ELSEVIER

Artificial Intelligence 97 (1997) 45–82

Artificial Intelligence

A logical notion of conditional independence: properties and applications

Adnan Darwiche[1]

Department of Mathematics, American University of Beirut, P.O. Box 11-236, Beirut, Lebanon

Received October 1995; revised April 1996

Abstract

We propose a notion of conditional independence with respect to propositional logic and study some of its key properties. We present several equivalent formulations of the proposed notion, each oriented towards a specific application of logical reasoning such as abduction and diagnosis. We suggest a framework for utilizing logical independence computationally by structuring a propositional logic database around a directed acyclic graph. This structuring explicates many of the independences satisfied by the underlying database. Based on these structural independences, we develop an algorithm for a class of structured databases that is not necessarily Horn. The algorithm is linear in the size of a database structure and can be used for deciding entailment, computing abductions and diagnoses. The presented results are motivated by similar results in the literature on probabilistic and constraint-based reasoning. © 1997 Elsevier Science B.V.

Keywords: Independence; Structure-based reasoning; Graphoids; Causal networks; Pruning knowledge bases; Relevance; Logic; Probability

1. Introduction

A major factor in slowing down logical computations is that reasoners tend to consider irrelevant parts of a given database when computing answers to queries. This has prompted a considerable amount of research on the notion of *irrelevance*, with the goal of identifying these irrelevant parts of a database so they can be avoided by logical reasoners [13,15,23,24].

Irrelevance has also been the subject of extensive research in probabilistic reasoning, where it is typically referred to as *independence* [19]. The scope of independence

[1] Email: darwiche@aub.edu.lb.

in probability, however, seems to be bigger than its scope in logic for at least two reasons. First, although there is a standard definition of independence in probability, there seems to be no agreement on a definition of irrelevance in logic. This also contributes to the lack of a comprehensive theory of logical irrelevance similar to the one for probabilistic independence. Second, the computational utility that irrelevance brings to logical reasoning is perceived as a *luxury* instead of a *necessity* since influential algorithms for logical reasoning have not been based on irrelevance. This is contrary to what one finds in probabilistic reasoning where independence is the building block of almost all state-of-the-art algorithms.

Motivated by the role of independence in probability, our goal in this paper is to show that independence can play a similar role in logical reasoning, at least in the context of propositional logic. Therefore, our definition of conditional independence with respect to propositional databases resembles the definition of conditional independence with respect to probability distributions. We use *L*ogical *C*onditional *I*ndependence, LCI, to refer to the proposed definition.

Our contribution here is not LCI per se, but rather

(a) the framework we propose for exploiting it computationally and
(b) the various formulations of LCI that we provide with respect to different reasoning tasks.

We show that LCI introduces to propositional reasoning many of the tools and techniques that conditional independence has introduced to probabilistic reasoning. This includes a paradigm for logical reasoning in which the amount of independence information decides the computational complexity of reasoning, just as independence controls the complexity of probabilistic reasoning. The various formulations of LCI that we present make it clear how independence can be computationally valuable to the corresponding reasoning tasks. We utilize these formulations by developing an algorithm that can be used for deciding entailment, computing abductions and diagnoses. The algorithm provides a good example of how to use independence information when deriving algorithms for logical reasoning.

In the following section, we provide a more extended introduction to the proposed notion of independence where we explain the choices we had to make in developing it. We also outline the structure of the paper in light of the results to be presented.

2. Key choices for independence

When formulating a notion of independence, one faces a number of choice points. We devote this section to enumerating some of these points and to presenting our position on them. This helps in relating our approach to the spectrum of other existing approaches. It also provides a good opportunity for outlining the structure of the paper.

What objects can appear in an independence relation? One finds proposals in which sentences, atomic propositions, predicates, even algorithms appear as part of an independence relation [13]. In our proposal, independence is a relation between four objects: a propositional database Δ and three sets of atomic propositions X, Y and Z. Specifically, LCI decides whether the database Δ finds X independent of Y given Z. When the independence holds, we write $Ind_{\Delta}(X, Z, Y)$ and refer to it as an LCI assertion. This is similar to probabilistic independence which tells us whether a probability distribution finds X independent of Y given Z.

What decides the correctness of a definition of independence? In considering the literature on independence (both probabilistic and logical), one finds two main positions:

(1) A *philosophical* position that starts with postulating some intuitive properties of independence and then proposes a definition that adheres to these properties. In most of these approaches, independence is formulated in terms of *belief change*, that is, formalizing the irrelevance of certain information to certain beliefs [11, 19]. The probabilistic notion of independence can clearly be motivated on these grounds where belief in a proposition corresponds to its probability.

(2) A *pragmatic* position, where independence is not an absolute notion but rather a task-specific one. That is, there is no correct or incorrect definition of independence but rather a useful or not-very-useful one. For example, in deciding an entailment test $\Delta \models \alpha$, a definition of independence may target the identification of sentences in Δ that if removed from Δ will not affect the test result.

LCI, at least as developed in this paper, is meant to be a pragmatic notion. In fact, we provide different and equivalent formulations of LCI, each explicating its usefulness to a specific reasoning task. The tasks we cover are: deciding entailment and satisfiability, which are dual tasks, and computing abductions and diagnoses, which are also dual tasks. This leads to a total of four formulations of LCI that are meant to explicate its computational role in these different reasoning tasks.

LCI, however, does have a formulation based on belief change, which happens to be the closest one to probabilistic independence. We therefore start in Section 3 with this formulation and then lead into the other four formulations in Section 4 (entailment and satisfiability) and Section 5 (abduction and diagnosis).

How should independence be used computationally? If one is adopting a pragmatic approach to independence, then the usage of independence will depend on the reasoning task of interest. A very popular use of independence though is in pruning a database before attempting certain computations. For example, in testing for logical entailment, the current practice is to use independence for reducing an entailment test $\Delta \models \alpha$ into a simpler test $\Delta' \models \alpha$, where Δ' is obtained by removing sentences from Δ that are irrelevant to the test.

Although LCI will support such usage, its main computational role is different. In deciding entailment, for example, LCI will be used to *decompose* a global entailment test $\Delta \models \alpha$ into a number of local tests $\Delta_1 \models \alpha_1$, $\Delta_2 \models \alpha_2, \ldots, \Delta_n \models \alpha_n$, where each Δ_i is so small that its corresponding test $\Delta_i \models \alpha_i$ can be performed in constant time.

To introduce the computational role of LCI, we point out that the following decompositions are generally *not valid* in logic:

(1) *Entailment*: Decomposing an entailment test $\Delta \models \alpha \vee \beta$ into two simpler tests $\Delta \models \alpha$ and $\Delta \models \beta$.
(2) *Satisfiability*: Decomposing a satisfiability test with respect to $\Delta \cup \{\alpha \wedge \beta\}$ into two tests with respect to the smaller databases $\Delta \cup \{\alpha\}$ and $\Delta \cup \{\beta\}$.
(3) *Abduction*: Decomposing the abductions of a finding $\alpha \vee \beta$ into the abductions of α and those of β.
(4) *Diagnosis*: Decomposing the diagnoses of an observation $\alpha \wedge \beta$ into the diagnoses of α and those of β.

We shall demonstrate, however, that each one of these decompositions becomes valid when certain independences hold between the atoms appearing in α and those appearing in β. In fact, Sections 4 and 5 provide examples of how such decompositions can be used to decompose a global computation into a number of local computations.

What is the source of independence information? Most existing approaches attempt to *discover* independence information by pre-processing a given database [15-17,23]. Although this is consistent with our utilization of independence, we advocate a different strategy that is motivated by the following result:

> If a database Δ is *graphically structured*—that is, satisfies some conditions that are dictated by a directed acyclic graph—then the topology of the structure reveals many of the independences satisfied by the database.

Therefore, instead of automatically discovering the independences satisfied by a database, we will propose *explicating* them by constructing a structured database in the first place. Such databases are defined precisely in Section 6, which also contains a key result on how to read independences from the topology of a database structure.

What is the measure of success when using independence? When using independence to prune a database, the measure of success is the degree of pruning it allows. But when using independence to decompose a global reasoning task into local tasks, the measure of success is the number of local tasks that result from the decomposition. In Section 7, we identify a class of structured databases for which we can decompose a reasoning task (entailment, abduction, diagnosis) into a number n of local tasks, where n is linear in the number of arcs and nodes of the database structure. We also discuss extensions of the algorithm to other classes of structured databases.

We start in the next three sections by providing five formulations of LCI, each oriented towards a specific reasoning task. The different formulations are then shown to be equivalent.

3. Belief change formulation

Common intuitions about independence suggest that it is strongly related to the notion of belief change. Therefore, many expect a formal definition of independence to be based on such a notion.

Although we shall present a few formulations of LCI in this paper, it is for these intuitions that we start with a formulation in terms of belief change. This formulation also happens to be the one most resembling probabilistic independence.

Before we present this first formulation, we need to define the notion of information.

Definition 1. *Information* about a set of atomic propositions X is a propositional sentence constructed from these propositions. We use $\tilde{X}$ to denote information about propositions X.

For example, if X contained the atoms p and q, then $p \vee q$, $\neg p$ and $p \supset q$ are three pieces of information about X.

Definition 2. *Full information* about a set of atomic propositions X is a conjunction of literals, one for each atomic proposition in X. We use $\widehat{X}$ to denote full information about propositions X.

For example, there are four pieces of full information about atoms p and q: $p \wedge q$, $p \wedge \neg q$, $\neg p \wedge q$ and $\neg p \wedge \neg q$. We will also use the term *conjunctive clause* over X to mean full information about X.

Intuitively, the first formulation of LCI says that atoms X are independent of Y given Z with respect to database Δ if obtaining information about Y is irrelevant to entailing more information about X given that we have already obtained full information about Z:

Definition 3. Let X, Y, and Z be disjoint sets of atomic propositions. Database Δ finds X independent of Y given Z, written $Ind^{b}_{\Delta}(X, Z, Y)$, iff

$$\Delta \cup \{\widehat{Z}\} \models \tilde{X} \quad \text{precisely when} \quad \Delta \cup \{\widehat{Z}, \tilde{Y}\} \models \tilde{X}$$

for all $\tilde{X}, \tilde{Y}, \widehat{Z}$ such that $\Delta \cup \{\widehat{Z}, \tilde{Y}\}$ is consistent. If Z is empty, we simply say that X is independent of Y.

The database $\Delta \cup \{\widehat{Z}\}$ results from adding the full information $\widehat{Z}$ to Δ, and the database $\Delta \cup \{\widehat{Z}, \tilde{Y}\}$ results from adding the extra information $\tilde{Y}$. Definition 3 says that X is independent of Y given Z if both of these databases entail the same information about X, that is, the information about Y is irrelevant.

Examples

To further illustrate Definition 3, consider the following database,

$$\Delta = \{\mathsf{it\_rained} \vee \mathsf{sprinkler\_was\_on} \equiv \mathsf{wet\_ground}\}.$$

Let $X = \{\mathsf{sprinkler\_was\_on}\}$, $Y = \{\mathsf{it\_rained}\}$ and $Z = \emptyset$. Then $Ind^{b}_{\Delta}(X, Z, Y)$ because

$$\begin{aligned} \Delta &\not\models \mathsf{sprinkler\_was\_on} \\ \Delta \cup \{\mathsf{it\_rained}\} &\not\models \mathsf{sprinkler\_was\_on} \\ \Delta \cup \{\neg\mathsf{it\_rained}\} &\not\models \mathsf{sprinkler\_was\_on}, \end{aligned}$$

and, moreover,

$$\Delta \not\models \neg\mathsf{sprinkler\_was\_on}$$
$$\Delta \cup \{\mathsf{it\_rained}\} \not\models \neg\mathsf{sprinkler\_was\_on}$$
$$\Delta \cup \{\neg\mathsf{it\_rained}\} \not\models \neg\mathsf{sprinkler\_was\_on}.$$

That is, adding information about it_rained to the database does not change the belief in any information about sprinkler_was_on. We say in this case that database Δ finds sprinkler_was_on independent of it_rained.

However, if we let $Z = \{\mathsf{wet\_ground}\}$, then we no longer have $Ind^{b}_{\Delta}(X, Z, Y)$ because

$$\Delta \cup \{\mathsf{wet\_ground}\} \not\models \mathsf{sprinkler\_was\_on}$$
$$\Delta \cup \{\mathsf{wet\_ground}, \neg\mathsf{it\_rained}\} \models \mathsf{sprinkler\_was\_on}.$$

That is, in the presence of complete information about wet_ground in the database, adding information about it_rained does change the belief in some information about sprinkler_was_on. We say in this case that the database Δ finds sprinkler_was_on dependent on it_rained given wet_ground.

Consider now the database,

$$\Delta = \{\mathsf{it\_rained} \supset \mathsf{wet\_ground}, \mathsf{wet\_ground} \supset \mathsf{wet\_shoes}\}.$$

Let $X = \{\mathsf{wet\_shoes}\}$, $Y = \{\mathsf{it\_rained}\}$ and $Z = \emptyset$. Then we do not have $Ind^{b}_{\Delta}(X, Z, Y)$ because

$$\Delta \not\models \mathsf{wet\_shoes}$$
$$\Delta \cup \{\mathsf{it\_rained}\} \models \mathsf{wet\_shoes}.$$

That is, adding information about it_rained to the database changes the belief in some information about wet_shoes. We say in this case that database Δ finds wet_shoes dependent on it_rained.

However, if we let $Z = \{\mathsf{wet\_ground}\}$, then we have $Ind^{b}_{\Delta}(X, Z, Y)$ because, in the presence of complete information about wet_ground in the database, adding information about it_rained does not change the belief in any information about wet_shoes—we leave this for the reader to verify. We say in this case that the database Δ finds wet_shoes independent of it_rained given wet_ground.

Properties of LCI

Definition 3 of LCI satisfies the following properties.

