Multi-state Directed Acyclic Graphs

Michael Wachter[1] and Rolf Haenni[1,2]

[1] University of Bern, Switzerland
{wachter,haenni}@iam.unibe.ch
[2] Bern University of Applied Sciences, Switzerland
rolf.haenni@bfh.ch

Abstract. This paper continues the line of research on the representation and compilation of propositional knowledge bases with propositional directed acyclic graphs (PDAG), negation normal forms (NNF), and binary decision diagrams (BDD). The idea is to permit variables with more than two states and to explicitly represent them in their most natural way. The resulting representation languages are analyzed according to their succinctness, supported queries, and supported transformations. The paper shows that most results from PDAGs, NNFs, and BDDs can be generalized to their corresponding multi-state extension. This implies that the entire knowledge compilation map is extensible from propositional to multi-state variables.

1 Introduction

Boolean functions play a crucial role in many areas of computer science and mathematics, most notably in Artificial Intelligence, digital system design, formal verification, mathematical logic, reliability theory, and combinatorial optimization. They are fundamental whenever knowledge is represented by propositional variables, i.e. through a set of possible states in the corresponding multi-dimensional Boolean space.

In practice, working with Boolean functions presupposes efficient ways to represent them. Among the existing approaches for representing Boolean functions are truth tables, Karnaugh maps, sum-of-products such as DNFs or prime implicants, product-of-sums such as CNFs or prime implicates, and most notably *binary decision diagrams* (BDD) [1,2,3], *negation normal forms* (NNF) [4,5], *propositional directed acyclic graphs* (PDAG) [6], and all their derivatives. Some of these forms are known to be impractical, as they impose representations of exponential size for *most* possible r-ary functions [7], but many BDD, NNF, and PDAG forms provide polynomial representations at least for *many* functions.

The restriction of these techniques to propositional variables does not entirely meet the requirements of real-world models, which are often not limited to Boolean variables. For example, the possible states of a traffic light (in most parts of the world) are red, yellow, and green. At a particular time, the traffic light is in exactly one of these states. We will use the following terminology to distinguish the different types of variables: *propositional*, *Boolean*, or *binary* variables have exactly two states, *non-binary* variables have more than two states,

Z. Kobti and D. Wu (Eds.): Canadian AI 2007, LNAI 4509, pp. 464–475, 2007.

and *multi-state* variables have two or more states.[1] In addition, we suppose that each (binary, non-binary, or multi-state) variable has a unique (but typically unknown) *true* state.

Multi-state variables have been discussed in the literature of *decision diagrams* [10], where *multivalued decision diagrams* (MDD) arise as an extension of BDDs to multi-state variables. They are an alternative to the usual replacement of multi-state variables by $\lceil \log_2 \ell \rceil$ Boolean variables, where ℓ denotes the number of possible states. In this way, MDDs can be transformed into BDDs with a linear growth in size, which relativizes the benefits of MDDs over BDDs, especially if ℓ is small. The conclusion in [10] is the following:

> "*MDDs are useful if the considered function has a natural description with multivalued variables.*" [10, Section 9.1, page 216]

In applications of NNFs, especially in the contexts of probabilistic reasoning, Bayesian networks, and model counting [11,12,13], it is common to use similar Boolean encodings for multi-state variables. These encodings typically use ℓ (or $\ell - 1$) auxiliary Boolean variables, i.e. one for each state (except for the last one). The exclusivity and exhaustiveness of these auxiliary variables requires explicit representations of corresponding *exclusive ORs*, which is a non-negligible overhead, especially if ℓ is large. Another problem of these Boolean encodings is the computation of probabilities, if independent probability mass functions are given for all multi-state variables. The core of the problem is the fact, that the auxiliary Boolean variables are no longer independent. It is possible to overcome this difficulty by transforming the given probabilities into conditional probabilities [11,12], but the existing solutions are rather cumbersome.

If we decide to work with multi-state variables from the beginning, these problems all disappear, including the one of selecting an appropriate Boolean encoding. The goal of this paper is thus to modify the existing PDAG, NNF, and BDD languages to multi-state variables (unless it is not yet done elsewhere). We will show that most theoretical results remain valid. In this way, we add an additional dimension to the knowledge compilation map promoted in [5,6].

The remainder of this paper is organized as follows. In Sect. 2, we extend the definition of PDAGs to multi-state variables, compare the resulting *multi-state directed acyclic graphs* (MDAG) with PDAGs, and finally define different MDAG sub-languages. In Sect. 3, some theoretical results about the succinctness, the supported queries, and the supported transformations of PDAGs are generalized to MDAGs. Section 4 concludes the paper.

