ELSEVIER

Artificial Intelligence 97 (1997) 83–136

Artificial Intelligence

Speeding up inferences using relevance reasoning: a formalism and algorithms

Alon Y. Levy [a,*], Richard E. Fikes [b,1], Yehoshua Sagiv [c,2]

[a] *AT&T Bell Laboratories, 180 Park Ave., Room A283, Florham Park, NJ 07932, USA*

[b] *KSL, Stanford University, Stanford, CA 94305, USA*

[c] *Department of Computer Science, Hebrew University, Jerusalem, Israel*

Received September 1995; revised May 1996

Abstract

Irrelevance reasoning refers to the process in which a system reasons about which parts of its knowledge are relevant (or irrelevant) to a specific query. Aside from its importance in speeding up inferences from large knowledge bases, relevance reasoning is crucial in advanced applications such as modeling complex physical devices and information gathering in distributed heterogeneous systems. This article presents a novel framework for studying the various kinds of irrelevance that arise in inference and efficient algorithms for relevance reasoning. We present a proof-theoretic framework for analyzing definitions of irrelevance. The framework makes the necessary distinctions between different notions of irrelevance that are important when using them for speeding up inferences. We describe the *query-tree* algorithm which is a sound, complete and efficient algorithm for automatically deriving certain kinds of irrelevance claims for Horn-rule knowledge bases and several extensions. Finally, we describe experimental results that show that significant speedups (often orders of magnitude) are obtained by employing the query-tree in inference. © 1997 Elsevier Science B.V.

Keywords: Relevance reasoning; Meta-level reasoning; Static analysis; Horn rules; Constraints; Knowledge representation

* Corresponding author. Email: levy@research.att.com.

[1] Email: fikes@cs.stanford.edu.

[2] Email: sagiv@cs.huji.ac.il.

PII S0004-3702(97)00049-0

1. Introduction

Many future applications of Artificial Intelligence (AI) will be in domains having large amounts of knowledge. In order for a system to perform efficiently in such a domain it is essential that it be able to determine which knowledge is relevant to any given query or task. In fact, the inability of current AI systems to ignore irrelevant information is a major obstacle in scaling up such systems. *Irrelevance reasoning* refers to the process in which the system reasons about which parts of its knowledge are relevant (or irrelevant) to a specific query, either by automatically inspecting the knowledge base or by exploiting explicit *irrelevance-claims* given by a user. Irrelevance reasoning is a specific form of meta-level reasoning [37,45,54], in which we reason *about* the knowledge in the knowledge base, as opposed to using the knowledge base to reason about the domain. Irrelevance reasoning is important in several contexts:

- *Speeding up inferences in large knowledge bases*: It is well known that the performance of inference engines in AI systems degrades quickly as the size of the knowledge base increases. Two of the major sources of inefficiency are due to irrelevant information:
  - Irrelevant facts in the knowledge base: In its search for a solution, the inference engine considers many facts in the knowledge base that are irrelevant to the query. Consequently, it spends significant effort pursuing useless solution paths.
  - Irrelevant distinctions in the representation: A knowledge base is designed to accommodate a variety of tasks. Therefore, its conceptualization of the domain must be detailed enough for all those tasks (i.e., it must include many refined relations, objects, etc.). As a result, the representation is likely to be too complex for any given task, thereby leading to inefficient reasoning.
- *Modeling complex physical devices*: Tasks such as diagnosis, design and simulation require a model of a given physical device [11,22]. However, the adequacy of a model depends heavily on the task for which it is used, and reasoning directly with the most detailed model of the system would be intractable. Therefore, it is important to be able to automatically create a model that is suited for a given query (e.g.,[2,32,46]), and doing so requires that we determine which aspects of the system are relevant to a given query.
- *Large scale distributed information systems*: Current communications technology (e.g., the Internet) enables easy access to many remote sources of information. At the moment, accessing this information can be mostly by browsing. A growing body of work in AI has the goal of designing architectures for integrating multiple sources of information and providing high level querying facilities over them, thereby freeing a user from the need to know about specific information sources [4, 31,33,38,51,60,65,90]. A key issue that needs to be addressed by these systems is the ability to automatically determine which information sources are relevant to a given query posed by a user.

Irrelevance reasoning also plays an important role in nonmonotonic reasoning [7,36, 71], belief revision [35] and learning (e.g., [30,66,74]). The focus of this paper is on using relevance reasoning to speed up inferences in large knowledge bases. The other applications of relevance reasoning are discussed briefly in Section 6.

In order for relevance reasoning to be a viable method for controlling inference, several issues need to be addressed. First, we must develop *efficient* algorithms for automatically detecting parts of a knowledge base that are irrelevant to a query. The input to these algorithms may vary; it could consist of certain parts (e.g., the rules) of the knowledge base, some meta-claims about other parts (e.g., integrity constraints on the ground facts in the knowledge base) and, possibly, some irrelevance claims supplied by the user. Second, we must investigate the *utility* of removing irrelevant knowledge and the tradeoff between meta-level reasoning about relevance and base-level reasoning about the domain. As a basis for addressing these two questions, we need a formal understanding of the possible meanings of irrelevance claims.

The notion of irrelevance has appeared in many contexts in research in AI and related fields. However, most of the time researchers use the term informally. Formal analyses of irrelevance have been discussed by philosophers as early as 1921 (Keynes [47]), 1950 (Carnap [16]) and 1978 (Gärdenfors [34]). The main thrust of these analyses was to try to capture our common-sense notions of irrelevance by a formal definition. Most of the work focuses on formulating properties of the notion of irrelevance and finding definitions that satisfy those properties. Consequently, the work has not been concerned with how to *use* irrelevance for speeding up inference or how to design algorithms for detecting irrelevance.

Within AI, the notion of irrelevance was investigated in the context of probabilistic reasoning [21,23,70] and used there to control inference in Bayesian belief networks. In the context of logical knowledge bases, Subramanian [88], and more recently Lakemeyer [53], investigated several formal definitions of irrelevance. However, the issues of automatically deriving irrelevance claims and the utility of irrelevance reasoning were left largely open, and consequently relevance reasoning has not been applied in any effective way.

This article presents a framework in which various definitions of irrelevance can be studied. We present efficient algorithms for automatically deriving irrelevance claims, and we describe the results of experiments that validate the utility of relevance reasoning. In particular, we make the following contributions.

- We present a proof-theoretic framework for analyzing definitions of irrelevance, yielding a space of possible definitions for the notion. We describe how properties of irrelevance claims vary as we move in the space. The framework encompasses and sheds new light on previous definitions of irrelevance. We show that the framework makes the necessary distinctions between the definitions needed to address the problem of automatically deriving irrelevance claims and the utility of removing irrelevant facts.
- We consider the problem of automatically deriving irrelevance claims for Horn-rule knowledge bases and several extensions, and present a novel tool, called the *query-tree*, for that purpose. We identify the important class of *strong-irrelevance* claims and show that the query-tree provides a sound and complete inference procedure for such claims. Strong irrelevance claims also have the property that removing strongly irrelevant facts is guaranteed not to slow down an inference engine (and often to speed it up significantly). The query-tree has two properties that make it especially useful in practice. First, irrelevance claims are derived by inspecting only

a small part of the knowledge base (e.g., inspecting the rules in the knowledge base and not the ground facts). Second, in deriving irrelevance claims, the query-tree considers the semantics of interpreted predicates appearing in the knowledge base (e.g., order predicates, sort predicates, etc.). Since interpreted predicates play a key role in many applications, this is an important feature of the query-tree. The query-tree can also be viewed as a tool for partial evaluation of constraint logic programs [15,67,85], but is distinguished from previous work in that area in that it provides completeness also in the presence of recursive rules and interpreted predicates.

- We describe experimental results that show that significant speedups (often orders of magnitude) are obtained by employing the query-tree in inference. The query-tree is used in two ways. First, it is used to determine which facts are irrelevant to a query. Based on that determination, we create specialized database indices that see only the ground facts that are (possibly) relevant to a class of queries. Given a query from the given class, we can then use these indices for fetching ground facts during inference, thereby significantly speeding up inference. The second use of the query-tree is based on the observation that it tells us exactly which *sequences* of rule applications and database lookups can yield solutions to the query. Therefore, we use the query-tree to guide our inference engine to follow only such sequences. The experiments show that the cost of building the query-tree and preprocessing the KB (to create the specialized indices) is negligible compared to the savings achieved. Furthermore, the savings grow as the size of the knowledge base grows, indicating that our methods will scale up. It should be noted that the savings achieved by employing the query-tree are orthogonal to methods that develop optimal strategies for searching the space (e.g., rule and subgoal ordering [42,84]) and methods that use run-time bindings (combined with tabulation) to prune the search [8,9,93]. Therefore, the experimental validation presented here does not follow from experiments validating these other methods.

Organization of the paper

The paper is organized as follows. Section 2 describes the space of definitions of irrelevance and compares properties of various definitions. Section 3 considers the problem of automatically deriving irrelevance claims, and Section 4 describes the query-tree algorithm. Section 5 describes how the query-tree can be used to speed up inferences and the experimental results. Section 6 briefly describes other applications of our framework, such as automatic creation of abstractions, database query optimization and global information systems. Section 7 discusses related work, and Section 8 contains concluding remarks.

2. Analyzing irrelevance

In this section, we consider what it means for a fact to be irrelevant to a query. We begin by introducing the terminology used throughout the paper.

2.1. Preliminaries

In our discussion, we assume that the theory of the domain is represented by a knowledge base Δ of facts which are closed formulas in first-order predicate calculus. We use lower-case letters for predicate names, upper-case letters for variables and bars to denote tuples of variables (e.g. $\bar{X}$). We assume that the inference mechanism employs a set of sound inference rules. A derivation D of a closed formula ψ from Δ is a sequence of closed formulas, $\alpha_1, \ldots, \alpha_n$, such that $\alpha_n = \psi$, and for each i $(1 \leqslant i \leqslant n)$, either $\alpha_i \in \Delta$, α_i is a logical axiom, or α_i is the result of applying an inference rule to some elements $\alpha_{i_1}, \ldots, \alpha_{i_l}$ that appear prior to α_i. The formulas $\alpha_{i_1}, \ldots, \alpha_{i_l}$ are said to be *subgoals* of α_i. The set of formulas in D that do not have any subgoals is called the *base* of the derivation, denoted by $Base(D)$. The set $Base(D)$ represents a "support set" for ψ in the knowledge base. We consider only derivations in which every α_i is connected to ψ through the subgoal relation.

A *query* is represented by a formula $\psi(\bar{X})$, where $\bar{X}$ is the tuple of free variables of ψ; we sometimes omit $\bar{X}$ and denote the query $\psi(\bar{X})$ just as ψ. When $\psi(\bar{X})$ is a closed formula (i.e., has no free variables and, hence, $\bar{X}$ is the empty tuple), the answer is **true** if there is a derivation of $\psi(\bar{X})$ from Δ; otherwise, the answer is **false**. When $\psi(\bar{X})$ contains free variables, the answer is the set of assignments from the free variables into the constants mentioned in Δ, such that the resulting closed formulas are derivable from Δ;[3] in this case, the phrase "a derivation of the query" refers to a set containing one derivation for each answer. The query ψ may have several derivations from a given knowledge base, and we denote the set of those derivations by $\mathcal{D}^{\psi}(\Delta)$ (when ψ has free variables, then $\mathcal{D}^{\psi}(\Delta)$ is a set of sets of derivations, where each set $\mathcal{D} \in \mathcal{D}^{\psi}(\Delta)$ contains one derivation for each answer of ψ). By a slight abuse of notation, we sometimes denote $\mathcal{D}^{\psi}(\Delta)$ simply as $\mathcal{D}^{\psi}$.

2.2. Defining irrelevance

Our goal is to express, reason with and automatically derive irrelevance claims, i.e., claims of the form "*Φ is irrelevant to the query ψ with respect to the knowledge base Δ*". To do so, it is essential that we give such claims a formal definition. Φ is called the *subject* of the irrelevance claim. In this paper, we consider the case in which Φ is a fact or set of facts. Other irrelevance subjects, such as objects, relation arguments and refinements of predicates are discussed briefly in Section 6 and in more detail in [56].

Broadly, we can take two possible approaches to analyzing irrelevance. The first approach, which has been pursued by several philosophers (e.g., [16,34,47]), is to try to capture our common-sense notion of irrelevance with a formal definition. In that approach, we would consider a formal definition of irrelevance and check whether it

[3] Alternative definitions are also possible. For example, in the case of a closed-formula query, one might return **unknown** if neither ϕ nor $\neg\phi$ is derivable; in the case of a query with free variables, one might return just the *first* variable binding that satisfies the query. However, these distinctions do not affect our discussion. Note that since the number of possible assignments for the free variables is finite, the set of answers to a query is finite.

satisfies properties that we consider natural for our intuitive notion of irrelevance. In this article, we pursue a second approach that focuses on analyzing how irrelevance arises in problem solving (and specifically in inference). In this approach, we are most interested in properties of definitions of irrelevance that are informative in designing inference methods that utilize irrelevance. For example, we are interested in whether irrelevance claims can be automatically derived, how the claims change when the KB changes, and the utility of removing irrelevant facts. The following example illustrates these properties.

Example 1. Consider the following knowledge base Δ_0, describing students and the courses in which they can serve as teaching assistants.

$r_1 : attendClass(X, Y) \Rightarrow pass(X, Y).$
$r_2 : passExam(X, Y) \Rightarrow pass(X, Y).$
$r_3 : pass(X, Y) \wedge tookGradCourse(X) \Rightarrow canTA(X, Y).$
$r_4 : pass(X, Y) \wedge (Y \geqslant 300) \Rightarrow tookGradCourse(X).$
$g_1 : attendClass(Fred, 101).$
$g_2 : passExam(Fred, 101).$
$g_3 : passExam(Fred, 201).$
$g_4 : passExam(Fred, 301).$

Suppose we want to find all the 100-level classes in which Fred is qualified to be a teaching assistant, i.e., the query is $q(Y) = canTA(Fred, Y) \wedge (Y < 200)$.