Theorem 4. *Let X, Y, Z and L be disjoint sets of atomic propositions and let Δ be a propositional database. Then*

(1) $Ind^{b}_{\Delta}(X, Z, \emptyset)$,

(2) $Ind^{b}_{\Delta}(X, Z, Y)$ *precisely when* $Ind^{b}_{\Delta}(Y, Z, X)$,

(3) $Ind^{b}_{\Delta}(X, Z, Y)$ *and* $Ind^{b}_{\Delta}(L, Z \cup X, Y)$ *precisely when* $Ind^{b}_{\Delta}(X \cup L, Z, Y)$.

Properties (2) and (3) are known as the semi-graphoid axioms [19]. Property (1) was added recently to them under the name *trivial independence* [26]. Property (2) is known as symmetry. Property (3) is typically broken into three other properties:

- *Decomposition*: $\mathit{Ind}^{b}_{\Delta}(X \cup L, Z, Y)$ only if $\mathit{Ind}^{b}_{\Delta}(X, Z, Y)$,
- *Weak union*: $\mathit{Ind}^{b}_{\Delta}(X \cup L, Z, Y)$ only if $\mathit{Ind}^{b}_{\Delta}(L, Z \cup X, Y)$ and
- *Contraction*: $\mathit{Ind}^{b}_{\Delta}(X, Z, Y)$ and $\mathit{Ind}^{b}_{\Delta}(L, Z \cup X, Y)$ only if $\mathit{Ind}^{b}_{\Delta}(X \cup L, Z, Y)$.

The semi-graphoid axioms are important for at least two reasons. First, they are intuitive properties that independence is expected to satisfy [19]. Second, they lead to an important result about the identification of LCI assertions by examining the topology of a structured database, which is the subject of Section 6.

Structured databases

We now introduce structured databases, which will be discussed in more detail in Section 6.

Figs. 1, 2 and 4 depict a number of structured databases. In general, a structured database has two parts:

(1) a directed acyclic graph over atomic propositions, and

(2) a number of local databases, one for each atom n in the graph.

The local database for atom n can only refer to n, its parents, and atoms that do not appear in the directed graph. The local database is typically depicted next to the atom as shown in Figs. 1, 2 and 4. If a database is structured, some of the LCI assertions it satisfies can be detected by visual inspection of the database structure, using a method that we shall describe in Section 6. For example, using this method, one can detect the following LCI assertions regarding the databases in Fig. 1: [2]

- $\mathit{Ind}^{b}_{\Delta}(\{A\}, \emptyset, \{B\})$ in Fig. 1.1.
- $\mathit{Ind}^{b}_{\Delta}(\{A\}, \{B\}, \{C\})$ in Fig. 1.2.
- $\mathit{Ind}^{b}_{\Delta}(\{B\}, \{A\}, \{C\})$ in Fig. 1.3.

We provide the formal definition of structured databases together with some of their key properties in Section 6.

The next two sections provide four other formulations of LCI, corresponding to its use in testing logical entailment and satisfiability, and in computing abductions and diagnoses.

4. Entailment and satisfiability formulations

This section discusses the computational role of LCI in deciding logical entailment and satisfiability. For this purpose, we present two formulations of LCI, each oriented towards one of these dual tasks.

[2] These assertions are meant to review what will be covered in Section 6. The reader is not expected to understand how these assertions are detected at this stage.

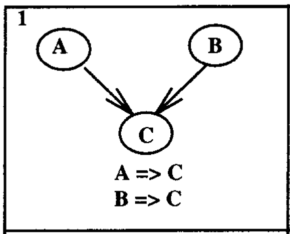	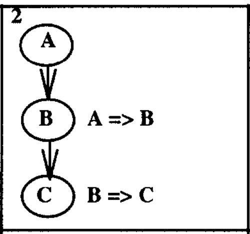	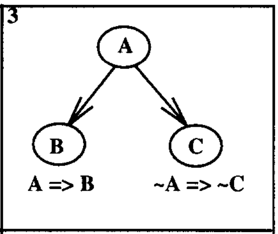
A = rain **B = sprinkler was on** **C= wet grass**	**A = rain** **B = wet ground** **C = slippery ground**	**A = battery ok** **B = lights on** **C = car starts**

Fig. 1. Three structured databases. Each graph identifies LCI assertions that are satisfied by its associated database.

We start with the formulation of LCI that is oriented towards testing logical entailment, that is, deciding whether some propositional sentence α is logically entailed by a database Δ. We first present the formulation and then show how it can be applied to logical entailment when the database is structured.

The computational difficulty in testing for logical entailment (at least in the propositional case) stems from the inability to do *decompositional* testing. That is, although one can decompose the test $\Delta \models \alpha \wedge \beta$ into testing whether $\Delta \models \alpha$ *and* $\Delta \models \beta$, one cannot (in general) decompose the test $\Delta \models \alpha \vee \beta$ into testing whether $\Delta \models \alpha$ *or* $\Delta \models \beta$. To see this, note that if $\Delta = \{p \vee q\}$, then $\Delta \models p \vee q$ holds while neither $\Delta \models p$ nor $\Delta \models q$ holds. Therefore, the decomposition

$$\Delta \models p \vee q \quad \text{precisely when} \quad \Delta \models p \text{ or } \Delta \models q$$

is not valid in this case.

If such a decomposition were valid, however, testing for entailment would be very easy. We would always be able to rewrite the test $\{\alpha_1, \ldots, \alpha_n\} \models \alpha$ into the equivalent test $\neg\alpha \models \neg\alpha_1 \vee \cdots \vee \neg\alpha_n$ and then decompose the latter into n tests $\neg\alpha \models \neg\alpha_1, \ldots,$ $\neg\alpha \models \neg\alpha_n$, which in turn are equivalent to $\alpha_1 \models \alpha, \ldots, \alpha_n \models \alpha$. This would make testing for logical entailment linear in the number n of sentences in a database.

Although the decomposition of a disjunctive test $\Delta \models \alpha \vee \beta$ is not valid in general, it can be valid when certain LCI assertions are satisfied by the database Δ. The second formulation of LCI is meant to explicate these assertions.

Entailment formulation

We first need the following supporting definition.

Definition 5. A *disjunctive clause* over a set of atomic propositions X is a disjunction of literals, one literal for each proposition in X. We use $\check{X}$ to denote a disjunctive clause over propositions X.

For example, there are four disjunctive clauses over p and q: $p \vee q$, $p \vee \neg q$, $\neg p \vee q$ and $\neg p \vee \neg q$.

The following formulation of LCI is in terms of decomposing disjunctive tests. Theorem 7 proves the equivalence between this formulation and the first one based on belief change.

Definition 6. Let Δ be a propositional database and let X, Y and Z be three disjoint sets of atomic propositions. Database Δ finds X independent of Y given Z, written $Ind^{\mathrm{c}}_{\Delta}(X, Z, Y)$, iff

$$\Delta \models \check{X} \vee \check{Y} \vee \check{Z} \quad \text{precisely when} \quad \Delta \models \check{X} \vee \check{Z} \text{ or } \Delta \models \check{Y} \vee \check{Z}$$

for all disjunctive clauses $\check{X}, \check{Y}, \check{Z}$.

Theorem 7. *$Ind^{\mathrm{b}}_{\Delta}(X, Z, Y)$ precisely when $Ind^{\mathrm{c}}_{\Delta}(X, Z, Y)$.*

Theorem 7 is then showing the equivalence between

(1) the validity of decomposing certain disjunctive tests, which is a computation-oriented formulation of LCI (Definition 6) and
(2) the irrelevance of certain information to certain beliefs, which is an intuition-oriented formulation of LCI (Definition 3).

An important special case of Definition 6 is when $Z = \emptyset$. Here, *false* is the only disjunctive clause over Z, and we have $Ind^{\mathrm{c}}_{\Delta}(X, \emptyset, Y)$ iff

$$\Delta \models \check{X} \vee \check{Y} \quad \text{precisely when} \quad \Delta \models \check{X} \text{ or } \Delta \models \check{Y}$$

for all disjunctive clauses $\check{X}$ and $\check{Y}$.

Example

Consider the database $\Delta = \{p \supset r, r \supset s\}$. This database satisfies the LCI assertion $Ind^{\mathrm{b}}_{\Delta}(\{s\}, \{r\}, \{p\})$. Therefore, the following decompositions are valid:

$$\begin{array}{rcl}
\Delta \models p \vee r \vee s & \text{precisely when} & \Delta \models p \vee r \text{ or } \Delta \models r \vee s \\
\Delta \models \neg p \vee r \vee s & \text{precisely when} & \Delta \models \neg p \vee r \text{ or } \Delta \models r \vee s \\
\Delta \models p \vee \neg r \vee s & \text{precisely when} & \Delta \models p \vee \neg r \text{ or } \Delta \models \neg r \vee s \\
\vdots & & \\
\Delta \models \neg p \vee \neg r \vee \neg s & \text{precisely when} & \Delta \models \neg p \vee \neg r \text{ or } \Delta \models \neg r \vee \neg s.
\end{array}$$

Note, however, that Δ does not satisfy the assertion $Ind^{\mathrm{b}}_{\Delta}(\{s\}, \emptyset, \{p\})$. Therefore, it should not be surprising that the decomposition

$$\Delta \models \neg p \vee s \quad \text{precisely when} \quad \Delta \models \neg p \text{ or } \Delta \models s$$

does not hold ($\Delta \models \neg p \vee s$ holds but neither $\Delta \models \neg p$ nor $\Delta \models s$ does).

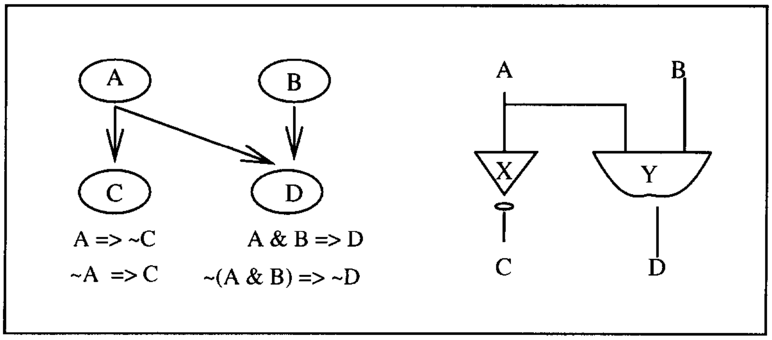

Fig. 2. A structured database representing a digital circuit.

The computational value of independence

We have shown so far that:

(1) disjunctive tests of the form $\Delta \models \alpha \vee \beta$ cannot be decomposed in general;

(2) if disjunctive tests were always decomposable, then testing for logical entailment would become linear in the number of sentences in a database;

(3) disjunctive tests could be decomposed when certain LCI assertions are satisfied by the database.

When all disjunctive tests are decomposable, taking advantage of this decomposability is easy as we have shown. But when only certain tests are decomposable—that is, limited independence is available—then the issue is no longer that simple.

The algorithm we present in Section 7 is a good example of how limited independence information can be utilized computationally. It decomposes a global test $\Delta \models \alpha$ into a number of local tests $\Delta_i \models \alpha_i$, which are assumed to require constant time. The decomposition is accomplished using two rewrite rules:

(1) *case-analysis*: $\Delta \models \alpha$ is rewritten into $\Delta \models \alpha \vee \beta$ and $\Delta \models \alpha \vee \neg\beta$ and

(2) *decomposition*: $\Delta \models \alpha \vee \beta \vee \gamma$ is rewritten into $\Delta \models \alpha \vee \gamma$ or $\Delta \models \beta \vee \gamma$.

The case-analysis rule is always valid because

(1) α is equivalent to $(\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$ and

(2) $\Delta \models (\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$ iff $\Delta \models \alpha \vee \beta$ and $\Delta \models \alpha \vee \neg\beta$.

However, the decomposition rule is valid only when certain LCI assertions hold according to Definition 6.

Fig. 3 shows an example of this rewrite process as applied to the test $\Delta \models \neg C \vee \neg D$ with respect to the database in Fig. 2. Specifically, the figure depicts a trace of a rewrite process that decomposes the global test $\Delta \models \neg C \vee \neg D$ into six local tests: $\Delta_1 \models \neg C \vee A$, $\Delta_1 \models \neg C \vee \neg A$, $\Delta_2 \models \neg D \vee A \vee B$, $\Delta_2 \models \neg D \vee A \vee \neg B$, $\Delta_2 \models \neg D \vee \neg A \vee B$, and $\Delta_2 \models \neg D \vee \neg A \vee \neg B$. Here Δ_1 and Δ_2 are the local databases:

$$\Delta_1 = \{A \supset \neg C, \neg A \supset C\}$$

and

$$\Delta_2 = \{A \wedge B \supset D, \neg(A \wedge B) \supset \neg D\}.$$

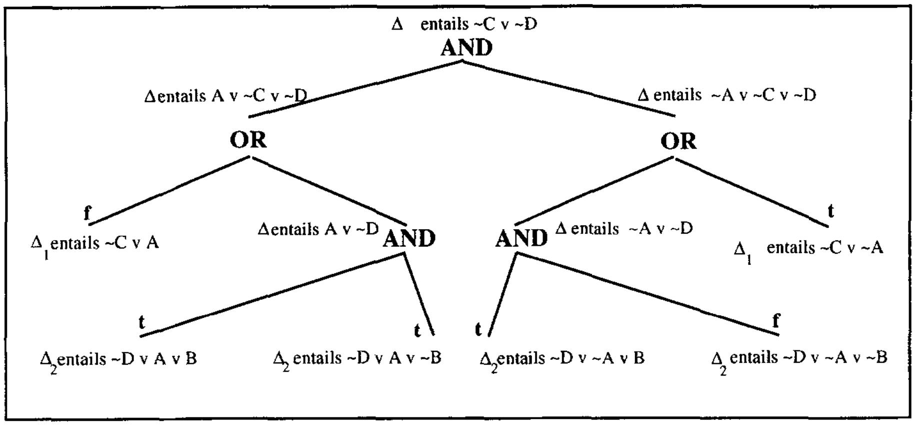

Fig. 3. A rewrite process for decomposing a global test for entailment into a number of local tests. Also shown is the evaluation of local tests.