2 Multi-state Directed Acyclic Graph

Let $\mathbf{V} = \{V_1, \ldots, V_r\}$ be a set of r variables and suppose that Ω_{V_i} denotes the finite set of states of V_i. A finite indicator function f is defined by $f : \Omega_{\mathbf{V}} \to \mathbb{B}$,

[1] To outline the difference to *multivalued* or *many-valued* logics, where logical sentences are mapped into more than two truth values [8,9], we prefer to use the term 'multi-state' instead of 'multivalued' or 'many-valued'.

where $\Omega_{\mathbf{V}} = \Omega_{V_1} \times \cdots \times \Omega_{V_r}$ and $\mathbb{B} = \{0, 1\}$. To emphasize the fact that f is a mapping from the Cartesian product $\Omega_{V_1} \times \cdots \times \Omega_{V_r}$ to $\{0, 1\}$, we will call it a *Cartesian indicator function* (CIF). The so-called *satisfying set* $S_f = \{\mathbf{x} \in \Omega_{\mathbf{V}} : f(\mathbf{x}) = 1\} = f^{-1}(1)$ of f is the set of r-dimensional vectors $\mathbf{x} \in \Omega_{\mathbf{V}}$ for which f evaluates to 1. Special cases of finite CIFs are Boolean functions (BF), where $\Omega_{V_i} = \mathbb{B}$, and therefore $\Omega_{\mathbf{V}} = \mathbb{B}^r$.

The most general forms for representing BFs are PDAGs. As shown in [6], the well-known NNFs, BDDs, and their derivatives correspond to subsets of PDAGs. While *multivalued decision diagrams* (MDD) have been proposed as an extension of BDDs to multi-state variables in the context of *decision diagrams* [10], there is no such extension for NNFs or PDAGs. PDAGs, and therewith NNFs, will be extended to multi-state variables below. We will see that the resulting language also includes the existing MDDs. As in the case of PDAGs and NNFs, the representation we propose here is based on directed acyclic graphs, but now we impose some particularities.

Definition 1. *A* multi-state DAG *(MDAG) is a rooted directed acyclic graph, where:*

1. Leaves *are represented by* $\Box$ *and labeled with* $\top$ *(true),* $\bot$ *(false), or* $X{=}x$*, where* $X \in \mathbf{V}$ *is a variable and* $x \in \Omega_X$ *is one of its states;*
2. Non-leaves *are represented by* $\triangle$ *(logical and),* $\triangledown$ *(logical or), or* $\Diamond$ *(logical not).* $\triangle$*- and* $\triangledown$*-nodes have at least one child,* $\Diamond$*-nodes have exactly one child.*

In a MDAG, each node α represents a finite CIF f_α by

$$f_\alpha = \begin{cases} \bigwedge_{i=1}^{t} f_{\beta_i} = \min_{i=1}^{t} f_{\beta_i}, & \text{if } \alpha \text{ is an } \triangle\text{-node with children } \beta_1, \ldots, \beta_t, \\ \bigvee_{i=1}^{t} f_{\beta_i} = \max_{i=1}^{t} f_{\beta_i}, & \text{if } \alpha \text{ is an } \triangledown\text{-node with children } \beta_1, \ldots, \beta_t, \\ \neg f_\psi = 1 - f_\psi, & \text{if } \alpha \text{ is a } \Diamond\text{-node with the child } \psi, \\ 1, & \text{if } \alpha \text{ is a } \Box\text{-node labeled with } \top, \\ 0, & \text{if } \alpha \text{ is a } \Box\text{-node labeled with } \bot, \\ f_{X=x}, & \text{if } \alpha \text{ is a } \Box\text{-node labeled with } X{=}x, \end{cases}$$

where $f_{X=x}(\mathbf{x})$ with $\mathbf{x} \in \Omega_{\mathbf{V}}$ is defined by

$$f_{X=x}(\mathbf{x}) = \begin{cases} 1, & \text{if } x \text{ is the corresponding value of } X \text{ in } \mathbf{x}, \\ 0, & \text{otherwise.} \end{cases}$$

The MDAG depicted in Fig. 1 represents the finite CIF $f = ([Y{=}y_1] \wedge [X{=}x_1]) \vee ([Y{=}y_2] \wedge \neg[X{=}x_2]) \vee ([X{=}x_2] \wedge [Y{=}y_3])$. Note that with this, Ω_X and Ω_Y are not necessarily restricted to $\{x_1, x_2\}$ and $\{y_1, y_2, y_3\}$ from the beginning.