For the given ground facts, the only answer to the query $q(Y)$ is $Y = 101$ and, so, we shall use q to denote the query $canTA(Fred, 101)$. The query q can be derived either by using g_1, g_4 and the rules r_1, r_3 and r_4, or by using g_2, g_4 and the rules r_2, r_3 and r_4. Hence, each of the ground atoms g_1, g_2 and g_3 is irrelevant to the query when considered alone, because for each one, the answer to the query can be derived without it. However, there are differences between these irrelevance claims. Specifically, g_3 is not used in any derivation of the query. More generally, facts of the form $passExam(Fred, Y)$, where $200 \leqslant Y < 300$, can be added to or retracted from the KB without changing the answers to the query. As for g_1 and g_2, even though the query can be derived without either one of them, it cannot be derived without both of them and, therefore, we cannot remove both. To summarize, some facts are irrelevant because they are not part of any derivation of an answer to the query, while some other facts are irrelevant because all answers can still be derived without those facts.

There are, however, other variants of relevance and irrelevance. For one, even though the ground atom $canTA(Fred, 301)$ is not part of any derivation of an answer to q, we may still consider it relevant to the query, because it is always entailed by the facts and rules used in any derivation of q. To see another type of irrelevance, consider the query $canTA(Fred, 301)$. The atom $passExam(Fred, 302)$ can be part of a derivation of an answer to this query (if it were in the KB), but such a derivation would not be *minimal*, in the sense that the set of ground atoms that it uses from the KB is not minimal (i.e., since g_4 must be used, along with the rules r_2, r_3 and r_4, there is no need to use any other ground fact from the KB). Hence, the atom $passExam(Fred, 302)$ may be deemed irrelevant.

Finally, rules may also be irrelevant. For example, if we add the rule

$$passExam(X,Y) \wedge (Y \geqslant 300) \quad \Rightarrow \quad canTA(X,Y),$$

it would be considered irrelevant, since answers can be derived without it. However, for some inference mechanisms, it may be the case that this rule will speed up inference, since fewer rule applications may be needed to derive answers if this rule is used. □

Clearly, the two approaches we described to analyzing irrelevance are not independent of each other. On the one hand, the analysis of irrelevance that we consider is inspired by our common-sense notion of the concept, and the definitions we examine mirror it in various ways. On the other hand, given a formalization of the common-sense notion of irrelevance, analyzing it in our framework will provide a way of using it for speeding up inference. However, it should be emphasized that the approach we have taken is intended to be evaluated on its usefulness for speeding up inference, not on how well it captures intuitive notions of irrelevance.

As illustrated by the above example, there is no *single* best definition of irrelevance. For example, we can define a formula ϕ to be irrelevant to ψ if there is some derivation of ψ that does not contain ϕ, or alternatively, we can require that *no* derivation of ψ contains ϕ. Therefore, we describe a space of possible definitions of irrelevance, and investigate the properties of various definitions within this space. Our space is based on a proof-theoretic analysis of irrelevance, i.e., on investigating the ways in which formulas can participate in derivations of the query. In contrast, Subramanian [88] described a *meta-theoretic* account of irrelevance. Her framework considers only the formulas in the KB, not the possible derivations of the query. Consequently, we are able to make finer distinctions than those made in Subramanian's framework. A more detailed comparison with Subramanian's work appears in Section 7.

2.3. A space of definitions

In this section, we assume that the given query ψ is a closed formula. Extending the definitions of this section to a query with free variables is straightforward.

Definitions in the space vary along two axes. In the first axis, we consider different ways of defining *derivation irrelevance*, i.e., irrelevance of a subject ϕ to a *single* derivation D of the query ψ. Derivation irrelevance is given by defining a binary predicate $DI(\phi, D)$. The following are a few examples of how DI can be defined:

- $DI_1(\phi, D)$ if $\phi \notin Base(D)$.
- $DI_2(\phi, D)$ if $\phi \notin D$.
- $DI_3(\phi, D)$ if $Base(D) \not\models \phi$.
- $DI_4(\phi, D)$ if $Base(D) \not\models \phi$ and $Base(D) \not\models \neg\phi$

Definition DI_1 requires that ϕ not be in the support set of the derivation D. Definition DI_2 is stronger and requires that ϕ not be anywhere in D. Definition DI_3 is even stronger and requires that ϕ not be a logical consequence of the formulas in $Base(D)$, and DI_4 requires that $\neg\phi$ not be a logical consequence either. The relationship between these definitions of DI can, therefore, be summarized as follows:

Proposition 2. $DI_4(\phi, D) \Rightarrow DI_3(\phi, D) \Rightarrow DI_2(\phi, D) \Rightarrow DI_1(\phi, D)$.

Requiring that $DI(\phi, D)$ holds for all possible derivations of the query may be too restrictive. Therefore, in the second axis, we consider different subsets of the derivations of the query for which we require $DI(\phi, D)$ to hold. Formally, given the set $\mathcal{D}^{\psi}(\Delta)$ of all possible derivations of ψ from Δ, we consider a subset $\mathcal{D}_0^{\psi}(\Delta)$ of $\mathcal{D}^{\psi}(\Delta)$ (which may be $\mathcal{D}^{\psi}(\Delta)$ itself), and require $DI(\phi, D)$ to hold only for derivations in $\mathcal{D}_0^{\psi}(\Delta)$. For example, we can require $DI(\phi, D)$ to hold only for the set of *minimal* derivations of ψ (in Section 3, we consider several definitions of minimality for a derivation). As another example, we can consider only the set of derivations bounded by some resource constraint.

Given a choice for DI and $\mathcal{D}_0^{\psi}(\Delta)$, we give two definitions of irrelevance, depending on whether DI is required to hold for *all* derivations in $\mathcal{D}_0^{\psi}(\Delta)$ or for *some* derivation in $\mathcal{D}_0^{\psi}(\Delta)$.[4] Formally, a definition of irrelevance in our space is given as follows:

Definition 3. Suppose we are given:

(1) A knowledge base Δ (i.e., a set of formulas).
(2) A closed formula ϕ (the subject of irrelevance).
(3) A closed-formula ψ (the query).
(4) A predicate $DI(\tau, D)$ specifying when a formula τ is irrelevant to a derivation D of the query.
(5) A subset $\mathcal{D}_0^{\psi}(\Delta)$ of the set $\mathcal{D}^{\psi}(\Delta)$ (of all possible derivations of ψ from Δ). By a slight abuse of notation, we usually denote $\mathcal{D}_0^{\psi}(\Delta)$ as $\mathcal{D}_0(\Delta)$ or simply as $\mathcal{D}_0$.

The formula ϕ is said to be *weakly irrelevant* to the query ψ with respect to Δ, DI and $\mathcal{D}_0$, denoted by $WI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$, if $DI(\phi, D)$ holds for some $D \in \mathcal{D}_0(\Delta)$.

The formula ϕ is said to be *strongly irrelevant* to the query ψ with respect to Δ, DI and $\mathcal{D}_0$, denoted by $SI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$, if $DI(\phi, D)$ holds for every $D \in \mathcal{D}_0(\Delta)$.

If $\mathcal{D}^{\psi}(\Delta)$ is empty (i.e., ψ is not derivable from Δ), the formula ϕ is both weakly and strongly irrelevant to ψ.

In our discussion, we want to refer to irrelevance of a *set* of formulas. Formally, we define irrelevance of a set of formulas by extending the definition of DI:

Definition 4. If Φ is a set of formulas, $DI(\Phi, D)$ holds if $DI(\phi_i, D)$ holds for every $\phi_i \in \Phi$.

The definitions of strong and weak irrelevance remain unchanged. It will also be useful to state irrelevance claims that hold for a set of knowledge bases. For example, in the context of Horn-rule knowledge bases, we may want to know whether a rule is irrelevant with respect to all the knowledge bases that have the same rules (but may

[4] We can also consider other ways of quantifying over the set $\mathcal{D}_0^{\psi}(\Delta)$, such as requiring that $DI(\phi, D)$ holds for some percent of the derivations in $\mathcal{D}_0^{\psi}(\Delta)$. In fact, different ways of quantifying over $\mathcal{D}_0^{\psi}(\Delta)$ could be considered a third axis in the space. Here, we consider only universal and existential quantification.

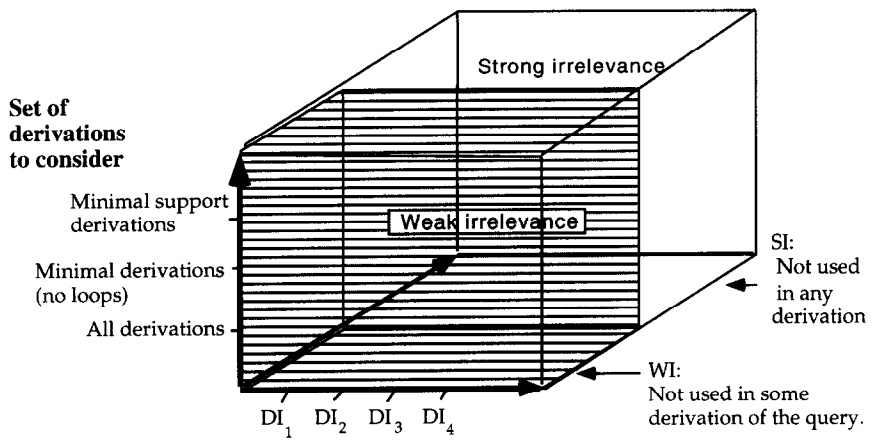

Fig. 1. A space of definitions of irrelevance. The first axis consists of different definitions of derivation irrelevance. The second axis consists of the set of derivations considered. Weak irrelevance and strong irrelevance differ in the way we quantify DI_i over the derivations chosen in the second axis.

have different ground atoms). We extend the definitions to sets of knowledge bases as follows:

Definition 5. Let Σ be a set of knowledge bases. We say that ϕ is weakly irrelevant to ψ with respect to Σ, denoted by $WI(\phi, \psi, \Sigma, DI, \mathcal{D}_0)$, if ϕ is weakly irrelevant to ψ with respect to every KB in Σ; i.e., if $WI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$ holds for every $\Delta \in \Sigma$ (note that $\mathcal{D}_0$ is a function that given a KB $\Delta \in \Sigma$ returns a subset of the derivations $\mathcal{D}^{\psi}(\Delta)$). The definition for strong irrelevance is extended likewise.

The space of definitions is summarized in Fig. 1. Going back to Example 1, we can see different kinds of irrelevance claims. The atom g_1 is weakly irrelevant to the query $q = canTA(Fred, 101)$, since there is a derivation of q that does not use g_1 (i.e., it uses g_2 instead). Consequently, $WI(g_1, q, \Delta_0, DI_2, \mathcal{D}^q(\Delta))$ holds. Similarly for the atom g_2. The atom g_3 is strongly irrelevant to q, because none of the derivations of q uses it. Consequently, $SI(g_3, q, \Delta_0, DI_2, \mathcal{D}^q(\Delta))$ holds.

The atom $q_1 = canTA(Fred, 301)$ is strongly irrelevant to q if we consider derivation irrelevance based on DI_2. However, if we consider derivation irrelevance based on DI_3, the atom q_1 is not strongly irrelevant to q, since the formulas used to derive q can also be used to derive q_1.

Finally, suppose that q_1 is the query. If we consider the set of all derivations of q_1, then the atom $passExam(Fred, 302)$ would not be strongly irrelevant to the query (had it been in the KB), since it can be used in a derivation of q_1 (to derive $tookGradCourse(Fred)$). However, if we consider only derivations in which $Base(D)$ is minimal (i.e., there is

no subset of $Base(D)$ that is enough to derive the query), then $passExam(Fred, 302)$ would not be part of any derivation of q_1 and, therefore, would be strongly irrelevant to the query q_1.

2.4. Properties of definitions in the space

When investigating irrelevance claims, several potential properties are of interest; the actual properties, however, vary quite a bit as we move from one definition in the space to another. One key property is whether irrelevance claims can be derived automatically. Sections 3 and 4 discuss this property in the context of several definitions of irrelevance. Below we summarize and explain the significance of some other properties of irrelevance claims that are of practical interest in speeding up inference. The proofs are given in Appendix A.

Theorem 6. *Properties* A0–A8 (*listed below*) *hold, given the following notation.*

- ψ *denotes a query.*
- ϕ, ϕ_1, *and* ϕ_2 *denote formulas.*
- Φ, Φ_1, *and* Φ_2 *denote sets of formulas.*
- Δ *denotes a knowledge base.*
- Σ, Σ_1, *and* Σ_2 *denote sets of knowledge bases.*
- $\mathcal{D}_0$, $\mathcal{D}_1$, *and* $\mathcal{D}_2$ *denote functions that given a KB* Δ *and a query* ψ *return a subset of* $\mathcal{D}^{\psi}(\Delta)$.
- DI, DI', *and* DI'' *denote definitions of derivation irrelevance.*
- DI_1, DI_2, DI_3, *and* DI_4 *are the definitions of derivation irrelevance from the beginning of Section* 2.3.