The process proceeds top-down, rewriting each test into a logical combination of the tests below it. The case-analysis rule rewrites a test into a conjunction of other tests. The decomposition rule rewrites a test into a disjunction of other tests. Therefore, the rewrite process constructs an and-or tree, the root of which is the global test, and the leaves of which are local tests. Note here that the two applications of the decomposition rule are validated by the LCI assertion $Ind^{\mathcal{E}}_{\Delta}(\{C\},\{A\},\{D\})$, which follows from the structure in Fig. 2.

Fig. 3 also depicts the result of evaluating local tests. It is clear that this tree evaluates to *true*, which is the answer computed for the global test $\Delta \models \neg C \vee \neg D$.

We have a few points to make with respect to this example. First, it exemplifies the computational paradigm that we are advocating in this paper: decompose a global computation into a number of local, constant time computations. Second, for each of the computational tasks we shall consider, there will be a corresponding case-analysis and decomposition rules. The case-analysis rule will always be valid, but the decomposition rule will be valid only when certain LCI assertions hold. Therefore, to develop an independence-based algorithm, one must

(1) know when the decomposition rule is valid (*soundness*)
(2) control the rewrite process so that:
    (a) it terminates (*completeness*)
    (b) it invokes the least number of rewrites possible (*efficiency*)

Structured databases that we shall introduce in Section 6 are central for achieving these goals: the topology of a database tells us which decompositions are valid and can be used to control the rewrite process so that it terminates.

The number of invoked rewrites decides the complexity of reasoning and is the parameter for measuring the superiority of one algorithm versus another. As we shall see, for certain database structures, we can decompose a task using only a linear number of rewrites. These computational issues will be discussed in more detail later.

Satisfiability formulation

We now provide the third formulation of LCI, which is oriented towards testing satisfiability:

Definition 8. Let Δ be a propositional database and let X, Y and Z be three disjoint sets of atomic propositions. Database Δ finds X independent of Y given Z, written $Ind^{s}_{\Delta}(X,Z,Y)$, iff $\Delta \cup \{\widehat{Z},\widehat{X}\}$ and $\Delta \cup \{\widehat{Z},\widehat{Y}\}$ are both satisfiable precisely when $\Delta \cup \{\widehat{Z},\widehat{Y},\widehat{X}\}$ is satisfiable for all conjunctive clauses $\widehat{Z}$, $\widehat{Y}$ and $\widehat{X}$.

This formulation is dual to the second formulation for testing entailment. The equivalence is established below:

Theorem 9. *$Ind^{s}_{\Delta}(X,Z,Y)$ precisely when $Ind^{e}_{\Delta}(X,Z,Y)$.*

Now that we have established the equivalence between Ind^{b}_{Δ}, Ind^{e}_{Δ} and Ind^{s}_{Δ}, we will sometimes drop the superscripts and simply write Ind_{Δ}. The specific interpretation of Ind_{Δ} will then be chosen depending on the context.

5. Diagnosis and abduction formulations

This section presents two more formulations of LCI oriented towards the dual tasks of diagnosis and abduction. We define the notion of a *consequence* for diagnosis tasks, which is a semantical characterization of all diagnoses. We also define the notion of an *argument* for abduction tasks, which is a semantical characterization of all abductions. We show that the difficulties in computing diagnoses and abduction are rooted in the inability to decompose the computations of consequences and arguments in general. We also show that independence assertions can validate this decomposition in certain cases. We present two more formulations of LCI to spell out these cases. The two formulations are dual since consequences are dual to arguments.

We start with the fourth formulation of LCI that is oriented towards diagnostic applications. Before we present it though, we need to review some basics of diagnosis [7].

Basics of diagnosis

In the diagnostic literature [7], a *system* is typically characterized by a tuple (Δ, P, W) where Δ is a database constructed from atomic propositions $P \cup W$. Moreover, a system *observation* is typically characterized by a sentence γ constructed from atoms P. The atoms in W are called assumables and those in P are called non-assumables. The intention is that the database Δ describes the system behavior, the assumables W represent the modes of system components and the sentence γ represents the observed system behavior. For example, in Fig. 4, the assumables are okX and okY, the non-assumables are A, B, C and D and a possible system observation γ is $C \wedge D$.

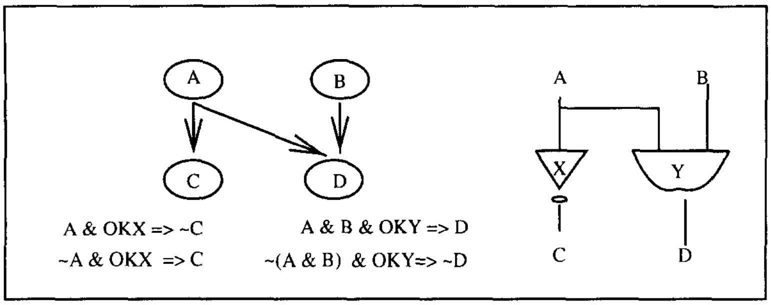

Fig. 4. A structured database representing a digital circuit.

A *diagnosis* is defined as a conjunctive clause over the set of assumables W that is consistent with $\Delta \cup \{\gamma\}$. Therefore, a diagnosis is an assignment of modes to components that is consistent with the system description and its observed behavior. In Fig. 4, *okX* and *okY* are the assumables and $okX \wedge \neg okY$ is a potential diagnosis.

One goal of diagnostic reasoning is to characterize all diagnoses compactly so they can be presented to a user. Another goal is to extract a subset of these diagnoses according to some preference criterion. We have shown elsewhere that these objectives can be achieved in two steps [4]. First, we compute the *consequence* of observation γ, which is a Boolean expression that characterizes all the diagnoses of γ. Second, we extract the most preferred diagnoses from the computed consequence.

The consequence of an observation is defined formally below:

Definition 10. Let Δ be a propositional database, α be a propositional sentence and W be a set of atomic propositions that do not appear in α. The *consequence* of sentence α with respect to database Δ and atoms W, written $Cons_W^{\Delta}(\alpha)$, is a logically strongest sentence $\tilde{W}$ such that $\Delta \cup \{\alpha\} \models \tilde{W}$.[3]

When Δ and W are clear from the context, we will write $Cons(\alpha)$ instead of $Cons_W^{\Delta}(\alpha)$ for simplicity. In Fig. 4, for example, the consequence of observation $C \wedge D$ is $\neg okX \vee \neg okY$ because it is a logically strongest sentence (constructed from assumables) that can be concluded from the given observation and system description.

A consequence characterizes all diagnoses in the following way:

Theorem 11. *$\widehat{W}$ is a diagnosis for system (Δ, P, W) and observation γ iff $\widehat{W} \models Cons_W^{\Delta}(\gamma)$.*

The consequence $\neg okX \vee \neg okY$ for example characterizes three diagnoses: $\neg okX \wedge \neg okY$, $okX \wedge \neg okY$ and $\neg okX \wedge okY$.

[3] A sentence α is stronger than sentence β iff $\alpha \models \beta$. The sentence β is said to be weaker than α in such a case.

Diagnosis formulation

Similar to testing for logical entailment, the difficulty with computing diagnoses is that it cannot be done compositionally. In particular, although $Cons(\alpha \vee \beta)$ is equivalent to $Cons(\alpha) \vee Cons(\beta)$, $Cons(\alpha \wedge \beta)$ is not equivalent to $Cons(\alpha) \wedge Cons(\beta)$ in general.

The following formulation of LCI is in terms of decomposing consequences. It is followed by a theorem that shows the equivalence between this formulation and the one based on belief change, therefore, identifying conditions under which decomposing consequences is valid.

Definition 12. Let Δ be a database and let X, Y, Z and W be disjoint sets of atomic propositions. The pair (Δ, W) finds X independent of Y given Z, written $Ind^{c}_{(\Delta,W)}(X, Z, Y)$, iff

$$\models Cons(\widehat{X} \wedge \widehat{Y} \wedge \widehat{Z}) \equiv Cons(\widehat{X} \wedge \widehat{Z}) \wedge Cons(\widehat{Y} \wedge \widehat{Z})$$

for all conjunctive clauses $\widehat{X}$, $\widehat{Y}$ and $\widehat{Z}$.

Theorem 13. *$Ind^{c}_{(\Delta,W)}(X, Z, Y)$ precisely when $Ind^{b}_{\Delta}(X, Z \cup W, Y)$.*

Given Theorem 13, we now have an equivalence between
(1) the irrelevance of information to beliefs (Definition 3),
(2) the validity of decomposing entailment tests (Definition 6),
(3) the validity of decomposing satisfiability tests (Definition 8), and
(4) the validity of decomposing consequences (Definition 12).

Before we present an example on the computational use of this LCI formulation, we note the following special case of Definition 12. When Z is empty, *true* is the only conjunctive clause over Z, and we have $Ind^{c}_{(\Delta,W)}(X, \emptyset, Y)$ iff

$$\models Cons(\widehat{X} \wedge \widehat{Y}) \equiv Cons(\widehat{X}) \wedge Cons(\widehat{Y})$$

for all conjunctive clauses $\widehat{X}$ and $\widehat{Y}$.

Example

We have presented elsewhere an algorithm for computing consequences with respect to a structured database [4]. The algorithm works by rewriting a global consequence of the form $Cons^{\Delta}(\alpha)$ into a Boolean expression that involves logical connectives and local consequences of the form $Cons^{\Delta_i}(\alpha_i)$, where Δ_i is a local database. The algorithm is based on the following two rewrites:
(1) *case-analysis*: $Cons(\alpha)$ is rewritten into $Cons(\alpha \wedge \beta) \vee Cons(\alpha \wedge \neg\beta)$;[4] and
(2) *decomposition*: $Cons(\alpha \wedge \beta \wedge \gamma)$ is rewritten into $Cons(\alpha \wedge \gamma) \wedge Cons(\beta \wedge \gamma)$ when the corresponding LCI assertion holds.

[4] The case-analysis rule follows because (a) α is equivalent to $(\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta)$ and (b) $Cons((\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta))$ is equivalent to $Cons(\alpha \wedge \beta) \vee Cons(\alpha \wedge \neg\beta)$.

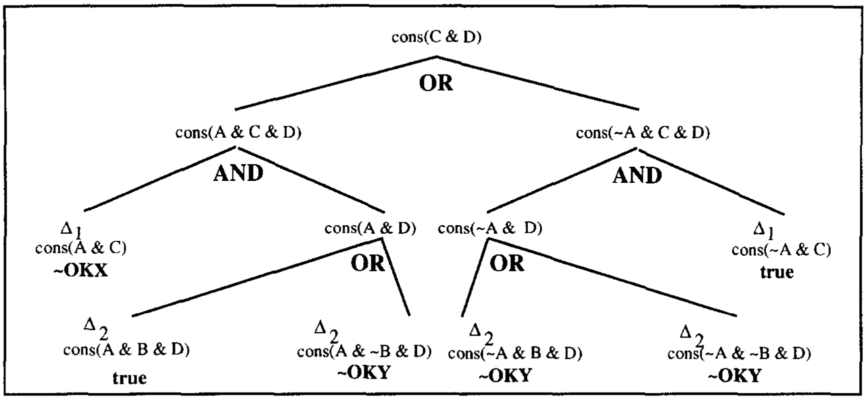

Fig. 5. A rewrite process for decomposing a global consequence into a number of local consequences. Also shown are the values of local consequences.

We will now consider an example using these rewrites with respect to Fig. 4. Our goal is to compute the consequence $Cons(C \wedge D)$, where the set W contains the atoms okX and okY.

Fig. 5 depicts a trace of the rewrite process that decomposes the global consequence $Cons(C \wedge D)$ into six local consequences: $Cons^{\Delta_1}(A \wedge C)$, $Cons^{\Delta_1}(\neg A \wedge C)$, $Cons^{\Delta_2}(D \wedge A \wedge B)$, $Cons^{\Delta_2}(D \wedge A \wedge \neg B)$, $Cons^{\Delta_2}(D \wedge \neg A \wedge B)$ and $Cons^{\Delta_2}(D \wedge \neg A \wedge \neg B)$. Here, Δ_1 and Δ_2 are the local databases:

$$\Delta_1 = \{A \wedge okX \supset \neg C, \neg A \wedge okX \supset C\}$$

and

$$\Delta_2 = \{A \wedge B \wedge okY \supset D, \neg(A \wedge B) \wedge okY \supset \neg D\}.$$

The process proceeds top-down, rewriting each consequence into the nodes below it. The case-analysis rule rewrites a consequence into a disjunction of other consequences, while the decomposition rule rewrites a consequence into a conjunction of other consequences. Therefore, the rewrite process constructs an and-or tree, the root of which is the global consequence, and the leaves of which are local consequences. Given the values of local consequences, the tree simplifies to $\neg okX \vee \neg okY$, which is the value of the global consequence $Cons(C \wedge D)$. Note that local consequences can be computed by operating on local databases, which is assumed to require constant time. Note also that the two applications of the decomposition rule are validated by the LCI assertion $Ind^{\mathcal{C}}_{(\Delta,W)}(\{C\},\{A\},\{D\})$, which can be inferred by examining the database structure.

Abduction formulation

The fifth formulation of LCI is oriented towards the computation of *abductions*.

There is no standard definition of an abduction, although existing definitions seem to agree on the following two properties:

(1) *Generality*: An abduction should be as general as possible while still explaining the finding.
(2) *Scope*: The language in which an abduction is phrased must be restricted in order to avoid trivial abductions (α being an abduction of itself).

The following definition gives a generic notion of an abduction, called an *argument*. The ATMS label of a proposition, for example, is only a syntactic variation on the argument for that proposition [6,21].

Definition 14. Let Δ be a propositional database, α be a propositional sentence, and W be a set of atomic propositions that do not appear in α. The *argument* for sentence α with respect to database Δ and atoms W, written $Arg_W^\Delta(\alpha)$, is the logically weakest sentence $\tilde{W}$ such that $\Delta \cup \{\tilde{W}\} \models \alpha$.[5]

The intention here is that Δ represents background information, α represents a finding, and W restricts the language used in phrasing an abduction. For example, if we choose W to contain okX and okY in Fig. 4, the argument for finding $\neg C \vee \neg D$ would then be $okX \wedge okY$.

Arguments are dual to consequences as the following theorem shows:

Theorem 15. $Arg_W^\Delta(\alpha) \equiv \neg Cons_W^\Delta(\neg\alpha)$.