Formally, we will write $\texttt{MDAG}_{\mathbf{V}}$ for the set of all possible MDAGs with respect to $\mathbf{V}$. We follow the view from [5,6] and call $\texttt{MDAG}_{\mathbf{V}}$ a *language*. When no confusion is anticipated, we omit the reference to the set $\mathbf{V}$, i.e. we simply write MDAG

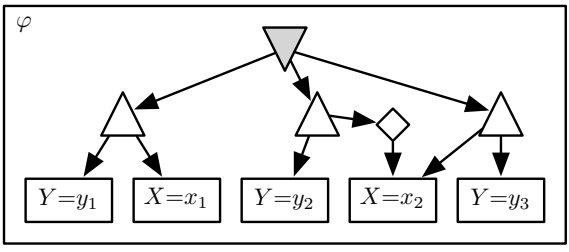

Fig. 1. The finite CIF f represented as the MDAG φ

instead of $\texttt{MDAG}_{\mathbf{V}}$ and Ω instead of $\Omega_{\mathbf{V}}$. Our convention is to denote MDAGs by lower-case Greek letters such as φ, ψ, or the like. Remember that any node α included in a MDAG φ defines its own (sub-) MDAG, and is thus another element of MDAG.

The number of edges of $\varphi \in \texttt{MDAG}$ is called its *size* and is denoted by $|\varphi|$. MDAGs are called *binary*, if no $\triangle$- or $\triangledown$-node has more than two children. The set of variables included in a sub-MDAG α of φ is denoted by $vars(\alpha)$. The *path-length* of a path from the root to a leave is the number of edges minus the number of $\Diamond$-nodes along the path. The *height* of φ, denoted by $h(\varphi)$, is its maximal path-length. Note that these concepts (size, binary, *vars*, path-length, height) have the same meaning for PDAGs.

Any finite CIF can be represented by a MDAG, so the MDAG language is *complete*. On the other hand, MDAGs are not *canonical*, i.e. we may have several equivalent MDAGs representing the same finite CIF. Two MDAGs $\varphi, \psi \in \texttt{MDAG}$ are *equivalent*, denoted by $\varphi \equiv \psi$, iff $f_\varphi(\mathbf{x}) = f_\psi(\mathbf{x})$ for all $\mathbf{x} \in \Omega$. Furthermore, φ *entails* ψ, denoted by $\varphi \models \psi$, iff $f_\varphi(\mathbf{x}) \leq f_\psi(\mathbf{x})$ for all $\mathbf{x} \in \Omega$. In terms of their satisfying sets, $S_{f_\varphi} = S_{f_\psi}$ means equivalence and $S_{f_\varphi} \subseteq S_{f_\psi}$ entailment. Again, both equivalence and entailment have the same meaning for PDAGs.

2.1 Sub-languages

We will now turn our attention to some sub-languages of MDAG. The classification of sub-languages is done according to the following properties, which are based on the ones given in [5,6]:

1. *Decomposability*: the sets of variables of the children of each $\triangle$-node α in φ are pairwise disjoint (i.e. if $\beta_1, \ldots, \beta_n$ are the children of α, then $vars(\beta_i) \cap vars(\beta_j) = \emptyset$ for all $i \neq j$);
2. *Determinism*: the children of each $\triangledown$-node α in φ are pairwise logically contradictory (i.e. if $\beta_1, \ldots, \beta_n$ are the children of α, then $S_{f_{\beta_i}} \cap S_{f_{\beta_j}} = \emptyset$ for all $i \neq j$);
3. *No-Negation*:[2] φ does not contain any $\Diamond$-node;

[2] No-negation corresponds to *simple-ngeation* in [5,6].

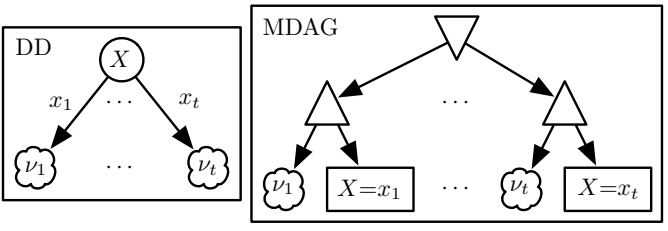

Fig. 2. A decision node X with $\Omega_X = \{x_1, \ldots, x_t\}$ and its decision structure. ν_i are further nodes of the DD, resp. MDAG.

4. *Flatness*: $h(\varphi) \leq 2$;
5. *Simple-Conjunction*: the children of each $\triangle$-node α in φ are leaves without any common variable (i.e. α is a proper *term*),
6. *Simple-Disjunction*: the children of each $\triangledown$-node α in φ are leaves without any common variable (i.e. α is a proper *clause*);
7. *Smoothness*: the children of each $\triangledown$-node α in φ include the same set of variables (i.e. if $\beta_1, \ldots, \beta_n$ are the children of α, then $vars(\beta_i) = vars(\beta_j)$ for all $i \neq j$).

Note that smoothness is not that important from a complexity viewpoint, unless we have flatness [5]. We will not further discuss smoothness in this paper. Simple-disjunction and simple-conjunction are characteristic for classical forms such as CNFs, DNFs, prime implicates, etc.