A0. *If* $WI(\phi, \psi, \Delta, DI_1, \mathcal{D}^{\psi}(\Delta))$ *holds, then the formula* ϕ *can be removed from* Δ *without changing the answer to* ψ. *That is, if* $\phi \in \Delta$ *and* $\Delta - \phi$ *denotes the set of formulas in* Δ *except for* ϕ, *then*

$$\Delta \vdash \psi \;\Leftrightarrow\; \Delta - \phi \vdash \psi$$

A1. *If* $DI'(\Phi, D) \Rightarrow DI''(\Phi, D)$ *for all derivations* $D \in \mathcal{D}_0$, *then*

$$SI(\Phi, \psi, \Sigma, DI', \mathcal{D}_0) \;\Rightarrow\; SI(\Phi, \psi, \Sigma, DI'', \mathcal{D}_0)$$
$$WI(\Phi, \psi, \Sigma, DI', \mathcal{D}_0) \;\Rightarrow\; WI(\Phi, \psi, \Sigma, DI'', \mathcal{D}_0)$$

A2. *If* $\mathcal{D}_1(\Delta) \subset \mathcal{D}_2(\Delta)$ *for all knowledge bases* $\Delta \in \Sigma$, *then*

$$SI(\Phi, \psi, \Sigma, DI, \mathcal{D}_2) \;\Rightarrow\; SI(\Phi, \psi, \Sigma, DI, \mathcal{D}_1)$$
$$WI(\Phi, \psi, \Sigma, DI, \mathcal{D}_1) \;\Rightarrow\; WI(\Phi, \psi, \Sigma, DI, \mathcal{D}_2)$$

A3. *The following is always true.*

$$SI(\Phi, \psi, \Sigma, DI, \mathcal{D}_0) \;\Rightarrow\; WI(\Phi, \psi, \Sigma, DI, \mathcal{D}_0)$$

A4. *If* $\Sigma_1 \subseteq \Sigma_2$, *then*

$$SI(\Phi, \psi, \Sigma_2, DI, \mathcal{D}_0) \Rightarrow SI(\Phi, \psi, \Sigma_1, DI, \mathcal{D}_0)$$
$$WI(\Phi, \psi, \Sigma_2, DI, \mathcal{D}_0) \Rightarrow WI(\Phi, \psi, \Sigma_1, DI, \mathcal{D}_0)$$

A5. *If the inference rules are complete,* $\phi_1 \equiv \phi_2$, *and DI is either* DI_3 *or* DI_4, *then*

$$WI(\phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow WI(\phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$$
$$SI(\phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow SI(\phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$$

A6. *If the inference rules are complete, then*

$$WI(\phi, \psi, \Sigma, DI_4, \mathcal{D}_0) \Rightarrow WI(\neg\phi, \psi, \Sigma, DI_4, \mathcal{D}_0)$$
$$SI(\phi, \psi, \Sigma, DI_4, \mathcal{D}_0) \Rightarrow SI(\neg\phi, \psi, \Sigma, DI_4, \mathcal{D}_0)$$

A7. *If* $DI(\Phi_1, D) \wedge DI(\Phi_2, D) \Rightarrow DI(\Phi_1 \cup \Phi_2, D)$ *for all derivations* $D \in \mathcal{D}_0$, *then*

$$SI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge SI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow SI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$$
$$SI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge WI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow WI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$$

A8. *If* $\Delta \vdash \tau$ *and* Δ *is consistent, then*

$$SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}^{\psi}(\Delta)) \Leftrightarrow SI(\Phi, \psi, \Delta \cup \tau, DI_2, \mathcal{D}^{\psi}(\Delta))$$

Property A0 guarantees that we can remove an irrelevant fact without changing the answer to the query. Properties A1–A4 show how the relative strength of irrelevance claims is affected as a result of changing some of the parameters of these claims.

In general, irrelevance claims in our space are not preserved under equivalence of the subject, query, or knowledge base. Although preservation under equivalence has been considered natural for a common-sense notion of irrelevance [34], we believe that it is not necessarily appropriate when analyzing irrelevance for the purpose of speeding up inferences. Property A5 identifies some cases in which irrelevance claims are preserved under equivalence. Property A6 is similar in the sense that it shows when irrelevance claims are closed under negation.

Property A7 shows when irrelevance claims can be added up. This property is important when a system needs to determine whether it can use all the irrelevance claims it has or whether using certain ones will falsify others. The same property does not hold for weak irrelevance. In general, adding new formulas to the knowledge base may cause a formula that was irrelevant to become relevant or vice versa. In particular, weak irrelevance claims can change even when the added formulas are logical consequences of the knowledge base. In contrast, as property A8 shows, strong irrelevance claims do not change when we reason with existing knowledge. For weak irrelevance the implication from right to left in A8 does not hold.

Utility of removing irrelevant facts

For any of the definitions in our space, an irrelevant fact can be removed from the knowledge base without affecting the answer to the query (property A0). However, the

utility of removing an irrelevant fact is a more subtle issue. Removing a fact that is only weakly irrelevant may not speed up inference. In fact, explanation based learning systems [69] do exactly the opposite; that is, they add redundant rules (which, in our framework, would be considered weakly irrelevant). The utility of adding such rules is a subject of ongoing research (e.g., [28, 30, 44, 68]).

For strong irrelevance, savings are guaranteed for many cases. For example, if the claim $SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}^{\psi})$ holds (i.e., all derivations of the query are considered), then deriving ψ from $\Delta - \Phi$ costs no more than deriving it from Δ. This property also holds if we consider a set of derivations $\mathcal{D}_0^{\psi}(\Delta)$, such that the inference engine is always guaranteed to find one of the derivations in $\mathcal{D}_0^{\psi}(\Delta)$ before it finds others. In subsequent sections, we will show that in some situations it is possible to efficiently determine the facts that are strongly irrelevant to a query and that removing such facts yields significant savings in practice. These savings come from several sources:

- Removing irrelevant facts prunes branches of the search space.
- Much of the cost of reasoning in a large knowledge base is in doing database lookups. Removing a large number of irrelevant ground facts at the outset significantly reduces the cost of each lookup operation.
- If updates that involve only irrelevant facts are made to the KB, then the answer to a query need not be recomputed.

2.5. *Encompassing previous definitions in our space*

An important contribution of our space of definitions is that it encompasses definitions of irrelevance previously discussed in the literature and, therefore, enables us to make comparisons among them. We mention several of these comparisons below.

Subramanian investigates several definitions of irrelevance. In our framework, all these definitions are instances of weak irrelevance. The main definition investigated in [88] is the following:

Definition 7. Let ϕ be a fact, ψ be a query, and Δ be a knowledge base. The fact ϕ is said to be irrelevant to ψ, denoted by $WI_1(\phi, \psi, \Delta)$, if there exists a subset Δ_1 of Δ such that $\Delta_1 \not\models \phi$ and $\Delta_1 \models \psi$.

This definition can be couched in our framework as follows:

Observation 8. *For a complete set of inference rules S,*

$$WI_1(\phi, \psi, \Delta) \quad \equiv \quad WI(\phi, \psi, \Delta, DI_3, \mathcal{D}^{\psi})$$

Proof. Suppose $WI_1(\phi, \psi, \Delta)$ holds. Therefore, there is some subset Δ_1 of Δ such that $\Delta_1 \models \psi$, $\Delta_1 \not\models \phi$, and there is some derivation D of ψ from Δ_1. Clearly, $Base(D) \not\models \phi$ and, consequently, $WI(\phi, \psi, \Delta, DI_3, \mathcal{D}^{\psi})$ holds.

Conversely, assume that $WI(\phi, \psi, \Delta, DI_3, \mathcal{D}^{\psi})$ holds. Consequently, there is some derivation D of ψ from Δ such that $Base(D) \not\models \phi$. The KB consisting of $Base(D)$ is a subset of Δ and does not entail ϕ. Consequently, $WI_1(\phi, \psi, \Delta)$ holds. □

A variation of this definition, which is described in [87], can be formulated in our space as $WI(\phi, \psi, \Delta, DI_4, \mathcal{D}^{\psi})$. Couching Subramanian's definitions in our framework shows how other definitions in the space overcome some of the limitations of her definitions. In particular, our space identifies irrelevance definitions such that removing irrelevant facts leads to speeding up inferences and for which some forms of monotonicity and adding-up hold (property A7).

An important contribution of our framework is that it sheds light on the problem of detecting when a query is independent of an update [10,26]. In [63], we show that the problem of independence of a query from a deletion update can be equivalently formulated as the problem of detecting weak irrelevance in our framework (specifically, detecting $WI(\phi, \psi, \Delta, DI_1, \mathcal{D}^{\psi})$). However, a close inspection of previous work on this problem revealed that the algorithms proposed for detecting independence were based on detecting strong irrelevance, which is a more restricted condition. Identifying this relationship has led to the development of novel algorithms for detecting independence, based on algorithms for determining strong and weak irrelevance (see [63] for details).[5]

A definition of irrelevance is described by Srivastava and Ramakrishnan in [86]. Their definition is equivalent to strong irrelevance when DI_2 is used for derivation irrelevance and is applied to the set of all derivations of the query; that is, their definition is equivalent to $SI(\phi, \psi, \Delta, DI_2, \mathcal{D}^{\psi})$. The notion of irrelevance discussed by Levy and Sagiv in [61] can be couched in our framework as $SI(\phi, \psi, \Delta, DI_2, \mathcal{D}_m)$, where $\mathcal{D}_m$ is the set of all minimal derivations of the query.

Finally, several resolution strategies are based on removing irrelevant clauses. For example, when using refutation resolution, clauses containing pure literals[6] can be shown to be strongly irrelevant (with respect to DI_1 and $\mathcal{D}^{\psi}$) and, therefore, can be removed. Tautologies can be shown to be weakly irrelevant (with respect to DI_1 and $\mathcal{D}^{\psi}$) and, therefore, are removed by the *tautology elimination* strategy [39].

3. Automatically deriving irrelevance claims

A key question that we address in this article is how (and under what conditions) irrelevance claims can be derived automatically. Specifically, we are interested in two problems. First, given a knowledge base, a query, and a specific definition of irrelevance, we want to find automatically which facts in the knowledge base are irrelevant to the query. Second, given an irrelevance claim, we want to derive other irrelevance claims that logically follow. We focus on solving the first problem. Later, we show how results pertaining to the first problem can be used to solve the second.

We examine the question of automatically deriving irrelevance claims for Horn knowledge bases that consist of a set of Horn rules $\mathcal{P}$ and a set of ground atomic facts

[5] As shown in Section A.2, there is a close connection between weak irrelevance and the problem of *query containment*. Using this relationship we obtain, in addition to the characterization of independence of deletion updates, also a complete characterization of independence of *insertion* updates.

[6] A literal is pure if and only if it has no instance that is complementary to an instance of another literal in the knowledge base [39].

G. We distinguish two sets of predicates in the KB: *base predicates* (often called EDB predicates) and *derived predicates* (IDB predicates). The base predicates are those that appear in the ground facts of G. The derived predicates are those that appear in the consequents of the rules. For syntactic convenience, we assume that base predicates do not appear in the consequents of rules. We assume that the rules in $\mathcal{P}$ are safe, i.e., any variable that appears in the consequent appears also in the antecedent.

Determining that a fact is irrelevant to a query requires that we establish properties of some (or all) possible derivations of the query. This is usually impractical and, in particular, may be more expensive than answering the query; hence, it defeats the original goal of relevance reasoning. In order for our algorithms to be of practical interest, we consider the problem of deriving irrelevance claims by examining only a (preferably small and stable) *portion* of the knowledge base.

In many applications involving Horn-rule knowledge bases, the bulk of the KB consists of ground facts and, moreover, the ground facts are much more prone to frequent changes than the rules of the KB. Therefore, we address the irrelevance problem for a set of knowledge bases that differ only on ground facts. Specifically, we address the following question. Let $\mathcal{P}$ be a set of rules, and let $\Sigma_{\mathcal{P}}$ be the set of knowledge bases of the form $\mathcal{P} \cup G$, where G is a set of ground atomic facts for the base predicates. The subjects of irrelevance we consider are either a rule in the knowledge base or a set of atomic ground facts, and the query ψ will be an atom, possibly with free variables. Note that conjunctive queries and disjunctions of conjunctive queries can be expressed by simply adding the appropriate predicates and the corresponding rules to the knowledge base. The question we address is whether ϕ is irrelevant to a query ψ with respect to the set of knowledge bases $\Sigma_{\mathcal{P}}$.

It should be emphasized that distinguishing between rules and ground facts is only one way to distinguish between *known* and *unknown* parts of a knowledge base. For example, some base predicates may have small and stable extensions and, therefore, one may want to include those extensions in every KB of $\Sigma_{\mathcal{P}}$. Alternatively, some derived predicates may appear in the consequents of numerous rules and, so, one may decide not to include those rules in the KBs of $\Sigma_{\mathcal{P}}$ (instead, those predicates would be treated as base predicates). The results and algorithms we describe in this article extend straightforwardly to such cases.

3.1. Interpreted predicates

A final key point about our analysis is the treatment of interpreted predicates. Many of the interactions between rules in a knowledge base can be deduced by considering the semantics of interpreted predicates that appear in them, such as order predicates ($=$, $\neq$, $\leqslant$, $<$) or sort predicates. For instance, in Example 1, g_3 was deemed strongly irrelevant by considering the semantics of the predicate $\leqslant$. The extensions of the interpreted predicates can be viewed as part of the ground facts in the knowledge base. However, enforcing their semantics in our analysis entails that we derive irrelevance claims that hold for any knowledge base of the form $\mathcal{P} \cup G$ such that G is a set of ground facts that satisfies the semantics of the interpreted predicates.

Formally, we distinguish a subset of the base predicates in the knowledge base as *constraint predicates*. A *constraint formula* is a formula in some language $\mathcal{L}$ for expressing constraints that involves only literals of constraint predicates and logical connectives (i.e., $\wedge$, $\vee$, $\neg$). For example the formula $c_1 = even(X) \wedge (X > 100)$ is a constraint if the predicate *even* is a sort predicate whose extension is the even numbers. A formula c in the language $\mathcal{L}$, with free variables $X_1, \ldots, X_n$, can also be viewed as describing a (possibly infinite) relation $R_c(X_1, \ldots, X_n)$, which is the set of all tuples satisfying the constraints expressed by c. For example, R_{c_1} denotes the unary relation containing all the even integers greater than 100. To denote the variables to which a constraint formula applies, we use the following conventions. The *standard form* of a constraint formula c will be that its variables are $X_1, \ldots, X_n$ (note that X_i corresponds to the ith argument of R_c). The expression $c(\bar{Y})$, where $\bar{Y} = Y_1, \ldots, Y_n$ will denote the formula c after substituting Y_i for X_i, for $1 \leqslant i \leqslant n$.

We assume the following properties of our constraint predicates:

- *Closure*: Given formulas c_1 and c_2, it is possible to effectively construct formulas that express:
  - The join of R_{c_1} and R_{c_2}.
  - A projection of R_{c_1} (i.e., a relation consisting of only a subset of the arguments of R_{c_1}).[7]
  - A selection $\sigma_{i=j}R_{c_1}$, where i and j are some columns of R_{c_1} (i.e., a relation consisting of only tuples of R_{c_1} in which columns i and j are equal).
  - A selection $\sigma_{i=c}R_{c_1}$, where i is some column of R_{c_1} and c is a constant in the language $\mathcal{L}$.
- *Equivalence*: Given formulas c_1 and c_2, it is decidable whether $R_{c_1} = R_{c_2}$.
- *Satisfiability*: Given a formula c, it is decidable whether R_c is nonempty.[8]
- *Finiteness*: Let C be a finite set of constants in the language $\mathcal{L}$, and let $\mathcal{F}$ be a finite set of formulas in the language $\mathcal{L}$ that have at most n free variables (for some fixed n) and only constants from C. Then applications of the operators (discussed in the Closure property) to formulas in $\mathcal{F}$ may create only a finite number of nonequivalent formulas over n (or fewer) free variables.