This also explains, indirectly, why ATMS engines are typically used for computing diagnoses.

Similar to logical entailment, the difficulty in computing abductions/arguments is related to the inability to decompose arguments. That is, although $Arg(\alpha \wedge \beta)$ is equivalent to $Arg(\alpha) \wedge Arg(\beta)$, $Arg(\alpha \vee \beta)$ is not equivalent to $Arg(\alpha) \vee Arg(\beta)$ in general. For example, if $\Delta = \{p \supset (r \vee s)\}$ and $W = \{p\}$, then $Arg(r \vee s) = p$ while $Arg(r) = \mathit{false}$ and $Arg(s) = \mathit{false}$.

The following formulation of LCI is in terms of decomposing arguments. It is followed by a theorem that shows the equivalence between this formulation and previous ones, thus establishing conditions under which it is valid to decompose arguments.

Definition 16. Let Δ be a database and let X, Y, Z and W be disjoint sets of atomic propositions. The pair (Δ, W) finds X independent of Y given Z, written $Ind^a_{(\Delta,W)}(X, Z, Y)$, iff

$$\models Arg(\check{X} \vee \check{Y} \vee \check{Z}) \equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Y} \vee \check{Z}),$$

for all disjunctive clauses $\check{X}$, $\check{Y}$, $\check{Z}$.

Theorem 17. *$Ind^a_{(\Delta,W)}(X, Z, Y)$ precisely when $Ind^c_{(\Delta,W)}(X, Z, Y)$.*

[5] We will typically drop Δ and W, thus writing $Arg(\alpha)$, when no confusion is expected.

Now that we have established the equivalence between $Ind^{a}_{(\Delta,W)}$ and $Ind^{c}_{(\Delta,W)}$, we will sometimes drop the superscripts and simply write $Ind_{(\Delta,W)}$. The specific interpretation of *Ind* will then be chosen depending on the context.

We close this section with an important special case of Definition 16. When Z is empty, *false* is the only disjunctive clause over Z, and we have $Ind^{a}_{(\Delta,W)}(X,\emptyset,Y)$ iff

$$Arg(\check{X} \vee \check{Y}) \equiv Arg(\check{X}) \vee Arg(\check{Y}),$$

for all disjunctive clauses $\check{X}$ and $\check{Y}$.

6. Structured databases

The different formulations of LCI show how independence information can validate the decomposition of a computation into smaller computations that can be performed in parallel, a decomposition that is not sound in general. Although this ability to decompose a computation could be valuable from a complexity viewpoint, exploiting such decompositions is not always straightforward.

Structured databases make this utilization of independence more feasible. In particular, the structure of a database plays two important roles in designing independence-based algorithms:

(1) it graphically explicates valid applications of the decomposition rule, and
(2) it specifies a control flow that guarantees the termination of the decomposition process.

This is also the role that directed acyclic graphs have been playing in probabilistic reasoning, and our goal here is to extend the role to propositional logic.[6]

We will now provide the formal definition of structured databases, and then present two important operations on their structures:

(1) reading independences off the database structure and
(2) pruning irrelevant parts of a database structure before performing certain computations.

6.1. The syntax of a structured database

A structured database is a pair $(\mathcal{G},\Delta)$ where

- $\mathcal{G}$ is a *directed acyclic graph* with nodes representing atomic propositions, and
- Δ is the union of *local databases*, one database for each node in $\mathcal{G}$.

The atoms that appear in the database Δ but do not appear in the structure $\mathcal{G}$ are called *exogenous* atoms. For example, in Fig. 4, the exogenous atoms are *okX* and *okY*. The local database corresponding to proposition n in $\mathcal{G}$, denoted by Δ_n, must satisfy the following conditions:

(1) *locality*: the only atoms in $\mathcal{G}$ that Δ_n can mention are the family of atom n;[7] and

[6] However, we later discuss a key difference between the propositional and probabilistic framework.

[7] Recall that the family of n consists of n and its parents in the database structure.

(2) *modularity*: if Δ_n entails a disjunctive clause that does not mention atom n, then the clause must be valid.

The first condition makes sure that each local database is restricted to specifying the relationship between an atom and its parents. The second condition ensures that a local database for atom n be only concerned with specifying how the parents of n determine the truth value of n. That is, the database should not be concerned with specifying a direct relationship between the parents of n. Note also that the modularity condition ensures the consistency of a local database since the empty clause (falsehood) does not mention atom n.

If the arcs of a structured database represent causal influences, then the conditions above are typically self-imposed. For example, suppose that a structured database is used to describe the functionality of a digital circuit as shown in Figs. 2 and 4. Each atom in the structure represents the state of a wire in the circuit, and the arcs point from the inputs of a gate into its output. The local database associated with atom n specifies the behavior of the gate having output n. For this class of structured databases, the locality and modularity conditions are typically self-imposed since

(1) the locality condition means that one should not mention any wire that is not an input or an output of the gate for which we are specifying a behavior, and
(2) the modularity condition means that one should not specify a relationship between the inputs of a gate in the process of specifying its behavior.

We have focused elsewhere on a causal interpretation of structured databases, referred to as *symbolic causal networks*, where we discussed non-computational applications of LCI that include reasoning about actions and identifying anomalous extensions of nonmonotonic theories [5].

6.2. Structure-based independence

By construction, a structured database satisfies some LCI assertions that can be easily detected from the database structure:

Theorem 18. *Let $(\mathcal{G}, \Delta)$ be a structured database, W be its exogenous atoms, n be a node in $\mathcal{G}$, U be its parents, and N be its non-descendants. Then $Ind_\Delta(\{n\}, U \cup W, N)$.*

That is, each atom in the database structure is independent of its non-descendants given its parents and exogenous atoms. With respect to the database in Fig. 4, this theorem states that

(1) $\{D\}$ is independent of $\{C\}$ given $\{A, B, okX, okY\}$,
(2) $\{B\}$ is independent of $\{A\}$ given $\{okX, okY\}$, and
(3) $\{C\}$ is independent of $\{D\}$ given $\{A, okX, okY\}$.

Theorem 18 brings up a key difference between structured databases and Bayesian networks: the independences characterized by this theorem are *properties* of a structured database, but their corresponding probabilistic independences are *part of the definition* of a Bayesian network. That is, the restrictions on a structured database are sufficient to imply these independences, but similar restrictions on a Bayesian network are not enough to imply the corresponding probabilistic independences. There are two implications of this:

- By constructing a structured database, one is making no explicit commitment to independences; one is committing only to the logical content of local databases. By constructing a Bayesian network, however, one is explicitly committing to independences which correspond to Theorem 18 [19].
- One can recover the independences satisfied by a structured database from only its local databases, that is, without having to consider the database structure.[8]

Note that having an explicit structure is very useful in deducing further independences. Specifically, since LCI is a symmetric relation, the LCI assertion in (3) above implies

(4) $\{D\}$ is independent of $\{C\}$ given $\{A, okX, okY\}$,

which *cannot* be detected by applying Theorem 18 directly. As this example illustrates, a structured database satisfies more LCI assertions than those characterized by Theorem 18. We now present a topological test, called d-separation, for identifying some of these additional assertions.

d-separation is a test on directed acyclic graphs that tells us whether two sets of nodes are d-separated by a third set [19]. According to this test, for example, the nodes $\{B, D\}$ are d-separated from the node $\{C\}$ by the node $\{A\}$ in Fig. 4. Therefore, d-separation could be viewed as a relation $Sep_{\mathcal{G}}$ where $Sep_{\mathcal{G}}(X, Z, Y)$ holds precisely when Z d-separates X from Y in the directed acyclic graph $\mathcal{G}$. The d-separation test has been instrumental in reasoning about independence in Bayesian networks. We shall use it analogously in structured databases.

We will define d-separation later, but we first go over how it can be used to identify LCI assertions. The following result, which is basically a corollary of a theorem in [25], reveals the importance of d-separation:

Theorem 19. *Let $(\mathcal{G}, \Delta)$ be a structured database and let W be its exogenous atoms. If Z d-separates X from Y in $\mathcal{G}$, then Δ finds X and Y independent given $Z \cup W$. That is, $Sep_{\mathcal{G}}(X, Z, Y)$ implies $Ind_{\Delta}(X, Z \cup W, Y)$.*

Therefore, d-separation allows one to infer LCI assertions satisfied by a given structured database by simply examining the topology of its structure.

Following are the key computational implications of Theorem 19. If atoms X and Y are d-separated by Z in the structure of database $(\mathcal{G}, \Delta)$, then the following decompositions are valid with respect to database Δ:

(1) $\Delta \models \check{X} \vee \check{Z} \vee \check{W} \vee \check{Y}$ iff $\Delta \models \check{X} \vee \check{Z} \vee \check{W}$ or $\Delta \models \check{Z} \vee \check{W} \vee \check{Y}$,

(2) $\Delta \cup \{\widehat{X} \wedge \widehat{Z} \wedge \widehat{W} \wedge \widehat{Y}\}$ is satisfiable iff $\Delta \cup \{\widehat{X} \wedge \widehat{Z} \wedge \widehat{W}\}$ and $\Delta \cup \{\widehat{Z} \wedge \widehat{W} \wedge \widehat{Y}\}$ are satisfiable,

(3) $Arg(\check{X} \vee \check{Z} \vee \check{Y})$ is equivalent to $Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Z} \vee \check{Y})$, and

(4) $Cons(\widehat{X} \wedge \widehat{Z} \wedge \widehat{Y})$ is equivalent to $Cons(\widehat{X} \wedge \widehat{Z}) \vee Cons(\widehat{Z} \wedge \widehat{Y})$,

for all disjunctive clauses $\check{X}, \check{Y}, \check{Z}$ and all conjunctive clauses $\widehat{X}, \widehat{Y}, \widehat{Z}$. These implications show why the complexity of reasoning in the framework we are proposing is decided by the topology of a structured database.

The d-separation test will be defined next. We first need the following supporting definition:

[8] We do not investigate this direction in this paper however.

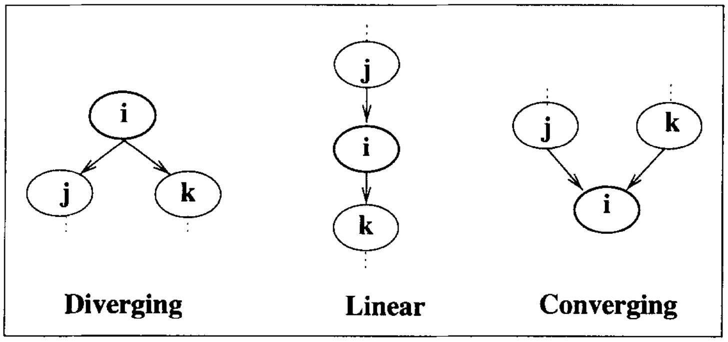

Fig. 6. There are three types of intermediate nodes on a given path. The type of a node is determined by its relation to its neighbors. A node is diverging if both neighbors are children. A node is linear if one neighbor is a parent and the other is a child. A node is converging if both neighbors are parents.

Definition 20. Let $\mathcal{G}$ be a directed acyclic graph and let X, Y, and Z be three disjoint sets of nodes in $\mathcal{G}$. A path between X and Y is Z*-active* precisely when its nodes satisfy the following conditions:

(1) each converging node belongs to Z or has a descendent in Z, and

(2) each diverging or linear node is outside Z.

When Z is empty, we say the path is *active*. Fig. 6 gives the definition of converging, diverging, and linear nodes.

In Fig. 4, for example,
- the path $A \rightarrow D \leftarrow B$ is $\{D\}$-active,
- the path $C \leftarrow A \rightarrow D$ is active, and
- the path $A \rightarrow D \leftarrow B$ is not active.

Following is the definition of d-separation:

Definition 21 (Pearl [19]). In a directed acyclic graph $\mathcal{G}$, nodes Z d-separate X from Y, written $Sep_{\mathcal{G}}(X, Z, Y)$, precisely when there is no Z-active path between X and Y in $\mathcal{G}$. When Z is empty, we say that X and Y are d-separated.

The d-separation test is not complete in the sense that it cannot identify all LCI assertions satisfied by a database. To illustrate this, consider the structured databases in Fig. 7. In the first one, $\{A\}$ and $\{C\}$ are independent although they are not d-separated. And in the second, $\{B\}$ and $\{C\}$ are also independent although not d-separated. Intuitively, in the first database, there is no information about A that could lead us to prove anything about C. Similarly in the second database, neither B nor $\neg B$ will help in proving anything about C.

Existing structure-based algorithms use only the database structure for identifying LCI assertions. It should be clear then that such algorithms cannot be optimal because they are bound to miss some independences that could be useful computationally. This also seems to be the practice in the probabilistic literature, except possibly for some recent work on utilizing non-structural independence [1, 18].

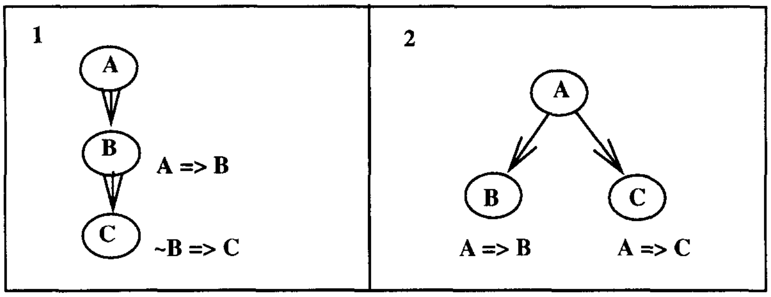

Fig. 7. Structured databases satisfying more independences than is revealed by their graphical structures.

6.3. Structure-based pruning

The modularity condition on local databases has strong implications that make structured databases attractive from computational complexity and knowledge acquisition viewpoints. To illustrate this, suppose that we are constructing a structured database incrementally by adding leaf nodes together with their local databases. When adding node n, the modularity condition ensures that the added database Δ_n does not contradict the structured database Δ constructed so far. For the database Δ_n to contradict Δ, it must entail some clause that is inconsistent with Δ. But this is impossible. Every non-valid clause that is entailed by Δ_n must mention the atom n. And any clause that mentions n cannot be inconsistent with Δ, since Δ does not mention n to start with.