In addition to the basic properties above, we also consider some properties of decision diagrams (DD). According to [10,14], a decision diagram consists of non-leaves, so-called *decision nodes*, and leaves, so-called *terminals*. Decision nodes are represented by $\bigcirc$, labeled with $X \in \mathbf{V}$, and have outdegree $|\Omega_X|$, see left part of Fig. 2. In addition, Fig. 2 shows the one-to-one mapping between a decision node and its `MDAG` representation, called *decision structure*. Terminals are represented by $\square$ and labeled with 1 or 0 in decision diagrams, resp. with $\top$ or $\bot$ in MDAGs.

8. *Decision*: φ contains only decision structures and terminals.
9. *Read-once*: each path from the root to a terminal contains at most one decision node/structure for each variable $X \in \mathbf{V}$;
10. *Ordering*: on each path from the root to a leaf, the occurrence of decision structures respects a total ordering on $\mathbf{V}$;
11. π-*Ordering*: on each path from the root to a leaf, the occurrence of decision structures respects the globally specified ordering π on $\mathbf{V}$.

The sub-languages of `MDAG` are defined via these properties in the same way as sub-languages are defined for `PDAG` [6]. Table 1 shows `MDAG` and some of its most

Table 1. MDAG and some of its sub-languages according to the 11 properties. With 'x' we indicate that a language satisfies the corresponding property, and '(x)' means that this property is implied by other properties.

	decomposability	determinism	no-negation	flatness	simple-conjunction	simple-disjunction	smoothness	decision	read-once	ordering	π-ordering	Description	Boolean Case
MDAG												multi-state directed acyclic graph (MDAG)	PDAG
c-MDAG	x											decomposable MDAG	c-PDAG
d-MDAG		x										deterministic MDAG	d-PDAG
n-MDAG			x									negation-free MDAG	NNF ($\equiv$ n-PDAG)
cd-MDAG	x	x										decomposable deterministic MDAG	cd-PDAG
cn-MDAG	x		x									decomposable negation-free MDAG	DNNF ($\equiv$ cn-PDAG)
dn-MDAG		x	x									deterministic negation-free MDAG	d-NNF ($\equiv$ dn-PDAG)
cdn-MDAG	x	x	x									decomposable deterministic negation-free MDAG	d-DNNF ($\equiv$ cdn-PDAG)
f-MDAG				x								flat MDAG	f-PDAG
fn-MDAG			x	x								flat negation-free MDAG	f-NNF ($\equiv$ fn-PDAG)
MCNF			x	x		x						multi-state conjunctive normal form (MCNF)	CNF
MPI			x	x		x						multi-state prime implicates	PI
MDNF	(x)		x	x	x							multi-state disjunctive normal form (MDNF)	DNF
MIP	(x)		x	x	x							multi-state prime implicants	IP
d-MDNF	(x)	x	x	x	x							deterministic MDNF	d-DNF
MMODS	(x)	x	x	x	x		x					multi-state models	MODS
MDD		(x)	x					x				multivalued decision diagram (MDD)	BDD
FMDD	(x)	(x)	x					x	x			free MDD	FBDD
OMDD	(x)	(x)	x					x	(x)	x		ordered MDD (OMDD)	OBDD
π-OMDD	(x)	(x)	x					x	(x)	(x)	x	OMDD using order π	π-OBDD, OBDD$_<$

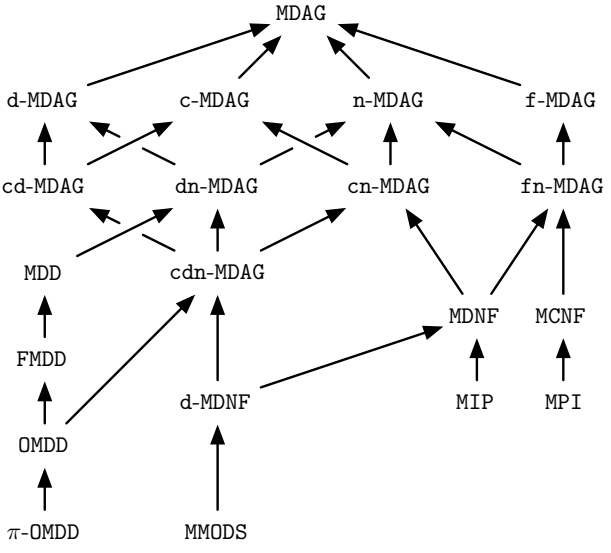

Fig. 3. Sub-language relationships for `MDAG`. An edge $L_1 \rightarrow L_2$ indicates that L_1 is a sub-language of L_2.

important sub-languages. Note that this table is far from being complete. We use `c`, `d`, `f`, and `n` to indicate that the properties decomposability, determinism, flatness, and no-negation hold. Figure 3 shows how the languages are related in terms of set inclusion.