The Closure condition guarantees that we can perform the basic manipulations on the relations denoted by formulas (i.e., conjunction, projection, selection) within our constraint language. The second and third conditions guarantee that we can identify two equivalent constraints. The Finiteness constraint guarantees that with a given set of constants and variables we only create a finite number of nonequivalent constraints (we later discuss the case in which the Finiteness condition does not hold). The procedures needed to compute the closure operations, equivalence, and satisfiability are assumed to be given. Typically, these procedures are efficient. For example, for conjunctive order constraints over dense domains, testing equivalence is cubic in the number of variables

[7] Note that this property corresponds to closure under existential quantification of the corresponding logical formula.

[8] Note that if we have a formula FALSE in our language denoting the empty relation, then the Satisfiability property will follow from the Equivalence property.

[89] (but over integers or when disjunctive constraints are allowd the problem is NP-hard; see [25] for a discussion of some of these issues).

The properties we require are satisfied by a wide class of interpreted predicates. For example:

- *Order constraints*: The language consisting of the predicates $=$, $\neq$, $\leqslant$, $<$, and the connectives $\wedge$ and $\vee$.
- *Sort constraints*: A constraint language based on a finite sort hierarchy, and the connectives $\wedge$, $\vee$, and $\neg$. In particular, any description logic with decidable subsumption can be viewed as a sort language (e.g. [6, 12, 72]).
- *Finite, given relation*: Often, a given relation that is relatively small and stable can be best viewed as a constraint. Any given finite relation satisfies the properties that we require.

Hereafter, a *constraint* will refer to a constraint formula in some constraint language $\mathcal{L}$.

3.2. Decidability of irrelevance

In our analysis, we consider definitions based on DI_2 (because in Horn-rule KBs, DI_1 and DI_2 are equivalent when the subject of the irrelevance claim is a rule or a base fact, and when the subject is a fact of a derived predicate, DI_1 is trivially true). We consider several sets of possible derivations. Recall that a derivation is a sequence $\alpha_1, \ldots, \alpha_n$, and it can be viewed as a tree formed by the subgoal relation. In addition to the set of all derivations of the query, $\mathcal{D}^{\psi}(\Delta)$, we consider two other choices of sets of derivations based on the following definitions of derivation minimality:

- A derivation D is *minimal* if does not have two identical facts α_i and α_j such that α_i is an ancestor of α_j.
- A derivation D is a *minimal support* derivation if there is no other derivation of the query, D', such that $Base(D') \subseteq Base(D)$ and $Base(D') \neq Base(D)$.

A summary of the decidability results pertaining to deriving irrelevance claims is shown in Table 1. The table also considers cases that go beyond Horn rules, where the rules contain negated literals in their antecedents. The important observation from the table is that weak irrelevance becomes undecidable whenever the rules contain recursion (shown in Lemma A.1 in Appendix A). In contrast, strong irrelevance is (efficiently) decidable for a larger class of languages, including recursion and some forms of negation. Allowing arbitrary function symbols in the Horn rules leads to undecidability of strong irrelevance, but only if there is recursion through the function symbols. If we allow negation in the function-free case but only allow negation on base predicates, then strong irrelevance remains decidable even in the presence of constraints. However, allowing stratified negation causes strong irrelevance to be undecidable.[9] It should be noted that when rules contain stratified negation, we consider stratified semantics as opposed to classical first order semantics [89]. Finally, the table shows that slight variations in defining the minimality of derivations can cause

[9] Rules are said to be stratified [89] if there are no dependency cycles that involve negations between the predicates in the KB. The dependency graph of the KB has one node for every predicate and there is an arc from p to q if p appears in the antecedent of a rule whose consequent is q.

Table 1
Decidability of deriving irrelevance claims

Language	Strong irrelevance			Weak irrelevance
	All derivations	Minimal derivations	Minimal support derivations	All derivations
Horn rules with no recursion or constraints	**Decidable** Follows from [48]		**Decidable** Follows from [78]	
Non-recursive Horn with constraints	**Decidable** Section 4		**Decidable** Follows from [49]	
Recursive function-free Horn, no constraints (datalog)	**Decidable** Section 4	[61]	**Undecidable** [55]	Lemma A.1
Function-free Horn with constraints	**Decidable** Section 4	[61]	**Undecidable** [55]	Lemma A.1
Arbitrary Horn rules	**Undecidable** Follows from [1]			
Function-free Horn with Stratified Negation	**Undecidable** Lemma A.2		[55]	Lemma A.1
Function-free Horn with negated base predicates	**Decidable** Section 4 + [59]		**Undecidable** [55]	Lemma A.1
Datalog with unary base predicates	**Decidable** [59]			

the decidability of strong irrelevance with respect to minimal derivations to change significantly.

In this article, we focus on the algorithm for deciding strong irrelevance. In the next section we present the *query-tree*, which is a tool for automatically and efficiently deriving strong irrelevance claims for a variety of languages. The query-tree will provide a sound, complete, and efficient inference procedure for deriving strong irrelevance claims for the decidable cases denoted in the table, and will provide a sound inference procedure for the undecidable cases.

4. The query-tree

Deriving strong irrelevance claims requires that we meet several challenges. First, as implied by the definition of strong irrelevance, in order to deem a fact f strongly irrelevant to a query q, we need to establish properties of the set of *all* possible derivations of q, which may even be an infinite set. Furthermore, we have restricted our algorithm to look only at the rules in the knowledge base, and therefore, the algorithm needs to consider the possible derivations that may arise for *any* set of possible ground facts in the KB. Finally, we required that we enforce the semantics of the interpreted predicates in the rules.

The knowledge base Δ consists of the following rules:

$r_1 : badPoint(X) \wedge path(X,Y) \wedge goodPoint(Y) \Rightarrow goodPath(X,Y).$
$r_2 : link(X,Y) \Rightarrow path(X,Y).$
$r_3 : link(X,Z) \wedge path(Z,Y) \Rightarrow path(X,Y).$
$r_4 : step(X,Y) \Rightarrow link(X,Y).$
$r_5 : bigStep(X,Y) \Rightarrow link(X,Y).$

The following constraints are given on the ground facts:

$badPoint(X) \Rightarrow 100 < X < 200.$
$step(X,Y) \Rightarrow X < Y.$
$goodPoint(X) \Rightarrow 150 < X < 170.$
$bigStep(X,Y) \Rightarrow X < 100 \wedge Y > 200.$

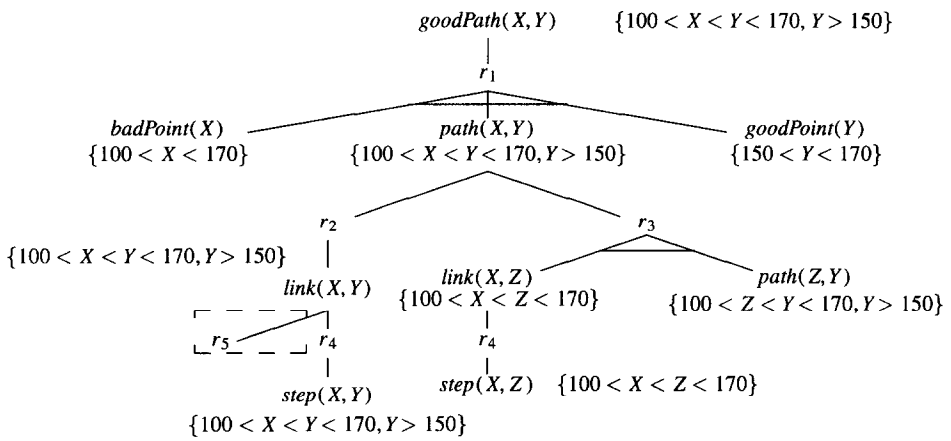

Fig. 2. An example query-tree. Note that expanding the node $link(X,Y)$ with the rule r_5 would result in an inconsistent label, and is therefore not expanded. The expanded equivalent of the node $path(Z,Y)$ is $path(X,Y)$.

This section presents a novel tool, the *query-tree,* that provides a compact representation of *precisely* the set of all derivations of the query that satisfy the semantics of the interpreted predicates. (See the example in Fig. 2.) Properties of this set of derivations can be established by simply examining the query-tree. For example, by inspecting the query-tree we can check whether a certain fact can be part of some derivation of the query, and therefore decide whether it is strongly irrelevant to the query.

Informally, the query-tree is a symbolic AND–OR tree consisting of goal nodes and rule nodes. The root of the tree is a goal node labeled with the query. A goal-node g has a child for every rule whose consequent unifies with g, and the actual child is the rule resulting from the unification with g. A rule-node has a goal-node child for every conjunct in its antecedent. The query-tree is made finite by attaching a *label* to each

node in the tree, and expanding only one goal-node from every equivalence class of labels. The label of a node contains the tightest constraint that needs to be satisfied by facts generated in that node. The label is inferred by the constraint literals appearing in the rules and the constraints known about the possible ground facts that may appear in the KB.

We begin in Section 4.1 by explaining the correspondence between derivations and symbolic derivations, and how the query-tree *encodes* a set of symbolic derivations. Section 4.2 describes the algorithm for constructing the query-tree, and Section 4.3 discusses the complexity of building the query-tree. The query-tree algorithm, as we describe here, is an instance of a general method for encoding sets of derivations [55]. Intuitively, by changing the contents of node labels in the query-tree, it can be designed to encode various sets of derivations (e.g., minimal derivations), and therefore, used for deriving other strong irrelevance claims. In Section 4.4, we briefly mention other encodings possible by the query-tree.

4.1. Derivations and symbolic derivations

In the context of Horn-rule knowledge bases, a derivation of a ground atom can be viewed as a tree consisting of goal-nodes and rule-nodes (see Fig. 3(a)). The root of the tree is a goal-node containing the query atom. If a goal-node g was derived using an instantiation of a rule r, then r is in the child rule-node of g and its children are the instantiations of the antecedents of r. To simplify some of the arguments in our proofs, we assume without loss of generality. that the children of the rule-node are ordered from left to right as they are ordered in the antecedent of r. The leaves of a derivation are the ground atomic facts from the knowledge base that were used in the derivation. For reasons that will become clearer shortly, we do not include nodes in the tree for interpreted predicates.

Since the query-tree will be built based only on the rules in the knowledge base, it will actually encode *symbolic derivations*. A symbolic derivation (see Fig. 3(b,c)) is like a derivation except that the constants are replaced by variables, and it includes only constants that appear in the query or in the rules. The root of a symbolic derivation tree is a goal-node of the query predicate. The child of a goal-node g is a rule-node containing a rule whose consequent unifies with the goal-node. The rule-node has a goal-node child for every conjunct in its antecedent, and the contents of each such goal-node is the corresponding conjunct in the unification of the rule with g. The leaves of a symbolic derivation tree contain symbolic atoms of the base predicates. A rule-node r is formally a renaming $\theta(r_0)$ of a rule r_0 in the knowledge base, where θ is a mapping on the variables of r_0. The expression $rule(r)$ denotes the actual rule r_0.

A symbolic derivation represents the set of derivations that can be obtained by assigning constants to its variables such that the literals of the interpreted predicates in the rules are satisfied. A symbolic derivation tree is said to be *satisfiable* if there is at least one such variable assignment. For example, in Fig. 3, (b) is a satisfiable symbolic derivation and (c) is not. The set of all derivations of the query is therefore represented by the set of instances of all satisfiable symbolic derivations.

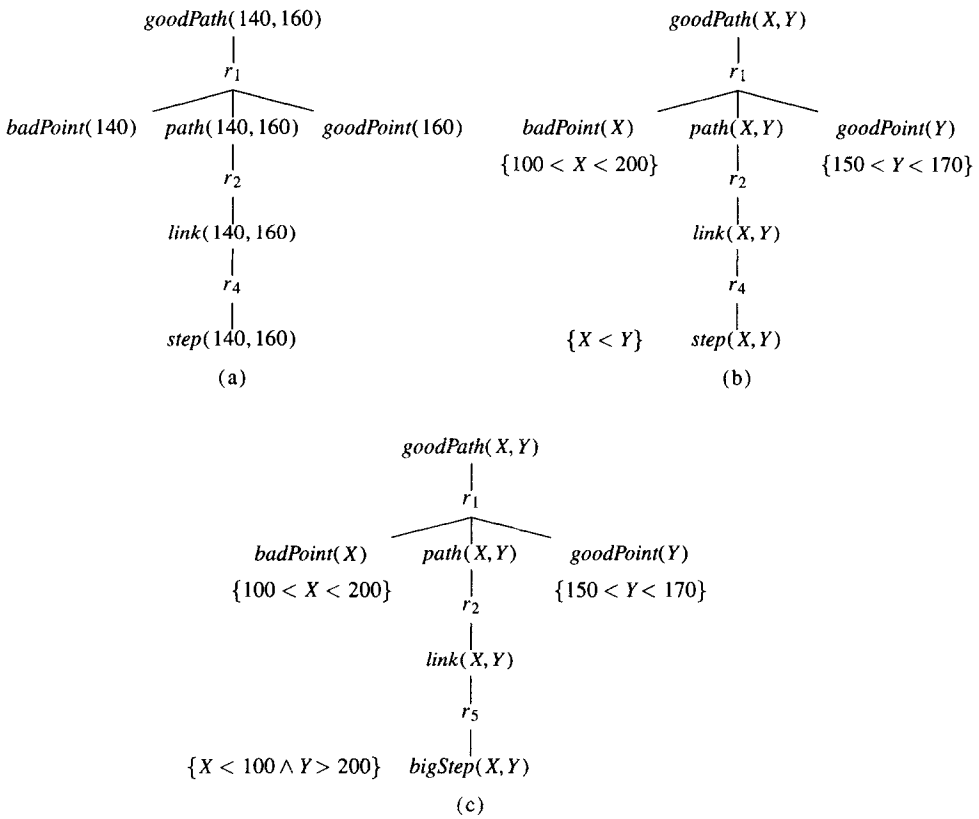

Fig. 3. (a) is a ground derivation. (b) is a satisfiable symbolic derivation and (c) is an unsatisfiable symbolic derivation.