Therefore, structured databases are guaranteed to be consistent by construction, which is a very attractive property from a knowledge acquisition viewpoint.[9]

Another important implication of the modularity condition on local databases is the ability to prune certain (irrelevant) parts of a structured database before computing answers to certain queries. The following theorem identifies cases where this pruning is possible.

Theorem 22. *Let $(\mathcal{G}, \Delta)$ be a structured database with exogenous variables $\mathbf{W}$, let $\mathbf{N}$ be some atoms in structure $\mathcal{G}$ and n be a leaf node in $\mathcal{G}$ that does not belong to $\mathbf{N}$. If $(\mathcal{G}', \Delta')$ is a structured database (with exogenous variables $\mathbf{W}'$) that results from removing node n and its local database Δ_n from $(\mathcal{G}, \Delta)$, then*

(1) *Entailment: $\Delta \models \check{\mathbf{N}}$ iff $\Delta' \models \check{\mathbf{N}}$,*
(2) *Satisfiability: $\Delta \cup \{\widehat{\mathbf{N}}\}$ is satisfiable iff $\Delta' \cup \{\widehat{\mathbf{N}}\}$ is satisfiable,*
(3) *Abduction: $Arg^{\Delta}_{\mathbf{W}}(\check{\mathbf{N}})$ is equivalent to $Arg^{\Delta'}_{\mathbf{W}'}(\check{\mathbf{N}})$ and*
(4) *Diagnosis: $Cons^{\Delta}_{\mathbf{W}}(\widehat{\mathbf{N}})$ is equivalent to $Cons^{\Delta'}_{\mathbf{W}'}(\widehat{\mathbf{N}})$*

for all disjunctive clauses $\check{\mathbf{N}}$ and conjunctive clauses $\widehat{\mathbf{N}}$.

[9] Modularity is also a key property responsible for the desirable properties of directed constraint networks [10].

For example, when attempting to test for the logical entailment of a clause $\check{N}$ that does not involve a leaf atom n, we can drop atom n and its local database Δ_n without affecting the result of the test. Applying this theorem recursively may prune a significant part of a structured database in certain cases. The amount of pruning, however, depends on how the atoms of clause $\check{N}$ are spread topologically in the structure. For example, at one extreme, clause $\check{N}$ may refer to all leaf nodes in the database structure. In this case, no pruning is possible. At another extreme, the clause may only mention root nodes in the structure, in which case all nodes except for those mentioned in the clause will be pruned.

From a computational complexity viewpoint, the significance of Theorem 22 is in showing how the complexity of a reasoning task is affected by the specific query.

Theorem 22 presents a result which is close in spirit to the traditional usage of irrelevance information in the literature on logical reasoning [15,23,24]. That is, before applying some algorithm for testing entailment, we prune some parts of the given database knowing that we will not compromise the soundness of the test. We have to mention, however, that although such pruning can lead to great computational savings in practice, it is only a secondary usage of independence information in the framework that we are proposing. This is also consistent with the way independence is used in probabilistic and constraint-based reasoning. The key usage of independence information is in decomposing a computation with respect to a global database into a number of independent computations with respect to local databases. This usage will be illustrated in the following section, which presents an independence-based algorithm for a class of structured databases.

We close this section with the following corollary which says that we can prune a leaf node from a structured database, without affecting the independences that hold between any of the remaining nodes.

Corollary 23. *Let $(\mathcal{G}, \Delta)$ be a structured database with exogenous variables W. Let X, Y and Z be atoms in structure $\mathcal{G}$ and let $(\mathcal{G}', \Delta')$ be a structured database (with exogenous variables W') that results from removing a leaf node from $(\mathcal{G}, \Delta)$ that does not belong to $X \cup Y \cup Z$. Then*

$$Ind_{\Delta}(X, Z \cup W, Y) \quad \textit{precisely when} \quad Ind_{\Delta'}(X, Z \cup W', Y).$$

7. Structure-based reasoning

This section presents an algorithm for computing arguments. The algorithm applies only to structured databases that are singly-connected, which are introduced in the following section. The algorithm can be generalized straightforwardly to arbitrary structured databases. The computational complexity of the algorithm and its extensions depend on the topology of the structured database to which it applies. This is why this class of algorithms is referred to as *structure-based.*

Using this algorithm, one can also decide entailment and satisfiability in addition to computing consequences. We provide more details on this later.

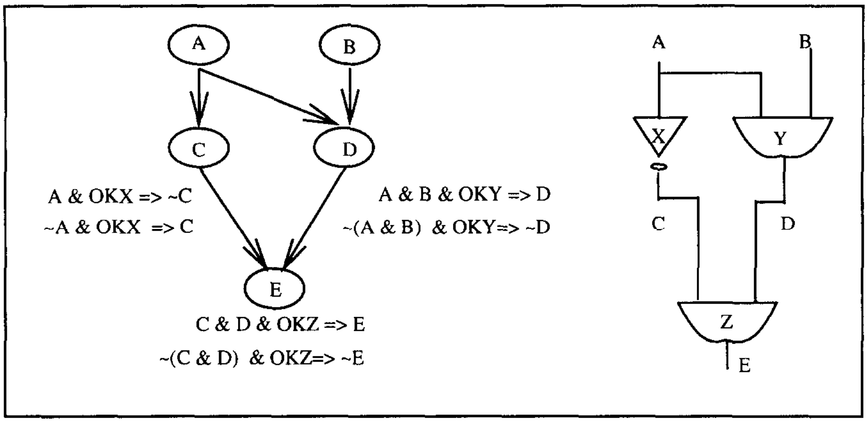

Fig. 8. A structured database representing a digital circuit. The structure of this database is multiply-connected because there is more than one undirected path between nodes A and E.

Singly-connected structures

A singly-connected structure is one in which there is only one undirected path between any two nodes—the structure has no undirected cycles. The database in Fig. 4 has a singly-connected structure, but the one in Fig. 8 has a multiply-connected structure.

Singly-connected databases are important for at least two reasons:

(1) Computation on a singly-connected structure is linear in the number of nodes and arcs of that structure, exponential only in the size of a family.[10]

(2) There is a simple algorithm for reducing a computation on a multiply-connected structure into a number n of computations on singly-connected structures, where n is exponential in the size of a structure cutset [2, 19].[11]

A singly-connected database is not necessarily Horn. For example, the databases in Figs. 2 and 4 are singly-connected and yet contain non-Horn clauses.

The rewrite paradigm

Given a singly-connected database $(\mathcal{G}, \Delta)$, and a disjunctive clause $\check{O}$ over some atoms in $\mathcal{G}$, the algorithm we shall present computes the argument for $\check{O}$ with respect to database Δ. The algorithm is symmetric to a well-known one in the probabilistic literature, known as the polytree algorithm [19], and has similar computational complexity: linear in the number of nodes and arcs in $\mathcal{G}$ and exponential in the size of each family

[10] In the literature on structure-based reasoning, the size of a family is typically assumed to be small enough to justify the assumption that any computation involving only a family will require constant time.

[11] A cutset of a directed acyclic graph is a set of nodes in the graph that satisfies the following property: if we remove the arcs outgoing from every node in the cutset, the resulting graph becomes singly-connected.

in $\mathcal{G}$. The algorithm can also be used to compute the argument of an arbitrary sentence α by first converting the sentence into conjunctive normal form $\check{O}_1 \wedge \cdots \wedge \check{O}_n$, and then applying the algorithm to each of the conjuncts:

$$Arg(\alpha) \equiv Arg(\check{O}_1 \wedge \cdots \wedge \check{O}_n) \equiv \bigwedge_i Arg(\check{O}_i).$$

The algorithm works by rewriting a global argument $Arg^{\Delta}(\check{O})$ into an expression that includes logical connectives and local arguments of the form $Arg^{\Delta_i}(\alpha_i)$, where Δ_i is a local database. A local argument can be evaluated by operating on a local database, which is assumed to require constant time operation. The algorithm alternates between applying the case-analysis and decomposition rewrite rules:

(1) *case-analysis*: $Arg(\alpha)$ is rewritten into $Arg(\alpha \vee \beta) \wedge Arg(\alpha \vee \neg\beta)$, and
(2) *decomposition*: $Arg(\alpha \vee \beta \vee \gamma)$ is rewritten into $Arg(\alpha \vee \gamma) \vee Arg(\beta \vee \gamma)$ when the corresponding LCI assertion holds.

The alternation between the two rules takes place because of the following. To decompose an argument $Arg(\check{X} \vee \check{Y})$, atoms X and Y must be independent. If they are not, the algorithm performs a case analysis on atoms Z (that make X and Y independent) in order to apply the decomposition rule. That is,

(1) the algorithm rewrites $Arg(\check{X} \vee \check{Y})$ into

$$\bigwedge_{\check{z}} Arg(\check{X} \vee \check{Y} \vee \check{Z})$$

using the case-analysis rule, and then
(2) rewrites the result into

$$\bigwedge_{\check{z}} Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Y} \vee \check{Z})$$

using the decomposition rule.

Therefore, when an argument cannot be decomposed, it is first expanded using case-analysis and then decomposed. This process continues until we have a Boolean expression in which all arguments are local.

The algorithm

We now introduce some notation that is needed to state the algorithm as a set of rewrite rules. Let

- x be an arbitrary node in the database structure,
- u and u' be distinct parents of x,
- y and y' be distinct children of x, and
- U be all parents of x.

The algorithm can be described as a recursive and deterministic[12] application of the following rewrite rules:

[12] Except for the first rewrite rule, which applies only once.

$$Arg(\check{O}) \longrightarrow (\pi(x) \vee \lambda(x)) \bigwedge (\pi(\neg x) \vee \lambda(\neg x)), \quad \text{for some node } x \text{ in the structure } \mathcal{G} \qquad (1)$$

$$\pi(\check{x}) \longrightarrow \bigwedge_{\check{U}} Arg^{\Delta_x}(\check{x} \vee \check{U}) \vee \bigvee_{\check{u} \models \check{U}} \pi_x(\check{u}) \qquad (2)$$

$$\lambda(\check{x}) \longrightarrow \Lambda(\check{x}) \vee \bigvee_{y} \lambda_y(\check{x}) \qquad (3)$$

$$\pi_y(\check{x}) \longrightarrow \pi(\check{x}) \vee \bigvee_{y'} \lambda_{y'}(\check{x}) \qquad (4)$$

$$\lambda_x(\check{u}) \longrightarrow \bigwedge_{\check{x}} \lambda(\check{x}) \vee \bigwedge_{\check{u} \models \check{U}} Arg^{\Delta_x}(\check{x} \vee \check{U}) \vee \bigvee_{\check{u}' \models \check{U}} \pi_x(\check{u}') \qquad (5)$$

$$\Lambda(\check{x}) \longrightarrow \begin{cases} \textit{true} & \text{if } \neg\check{x} \text{ appears in } \check{O}, \\ \textit{false} & \text{otherwise.} \end{cases} \qquad (6)$$

The algorithm starts with the argument $Arg(\check{O})$. It then keeps on applying the rewrites given above until it reaches an expression that contains only the connectives $\wedge$ and $\vee$ in addition to *true*, *false* and local arguments of the form $Arg^{\Delta_x}(\check{x} \vee \check{U})$. In its intermediate stages, the constructed expression will also include the following intermediate terms π, λ, π_y, λ_x and Λ, which will be rewritten into other expressions using the above rewrites. The algorithm is guaranteed to construct an expression that is free of these intermediate terms. Given that the database structure is singly connected, it is easy to verify that the rewrite process will terminate and

- Rewrite (1) will be applied only once,
- Rewrites (2), (3) and (6) will be applied at most once per node in the database structure, and
- Rewrites (4) and (5) will be applied at most once for each arc in the database structure.

Therefore, the complexity of the algorithm and the size of the constructed Boolean expression are linear in the number of nodes and arcs, and exponential only in the size of families of the database structure.

Deriving the rewrites

We will now go over the derivation of this algorithm. This discussion would typically be part of Appendix A, but we include it here to provide an example of the techniques involved in developing a structure-based algorithm. This is important because in certain applications, one may arrive at database structures that imply strong independences and therefore may be easy to solve using a specialized structure-based algorithm. To develop such an algorithm, one needs to go into an exercise similar to the following.

The algorithm starts by rewriting $Arg(\check{O})$ using the case-analysis rule into

$$Arg(x \vee \check{O}) \wedge Arg(\neg x \vee \check{O})$$

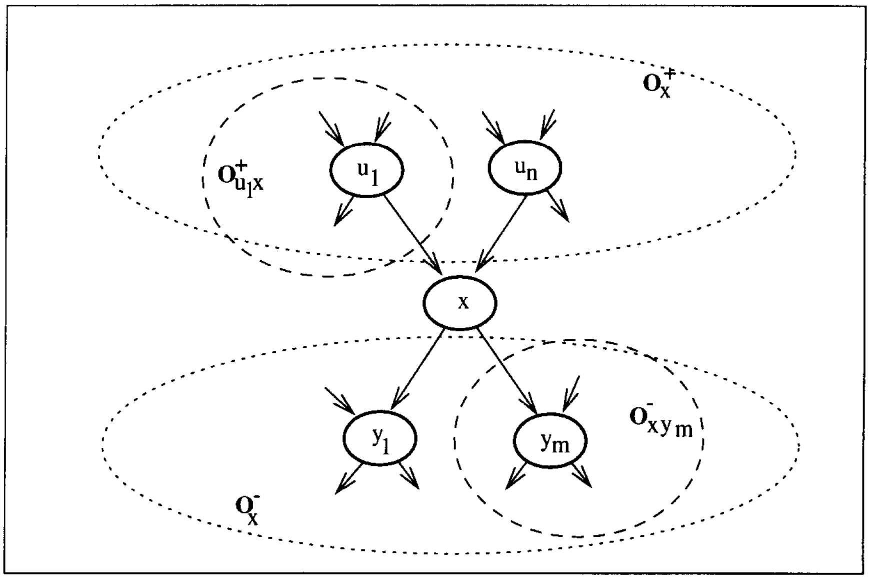

Fig. 9. Decomposing a disjunctive clause in a singly-connected structure.

for some atom x in the structure (we will assume that x does not appear in $\check{O}$ without loss of generality). To decompose the argument $Arg(\check{x} \vee \check{O})$, the algorithm partitions the disjunctive clause $\check{O}$ into two parts: a clause $\check{O}_x^-$ mentioning only descendants of x and a clause $\check{O}_x^+$ mentioning only non-descendants of x; see Fig. 9. This validates the decomposition:

$$\begin{aligned} Arg(\check{x} \vee \check{O}) &\longmapsto Arg(\check{x} \vee \check{O}_x^+ \vee \check{O}_x^-) \\ &\longmapsto \underbrace{Arg(\check{x} \vee \check{O}_x^+)}_{\pi(\check{x})} \vee \underbrace{Arg(\check{x} \vee \check{O}_x^-)}_{\lambda(\check{x})} \end{aligned}$$

because x d-separates any of its descendants from any of its non-descendants.