2.2 MDAGs vs. PDAGs

In this subsection, the usage of MDAGs and PDAGs will be compared. For this purpose, we will first examine the special case of BFs, before considering the general case of finite CIFs. The reader may skip this subsection, if the interest is primarily on MDAGs.

Formally, PDAGs are almost like MDAGs, except that leaves are represented by $\bigcirc$ and labeled with $\top$ (true), $\bot$ (false), or X, where $X \in \mathbf{V}$ is a Boolean variable with $\Omega_X = \{0, 1\}$ [6]. The language of all possible PDAGs is denoted by `PDAG`. The left hand side of Fig. 4 depicts a PDAG ψ with $f_\psi = (\neg[X{=}1] \wedge [Y{=}1]] \vee ([X{=}1] \wedge \neg[Y{=}1])$. Leaves labeled with $\top$ ($\bot$) represent the constant BF which always evaluates to 1 (0). A leaf labeled with the propositional symbol X is interpreted as the assignment $X{=}1$, i.e. it represents the BF which evaluates to 1 iff $X = 1$. All other nodes ($\triangle$, $\triangledown$, $\Diamond$) have the same meaning as for MDAGs.

From a PDAG ψ representing a BF f, we obtain an MDAG φ representing the same BF f by simply replacing $\bigcirc$-nodes labeled with X by $\square$-nodes labeled with $X{=}1$, as shown in Fig. 4. Conversely, i.e. to obtain a PDAG ψ from a MDAG φ, $\square$-nodes labeled with $X{=}1$ are replaced by $\bigcirc$-nodes labeled with X, and

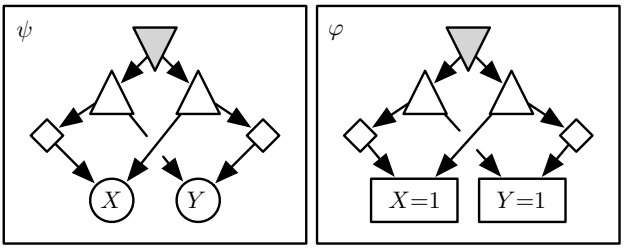

Fig. 4. The BF f represented by a PDAG ψ and a MDAG φ

$\Box$-nodes labeled with $X{=}0$ are replaced by $\Diamond$-nodes, whose children are $\bigcirc$-nodes labeled with X.

Proposition 1. *For every PDAG ψ, there is an equivalent MDAG φ with $|\psi| = |\varphi|$. Similarly, for every MDAG φ representing a Boolean function, there is an equivalent PDAG ψ with $|\varphi| \leq |\psi| \leq |\varphi| + r$.*

To represent CIFs by PDAGs, we need ways to transform them into BFs, i.e. each multi-state variable has to be replaced by auxiliary Boolean variables.

Decision Diagrams: Each multi-state variable can be replaced by $d = \lceil \log_2 \ell \rceil$ auxiliary Boolean variables, where ℓ denotes the number of possible states [10]. For each decision node in the MDD, this replacement induces a tree-shaped decision diagram of depth d and with $l - 1$ binary decision nodes. Figure 5 depicts the decision diagram for the multi-state variable X with $\Omega_X = \{x_1, x_2, x_3, x_4\}$ and its replacement with the auxiliary variables X_1 and X_2, i.e. for $d = 2$ and $\ell = 3$.

In [10], it is argued that this encoding corresponds to a linear transformation with a small constant. This argument is used to put the usefulness of MDDs into question. Such a conclusion is partly valid from a purely logical point of view and for small ℓ, but it does no longer hold when ℓ is large or when MDAGs are used to compute probabilities. In the latter case, the Boolean variables used to replace a multi-state variable are no longer independent, which disallows the classical method of probability computation. One way to overcome this is to derive respective conditional probabilities and to attach them to the edges as depicted in Fig. 5. This shows that the variable X_2 depends on X_1. Since the outgoing edges of the X_2 node have different probabilities, this is like using $\ell - 1$ new variables.

PDAGs and NNFs: In this context, the probabilities are attached to the variables. In our example, we would have to attach two different conditional probabilities to X_2, which is impossible. An obvious alternative replacement considers each decision node of the decision diagram as an auxiliary variable, i.e. $l-1$ variables in total, where the probabilities of the variables correspond to the probabilities attached to the edges. This is essentially the replacement proposed in [13].