The query-tree is an AND–OR tree *encoding* a set of symbolic derivations. Its root is a goal-node containing the atom of the query. In contrast to a symbolic derivation tree, a goal-node g (an OR node) may have several rule-node children, each for one of the rules whose consequent unifies with g.

When rules contain recursion, there will be an infinite number of symbolic derivations. Therefore, in order to encode an infinite number of derivations by a finite structure, we expand only part of the goal-nodes in the query-tree. To decide which nodes to expand, we attach *labels* to every node in the tree. The label describes all the constraints that need to be satisfied by instances of that node. The labels induce an equivalence relation on nodes in the query-tree, and in our construction, we only expand *one* goal-node from every equivalence class. If nodes g_1 and g_2 have equivalent labels, and g_1 was the one expanded, we call g_1 the *expanded equivalent* of g_2, denoted $Eq(g_2)$. As we will see, these labels cannot be completely computed during the top-down construction of the tree, and we will need a bottom-up computation to precede the top-down phase.

Based on expanded equivalents, we can now define how the query-tree *encodes* a symbolic derivation:

Definition 9. A symbolic derivation d is encoded by the query-tree T if there exists a mapping ψ from the nodes of d to the nodes of T that satisfies the following conditions:

E0. If $g_1, \ldots, g_n$ are the children goal-nodes of the rule-node r in d, then $\psi(g_1), \ldots, \psi(g_n)$ are the children of $\psi(r)$ in T, respectively.

E1. For every rule-node $r \in d$, the rule in $\psi(r)$ is the same as the rule in r; i.e., $rule(r) = rule(\psi(r))$.

E2. The node $\psi(root(d))$ is the root of the query-tree.

E3. If r is a child of the goal-node g in d, then:

(1) If $\psi(g)$ is expanded in T, then $\psi(r)$ is a child of $\psi(g)$.

(2) If $\psi(g)$ is not expanded in T, then $\psi(r)$ is a child of its expanded equivalent, $Eq(\psi(g))$.

In the next section we describe how to construct the query-tree. The main challenge in the construction is to compute the labels of the nodes in the tree such that the resulting tree will encode *precisely* the set of satisfiable symbolic derivations of the query.

4.2. Constructing the query-tree

The input to our algorithm is the following:

- A set of rules $\mathcal{P}$. We assume without loss of generality that all literals of non-interpreted predicates in the rules contain a distinct variable in every argument position. (All constants and all equalities between arguments are expressed as equality literals.) Note that this implies that all unifications will be trivial.
- Constraints on the base relations. For every base relation e, we are given a constraint c_e on the arguments of e that describes conditions that are known to hold for facts of e in any given knowledge base. If there are no such constraints, then $c_e = TRUE$. We assume that c_e is given in its standard form.
- A query, which we assume is of the form $q(X_1, \ldots, X_n) \wedge c_q$, where $X_1, \ldots, X_n$ are n distinct variables and c_q is a constraint on $X_1, \ldots, X_n$.

A key difficulty in building the query-tree stems from the observation that the tree cannot be built in a single top-down construction beginning from the root. This is because the label of a node in the tree may depend on its descendents and therefore cannot be determined in a single top-down construction. However, the label of a node must be known in order to decide whether or not to expand the node. The solution to the problem is to precede the top-down construction by a bottom-up phase.

The query-tree algorithm consists of two steps:

Bottom-up phase. In this step we compute a set of *adorned predicates* and a set of *adorned rules*, $\mathcal{P}_1$. An adorned predicate is a predicate of the form p^c, where p is a predicate in the KB and c is a constraint on the arguments of p. Intuitively, the predicate p^c is intended to represent the subset of the extension of the predicate p which includes the tuples satisfying the constraint c. The adorned rules are specializations of the original rules, where the original predicates are replaced by adorned predicates. Note that since a predicate in $\mathcal{P}$ may have several adornments in $\mathcal{P}_1$, the number of rules in $\mathcal{P}_1$ may be greater than the number of rules in $\mathcal{P}$.

We begin with the predicates e^{c_e}, where e is a base predicate and c_e is the given constraint on facts of e (as given in the input). We compute new adorned predicates using the rules in the KB as follows. Suppose $r \in \mathcal{P}$ is a rule of the form

$$r : q_1(\bar{X}_1) \wedge \cdots \wedge q_m(\bar{X}_m) \wedge c_r \Rightarrow p(\bar{X})$$

where c_r is the conjunction of interpreted literals in r, and suppose that we have computed a predicate $q_i^{c_i}$ for each predicate q_i, $1 \leqslant i \leqslant m$. Let c be the constraint $c = c_1(\bar{X}_1) \wedge \cdots \wedge c_m(\bar{X}_m) \wedge c_r$, and let c_h be the projection of c on the head variables $\bar{X}$. If c is satisfiable, then we create the adorned predicate p^{c_h} and we add the following adorned rule to $\mathcal{P}_1$:

$$r' : q_1^{c_1}(\bar{X}_1) \wedge \cdots \wedge q_m^{c_m}(\bar{X}_m) \wedge c \Rightarrow p^{c_h}(\bar{X}).$$

Note that the rule r may be recursive, and therefore we may use one adornment of the predicate p in order to compute a new adornment. We apply the rules of $\mathcal{P}$ until no new adornments are computed. Note that the Finiteness property of the constraint language guarantees that this phase will terminate.

Top-down construction of the query-tree. In this step, we construct a *forest* of trees using the adorned rules $\mathcal{P}_1$. We attach to every node g in the forest a label, denoted by $L(g)$. We begin by inserting a node for each adorned predicate of the query predicate q of $\mathcal{P}$, i.e., a node for each predicate of the form q^c, where $q^c \in \mathcal{P}_1$, when $c \wedge c_q$ is satisfiable. The label of the node is $c \wedge c_q$. We proceed to build the forest as follows. A goal-node g for a predicate p^c can be unified only with adorned rules whose consequent has the predicate p^c, i.e., of the form:

$$r : q_1^{c_1}(\bar{X}_1) \wedge \cdots \wedge q_m^{c_m}(\bar{X}_m) \wedge c_r \Rightarrow p^c(\bar{X}).$$

If $L(g) \wedge c_r$ is satisfiable, then we create a new rule-node g_r as a child of g, where g_r is a rule-node of rule r and its label is $L(g_r) = L(g) \wedge c_r$; moreover, we also create a child goal-node of g_r for every literal in the antecedent of r that has a non-interpreted predicate. The label of the child goal-node corresponding to $q_i^{c_i}$ is the projection of $L(g_r)$ onto the variables that appear in $q_i^{c_i}$. The termination of the top-down phase is based on the following conditions:

- We do not expand goal-nodes of base predicates.
- We do not expand a goal-node g if the resulting rule-node will have an unsatisfiable label.
- We do not expand a goal-node of the form $p(\bar{X})$ if there exists an expanded node $p(\bar{Y})$ in the forest such that the isomorphism mapping the variables $\bar{X}$ to $\bar{Y}$ is also an isomorphism between $L(p(\bar{X}))$ and $L(p(\bar{Y}))$.[10]

The details of the top-down phase are given in Fig. 4.

Example 10. Consider the application of the algorithm to the example in Fig. 2. The bottom-up phase begins with the adorned predicates

[10] Note that $p(\bar{Y})$ can be any node in the forest, not necessarily an ancestor of $p(\bar{X})$.

procedure build-forest$(\mathcal{P}_1, q(\bar{X}) \wedge c_q)$
begin
/* Creating a forest of trees T for the adorned rules $\mathcal{P}_1$ and query $q(\bar{X}) \wedge c_q$. */
for all nodes g in the tree, $Eq(g) = g$ unless otherwise stated.
for every predicate of the form $q^c \in \mathcal{P}_1$ such that $c \wedge c_q$ is satisfiable,
  Let $\bar{Y} = Y_1, \ldots, Y_n$ be n new variables that do not appear in T.
  Insert a goal-node $q^c(\bar{Y})$ into T with label $(c \wedge c_q)(\bar{Y})$.
repeat
  Let $g = p^c(\bar{X}_1)$ be a leaf in T of a derived predicate with label $L(g)$.
  if there is an expanded goal-node $g_1 = p^c(\bar{X}_2)$ in T, such that ϕ is
    the isomorphism from $\bar{X}_1$ to $\bar{X}_2$ **and** $\phi(L(g))$ is equivalent to $L(g_1)$ **then**
    Set $Eq(g) = g_1$.
  else
    for each rule $r \in \mathcal{P}_1$ with consequent predicate p^c **do**
      Let θ be a 1–1 variable mapping that maps the consequent of r to g,
      and the other variables of r to new variables that don't appear in T.
      if $\theta(c_r) \wedge L(g)$ is satisfiable **then**
        Create a child rule-node of g, containing the rule $\theta(r)$,
        with label $\theta(c_r) \wedge L(g)$.
        for every literal g_i of non interpreted predicate in the antecedent of $\theta(r)$,
          Create a child goal-node g_i for the rule-node, where
          $L(g_i)$ is the projection of $\theta(c_r) \wedge L(g)$ on the variables of g_1.
until no changes are made to T.
Remove all adornments from predicates in nodes of T.
return T.
end build-forest.

Fig. 4. Top-down creation of the query-tree.

$step^{X_1<X_2}$, $bigStep^{X_1<100,X_2>200}$,
$badPoint^{100<X_1,X_1<200}$, $goodPoint^{150<X_1,X_1<170}$.

With rule r_4, we create the adorned predicate $link^{c_1}$, where $c_1 = \{X_1 < X_2\}$; and with rule r_5, we create $link^{c_2}$, where $c_2 = \{X_1 < 100, X_2 > 200\}$.

Substituting $link^{c_1}$ in r_2 results in $path^{c_1}$, and similarly for $link^{c_2}$ and $path^{c_2}$. Rule r_3 does not produce any new adornments for *path*. For example, substituting $link^{c_1}$ for the first subgoal and $path^{c_1}$ for the second results in $path^{c_1}$, while the other three combinations result in $path^{c_2}$. Finally, substituting $path^{c_1}$ in r_1 we compute $goodPath^{c_3}$, where $c_3 = \{100 < X_1, X_1 < X_2, X_2 < 170, X_2 > 150\}$. Note that trying to substitute $path^{c_2}$ in r_1 will yield an unsatisfiable label for *goodPath*, therefore we do not create additional adornments of *goodPath*.

The bottom-up phase creates the following adorned rules (omitting adornments of predicates with a single adornment):

r_1 : $badPoint(X) \wedge path^{c_1}(X,Y) \wedge goodPoint(Y) \wedge (X > 100) \wedge (X < Y) \wedge (Y < 170) \wedge (Y > 150) \Rightarrow goodPath^{c_3}(X,Y)$.
r_2^1 : $link^{c_1}(X,Y) \wedge (X < Y) \Rightarrow path^{c_1}(X,Y)$.
r_2^2 : $link^{c_2}(X,Y) \wedge (X < 100) \wedge (Y > 200) \Rightarrow path^{c_2}(X,Y)$.
r_3^1 : $link^{c_1}(X,Z) \wedge path^{c_1}(Z,Y) \wedge (X < Z) \wedge (Z < Y) \Rightarrow path^{c_1}(X,Y)$.
r_3^2 : $link^{c_2}(X,Z) \wedge path^{c_1}(Z,Y) \wedge (X < 100) \wedge (Z > 200) \wedge (Z < Y) \Rightarrow path^{c_2}(X,Y)$.
r_3^3 : $link^{c_1}(X,Z) \wedge path^{c_2}(Z,Y) \wedge (X < Z) \wedge (Z < 100) \wedge (Y > 200) \Rightarrow path^{c_2}(X,Y)$.
r_4 : $step(X,Y) \wedge (X < Y) \Rightarrow link^{c_1}(X,Y)$.
r_5 : $bigStep(X,Y) \wedge (X < 100) \wedge (Y > 200) \Rightarrow link^{c_2}(X,Y)$.

The result of the top-down phase is shown in Fig. 2. Note that the predicates $link^{c_2}$ and $path^{c_2}$ cannot be used in derivations of $goodPath^{c_3}$, and therefore rule r_5 is not in the tree. The expanded equivalent of the node $path(Z,Y)$ is $path(X,Y)$.

The following theorem shows that the query-tree tells us exactly which facts are strongly irrelevant to the query:

Theorem 11. *Let $\mathcal{P}$ be a set of Horn rules with interpreted predicates that satisfy the Closure, Equivalence, Satisfiability, and Finiteness properties. Let T be the query-tree created for the rules $\mathcal{P}$ and the query of the form $q(\bar{X}) \wedge c_q$.*

(1) *A fact $p(a_1,\ldots,a_n)$ is strongly irrelevant to any instance q_0 of $q(\bar{X}) \wedge c_q$ with respect to $\Sigma_{\mathcal{P}}$, (i.e., the irrelevance claim $SI(p(a_1,\ldots,a_n), q_0, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{q_0})$ holds), if and only if there is no node g of p in T, such that $a_1,\ldots,a_n$ satisfies the label $L(g)$ of g.*

(2) *A rule r is strongly irrelevant to any instance q_0 of $q(\bar{X}) \wedge c_q$ with respect to $\Sigma_{\mathcal{P}}$ (i.e., the irrelevance claim $SI(r, q_0, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{q_0})$ holds) if and only if r does not appear in T.*

It is important to emphasize that the labels in the query-tree are as tight as possible, and therefore the theorem provides not only a sufficient condition for strong irrelevance, but also a necessary condition. The proof of the theorem is given in Appendix A. Returning to the example in Fig. 2, the rule r_5 is strongly irrelevant to *goodPath* because it does not appear in the query-tree. Using the query-tree, we can also derive that the atomic facts of $step(X,Y)$ for which $X \leqslant 100$ or $Y \geqslant 170$ are strongly irrelevant to the query.