To decompose the argument $Arg(\check{x} \vee \check{O}_x^-)$, the algorithm partitions the clause $\check{O}_x^-$ into a number of clauses $\check{O}_{xy}^-$, each about the nodes connected to one child y of x; see Fig. 9. This validates the decomposition:

$$\begin{aligned} Arg(\check{x} \vee \check{O}_x^-) &\longmapsto Arg\Big(\check{x} \vee \bigvee_y \check{O}_{xy}^-\Big) \\ &\longmapsto \bigvee_y \underbrace{Arg(\check{x} \vee \check{O}_{xy}^-)}_{\lambda_y(\check{x})} \end{aligned}$$

because x d-separates any nodes connected to one of its children from any nodes connected to other children.

To decompose the argument $Arg(\check{x} \vee \check{O}_x^+)$, the algorithm applies the case-analysis rule:

$$Arg(\check{x} \vee \check{O}_x^+) \longmapsto \bigwedge_{\check{U}} Arg(\check{x} \vee \check{U} \vee \check{O}_x^+),$$

where $\check{U}$ is a conjunctive clause over U, the parents of x.

Using Theorem A.1 in Appendix A, we can decompose the argument

$$Arg(\check{x} \vee \check{U} \vee \check{O}_x^+)$$

into

$$Arg^{\Delta_x}(\check{x} \vee \check{U}) \vee Arg(\check{U} \vee \check{O}_x^+).$$

Note here that the first argument is local, that is, can be computed using only the local database for x.

To decompose the argument $Arg(\check{U} \vee \check{O}_x^+)$, the algorithm partitions the clause $\check{O}_x^+$ into a number of clauses $\check{O}_{ux}^+$, each about the nodes connected to x through its parent u; see Fig. 9. This allows the application of the decomposition rule:

$$\begin{aligned} Arg(\check{U} \vee \check{O}_x^+) \quad &\longmapsto \quad Arg\Big(\bigvee_{\check{u} \models \check{U}} \check{u} \vee \check{O}_{ux}^+\Big) \\ &\longmapsto \quad \bigvee_{\check{u} \models \check{U}} \underbrace{Arg(\check{u} \vee \check{O}_{ux}^+)}_{\pi_x(\check{u})}, \end{aligned}$$

which is validated by d-separation: each parent u of x and the nodes in clause $\check{O}_{ux}^+$ are d-separated from every other parent u' of x and the nodes in clause $\check{O}_{u'x}^+$. The rewrite for $\lambda_y(\check{x})$ and that for $\pi_x(\check{u})$ can be derived similarly. Moreover, given that x may appear in $\check{O}$, the term A must be inserted in the Rewrites (1)-(6) as shown.

The algorithm presented in this section can be extended to multiply-connected databases straightforwardly, leading to an algorithm called *cutset conditioning* that is exponential in the size of a structure cutset [2, 19, 20]. There are more sophisticated extensions to multiply-connected structures, however, but they are outside the scope of this paper [3, 14, 22]. Note that multiply-connected structures are not necessarily harder than singly-connected ones in general. For example, n-bit adders lead to multiply-connected structures that behave computationally like singly-connected ones.

Other reasoning tasks

The algorithm we just presented computes the argument for a given disjunctive clause $\check{O}$. Given the duality between arguments and consequences, the algorithm can also be used to compute the consequence of a conjunctive clause $\widehat{O}$ using

$$Cons(\widehat{O}) \equiv \neg Arg(\neg\widehat{O}).$$

The algorithm can also be used to decide entailment since

$$\Delta \models \alpha \quad \text{precisely when} \quad Arg_{\emptyset}^{\Delta}(\alpha) \equiv \mathit{true}.$$

Therefore, if the structured database $(\mathcal{G}, \Delta)$ has no exogenous variables—that is, all atoms that appear in Δ also appear in $\mathcal{G}$—then one can compute arguments and declare a clause $\check{O}$ as entailed by the database Δ iff the computed argument for $\check{O}$ is equivalent to *true*.

Given the duality between satisfiability and entailment in propositional logic, one can also use the algorithm for testing satisfiability.

8. Conclusion

We have presented a notion of conditional independence with respect to propositional databases that resembles its probabilistic counterpart. We have also presented several formulations of logical independence, together with their applications to deciding logical entailment, computing abductions and diagnoses. We have demonstrated how structuring a database around a directed acyclic graph can lead to explicating the independences it satisfies and how independence-based algorithms can then take advantage of this structure.

Our proposed approach for utilizing independence ties the computational complexity of logical reasoning to the availability of independence information and therefore to database structure. This leads to a computational paradigm in which the complexity of reasoning is parameterized by the topology of a database structure. This structure-based approach is at the heart of causal-networks in the probabilistic literature and constraint-networks in the constraint satisfaction literature [8,9,12]. In both cases, a graphical structure is the key aspect deciding the difficulty of a reasoning problem. This structure is what users need to control in order to ensure an appropriate response time for their applications. The probabilistic literature, for example, contains techniques for tweaking this structure to ensure certain response times, most of which can be adopted by our proposed framework.

Appendix A. Proofs

The proofs will not have the same order as the theorems. We will first establish the equivalence between all LCI formulations, then prove the semi-graphoid axioms using the abduction formulation in Definition 16.

Proving equivalence between LCI definitions

We first prove the equivalence between dualities: entailment/satisfiability and diagnosis/abduction. We then prove the equivalence between belief-change and abduction formulations followed by the equivalence between belief-change and entailment formulations. We will then have covered Theorems 7, 9, 13 and 17.

Equivalence between entailment and consistency formulations

Suppose that

$$\Delta \models \check{X} \vee \check{Y} \vee \check{Z} \quad \text{precisely when} \quad \Delta \models \check{X} \vee \check{Z} \text{ or } \Delta \models \check{Y} \vee \check{Z}$$

for all disjunctive clauses $\check{X}, \check{Y}, \check{Z}$. Then $\Delta \cup \{\neg\check{Z}, \neg\check{Y}, \neg\check{X}\}$ is inconsistent precisely when either $\Delta \cup \{\neg\check{Z}, \neg\check{X}\}$ or $\Delta \cup \{\neg\check{Z}, \neg\check{Y}\}$ is inconsistent for all disjunctive clauses $\check{X}, \check{Y}, \check{Z}$. This means that $\Delta \cup \{\widehat{Z}, \widehat{Y}, \widehat{X}\}$ is satisfiable precisely when both $\Delta \cup \{\widehat{Z}, \widehat{X}\}$ and $\Delta \cup \{\widehat{Z}, \widehat{Y}\}$ are satisfiable for all conjunctive clauses $\widehat{X}$, $\widehat{Y}$ and $\widehat{Z}$.

The other direction follows similarly. □

Equivalence between diagnosis and abduction formulations

We now prove the equivalence between the diagnosis and abduction formulations of LCI, therefore, proving the equivalence between Definitions 12 and 16 of independence.

By Theorem 15, we have that $Arg_W^\Delta(\alpha) \equiv \neg Cons_W^\Delta(\neg\alpha)$. Therefore,

$$\models Arg(\check{X} \vee \check{Y} \vee \check{Z}) \equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Y} \vee \check{Z})$$

is equivalent to

$$\models \neg Cons(\neg\check{X} \wedge \neg\check{Y} \wedge \neg\check{Z}) \equiv \neg Cons(\neg\check{X} \wedge \neg\check{Z}) \vee \neg Cons(\neg\check{Y} \wedge \neg\check{Z}),$$

which is equivalent to

$$\models \neg Cons(\neg\check{X} \wedge \neg\check{Y} \wedge \neg\check{Z}) \equiv \neg(Cons(\neg\check{X} \wedge \neg\check{Z}) \wedge Cons(\neg\check{Y} \wedge \neg\check{Z})),$$

and

$$\models Cons(\neg\check{X} \wedge \neg\check{Y} \wedge \neg\check{Z}) \equiv Cons(\neg\check{X} \wedge \neg\check{Z}) \wedge Cons(\neg\check{Y} \wedge \neg\check{Z}).$$

Therefore,

$$\models Arg(\check{X} \vee \check{Y} \vee \check{Z}) \equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Y} \vee \check{Z})$$

for all $\check{X}$, $\check{Y}$ and $\check{Z}$ iff

$$\models Cons(\widehat{X} \wedge \widehat{Y} \wedge \widehat{Z}) \equiv Cons(\widehat{X} \wedge \widehat{Z}) \wedge Cons(\widehat{Y} \wedge \widehat{Z})$$

for all $\widehat{X}$, $\widehat{Y}$ and $\widehat{Z}$. □

Equivalence between belief change and abduction formulations

We now prove that $Ind_\Delta^b(X, Z \cup W, Y)$ (Definition 3) is equivalent to $Ind_{(\Delta,W)}^a(X, Z, Y)$ (Definition 16).

The following observations (lemmas) are useful in understanding the proof of this theorem:

- Each sentence $\tilde{X}$ is equivalent to the conjunction of some $\check{X}$'s. Example: $a \wedge b$ is equivalent to $(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b)$.
- Each sentence $\tilde{X}$ is equivalent to the disjunction of some $\widehat{X}$'s. Example: $a \supset b$ is equivalent to $(a \wedge b) \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b)$.

- Suppose that there is no $\check{X}$ and $\widehat{Y}$ such that
  (1) $\Delta \cup \{\widehat{Y}\}$ is consistent,
  (2) $\Delta \not\models \check{X}$,
  (3) $\Delta \cup \{\widehat{Y}\} \models \check{X}$.
  Then there is no $\check{X}$ and $\tilde{Y}$ such that
  (1) $\Delta \cup \{\tilde{Y}\}$ is consistent,
  (2) $\Delta \not\models \check{X}$,
  (3) $\Delta \cup \{\tilde{Y}\} \models \check{X}$.
  Because if $\tilde{Y}$ is consistent with Δ and $\Delta \cup \{\tilde{Y}\} \models \check{X}$, then there must exist some $\widehat{Y}$ such that $\widehat{Y}$ is consistent with Δ and $\Delta \cup \{\widehat{Y}\} \models \check{X}$.
- Suppose that there is no $\tilde{Y}$ and $\check{X}$ such that
  (1) $\Delta \cup \{\tilde{Y}\}$ is consistent,
  (2) $\Delta \not\models \check{X}$, and
  (3) $\Delta \cup \{\tilde{Y}\} \models \check{X}$.
  Then there is no $\tilde{Y}$ and $\tilde{X}$ such that
  (1) $\Delta \cup \{\tilde{Y}\}$ is consistent,
  (2) $\Delta \not\models \tilde{X}$, and
  (3) $\Delta \cup \{\tilde{Y}\} \models \tilde{X}$.
  Because if $\Delta \not\models \tilde{X}$ and $\Delta \cup \{\tilde{Y}\} \models \tilde{X}$, then there must exist some $\check{X}$ such that $\Delta \not\models \check{X}$ and $\Delta \cup \{\tilde{Y}\} \models \check{X}$.
- The following is always true:

$$Arg(\check{Z} \vee \check{X}) \vee Arg(\check{Z} \vee \check{Y}) \models Arg(\check{Z} \vee \check{X} \vee \check{Y}).$$

  Because if

$$\Delta \cup \{\tilde{W}\} \models \check{Z} \vee \check{X} \text{ or } \Delta \cup \{\tilde{W}\} \models \check{Z} \vee \check{Y},$$

  then also

$$\Delta \cup \{\tilde{W}\} \models \check{Z} \vee \check{X} \vee \check{Y}.$$

The proof of the theorem is given below.
($\Rightarrow$) Suppose that $Ind^{a}_{(\Delta,W)}(X, Z, Y)$ does not hold. That is, we have

$$Arg(\check{Z} \vee \check{X} \vee \check{Y}) \not\models Arg(\check{Z} \vee \check{X}) \vee Arg(\check{Z} \vee \check{Y})$$

for some $\check{X}$, $\check{Y}$ and $\check{Z}$. We want to show that $Ind^{b}_{\Delta}(X, Z \cup W, Y)$ does not hold either.
By supposition, there must exist some $\widehat{W}$ such that
(1) $\Delta \cup \{\widehat{W}\} \models \check{Z} \vee \check{X} \vee \check{Y}$,
(2) $\Delta \cup \{\widehat{W}\} \not\models \check{Z} \vee \check{X}$,
(3) $\Delta \cup \{\widehat{W}\} \not\models \check{Z} \vee \check{Y}$.
This means that
(1) $\Delta \cup \{\widehat{W}, \neg\check{Z}, \neg\check{Y}\} \models \check{X}$,
(2) $\Delta \cup \{\widehat{W}, \neg\check{Z}\} \not\models \check{X}$,
(3) $\Delta \cup \{\widehat{W}, \neg\check{Z}, \neg\check{Y}\}$ is consistent.
Therefore, $Ind^{b}_{\Delta}(X, Z \cup W, Y)$ does not hold. We have then proved that $Ind^{b}_{\Delta}(X, Z \cup W, Y)$ of implies $Ind^{a}_{(\Delta,W)}(X, Z, Y)$.