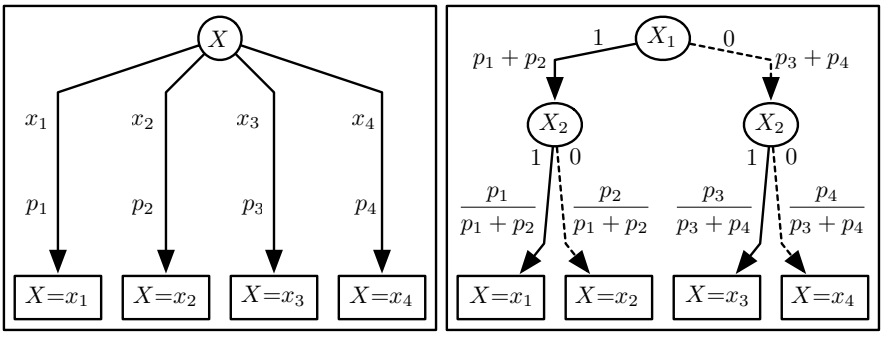

Fig. 5. Decision diagram for variable X and its replacement. The labels of the edges correspond to states and (conditional) probabilities.

An alternative replacement considers each state of a multi-state variable as a binary variable. This requires the explicit inclusion of an *exclusive or* over these auxiliary variables [11,12]. In this way, the switch to conditional probabilities is not necessary, but still the computation of probabilities becomes more difficult. A possible solution is to do some sort of *weighted model counting*, where the probabilities are attached to the leaves only, and their negations get the constant value 1.

The size of the different replacements and the additional effort strengthens our conclusion, namely that multi-state variables are useful and should be used, not encoded.

3 Succinctness, Queries and Transformations

The crucial properties of a language are its succinctness and the sets of queries and transformations supported in polynomial time. Depending on the application, we may come up with a set of queries and transformations, which the chosen language should support in polynomial time. If more than one language qualifies, the most succinct language provides the most compact representation. This is then the most appropriate language for the considered application.

In the following analysis of the MDAG language family, we will try to generalize as many results as possible from corresponding PDAG languages.

3.1 Succinctness

With respect to two languages $\mathtt{L}_1$ and $\mathtt{L}_2$, the intuitive idea of succinctness is to figure out whether finite CIFs are represented more compactly by elements of $\mathtt{L}_1$ or by elements of $\mathtt{L}_2$. The following definition corresponds to the one given in [5,6].

Definition 2. *Let* $\mathtt{L}_1$ *and* $\mathtt{L}_2$ *be two languages.* $\mathtt{L}_1$ *is* equally or more succinct than $\mathtt{L}_2$ *(or* $\mathtt{L}_1$ *is* at least as succinct as $\mathtt{L}_2$*), denoted by* $\mathtt{L}_1 \preceq \mathtt{L}_2$ *iff for every*

$\varphi_2 \in \mathtt{L}_2$, there is a $\varphi_1 \in \mathtt{L}_1$ such that $\varphi_1 \equiv \varphi_2$ and $|\varphi_1|$,the size of φ_1, is polynomial in $|\varphi_2|$, the size of φ_2.

The relation $\preceq$ is clearly *reflexive*, *anti-symmetric*, and *transitive*, i.e. it defines a *partial order* over all possible subsets of MDAG. Two languages $\mathtt{L}_1$ and $\mathtt{L}_2$ are called *equally succinct*, denoted by $\mathtt{L}_1 \equiv \mathtt{L}_2$, iff $\mathtt{L}_1 \preceq \mathtt{L}_2$ and $\mathtt{L}_2 \preceq \mathtt{L}_1$. The language $\mathtt{L}_1$ is called *strictly more succinct* than $\mathtt{L}_2$, denoted by $\mathtt{L}_1 \prec \mathtt{L}_2$, iff $\mathtt{L}_1 \preceq \mathtt{L}_2$ and $\mathtt{L}_2 \not\preceq \mathtt{L}_1$. They are *incomparable*, iff $\mathtt{L}_1 \not\preceq \mathtt{L}_2$ and $\mathtt{L}_2 \not\preceq \mathtt{L}_1$.

To generalize the succinctness results of PDAGs to MDAGs, let $\mathtt{L}_1^{\mathtt{P}}$, $\mathtt{L}_2^{\mathtt{P}}$ be two different PDAG sub-languages and let $\mathtt{L}_1^{\mathtt{M}}$, $\mathtt{L}_2^{\mathtt{M}}$ be their corresponding MDAG sub-languages (see Table 1). The following proposition is direct consequence of Proposition 1.

Proposition 2. $\mathtt{L}_1^{\mathtt{P}} \not\preceq \mathtt{L}_2^{\mathtt{P}} \Rightarrow \mathtt{L}_1^{\mathtt{M}} \not\preceq \mathtt{L}_2^{\mathtt{M}} \qquad (\equiv \mathtt{L}_1^{\mathtt{M}} \preceq \mathtt{L}_2^{\mathtt{M}} \Rightarrow \mathtt{L}_1^{\mathtt{P}} \preceq \mathtt{L}_2^{\mathtt{P}})$.