4.3. Complexity

The time taken to build the query-tree (and therefore to decide strong irrelevance) is linear in the number of rules in the knowledge base and may be exponential in the arity of the predicates. To see this, let l be the number of non-equivalent labels (i.e., adornments in the bottom-up or labels in the top-down phase) that can be generated for a predicate in the KB, and let e be the number of time units needed to check equivalence

of two labels. The bottom-up phase of the algorithm (i.e., creating the refined rules) can take time proportional to the maximum number of refined rules that can be created. If b is the maximum number non-interpreted literals in the antecedent of a rule, and R is the number of rules in the KB, then we can create at most Rl^b refined rules. Creating a refined rule requires that we check the satisfiability of the resulting constraint in the rule, and therefore the bottom-up phase will take at most Rel^b time units. The time to build the forest in the top-down phase will be $O(|T|e)$, where $|T|$ is the number of goal-nodes in the query-tree. However, since only one goal-node of every equivalence class is expanded, the number of internal goal-nodes in the tree is at most lP, where P is the number of derived predicates in the KB. Consequently, the number of leaves in the query-tree is at most $lPRb$ (each internal node can be expanded with R rules and Rb grand-children). Therefore, the time to construct the query-tree is $O(RPebl^b)$.

The value of l (the number of different adornments) and e (the time to check equivalence of labels) depends on the constraint language under consideration. In the case of order constraints (with conjunction and disjunction), l is at worst doubly exponential in *arity* of the predicates in the KB. This is because a label essentially describes some constraint on the ordering of the arguments of the relation (and constants that appear in the rules). There are an exponential number of total orderings of the variables, and therefore a doubly exponential number of non-equivalent constraints (each constraint describes a set of possible orderings). The time to check equivalence of two labels is at worst exponential in the arity (and polynomial if only conjunctive labels are allowed).

It is important to emphasize that the complexity of building the query-tree is only linear in the number of rules (and, of course, does not depend on the number of ground facts!), and possibly exponential only in the arity of the predicates. This is an important distinction because arities of predicates tend to be small (e.g., frame systems employ mostly binary predicates), and therefore, the algorithms will scale up to knowledge bases with many rules and ground facts. Moreover, we believe that an exponential number of labels occurs only in pathological cases. Furthermore, the following theorem shows that we cannot expect to do much better than we have with the query-tree. Specifically, it shows that once we introduce the predicate $\neq$, the lower bound on the problem of detecting strong irrelevance is exponential in the arity.

Theorem 12. *Given a set of rules $\mathcal{P}$, a query schema $q(\bar{X}) \wedge c_q$, and a rule $r \in \mathcal{P}$, deciding $SI(r, q(\bar{X}) \wedge c_q, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^q)$ is hard for exponential time if the rules may contain the predicate $\neq$.*

The proof, given in Appendix A, is based on reducing the acceptance problem of a linear-space alternating Turing machine (ATM) to the problem of detecting strong irrelevance of rules.

4.4. Extensions of the query-tree

The query-tree method. The query-tree algorithm, as described above, is one instance of a general method for encoding a possibly infinite set of derivations via a finite structure [55, 59, 61, 64]. The method is based on encoding an infinite number of derivations

by identifying a labeling scheme, i.e., a *finite* set of labels, such that terminating the construction of the tree based on label equivalence guarantees that the query-tree encodes *exactly* a desired set of derivations.

In the previous section, the label of a node described the tightest constraint formula that needs to be satisfied by facts generated at the node, and the query-tree encoded exactly the set of satisfiable symbolic derivations. Several other instances of this method have been used to encode different sets of derivations, and are of interest here because they yield algorithms for other types of strong-irrelevance claims (see Table 1):

(1) *Tag labels.* The label of a node also includes information on the ancestry of that node. As a result, the query-tree encodes exactly the set of minimal derivations of the query [61].

(2) *EDB-labels.* The label of a node includes information on negative and positive literals that can appear in the derivation tree. Using these labels, we encode exactly the set of satisfiable derivations when rules include negated base predicates in the antecedents [59].

(3) *IC-labels.* The label of a node includes partial instantiations of integrity constraints (i.e., clauses with only negative literals) that are satisfied by the derivation tree. Using this labeling schemes we can decide strong irrelevance for some classes of Horn theories that include integrity constraints [64].

Relaxing the finiteness property. The most stringent requirement we imposed on the constraint language is the Finiteness property, which requires that we can only generate a finite number of labels on the predicates, using operations of join, selection, and projection. When the rules contain *both* function symbols and recursion, this property may not hold. For example consider the rules:

$s_1 : (X = 0) \Rightarrow integer(X)$.
$s_2 : integer(X) \Rightarrow integer(X + 1)$.

As shown in Fig. 5(a), a top-down expansion of a tree for these rules will result in an infinite number of labels of the form $\{Z_i = X - i\}$ for every integer i. Therefore, the construction of the query-tree will not terminate.

To build a query-tree in such cases, we choose a finite set of labels $\mathcal{C}$ to assign to nodes in the query-tree. When we project a constraint on a rule-node onto its father (or children) goal-node, we proceed by the following strategy. Given a constraint c and a subset of its variables $\bar{X}$ that appear in the goal-node, if there is no label in $\mathcal{C}$ which describes the exact projection $c|_{\bar{X}}$, we return a member c_1 of $\mathcal{C}$ such that $c|_{\bar{X}} \models c_1$, and such that there is no other constraint $c_2 \in \mathcal{C}$ such that $c_2 \models c_1$ and $c|_{\bar{X}} \models c_2$. The constraint c_1 can be viewed as the best *approximation* to $c|_{\bar{X}}$ out of the finite number of labels $\mathcal{C}$. Consequently, the resulting labels in the query-tree are weaker than the tightest ones possible, and therefore, the query-tree provides only a sufficient condition for strong irrelevance. That means that a ground atomic fact which does not match any of the nodes in the tree is strongly irrelevant, but not vice versa. One simple method to choose such a finite set of labels $\mathcal{C}$ is to not allow new terms to be created in the labels (or to allow a maximum of k new terms, where k is fixed). For instance, in our example, if we do not allow new terms, we get the query-tree shown in Fig. 5(b).

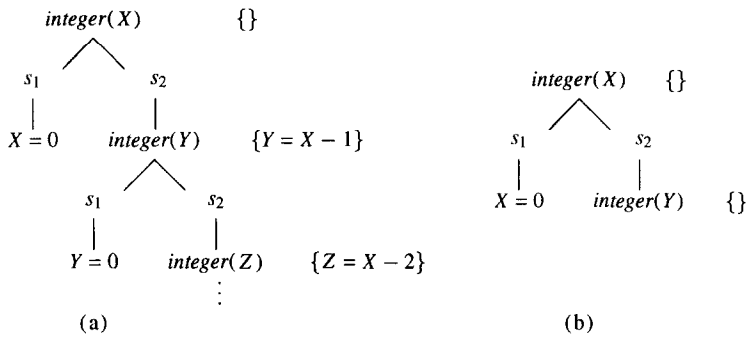

Fig. 5. Query-tree with function symbols.

Inferring irrelevance claims from external sources. As stated earlier, irrelevance claims can either be derived solely based on inspecting (parts of) the knowledge base, or can be inferred from irrelevance claims that are given by a user. In [55,62], it is shown that even for strong irrelevance, deriving all the irrelevance claims that follow from a given irrelevance claim is undecidable in function-free Horn-rule theories. However, the query-tree can be used as a sound inference procedure for irrelevance claims. Specifically, if we are told that a fact f is strongly irrelevant to the query, we build a query-tree that does not contain f (if f is a rule, we build the query-tree without f. If f is the set of facts of a predicate p that satisfy a constraint c, we simply impose $\neg c$ as an integrity constraint on p). As a result, facts that can only be parts of derivations of the query that contain f (and are therefore strongly irrelevant if f is strongly irrelevant) will also not appear in the query-tree. The details of the algorithm are described in [55].

5. Using the query-tree to speed up inferences

In this section, we explore two ways in which the query-tree can be used to speed up inferences. In the first, the query-tree is used to decide which ground facts and rules are strongly irrelevant to a given class of queries. Based on that determination, we create specialized database indices that see only the ground facts that are (possibly) relevant to a class of queries. Given a query from the given class, we can then use these indices for fetching ground facts during inference, thereby significantly speeding up inference. The second use of the query-tree is based on the observation that the tree also encodes all the possible *sequences* of rule applications and database lookups that can result in derivations of the query. We can therefore use the query-tree to guide the search of a backward chainer to follow only these sequences.

Although it is clear from theoretical analysis that ignoring irrelevant facts and solution paths can yield speedups, the impact of these savings in practice and their dependence on various factors is unclear. Such factors include the cost of building the indices (versus

the savings achieved from utilizing them), the percent of facts deemed irrelevant, the size of the knowledge base and variations on the form of the rules. This section presents an empirical validation of the savings achieved by using the query-tree and an empirical analysis of the significance of the factors affecting the speedups.

5.1. Two uses of the query-tree

The first use of the query-tree is based on the fact that it tells us exactly which facts are irrelevant to a given set of queries (Theorem 11). Specifically, a rule is strongly irrelevant to the query if and only if it does not appear in the tree. A ground fact is strongly irrelevant if and only if it does not satisfy the label of any goal-node in the tree with which it can be unified. Consequently, when answering the query, these rules and ground facts can be ignored. Recall that this determination is independent of the ground facts in the KB. For instance, in the *goodPath* example (repeated in Fig. 2), the rule r_5 can be ignored. Similarly, facts of the relation *step* that do not satisfy $\{100 < X < Y < 170\}$ can also be ignored.

We can use this property of the query-tree to speed up inferences for sets of queries that occur frequently. Given such a set of queries, we build a query-tree for it and create specialized indices *only* on the facts that are not strongly irrelevant to queries in the set. The cost of preprocessing the knowledge base in such a way involves the cost of building the query-tree and the cost of one pass over the knowledge base to build the specialized indices. However, the payoff of removing irrelevant facts can be significant because the size of the space that an inference mechanism needs to search can be drastically reduced. Specifically, it is guaranteed that every time a ground fact is retrieved, the fact *may* be part of a derivation of the query since it satisfies the constraint label of some node in the query-tree. This is especially significant when lookups are made with some unbound variables. For instance, in our example, there will be many lookups of the form $step(a, Y)$, where a is some constant and Y is unbound. Using the specialized index on the facts of the predicate *step* guarantees that every fact retrieved will satisfy $\{100 < Y < 170\}$. In contrast, retrieving a fact that does not satisfy this constraint can generate a whole search subtree that is guaranteed to be useless. Note that even if the reasoning mechanism detects immediately (by checking the available constraints) that the retrieved fact is irrelevant, the cost of doing all the useless lookups and checking the constraints can be arbitrarily large. [11]

The second use of the query-tree is based on the observation that the tree also encodes the *sequences* of rule applications and database lookups that can result in derivations of the query. We can use this observation to further control our search. To illustrate, consider the following example.

Example 13. Consider a knowledge base defining a relation *dessertMeal* with the following rules. Its query-tree is shown in Fig. 6.

[11] Moreover, note that in order to always be able to detect irrelevant facts immediately, the reasoning mechanism must propagate the constraints in the same fashion done in creating the query-tree.

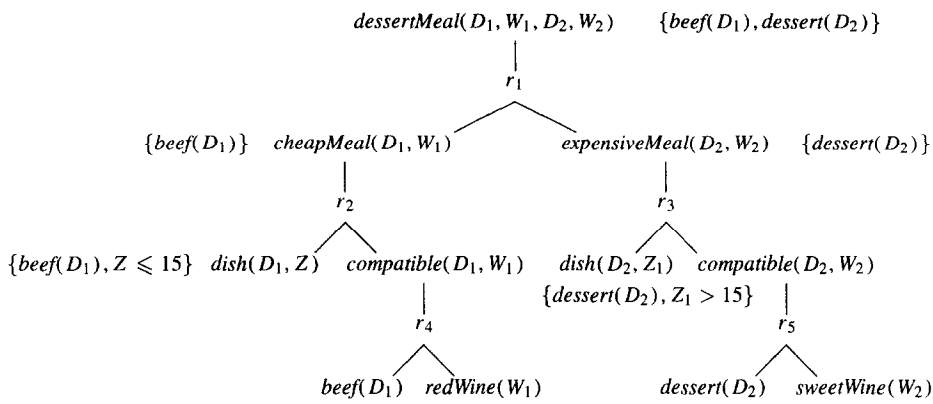

Fig. 6. Avoiding search paths using the query-tree.

r_1 : $cheapMeal(D_1, W_1) \wedge meat(D_1) \wedge expensiveMeal(D_2, W_2) \wedge dessert(D_2) \Rightarrow dessertMeal(D_1, W_1, D_2, W_2)$
r_2 : $dish(X, Z) \wedge (Z \leqslant 15) \wedge compatible(X, Y) \Rightarrow cheapMeal(X, Y)$
r_3 : $dish(X, Z) \wedge (Z > 15) \wedge compatible(X, Y) \Rightarrow expensiveMeal(X, Y)$
r_4 : $beef(X) \wedge redWine(Y) \Rightarrow compatible(X, Y)$
r_5 : $dessert(X) \wedge sweetWine(Y) \Rightarrow compatible(X, Y)$

The predicates *meat*, *beef*, and *dessert* are sort predicates (*dessert* is disjoint from the other two). The relation *compatible* represents pairs consisting of a wine and a dish that are compatible with each other. The relation *dessertMeal* identifies two pairs of compatible dishes and wines (one pair for the main dish and another for dessert), such that more attention is given to the dessert part of the meal. The relation *dish* represents the available dishes and their prices. Consider facts of the relation *dish*. Any fact $dish(X, Y)$ that satisfies either $(beef(X) \wedge Y \leqslant 15)$ or $(dessert(X) \wedge Y > 15)$ may be relevant to the query *dessertMeal* (because of the rules r_2 & r_4 and r_3 & r_5 respectively). However, as a subgoal of r_2, we need only consider facts of *dish* that satisfy the first constraint, whereas as a subgoal of r_3, only facts that satisfy the second constraint are needed. Moreover, the query-tree shows that rule r_4 can only be applied to a subgoal of r_2, and not of r_3 (and vice versa for r_5).

To exploit this additional control knowledge, we create specialized indices for every *leaf* in the query-tree (as opposed to an index for every relation in the KB), and modify the inference mechanism to follow *only* the paths permitted by the query-tree. In our example, we create one index for beef dishes under \$15 and another for dessert dishes over \$15. To follow the query-tree during inference, we attach to every subgoal g in our search a node in the query-tree $\phi(g)$. We start by assigning the root of the query-tree to the query. At every step, if g is a database lookup subgoal (i.e., a subgoal of a base predicate), we perform the lookup using the specialized index of $\phi(g)$. Otherwise, we expand g only with the rules that are children of the expanded equivalent of $\phi(g)$

(which may be the node $\phi(g)$ itself). We assign to the subgoals of g the appropriate subgoals of the rule-node in the query-tree. As a result, the inference engine follows only the paths encoded by the query-tree, and in every database lookup it retrieves only ground facts that can be used in derivations in the current path.