($\Leftarrow$) Suppose that $Ind^b_\Delta(X, Z \cup W, Y)$ does not hold. Then there must exist $\widehat{W}$, $\widehat{Z}$, $\tilde{Y}$, and $\tilde{X}$ such that
(1) $\Delta \cup \{\widehat{W}, \widehat{Z}\} \not\models \tilde{X}$,
(2) $\Delta \cup \{\widehat{W}, \widehat{Z}, \tilde{Y}\} \models \tilde{X}$,
(3) $\Delta \cup \{\widehat{W}, \widehat{Z}, \tilde{Y}\}$ is consistent.
By the above lemmas, there must exist $\widehat{Y}$ and $\check{X}$ such that
(1) $\Delta \cup \{\widehat{W}, \widehat{Z}\} \not\models \check{X}$,
(2) $\Delta \cup \{\widehat{W}, \widehat{Z}, \widehat{Y}\} \models \check{X}$,
(3) $\Delta \cup \{\widehat{W}, \widehat{Z}, \widehat{Y}\}$ is consistent.
That is, we have
(1) $\Delta \cup \{\widehat{W}\} \not\models \neg\widehat{Z} \vee \check{X}$,
(2) $\Delta \cup \{\widehat{W}\} \models \neg\widehat{Z} \vee \check{X} \vee \neg\widehat{Y}$,
(3) $\Delta \cup \{\widehat{W}\} \not\models \neg\widehat{Z} \vee \neg\widehat{Y}$.
Therefore,

$$Arg(\neg\widehat{Z} \vee \check{X} \vee \neg\widehat{Y}) \not\models Arg(\neg\widehat{Z} \vee \check{X}) \vee Arg(\neg\widehat{Z} \vee \neg\widehat{Y}),$$

which means that $Ind^a_{(\Delta,W)}(X, Z, Y)$ does not hold. We have then proved that $Ind^a_{(\Delta,W)}(X, Z, Y)$ implies $Ind^b_\Delta(X, Z \cup W, Y)$. □

Equivalence between belief change and entailment formulations
Suppose that $Ind^b_\Delta(X, Z, Y)$. Then $Ind^a_{(\Delta,W=\emptyset)}(X, Z, Y)$ and, hence,

$$\models Arg(\check{X} \vee \check{Y} \vee \check{Z}) \equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Y} \vee \check{Z}).$$

Since W is empty, arguments with respect to (Δ, W) can only be *true* or *false*. Moreover,
- $Arg(\alpha) = true$ iff $\Delta \models \alpha$.
- $Arg(\alpha) = false$ iff $\Delta \not\models \alpha$.

Therefore,

$$\models Arg(\check{X} \vee \check{Y} \vee \check{Z}) \equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Y} \vee \check{Z})$$

is equivalent to

$$\Delta \models \check{X} \vee \check{Y} \vee \check{Z} \quad \text{precisely when} \quad \Delta \models \check{X} \vee \check{Z} \text{ or } \Delta \models \check{Y} \vee \check{Z}$$

and we have $Ind^e_\Delta(X, Z, Y)$. The other direction follows similarly. □

Proof of Theorem 4

The proof uses Definitions 12 of independence. If we choose $W = \emptyset$, we obtain a proof with respect to Definitions 3 of independence.

(1) *Trivial independence*: $Ind_\Delta(X, Z, \emptyset)$. Note that $\check{\emptyset}$ is *false* and that $Arg(\check{Z}) \models Arg(\check{X} \vee \check{Z})$ since $\check{Z} \models \check{X} \vee \check{Z}$. We have:

$$\begin{aligned} \models Arg(\check{X} \vee \check{\emptyset} \vee \check{Z}) &\equiv Arg(\check{X} \vee false \vee \check{Z}) \\ &\equiv Arg(\check{X} \vee \check{Z}) \end{aligned}$$

$$\equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{Z})$$
$$\equiv Arg(\check{X} \vee \check{Z}) \vee Arg(\check{\emptyset} \vee \check{Z}).$$

Therefore, $Ind_{\Delta}(X, Z, \emptyset)$.

(2) *Symmetry*: $Ind_{(\Delta,W)}(X, Z, Y)$ precisely when $Ind_{(\Delta,W)}(Y, Z, X)$. Follows directly from definition.

(3) *Decomposition*: If $Ind_{(\Delta,W)}(X \cup L, Z, Y)$, then $Ind_{(\Delta,W)}(X, Z, Y)$. This proof uses the following facts:

(a) If $\models \phi_i \equiv \psi_i$, then $\models \bigwedge_i \phi_i \equiv \bigwedge_i \psi_i$.

(b) $\models \bigwedge_i (\phi \vee \psi_i) \equiv \phi \vee \bigwedge_i \psi_i$.

(c) $\models \bigwedge_{\check{X}} Arg(\check{X} \vee \check{Y}) \equiv Arg(\check{Y})$.

Suppose that $Ind_{(\Delta,W)}(X \cup L, Z, Y)$:

$$\models Arg(\check{Z} \vee \check{X} \vee \check{L} \vee \check{Y}) \equiv Arg(\check{Z} \vee \check{X} \vee \check{L}) \vee Arg(\check{Z} \vee \check{Y}).$$

We have

$$\models \bigwedge_{\check{L}} Arg(\check{Z} \vee \check{X} \vee \check{L} \vee \check{Y}) \equiv \bigwedge_{\check{L}} Arg(\check{Z} \vee \check{X} \vee \check{L}) \vee Arg(\check{Z} \vee \check{Y}),$$

$$\models Arg(\check{Z} \vee \check{X} \vee \check{Y}) \equiv Arg(\check{Z} \vee \check{Y}) \vee \bigwedge_{\check{L}} Arg(\check{Z} \vee \check{X} \vee \check{L})$$
$$\equiv Arg(\check{Z} \vee \check{Y}) \vee Arg(\check{Z} \vee \check{X}).$$

Hence, $Ind_{(\Delta,W)}(X, Z, Y)$.

(4) *Weak union*: If $Ind_{(\Delta,W)}(X \cup L, Z, Y)$, then $Ind_{(\Delta,W)}(L, Z \cup X, Y)$. Suppose that $Ind_{(\Delta,W)}(X \cup L, Z, Y)$:

$$\models Arg(\check{Z} \vee \check{X} \vee \check{L} \vee \check{Y}) \equiv Arg(\check{Z} \vee \check{X} \vee \check{L}) \vee Arg(\check{Z} \vee \check{Y}).$$

Since $Arg(\check{Z} \vee \check{X}) \models Arg(\check{Z} \vee \check{X} \vee \check{L})$, we have

$$\models Arg(\check{Z} \vee \check{X} \vee \check{L}) \equiv Arg(\check{Z} \vee \check{X} \vee \check{L}) \vee Arg(\check{Z} \vee \check{X}),$$

and

$$\models Arg(\check{Z} \vee \check{X} \vee \check{L} \vee \check{Y}) \equiv [Arg(\check{Z} \vee \check{X} \vee \check{L}) \vee Arg(\check{Z} \vee \check{X})] \vee Arg(\check{Z} \vee \check{Y}).$$

Moreover, by decomposition, we have $Ind_{(\Delta,W)}(X, Z, Y)$:

$$\models Arg(\check{Z} \vee \check{X} \vee \check{Y}) \equiv Arg(\check{Z} \vee \check{X}) \vee Arg(\check{Z} \vee \check{Y}).$$

Hence,

$$\models Arg(\check{Z} \vee \check{X} \vee \check{L} \vee \check{Y}) \equiv Arg(\check{Z} \vee \check{X} \vee \check{L}) \vee Arg(\check{Z} \vee \check{X} \vee \check{Y})$$

and $Ind_{(\Delta,W)}(L, Z \cup X, Y)$.

(5) *Contraction*: If $Ind_{(\Delta,W)}(X,Z,Y)$ and $Ind_{(\Delta,W)}(L,Z\cup X,Y)$, then $Ind_{(\Delta,W)}(X\cup L,Z,Y)$. Suppose that $Ind_{(\Delta,W)}(X,Z,Y)$ and $Ind_{(\Delta,W)}(L,Z\cup X,Y)$:

$$\models Arg(\check{Z}\vee\check{X}\vee\check{Y})\equiv Arg(\check{Z}\vee\check{Y})\vee Arg(\check{Z}\vee\check{X})\text{ and}$$

$$\models Arg(\check{Z}\vee\check{X}\vee\check{L}\vee\check{Y})\equiv Arg(\check{Z}\vee\check{X}\vee\check{L})\vee Arg(\check{Z}\vee\check{X}\vee\check{Y}).$$

Expanding the second equation using the first:

$$\models Arg(\check{Z}\vee\check{X}\vee\check{L}\vee\check{Y})\equiv Arg(\check{Z}\vee\check{X}\vee\check{L})\vee Arg(\check{Z}\vee\check{Y})\vee Arg(\check{Z}\vee\check{X}),$$

which reduces to

$$\models Arg(\check{Z}\vee\check{X}\vee\check{L}\vee\check{Y})\equiv Arg(\check{Z}\vee\check{X}\vee\check{L})\vee Arg(\check{Z}\vee\check{Y}),$$

because

$$\models Arg(\check{Z}\vee\check{X})\models Arg(\check{Z}\vee\check{X}\vee\check{L}).$$

Hence, $Ind_{(\Delta,W)}(X\cup L,Z,Y)$. □

Proof of Theorem 11

Suppose that $Cons^{\Delta}_{W}(\gamma)=\widehat{W}_1\vee\cdots\vee\widehat{W}_n$. Then $\Delta\cup\{\gamma\}\models\widehat{W}_1\vee\cdots\vee\widehat{W}_n$. We need to prove that
(1) each $\widehat{W}_i$ is a diagnosis, and
(2) if $\widehat{W}$ is a diagnosis, then $\widehat{W}\models\widehat{W}_1\vee\cdots\vee\widehat{W}_n$.
To show that $\widehat{W}_i$ is a diagnosis, we need to show that $\Delta\cup\{\gamma,\widehat{W}_i\}$ is consistent. Suppose that $\Delta\cup\{\gamma,\widehat{W}_i\}$ is not consistent. Then $\Delta\cup\{\gamma\}\models\neg\widehat{W}_i$ and

$$\Delta\cup\{\gamma\}\models\widehat{W}_1\vee\cdots\vee\widehat{W}_{i-1}\vee\widehat{W}_{i+1}\vee\cdots\vee\widehat{W}_n.$$

But this contradicts with $\widehat{W}_1\vee\cdots\vee\widehat{W}_n$ being a logically strongest sentence entailed by $\Delta\cup\{\gamma\}$.

Suppose that $\widehat{W}$ is a diagnosis. Then $\Delta\cup\{\gamma,\widehat{W}\}$ is consistent and $\Delta\cup\{\gamma,\widehat{W}\}\models\widehat{W}_1\vee\cdots\vee\widehat{W}_n$. This means that $\widehat{W}$ must belong to $\widehat{W}_1\vee\cdots\vee\widehat{W}_n$, otherwise $\Delta\cup\{\gamma,\widehat{W}\}\models\neg\widehat{W}_1,\ldots,\Delta\cup\{\gamma,\widehat{W}\}\models\neg\widehat{W}_n$. This means that $\Delta\cup\{\gamma,\widehat{W}\}$ is inconsistent, which we know it is not. □

Proof of Theorem 15

Suppose that $Arg^{\Delta}_{W}(\alpha)\equiv\beta$. Then β is a logically weakest sentence that is constructed from atoms W such that

$$\Delta\cup\{\beta\}\models\alpha.$$

It then follows that $\neg\beta$ is a logically strongest sentence constructed from W such that

$$\Delta\cup\{\neg\alpha\}\models\neg\beta.$$

Therefore, $\neg\beta$ is the consequence of $\neg\alpha$, that is, $Cons^{\Delta}_{W}(\neg\alpha)\equiv\neg\beta$. This implies that $Cons^{\Delta}_{W}(\neg\alpha)\equiv\neg Arg^{\Delta}_{W}(\alpha)$ and $Arg^{\Delta}_{W}(\alpha)\equiv\neg Cons^{\Delta}_{W}(\neg\alpha)$. □

Proof of Theorem 18

We will prove $Ind_{\Delta}(\{n\}, U \cup W, N)$ by assuming that n is a leaf node in $\mathcal{G}$. The proof will also work when n is not a leaf node since, by Corollary 23 of Theorem 22, we can prune the descendants of n to make it a leaf node without invalidating the proof.

To prove $Ind_{\Delta}(\{n\}, U \cup W, N)$, we need to prove

$$\Delta \cup \{\widehat{W}, \widehat{U}\} \models \tilde{n} \quad \text{precisely when} \quad \Delta \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \models \tilde{n},$$

for all $\widehat{W}, \widehat{U}, \tilde{N}, \tilde{n}$ where $\Delta \cup \{\widehat{W}, \widehat{U}, \tilde{N}\}$ is consistent. The direction

$$\Delta \cup \{\widehat{W}, \widehat{U}\} \models \tilde{n} \quad \text{only if} \quad \Delta \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \models \tilde{n}$$

is trivial. The direction

$$\Delta \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \models \tilde{n} \quad \text{only if} \quad \Delta \cup \{\widehat{W}, \widehat{U}\} \models \tilde{n}$$

is equivalent to

$$\Delta \cup \{\widehat{W}, \widehat{U}\} \not\models \tilde{n} \quad \text{only if} \quad \Delta \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \not\models \tilde{n}$$

and

$$\Delta' \cup \Delta_n \cup \{\widehat{W}, \widehat{U}\} \not\models \tilde{n} \quad \text{only if} \quad \Delta' \cup \Delta_n \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \not\models \tilde{n}$$

where $\Delta' = \Delta \setminus \Delta_n$.

Suppose that

$$\Delta' \cup \Delta_n \cup \{\widehat{W}, \widehat{U}\} \not\models \tilde{n}.$$

One of the following must be true:

(1) $\Delta_n \cup \{\widehat{W}, \widehat{U}\}$ is equivalent to $\{\widehat{W}, \widehat{U}\}$ or

(2) $\Delta_n \cup \{\widehat{W}, \widehat{U}\}$ is equivalent to $\{\neg\tilde{n}, \widehat{W}, \widehat{U}\}$.