Proving the converse, i.e. $\mathtt{L}_1^{\mathtt{P}} \preceq \mathtt{L}_2^{\mathtt{P}} \Rightarrow \mathtt{L}_1^{\mathtt{M}} \preceq \mathtt{L}_2^{\mathtt{M}}$, is more difficult. Although this proof is missing in general, most of the results can be transfered, since only two methods are used to proof $\mathtt{L}_1^{\mathtt{P}} \preceq \mathtt{L}_2^{\mathtt{P}}$:

- Sub-language relationships: if $\mathtt{L}_2^{\mathtt{P}}$ is a sub-language of $\mathtt{L}_1^{\mathtt{P}}$, then $\mathtt{L}_1^{\mathtt{P}} \preceq \mathtt{L}_2^{\mathtt{P}}$ holds trivially. The corresponding languages $\mathtt{L}_1^{\mathtt{M}}, \mathtt{L}_2^{\mathtt{M}}$ have of course the same sub-language relationship. Thus, $\mathtt{L}_1^{\mathtt{M}} \preceq \mathtt{L}_2^{\mathtt{M}}$ holds.
- Providing an algorithm that obtains $\varphi_1^P \in \mathtt{L}_1^{\mathtt{P}}$ from $\varphi_2^P \in \mathtt{L}_2^{\mathtt{P}}$ while meeting the size restriction. Taking a closer look at these algorithms reveals that they can be adapted to multi-state variables, i.e. $\varphi_1^M \in \mathtt{L}_1^{\mathtt{M}}$ is obtainable from $\varphi_2^M \in \mathtt{L}_2^{\mathtt{M}}$ while meeting the size restriction. Thus, $\mathtt{L}_1^{\mathtt{M}} \preceq \mathtt{L}_2^{\mathtt{M}}$ holds.

In this sense the succinctness relation between $\mathtt{L}_1^{\mathtt{M}}$, $\mathtt{L}_2^{\mathtt{M}}$ matches the succinctness relation between $\mathtt{L}_1^{\mathtt{P}}$, $\mathtt{L}_2^{\mathtt{P}}$ as given in [5,6].

3.2 Queries

A *query* is an operation that returns information about a MDAG representing a finite CIF without changing it. Among the important queries for finite CIFs are: *consistency* (CO) or *satisfiability* (SAT), *validity* (VA), *clause entailment* (CE), *term implication* (IM), *sentential entailment* (SE), *equivalence* (EQ), *model counting* (CT), *model enumeration* (ME), *counter-model enumeration* ($\mathtt{ME}^{\mathtt{C}}$), *probabilistic equivalence* (PEQ), and *probability computation* (PR).

If a language supports a query in polynomial time with respect to the size of the PDAG(s)/MDAG(s) (in the case of model or counter-model enumeration, the reference size is both the size of the PDAG/MDAG and size of the satisfying set or its compliment), we simply say that it *supports* this query. In the following, let $\mathtt{L}^{\mathtt{M}}$ be a MDAG sub-language, $\mathtt{L}^{\mathtt{P}}$ be the corresponding PDAG sub-language, and Q be a query. A direct consequence of Proposition 1 is: If Q is supported by $\mathtt{L}^{\mathtt{M}}$, then Q is supported by $\mathtt{L}^{\mathtt{P}}$. Or equivalently:

Proposition 3. *If* Q *is not supported by* $\mathtt{L}^{\mathtt{P}}$*, then* Q *is not supported by* $\mathtt{L}^{\mathtt{M}}$.

Unfortunately, the converse, i.e. ($\mathtt{L}^{\mathtt{P}}$ supports $\mathtt{Q}$) $\Rightarrow$ ($\mathtt{L}^{\mathtt{M}}$ supports $\mathtt{Q}$), is not proofed in general. However, it is easy to proof it for the languages given in Table 1. If $\mathtt{Q}$ is supported by a language $\mathtt{L}$, it is also supported by the sub-languages of $\mathtt{L}$, i.e. it is enough to consider the algorithms of the super-languages. Furthermore, it is sufficient to consider $\mathtt{Q} \in \{\mathtt{CO}, \mathtt{IM}, \mathtt{CT}, \mathtt{EQ}, \mathtt{SE}\}$ due to the correlations between the queries, see [6] for details.

In this sense the supported queries of $\mathtt{L}^{\mathtt{M}}$ matches the supported queries of $\mathtt{L}^{\mathtt{P}}$ as given in [5,6].

3.3 Transformations

A *transformation* is an operation that returns a MDAG representing a modified finite CIF. The new MDAG is supposed to satisfy the same properties as the language in use. Let's consider the following transformations: *term conditioning* (TC), *forgetting* (FO), *singleton forgetting* (SFO), *conjunction* (AND), *binary conjunction* ($\mathtt{AND}_2$), *disjunction* (OR), *binary disjunction* ($\mathtt{OR}_2$), and *negation* (NOT).