5.2. Experimental results and analysis

We tested the impact of the savings achieved by using the query-tree using a depth first search backward chainer on Horn rules.[12] Given a knowledge base Δ and a query schema $q(\bar{X})$, we built a query-tree for $q(\bar{X})$ and two sets of indices on ground facts in Δ. The first set $\mathcal{I}_1$ included an index on every relation that includes only the facts that were deemed not strongly irrelevant to $q(\bar{X})$ by the query-tree. Specifically, a ground fact $e(a_1,\ldots,a_n)$ is included in the index for the relation e in $\mathcal{I}_1$ if $a_1,\ldots,a_n$ satisfies the constraint label of *some* leaf of the predicate e in the query-tree.[13] The second set of indices $\mathcal{I}_2$ included one index for every leaf of a base predicate in the query-tree. We measured the running times and number of nodes expanded of three versions of the backward chainer, all of which returned the same answers:

- BC1—the backward chainer on Δ using the original indices in the KB.
- BC2—the backward chainer on Δ using the indices $\mathcal{I}_1$, i.e., ignoring strongly irrelevant facts.
- BC3—A backward chainer that uses the indices $\mathcal{I}_2$ and only follows the paths allowed by the query-tree.

We tested over 20 query schemas taken from the following four domains:

(1) A travel domain using a fragment of a database of real airline data describing flight times between cities in the U.S. (examples 3–6 in the tables).
(2) A wine domain consisting of a knowledge base of 50 rules describing various wines and dishes and compatibilities between them (based in part on [75]) (examples 7–8).
(3) A student–advisor domain using a knowledge base about computer science Ph.D graduates, including advisor, school and graduation dates (examples 9–10).
(4) The *goodPath* example, using the rules in Example 2 (examples 1–2).

The first and fourth domains usually yield deep recursive search trees, even though the number of rules is small. The second domain is non-recursive and yields shallow but bushy (i.e., large branching factor) search trees. In the third domain, search trees have a low branching factor (which was from student to advisor).

Table 2 presents the results of the experiments for the case where we are looking for all solutions to a query (e.g., find all X, Y such that *goodPath*(X,Y) is entailed by the KB). In the table, *Filtering Time* includes the time taken to build the query-tree and

[12] The speedups attained by removing irrelevant facts (the ratio of BC1 to BC2) were also tested using the backward chainer of Epikit (a commercial implementation of MRS [76]) and with Quintus Prolog. The speedups attained were even better than those reported here. However, we could not use these tools for testing BC3 because we could not modify the control of the backward chainers. In the experiments, we tested several rule and goal orderings. The results are shown for the ordering that yielded the best results consistently for *all* three versions of the backward chainer.

[13] Note that the original knowledge base had an index for each base relation.

Table 2
Experimental results: finding all solutions

Ex. #	KB size		Filtering time (sec)	Percent irrelevant	Solution time (sec)					Nodes expanded	
	Facts	Rules			BC1	BC2	BC1/BC2	BC3	BC1/BC3	BC1/BC2	BC1/BC3
1.	350	6	1.8	63	2780	182	15	183	15	10.5	10.5
2.	350	6	1.8	63	618	231	2.7	233	2.7	2.5	2.5
3.	200	18	6.5	69	372	14	27	8.6	43	22	28
4.	200	18	5.6	69	25	5.5	4.5	4.5	5.6	4.7	6
5.	200	18	20.7	64	3975	205	19	13	306	17	295
6.	200	18	14.7	68	1278	41	31	1.1	1190	31	1630
7.	1300	47	25	59	8740	8720	1	363	24	1	14
8.	1300	47	11.6	60	50	42	1.2	11	4.5	1.2	2.8
9.	150	17	0.8	59	35	7.5	4.6	7.5	4.6	4.5	4.5
10.	150	17	0.6	59	2.8	0.4	7.6	0.4	7.6	4.8	4.8

create all the indices (both $\mathcal{I}_1$ and $\mathcal{I}_2$). *Percent irrelevant* is the percent of ground facts in the knowledge base that were deemed strongly irrelevant (and therefore not included in the indices $\mathcal{I}_1$). The next columns show the time taken to find all the solutions to the query. The respective running times of BC1, BC2, and BC3 are shown, as well as the ratios of running times. The last two columns show the ratios of the number of nodes expanded by BC2 and BC3 compared to BC1.

The results show significant speedups for both BC2 and BC3. For BC2, the speedups were usually in excess of a factor of 3, ranging up to 31 (mean: 10.4). The results show that by following the query-tree using BC3, we often get additional improvements. The speedups of BC3 over BC1 were usually in excess of 5, ranging up to 1190 (mean: 41, excluding example 6).[14] In terms of nodes expanded, the average speedup for BC2 was 10, while the average speedup for BC3 was 37 (excluding example 6). The results clearly show that if we are looking for all solutions to the query, building the query-tree and the specialized indices will yield significant savings.

Table 3 shows that similar results are obtained in cases in which we use the query-tree built for a query schema to answer ground queries or to find the *first* solution to a query with free variables (i.e., the query-tree was built for $goodPath(X,Y)$ and the query is $goodPath(130,160)$, or we are trying to find the first binding for X and Y). The second and third columns show the ratios of the number of nodes expanded for ground queries. The next columns show the node ratios of finding the first solution to the query. The next column compares the preprocessing time and the time to find solutions to the query. It shows the number of calls (each looking for the *next* solution) after which the preprocessing time equals the time to answer the queries. The last column shows the number of solutions found for the query. The results indicate that often the preprocessing pays off after a very small number of solutions and therefore it is beneficial to build a query-tree even in cases when we are searching for few solutions.

[14] Example 6 was excluded from the mean because the speedups it yielded were exceptionally high. In this example, the resulting query-tree was empty, showing that there cannot be answers to the query.

Table 3
Experimental results: ground queries and finding the first solution

Example	Ground queries		Find first solution		Solutions needed to break even	Number of solutions
	BC1/BC2	BC1/BC3	BC1/BC2	BC1/BC3		
1.	5.8	6.1	85	85	1	187
2.	1.8	1.8	4.5	4.5	1	187
3.	33	74	2.5	2.6	1	49
4.	4.6	7	6.1	6.2	6	37
5.	15	290	1	1	1	12
6.	33	1550	31	1630	1	0
7.	1	2	1.4	25	115	41000
8.	1.1	1.4	1.6	4.5	81	420
9.	1.3	1.3	123	123	2	353
10.	1	1	4.8	4.8	1	0

Table 4
Changing the percent of irrelevant facts

Example 1			Example 3			Example 9		
Percent irrelevant	Solution time		Percent irrelevant	Solution time		Percent irrelevant	Solution time	
	BC1/BC2	BC1/BC3		BC1/BC2	BC1/BC3		BC1/BC2	BC1/BC3
45	5	5	21	1.8	3.1	4	1.1	1.1
63	15	15	45	4.5	7.6	15	1.3	1.3
79	101	101	69	27	43	59	4.6	4.6
92	1250	1240	92	381	447	82	23.5	23.5

The experiments showed that the savings achieved by using the query-tree are affected by several factors which we discuss below.

5.2.1. *Percent of irrelevant facts*

The analysis of the algorithm suggests that the speedups obtained will be significantly affected by the percent of facts in the knowledge base that are found to be irrelevant to the query. To test this effect, we ran several variants of each example that differed only in the constants appearing in the rules (which had the effect of varying the percent of irrelevant facts). The results, shown in Table 4, show that the speedups grow significantly as the percent of irrelevant facts increases. For example, when 90% of the facts are found to be strongly irrelevant, we get speedups greater than a factor of 100.

It is important to note that we have the flexibility of building a query-tree at different levels of generality of the query schema, and thereby to achieve varying percents of irrelevant facts. For example, instead of building a query-tree for the query schema $goodPath(X, Y)$, we can build one for $goodPath(120, Y)$. Doing so will result in deeming additional facts irrelevant (e.g., $step(X, Y)$ for $100 < X < 120$ in this case). However, the indices created by this query-tree will be usable for a smaller set of queries. Consequently, in using the query-tree one should attempt to identify the most accurate characterizations of frequently occurring sets of queries.

Table 5
Changing the size of the database

Example 1			Example 3			Example 7		
KB size	Solution time		KB size	Solution time		KB size	Solution time	
	BC1/BC2	BC1/BC3		BC1/BC2	BC1/BC3		BC1/BC2	BC1/BC3
250	12.6	12.4	100	23	31	540	1.3	11.3
350	15	15	200	27	43	930	1	20
550	20	20	300	35	58	1300	1	24

5.2.2. *The number of ground facts in the knowledge base*

The second factor that affects the speedups that was suggested by the initial results is the number of ground facts in the original knowledge base. To test this effect, we ran each of the examples with databases containing a different number of ground facts. The results, shown in Table 5, show that the speedups increased as the size of the databases grew, even if the percent of irrelevant facts remained roughly the same. The growth can be explained by the fact that the cost of backward chaining is more than linear in the number of facts. Therefore, the effect of removing some constant percent of facts will be greater when the overall number of facts is greater. These results are significant in that they suggest that our methods will scale up to large knowledge bases and be even more effective there. (Recall that the cost of building the query-tree is independent of the number of ground facts.)

5.2.3. *Placement of interpreted literals in the rules*

A final factor affecting the speedups achieved from using the query-tree is the way the interpreted literals are placed in the rules. To illustrate, consider the following set of rules defining the existence of a flight (perhaps with stops) between two cities in the country subject to time constraints (given by the constants s_0 and e_0):

$$u_1 : p(X, Y, S_1, E_1) \wedge (s_0 \leqslant S_1) \wedge (e_0 \geqslant E_1) \Rightarrow \mathit{timelyConnect}(X, Y)$$
$$u_2 : \mathit{fl}(X, Y, S, E) \Rightarrow p(X, Y, S, E)$$
$$u_3 : \mathit{fl}(X, Z, S, T) \wedge (T \leqslant T_1) \wedge p(Z, Y, T_1, E) \Rightarrow p(X, Y, S, E)$$

To describe such paths, the rules can also be written as follows:

$$v_1 : p(X, Y, S_1, E_1) \Rightarrow \mathit{timelyConnect}(X, Y)$$
$$v_2 : \mathit{fl}(X, Y, S, E) \wedge (S \geqslant s_0) \wedge (E \leqslant e_0) \Rightarrow p(X, Y, S, E)$$
$$v_3 : \mathit{fl}(X, Z, S, T) \wedge (T \leqslant T_1) \wedge (S \geqslant s_0) \wedge (T \leqslant e_0) \wedge p(Z, Y, T_1, E) \Rightarrow p(X, Y, S, E)$$

The difference between the two sets of rules is that the second set is crafted to exploit the constraints entailed by the interpreted constraints on *timelyConnect*. Specifically, whenever we retrieve a flight fact that violates the constraints (i.e., ends later than e_0 or begins before s_0), we will immediately backtrack. In contrast, when using the first set of rules, we will compute all possible paths and check the constraints in the last step of the derivation. Consequently, when using the first set of rules, a strongly irrelevant

fact may be the root of an arbitrarily large tree, whereas when using the second set, no such tree will be generated. As a result, removing strongly irrelevant facts will have a greater effect for a set of rules like the first one. The experimental results confirm this observation. The example pairs 1 & 2 and 3 & 4 are instances of rules differing exactly in this fashion.

Several points should be noted with respect to this issue:

- Although the speedups are significantly bigger using the first set of rules in each pair, we still achieve significant savings even when the rules are carefully crafted such that the constraints are used to control the search.
- Writing rules with such built-in control has many disadvantages (see [19]). It is extremely difficult to write such rules in practice and is a very error-prone task. Consequently, we expect rules would usually be written without such crafting.
- Crafting a set of rules with such built-in control can however be done easily using the query-tree. Specifically, for every rule-node in the query-tree,we can create a new rule that includes the constraints in the label of that node. The resulting set of rules will be equivalent to the original set with respect to the query predicate (i.e., will produce the same answer regardless of the database of ground facts). However, using the new set, the tightest constraints will be enforced on the bindings immediately when they appear.

5.3. *Discussion*

It is clear from the experiments and from the theoretical analysis that the methods described for using the query-tree will be most useful when we are able to identify classes of queries that occur frequently. Moreover, the query-tree will find irrelevant facts only in cases in which interpreted predicates play a role. The experiments we described deal with knowledge bases with large number of ground facts and relatively small number of rules. As shown in Section 4, the time to build the query-tree is linear in the number of rules; therefore, the algorithms will scale up to cases with large number of rules. It should be noted that he indices computed by the query can be used in conjunction with existing indexing methods. In particular, the indices computed by the query-tree are tailored for a specific class of queries and take into consideration the semantics of the interpreted predicates. These indices can be refined using other considerations (e.g., selectivity).

The experiments described above were done with a depth first search backward chaining inference mechanism. However, the techniques we described can be applied to a wide range of reasoning mechanisms. The first use of the tree, the removal of strongly irrelevant facts, is independent of the reasoning scheme used. Following the query-tree can also be integrated easily into any reasoning mechanism. The only requirement is that nodes in the search space be associated with nodes in the query-tree. The particular order in which the space is searched is unimportant.

There are several possible schemes for integrating the construction of the query-tree and the indices with search for the solution in order to minimize the overhead associated with building the query-tree. First, the bottom-up phase of building the query-tree is independent of the query. Second, in the top-down phase, we can create a specialized

index for a relation only if it is actually referenced in the search. This way one can avoid creating indices that will not be used.[15]

Finally, it should be emphasized that the optimizations obtained by using the query-tree are orthogonal to those achieved by methods that derive optimal rule and goal orderings [42,84] or methods that use run-time bindings (combined with tabulation) to prune the search [8,9,93]. Whereas these methods look for optimal ways to search the space, the query-tree identifies parts of the space that are guaranteed not to contain solutions and is independent of any run-time bindings. Therefore, the results obtained in the experimental validation of the query-tree do not follow from other savings usually obtained from using such methods.