This holds because by definition of a local database, Δ_n must be equivalent to a set of disjunctive clauses, each referring only to W, U and n and containing either $\tilde{n}$ or $\neg\tilde{n}$. Moreover, each one of these clauses must either

(1) be entailed by $\{\widehat{W}, \widehat{U}\}$ in which case it can be removed from Δ_n or

(2) resolve with $\{\widehat{W}, \widehat{U}\}$ to yield $\neg\tilde{n}$ in which case it can be replaced by $\neg\tilde{n}$.[13]

This means that we either have

(1) $\Delta' \cup \Delta_n \cup \{\widehat{W}, \widehat{U}\}$ is equivalent to $\Delta' \cup \{\widehat{W}, \widehat{U}\}$ and $\Delta' \cup \{\widehat{W}, \widehat{U}\} \not\models \tilde{n}$ or

(2) $\Delta' \cup \Delta_n \cup \{\widehat{W}, \widehat{U}\}$ is equivalent to $\Delta' \cup \{\neg\tilde{n}, \widehat{W}, \widehat{U}\}$ and $\Delta' \cup \{\neg\tilde{n}, \widehat{W}, \widehat{U}\} \not\models \tilde{n}$,

and therefore either

(1) $\Delta' \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \not\models \tilde{n}$ since $\Delta' \cup \{\widehat{W}, \widehat{U}, \tilde{N}\}$ is consistent and does not refer to n, or

(2) $\Delta' \cup \{\neg\tilde{n}, \widehat{W}, \widehat{U}, \tilde{N}\} \not\models \tilde{n}$ since $\Delta' \cup \{\neg\tilde{n}, \widehat{W}, \widehat{U}, \tilde{N}\}$ is consistent.

In either case, it follows that

$$\Delta \cup \{\widehat{W}, \widehat{U}, \tilde{N}\} \not\models \tilde{n}. \qquad \square$$

[13] The result of the resolution cannot be $\tilde{n}$ since $\Delta' \cup \Delta_n \cup \{\widehat{W}, \widehat{U}\} \not\models \tilde{n}$.

Proof of Theorem 19

Verma has shown in [25] that if

(1) N is a set of atomic propositions,

(2) $\mathcal{I}$ is an independence relation over N, a subset of $2^N \times 2^N \times 2^N$, that satisfies the semi-graphoid axioms,

(3) $\mathcal{G}$ is a DAG over nodes N,

(4) each node n in $\mathcal{G}$ is independent of its non-descendants given its parents according to $\mathcal{I}$,

then whenever X and Y are d-separated by Z in $\mathcal{G}$, we must have $I(X, Z, Y)$.

Given a structured database $(\mathcal{G}, \Delta)$, we know that the independence relation Ind_Δ satisfies the semi-graphoid axioms. And by Theorem 18, we know that each node in $\mathcal{G}$ is independent of its non-descendants given its parents, according to Ind_Δ. Therefore, whenever X and Y are d-separated by Z in $\mathcal{G}$, we must have $Ind_\Delta(X, Z, Y)$. □

Proof of Theorem 22

(1) The equivalence between $\Delta \models \check{N}$ and $\Delta' \models \check{N}$ follows immediately from the proof in (2) below.

(2) The consistency of $\Delta \cup \{\widehat{N}\}$ is equivalent to the consistency of

$$\Delta \cup \Big\{\bigvee_{\widehat{W},\widehat{A}} \widehat{N} \wedge \widehat{W} \wedge \widehat{A}\Big\},$$

where A are the other atoms of $\mathcal{G}$ (different from n and N). Similarly, the consistency of $\Delta' \cup \{\widehat{N}\}$ is equivalent to the consistency of

$$\Delta' \cup \Big\{\bigvee_{\widehat{W},\widehat{A}} \widehat{N} \wedge \widehat{W} \wedge \widehat{A}\Big\}.$$

Therefore, it suffices to show that the consistency of each $\Delta \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ is equivalent to the consistency of its corresponding $\Delta' \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$. Suppose that $\Delta \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ is consistent. Then $\Delta' \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ must also be consistent since $\Delta' \subseteq \Delta$. Now suppose that $\Delta' \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ is consistent. Then it must be equivalent to $\{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ since $N \cup W \cup A$ are all the atoms appearing in Δ. Therefore, $\Delta \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ must be equivalent to $\Delta_n \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ since $\Delta = \Delta' \cup \Delta_n$. By the definition of local database Δ_n, it follows that $\Delta_n \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ must be consistent.[14]

We have therefore established the equivalence between the consistency of $\Delta \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$ and that of $\Delta' \cup \{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$. By that we have also established that $\Delta \cup \{\widehat{N}\}$ is consistent iff $\Delta' \cup \{\widehat{N}\}$ is.

(3) Note that $Arg^{\Delta}_{W}(\check{N})$ is equivalent to $Arg^{\Delta}_{W \cup Z}(\check{N})$ if atoms Z do not appear in Δ. Therefore, $Arg^{\Delta'}_{W'}(\check{N})$ is equivalent to $Arg^{\Delta'}_{W}(\check{N})$ since $W \setminus W'$ do not appear in Δ'. All

[14] Recall that any disjunctive clause entailed by database Δ_n must include atom n, which does not appear in $\{\widehat{N} \wedge \widehat{W} \wedge \widehat{A}\}$.

we need to show then is the equivalence between $Arg^{\Delta}_{W}(\check{N})$ and $Arg^{\Delta'}_{W}(\check{N})$, which can be established by proving that $\Delta \cup \{\widehat{W}\} \models \check{N}$ iff $\Delta' \cup \{\widehat{W}\} \models \check{N}$. This can be proved as shown in (2) above.

(4) The equivalence between $Cons^{\Delta}_{W}(\widehat{N})$ and $Cons^{\Delta'}_{W'}(\widehat{N})$ follows immediately from (3) and Theorem 15. □

Proof of Corollary 23

What we need to show is

$$Arg^{\Delta}_{W}(\check{X} \vee \check{Z} \vee \check{Y}) \equiv Arg^{\Delta}_{W}(\check{X} \vee \check{Z}) \vee Arg^{\Delta}_{W}(\check{Y} \vee \check{Z})$$

precisely when

$$Arg^{\Delta'}_{W'}(\check{X} \vee \check{Z} \vee \check{Y}) \equiv Arg^{\Delta'}_{W'}(\check{X} \vee \check{Z}) \vee Arg^{\Delta'}_{W'}(\check{Y} \vee \check{Z}).$$

This follows from Theorem 22, according to which

$$\models Arg^{\Delta}_{W}(\check{X} \vee \check{Z} \vee \check{Y}) \equiv Arg^{\Delta'}_{W'}(\check{X} \vee \check{Z} \vee \check{Y}),$$
$$\models Arg^{\Delta}_{W}(\check{X} \vee \check{Z}) \equiv Arg^{\Delta'}_{W'}(\check{X} \vee \check{Z}) \text{ and}$$
$$\models Arg^{\Delta}_{W}(\check{Y} \vee \check{Z}) \equiv Arg^{\Delta'}_{W'}(\check{Y} \vee \check{Z}). \qquad \square$$

Theorem A.1. *Let $(\mathcal{G}, \Delta)$ be a structured database, n be a node in $\mathcal{G}$, U be its parents and N be some of its other ancestors. We then have*

$$Arg^{\Delta}(\check{n} \vee \check{U} \vee \check{N}) \equiv Arg^{\Delta_n}(\check{n} \vee \check{U}) \vee Arg^{\Delta}(\check{U} \vee \check{N}).$$

Proof. By Theorem 18, we have

$$\models Arg^{\Delta}_{W}(\check{n} \vee \check{U} \vee \check{N}) \equiv Arg^{\Delta}_{W}(\check{n} \vee \check{U}) \vee Arg^{\Delta}_{W}(\check{U} \vee \check{N})$$

since $Ind_{\Delta}(\{n\}, U \cup W, N)$. Therefore, it suffices to show

$$\models Arg^{\Delta}_{W}(\check{n} \vee \check{U}) \equiv Arg^{\Delta_n}_{W}(\check{n} \vee \check{U}) \vee Arg^{\Delta}_{W}(\check{U})$$

since $Arg^{\Delta}_{W}(\check{U})$ entails $Arg^{\Delta}_{W}(\check{U} \vee \check{N})$. The direction

$$Arg^{\Delta_n}_{W}(\check{n} \vee \check{U}) \vee Arg^{\Delta}_{W}(\check{U}) \models Arg^{\Delta}_{W}(\check{n} \vee \check{U})$$

is trivial. To show the direction

$$Arg^{\Delta}_{W}(\check{n} \vee \check{U}) \models Arg^{\Delta_n}_{W}(\check{n} \vee \check{U}) \vee Arg^{\Delta}_{W}(\check{U})$$

we need to prove that $\Delta \cup \{\widehat{W}\} \models \check{n} \vee \check{U}$ implies either $\Delta_n \cup \{\widehat{W}\} \models \check{n} \vee \check{U}$ or $\Delta \cup \{\widehat{W}\} \models \check{U}$. If we take $\widehat{n}$ to be $\neg\check{n}$ and $\widehat{U}$ to be $\neg\check{U}$, then we need to show that $\Delta \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ is inconsistent only if $\Delta_n \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ or $\Delta \cup \{\widehat{W}, \widehat{U}\}$ is inconsistent.

Suppose that $\Delta \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ is inconsistent and $\Delta_n \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ is consistent. Then $\Delta_n \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ is equivalent to $\{\widehat{W}, \widehat{n}, \widehat{U}\}$ and therefore $\Delta \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ is equivalent to

$\Delta/\Delta_n \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$. This means that $\Delta/\Delta_n \cup \{\widehat{W}, \widehat{n}, \widehat{U}\}$ is inconsistent, which implies that $\Delta/\Delta_n \cup \{\widehat{W}, \widehat{U}\}$ is also inconsistent since atom n does not appear in $\Delta/\Delta_n \cup \{\widehat{W}, \widehat{U}\}$. Therefore, $\Delta \cup \{\widehat{W}, \widehat{U}\}$ is inconsistent. □

References

[1] B.D. D'Ambrosio, Local expression languages for probabilistic dependence, *Internat. J. Approximate Reasoning* 11 (1994) 1-15.

[2] A. Darwiche, A symbolic generalization of probability theory, Ph.D. Thesis, Stanford University, Stanford, CA (1992).

[3] A. Darwiche, Conditioning algorithms for exact and approximate inference in causal networks, in: *Proceedings 11th Conference on Uncertainty in Artificial Intelligence (UAI)* (1995) 99-107.

[4] A. Darwiche, Model-based diagnosis using causal networks, in: *Proceedings IJCAI-95*, Montreal, Que. (1995) 211-217.

[5] A. Darwiche and J. Pearl, Symbolic causal networks, in: *Proceedings AAAI-94*, Seattle, WA (1994) 238-244.

[6] J. de Kleer, An assumption-based TMS, *Artificial Intelligence* 28 (1986) 127-162.

[7] J. de Kleer, A.K. Mackworth and R. Reiter, Characterizing diagnoses and systems, *Artificial Intelligence* 56 (1992) 197-222.

[8] R. Dechter and A. Dechter, Belief maintenance in dynamic constraint networks, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 37-42.

[9] R. Dechter and A. Dechter, Structure-driven algorithms for truth maintenance, *Artificial Intelligence* 82 (1996) 1-20.

[10] R. Dechter and J. Pearl, Directed constraint networks: A relational framework for causal modeling, in: *Proceedings IJCAI-91*, Sydney, Australia (1991) 1164-1170.

[11] P. Gärdenfors, *Knowledge in Flux: Modeling the Dynamics of Epistemic States* (MIT Press, Cambridge, MA, 1988).

[12] H. Geffner and J. Pearl, An improved constraint-propagation algorithm for diagnosis, in: *Proceedings IJCAI-87*, Milan, Italy (1987) 1105-1111.

[13] R. Greiner and D. Subramanian, Relevance, in: *Working Notes, AAAI-94 Fall Symposium Series*, New Orleans, LA (1994).

[14] F.V. Jensen, S.L. Lauritzen and K.G. Olesen, Bayesian updating in recursive graphical models by local computation, *Comput. Statist. Quart.* 4 (1990) 269-282.

[15] A.Y. Levy, Irrelevance reasoning in knowledge based systems, Ph.D. Thesis, Stanford University, Stanford, CA (1993).

[16] A. Levy, I.S. Mumick and Y. Sagiv, Query optimization by predicate movearound, in: *Proceedings 20th VLDB Conference*, Santiago, Chile (1994).

[17] A.Y. Levy and Y. Sagiv, Queries independent of updates, in: *Proceedings 19th VLDB Conference*, Dublin, Ireland (1993) 171-181.

[18] Z. Li and B.D. D'Ambrosio, Efficient inference in Bayes networks as a combinatorial optimization problem, *Internat. J. Approximate Reasoning* 11 (1994) 55-81.

[19] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann, San Mateo, CA, 1988).

[20] M.A. Peot and R.D. Shachter, Fusion and propagation with multiple observations in belief networks, *Artificial Intelligence* 48 (1991) 299-318.

[21] R. Reiter and J. de Kleer, Foundations of assumption-based truth maintenance systems: preliminary report, in: *Proceedings AAAI-87*, Milan, Italy (1987) 183-188.

[22] R. Shachter, S.K. Andersen and P. Szolovits, Global conditioning for probabilistic inference in belief networks, in: *Proceedings 10th Conference on Uncertainty in AI*, Seattle, WA (1994) 514-522.

[23] D. Subramanian, A theory of justified reformulations, Ph.D. Thesis, Stanford University, Stanford, CA (1989).

[24] D. Subramanian and M.R. Genesereth, The relevance of irrelevance, in: *Proceedings IJCAI-87*, Milan, Italy (1987) 416-422.

[25] T. Verma and J. Pearl, Causal networks: semantics and expressiveness, in: *Proceedings 4th Workshop on Uncertainty in AI*, Minneapolis, MN (1988) 352-359.

[26] N. Wilson, Generating graphoids from generalised conditional probability, in: *Proceedings 10th Conference on Uncertainty in AI*, Seattle, WA (1994) 583-590.