Note that conditioning, denoted by $\varphi|[X{=}x_i]$, includes the implicit *exclusive or*. This means the leaf labeled with $X{=}x_i$ is replaced by the leaf labeled with $\top$ and, in addition, leaves labeled with $X{=}x_j$, $j \neq i$, are replaced by the leaf labeled with $\bot$.

If a language supports a transformation in polynomial time with respect to the size of the PDAG(s)/MDAG(s), we simply say that it *supports* this transformation. In the following, let $\mathtt{L}^{\mathtt{M}}$ be a MDAG sub-language, $\mathtt{L}^{\mathtt{P}}$ be the corresponding PDAG sub-language, and T be a transformation. Another direct consequence of Proposition 1 is: If T is supported by $\mathtt{L}^{\mathtt{M}}$, then T is supported by $\mathtt{L}^{\mathtt{P}}$. This is equivalent to:

Proposition 4. *If* T *is not supported by* $\mathtt{L}^{\mathtt{P}}$*, then* T *is not supported by* $\mathtt{L}^{\mathtt{M}}$*.*

Once more, proving the converse, i.e. ($\mathtt{L}^{\mathtt{P}}$ supports T) $\Rightarrow$ ($\mathtt{L}^{\mathtt{M}}$ supports T), is an open task. Nevertheless, the results can be generalized, since the proof for $\mathtt{L}^{\mathtt{P}}$ supporting T can be adapted to $\mathtt{L}^{\mathtt{M}}$. Therefore, we have to consider the proof of Proposition 5.1 in [5], but only the parts were a language $\mathtt{L}^{\mathtt{P}}$ supports a transformation T. These proofs can be extended to hold also for $\mathtt{L}^{\mathtt{M}}$. In this sense the set of transformations supported by $\mathtt{L}^{\mathtt{M}}$ matches the set of transformations supported by $\mathtt{L}^{\mathtt{P}}$ as given in [5,6].

4 Conclusion

By allowing multi-state variables, this paper extends the family of graph-based languages for representing BFs to the corresponding family of graph-based languages for representing finite CIFs. Our main result is the observation, that properties w.r.t. succinctness, supported queries, and supported transformation are inherited, i.e. the mostly entire knowledge compilation map is extensible from propositional to multi-state variables. This allows us to avoid the usual approach of transforming the given CIF into a BF and the resulting linear growth.

Acknowledgments

This research supported by the *Swiss National Science Foundation*, Project No. PP002-102652/1, and *The Leverhulme Trust*.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers **27**(6) (1978) 509—516
2. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
3. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys **24**(3) (1992) 293–318
4. Darwiche, A.: Decomposable negation normal form. Journal of the ACM **48**(4) (2001) 608–647
5. Darwiche, A., Marquis, P.: A knowlege compilation map. Journal of Artificial Intelligence Research **17** (2002) 229–264
6. Wachter, M., Haenni, R.: Propositional DAGs: a new graph-based language for representing Boolean functions. In Doherty, P., Mylopoulos, J., Welty, C., eds.: KR'06, 10th International Conference on Principles of Knowledge Representation and Reasoning, Lake District, U.K., AAAI Press (2006) 277–285
7. Hill, F.J., Peterson, G.R.: Introduction to Switching Theory and Logical Design. John Wiley and Sons, New York, USA (1974)
8. Lukasiewicz, J., Tarski, A.: Untersuchungen über den Aussgenkalkül. Comptes rendus des séances de la Société des Sciences et des Lettres de Varsovie Cl. III **23** (1930) 30–50
9. Rosser, J.B., Turquette, A.R.: Many-Valued Logics. North-Holland (1952)
10. Wegener, I.: Branching Programs and Binary Decision Diagrams – Theory and Applications. Number 56 in Monographs on Discrete Mathematics and Applications. SIAM (2000)
11. Chavira, M., Darwiche, A.: Compiling Bayesian networks with local structure. In: IJCAI'05, 19th International Joint Conference on Artificial Intelligence, Edinburgh, U.K. (2005)
12. Palacios, H., Bonet, B., Darwiche, A., Geffner, H.: Pruning conformant plans by counting models on compiled d-DNNF representations. In: ICAPS'05, 15th International Conference on Planning and Scheduling, Monterey, USA (2005) 141–150
13. Sang, T., Beame, P., Kautz, H.: Solving Bayesian networks by weighted model counting. In: AAAI'05, 20th National Conference on Artificial Intelligence. Volume 1., Pittsburgh, USA (2005) 475–482
14. Bollig, B., Sauerhoff, M., Sieling, D., Wegener, I.: Binary decision diagrams. In Crama, Y., Hammer, P., eds.: Boolean Functions. Volume II. (2006 (to appear))