDs cannot include free variables and therefore regression is correct regardless of the aggregation function.

Add Action Variants: The Q-function $QReg_i^a = \Sigma_j Pr(a_i^j) RegV_i^j$ for each action a_i is generated by combining regressed diagrams using Ex-apply. Since all argument GFODDs to Ex-apply have max^*min^* aggregation, by Theorem 3 all max operators can be pushed to the head of the aggregation function, and the result also has max^*min^* aggregation. Correctness is guaranteed because both max and min form a safe pair with all combination operators required for Value Iteration ($+$, $\times$, and max).

Object Maximization: This involves converting action parameters in $QReg_i^a$ to max aggregated variables and ap-

pending these to the head of the aggregation function to produce Q^{a_i}. Aggregation in Q^{a_i} chooses a single binding for the new variables that provides the best value in $QReg_i^a$ over all action variants, thereby ensuring correctness. Again max^*min^* aggregation is preserved in Q^{a_i}.

Maximize over Actions: The n step-to-go value function $V^n = Max_i[R(S) + \gamma Q^{a_i}]$, is generated by combining diagrams using Ex-apply. As above, by Theorem 3 the result can be written using max^*min^* aggregation.

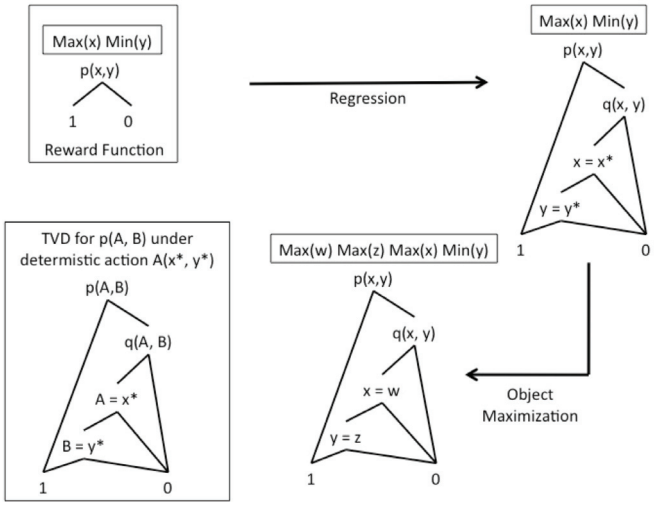

Figure 5: Example of GFODD Regression and Object Maximization

Figure 5 shows an example of the algorithm using GFODDs for a simple domain with one deterministic action $A(x^*, y^*)$. Notice that the 2^{nd} and 4^{th} steps in the algorithm are not required in this case. The reward is 1 if $\exists x, \forall y, p(x, y)$ and 0 otherwise. The action $A(x^*, y^*)$, is defined such that $p(x, y)$ is true after the action if either $p(x, y)$ was true before or $q(x, y)$ was true and $A(x, y)$ was performed. Since the action can make at most one $p(x, y)$ true at a time, intuitively, the regressed diagram should capture the following conditions for returning a value of 1. Either $\exists x, \forall y, p(x, y)$ or $\exists x$, such that for all but one y, $p(x, y)$ is true and for that y, $q(x, y)$ is true. It is easily verified that the resultant diagram captures these cases. The next theorem shows that the result is correct; using R12 we can also keep the diagrams compact.

Theorem 5 *For any First Order MDP with a reward function of the form $<max^*min^*, D>$, algorithm VI-GFODD produces the correct value function at every iteration.*

6 Conclusions and Future Work

This paper significantly extends the representation power of first order decision diagrams and our algorithmic understanding of their reduction. Generalized FODDs allow for arbitrary aggregation functions and basic operations on them can be done just as in the existential case. 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 the more expressive setting. Previous implementations of first order Value Iteration [Sanner and Boutilier, 2009; Joshi and Khardon, 2008] have resorted to heuristic treatment of universal goals. Using the new formulation, this can be captured and handled naturally. The other main contribution in the paper is the idea and analysis of model checking reductions. The completeness result for the FODD case falls short of being a normal form, but is much stronger than previous reductions. Examples by Wang *et al.* [2008] using a simple decidable fragment show that for normal form we may need some syntactic manipulation of diagrams so going beyond completeness may be hard or expensive to compute.

This work suggests several questions for future. The model checking reduction can probably be improved by further analysis, as discussed in Section 4. The model checking reductions currently require enumeration of substitutions. A promising idea is to use a sample of interpretations, judicially chosen, and reduce the diagrams relative to these interpretations.

References

[Bahar *et al.*, 1993] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *IEEE /ACM International Conference on Computer Aided Design*, 1993.

[Boutilier *et al.*, 2001] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order mdps. In *Proceedings of IJCAI*, pages 690–700, 2001.

[Groote and Tveretina, 2003] J. Groote and O. Tveretina. Binary decision diagrams for first order predicate logic. *Journal of Logic and Algebraic Programming*, 57:1–22, 2003.

[Hoey *et al.*, 1999] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proceedings of UAI*, pages 279–288, 1999.

[Hölldobler *et al.*, 2006] S. Hölldobler, E. Karabaev, and O. Skvortsova. FluCaP: a heuristic search planner for first-order MDPs. *Journal of Artificial Intelligence Research*, 27:419–439, 2006.

[Joshi and Khardon, 2008] S. Joshi and R. Khardon. Stochastic planning with first order decision diagrams. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2008.

[Kersting *et al.*, 2004] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In *Proceedings of ICML*, 2004.

[Lloyd, 1987] J.W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987. Second Edition.

[Puterman, 1994] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley, 1994.

[Sanner and Boutilier, 2009] Scott Sanner and Craig Boutilier. Practical solution techniques for first-order mdps. *Artif. Intell.*, 173:748–488, 2009.

[Wang *et al.*, 2008] C. Wang, S. Joshi, and R. Khardon. First order decision diagrams for relational mdps. *JAIR*, 31:431–472, 2008.