6. Other applications of relevance reasoning

So far we have focussed on how to use relevance reasoning to speed up inferences. In this section we briefly describe how our algorithms can be used in other applications of relevance reasoning.

Automatic creation of abstractions. Reasoning with multiple levels of abstraction is an effective method for reasoning in complex domains, and has been used in several fields of AI (e.g., planning [50,77], theorem proving [40,73] and constraint satisfaction [27]). An abstraction is obtained by removing some detail from the representation, such as collapsing a set of predicates into one denoting their union or projecting out arguments of relations. An abstraction will be useful if the detail removed is irrelevant to the particular query, i.e., any answer obtained in the abstract theory holds in the original one as well. Otherwise the system will have to backtrack between different representations. In [56] we describe a generalization of the framework discussed in this paper, where we consider different *subjects* of irrelevance, such as arguments of relations and distinctions between predicates. We also show how to use the query-tree to automatically create abstractions that are especially suited for a given class of queries, by automatically deriving irrelevance claims about such subjects.

Query optimization in database systems. Relevance reasoning is extremely important for query optimization in database systems. In most database systems, query evaluation proceeds in a bottom-up fashion, i.e., by evaluating subqueries before the query itself. Therefore, substantial savings are achieved by pruning as early as possible database tuples that cannot contribute to the solution of the query. The query-tree can be used for optimization of queries in relational and deductive systems. It tells us which tuples in the database are guaranteed to be irrelevant to the query; therefore, we can apply a filter to the database relations as a first step in query evaluation. Moreover, the query-tree can also be combined with other optimization methods such as magic-sets [8,9] and algorithms based on message-passing [91].

[15] Theoretically, we can interleave steps of reasoning with steps of building the query-tree, and in that way never perform worse than twice the optimal search.

A novel query optimization algorithm for SQL queries, *predicate move-around,* based on the query-tree is described in [58]. The predicate move-around[16] algorithm is an extension of the query-tree that deals with additional features of SQL such as aggregate queries (Min, Max, Avg), functional dependencies, and negation. The predicate move-around algorithm is a generalization of previous predicate push-down techniques for optimizing queries and is very easy to incorporate in existing query optimizers. The optimizations obtained by the predicate move-around algorithm are especially useful in the growing application area of decision support queries. Such queries are usually complex (built from many sub-queries), and interpreted constraints play a major role in them.

Information gathering in distributed heterogeneous environments. An important application of relevance reasoning is the integration of multiple information sources [4,31,38,60,90]. So called *mediator systems* provide access to a large number of information sources (such as databases, knowledge bases and text files). The mediator makes use of *descriptions* of the contents and the capabilities of the information sources (e.g., constraints on the property of data they contain). In order for such a system to answer queries effectively, it must be able, given a query, to efficiently decide which information sources contain data relevant to the query. Since the main cost in answering queries in such a setting is the cost of communication (either in time or monetary cost of access), it is crucial that the system access as few information sources as possible. The application of our framework and algorithms to such a setting is described in [65]. In [65] as in the TSIMMIS system [90], a mediator contains a set of Horn rules describing how to combine information from the various sources. In such an architecture, the information sources are viewed as containing the base relations, and their descriptions provide integrity constraints on these relations. Given a query, we use the query-tree to determine which base relations are contain tuples that are relevant to the query, and therefore which information sources need to be accessed.

7. Related work

7.1. Analysis of irrelevance

As mentioned earlier, the notion of irrelevance has been studied in the philosophy literature [16,34,47], but the thrust of these works was to analyze the common-sense notion of irrelevance and not its computational uses. A related concept discussed in the formal logic community is *relevance logics* (e.g., [3,5,24]). The key idea in relevance logics is to modify the logic and the inference rules such that only *relevant implications* can be made. However, two issues are still largely open in this field. The first is devising clean and intuitive semantics for these logics, and the second is providing tractable inference for them. In contrast, our analysis of irrelevance assumes that the underlying logic remains unchanged.

[16] In SQL terminology a *predicate* refers to a constraint in the query.

Within AI, the notion of irrelevance was used rather informally in various works, such as RLL [41] and compositional modeling [32]. Irrelevance was also investigated extensively in the context of probabilistic reasoning [21,23,70]. However, in that context, irrelevance has a natural definition based on the notion of conditional independence which does not carry over to the context of logical knowledge bases.

The work most related to ours is the analysis of irrelevance given by Subramanian [87,88]. Subramanian's motivations for analyzing irrelevance are similar to ours; namely, reformulating the knowledge base to create one that is simpler and will therefore lead to more efficient inference. However, her framework does not provide sufficient distinctions to enable one to analyze the issues of deriving irrelevance claims and the utility of doing so. Our framework can be viewed as a refinement of hers, where in addition to considering the form of the KB, we also consider the possible derivations that an inference mechanism can pursue. The specific definitions that she considers are formulated in our framework as variations of weak irrelevance. Subramanian also defined a class of *computational-irrelevance* claims whose exploitation leads to computational savings, but only gave some straightforward examples of such claims. Our class of strong irrelevance claims is a prime example of computational-irrelevance claims. It should be noted that in [87], a definition of strong-irrelevance is given. However, instances satisfying this definition are not necessarily instances of computational irrelevance. Finally, Subramanian discusses several algorithms for detecting irrelevance. However, these algorithms focus on the case of propositional logic KBs and require answering the query as part of the algorithm. She considers an extension of the algorithm to the first order case, using the concept of a *definability graph*. This graph shows only simple dependency relations between predicates in the KB (i.e., a predicate p_1 depends on p_2 if p_2 appears in a rule whose consequent is p_1), and therefore does not enable relevance reasoning beyond simple reachability tests.

7.2. *Static analysis of rules*

Several other authors have considered static analysis of rules for different purposes, such as explanation based learning [29,81], partial evaluation of logic programs [15, 67,85], automated reasoning [14,52], and deductive databases [86,89]. Some have also used graph-like representations of the rules, such as problem space graphs [29], connection graphs [52], compilation graphs [14], tree-automata [92] and rule/goal graphs [89]. Others have used rule folding/unfolding in their analysis.

In all these works, the common idea is to evaluate the rules over an abstract interpretation of the domain [20], i.e., to evaluate the rules over a domain consisting of abstract descriptions of possible domain elements. The key issue common to these works is when to terminate the application of the rules (i.e., when to terminate the the creation of the graph). The query-tree is novel in that it gives a well motivated termination criterion based on considering the semantics of interpreted predicates that appear in the rules. Moreover, the query-tree algorithm combines a forward chaining evaluation of the rules followed by a backward chaining evaluation of refined rules. Consequently, with the exception of [86], only the query-tree can be shown to be complete in more than

straightforward cases (i.e., in the presence of recursion and interpreted predicates). Recall that completeness guarantees that the query-tree encodes *precisely* the set of desired derivations.

A completeness result closely related to Theorem 11 was obtained independently by Srivastava and Ramakrishnan [86]. They describe an algorithm that uses fold/unfold transformations to obtain a rewritten set of rules in which the most tight constraint is computed for every relation mentioned in the rules. Their technique only applies to Horn rules with interpreted predicates, and does not extend to encoding other sets of derivations (such as minimal derivations and satisfiable derivations for rules with negated base predicates). Furthermore, they do not characterize the constraint languages for which their algorithm is guaranteed to be complete. Finally, the result of their algorithm is a new set of rules. While it is straightforward to generate these rules from the query-tree, the query-tree representation has several advantages over their transformed rule set. Most notably, the tree tells us *how* facts can be used in derivations, therefore enabling us to further focus the search of a backward-chainer by following relevant sequences of rule applications and database lookups (see Example 13). As seen in Section 5, such control leads to significant speedups. This finer level of control cannot be achieved using their rules.

Connection graphs [52] were also developed for the purpose of focusing a theorem prover by precomputing all the possible pairs of resolvable clauses. Clearly, if a certain clause appears in a component of the graph that is not connected to the component of the negation of the query, it can be removed from the KB (i.e., it is strongly irrelevant). However, connection graphs only capture a subset of the possible dependencies between clauses. Specifically, they only show that two clauses connected to a link are unifiable, but say nothing about the relationship between clauses connected via longer paths in the graph. Other work [18,83] has considered following only certain *walks* on the graph. However, these walks are not guaranteed to encode valid derivations, as are the paths encoded in the query-tree. Work on connection graphs has also not considered semantics of interpreted predicates.

The query-tree encodes exactly the space of derivations that an inference engine should search, thereby identifying parts of the space that can be safely ignored. Another approach to speeding up inference that was considered is finding optimal strategies for searching a given space [42,43,84]. The query-tree can be used to complement and extend these methods in two ways. First, by delimiting the actual space that needs to be searched, some search paths can be eliminated from consideration when looking for the optimal search strategy. Second, the methods described by Smith [84] and Greiner [42] require a graph-like representation of the possible derivations of the query. The query-tree provides such a representation which treats recursion and interpreted atoms in a principled way, unlike the representations that are currently used. Consequently, it can be used as a basis for extending such techniques to fully incorporate knowledge about interpreted atoms.

The goal of Explanation Based Learning [69] is also to speed up inferences. In EBL, new rules are added to the knowledge base that compress sequences of inference into a single rule. The key issue in this approach is the utility of the added rules [30,68]. Adding too many rules may have the inverse effect of slowing down inference. Etzioni

[29] has shown that much of the speedups obtained by EBL can be obtained by merely doing static analysis of the rules in the knowledge base. Using a tree-like representation of the rules in the knowledge base, called a *Problem Space Graph* (PSG), he showed how to glean from it new rules that were more effective than those learned by standard EBL techniques.

The problem space graph is similar in principle to the query-tree. However, it does not consider the semantics of interpreted literals in the rules. It also uses a very simple termination condition in the case of recursion; a node is not expanded if it is a variable renaming of one of its ancestors. A key difference between the PSG and the query-tree is that the decision whether to expand a node in the PSG depends not only on the node itself but also on its ancestry. As a result, the size of the PSG can be exponential in the number of rules, whereas the size of the query-tree may be exponential only in the arity of the predicates.

8. Conclusions

The ability of a system to identify and ignore irrelevant information is a key in scaling up knowledge based systems and to future knowledge based applications. This article showed that it is possible to reason effectively about relevance of knowledge in a principled manner, and that such reasoning can significantly impact the performance of knowledge based systems. As a basis for exploring the key questions regarding relevance reasoning, we presented a formal framework for analyzing irrelevance based on a proof-theoretic analysis of the notion. The framework enabled us to compare the properties of different definitions of irrelevance and shed light on previous definitions discussed in the literature. Within the space of definitions, we identified the class of strong irrelevance claims that has two desirable properties; namely, strong irrelevance claims can be efficiently derived automatically and are guaranteed to lead to savings in inference.

We investigated in detail the problem of automatically deriving irrelevance claims for Horn-rule knowledge bases. Our analysis was based on the observation that in order for relevance reasoning to be practical, we must derive irrelevance claims by considering only a small and stable part of the knowledge base, while not assuming anything about the unexamined parts. We considered the problem of automatically deriving irrelevance claims that were based only on the rules in the KB and were independent of the ground facts. As a result, our algorithms were efficient, and the irrelevance claims derived were independent of changes to the ground facts. We described a novel tool, the query-tree, for automatically deriving strong irrelevance claims. The query-tree is a finite structure that encodes precisely the set of derivations of the query, and therefore tells us exactly which rules and ground facts can appear in derivations of the query, thus providing the basis for a sound and complete inference procedure for several classes of strong irrelevance claims. One of the key aspects of the query-tree is that it considers the semantics of the interpreted literals that appear in the rules, which often enables the detection of additional interactions between the rules. Furthermore, the query-tree is a general technique that can be used to compute several variants of strong irrelevance.

Finally, we presented experimental results which showed that using the query-tree to filter out irrelevant facts often yields speedups of orders of magnitude, while the cost of building the query-tree is negligible. Both the theoretical analysis and the experimental results showed that our methods will scale up and be even more effective in larger knowledge bases.

There are several ways to extend this work. We mention only a couple here. One important extension is to incorporate probabilistic irrelevance claims into the framework, i.e., claims stating that some fact in the knowledge base is irrelevant to a query with some probability. A clear understanding of the meaning of such irrelevance claims is needed as well as algorithms for automatically deriving them and methods for exploiting them in inference. A second direction for future research is to consider extensions of the query-tree. One particular issue is to consider how to terminate the construction of the tree when rules contain recursion through function symbols. Although, in general, there is no termination condition that will guarantee completeness of the query-tree, it is important to find limited cases in which it does, and to find cases in which it captures most irrelevance claims encountered in practice. Recent work on termination [13, 79, 80] may be helpful in this context.

Acknowledgements

The authors would like to thank Ed Feigenbaum, Michael Genesereth, Matt Ginsberg and Pandu Nayak for many discussions about the material presented in this paper. Special thanks is due to Jim Rice, without whom the experimental results described in this paper would not be possible or meaningful.

Appendix A. Proofs

A.1. Properties of irrelevance

The properties of irrelevance given in Theorem 6 are proved as follows.

Proof of A0. This is an immediate corollary of the definitions. □

Proof of A1. Suppose $SI(\Phi, \psi, \Sigma, DI_i, \mathcal{D}_0)$ holds and let $\Delta \in \Sigma$. Therefore, for every derivation $D \in \mathcal{D}_0(\Delta)$, $DI_i(\Phi, D)$ holds and therefore, by the assumption of the claim, $DI_j(\Phi, D)$ holds. Consequently, $DI_j(\Phi, D)$ holds for every $D \in \mathcal{D}_0(\Delta)$, and so $SI(\Phi, \psi, \Sigma, DI_j, \mathcal{D}_0)$ holds. The proof for WI is similar. □

Proof of A2. The part about strong irrelevance follows from the observation that if $DI(\Phi, D)$ holds for all derivations $D \in \mathcal{D}_2(\Delta)$, then $DI(\Phi, D)$ holds also for all derivations $D \in \mathcal{D}_1(\Delta)$. For weak irrelevance, the claim follows from the observation that if a property holds for some derivation in $\mathcal{D}_1(\Delta)$, it will obviously hold for some derivation in $\mathcal{D}_2(\Delta)$. □

Proof of A3, A4. These are immediate consequences of the definitions. □

Proof of A5. This follows from the observation that if $\phi_1 \equiv \phi_2$, then $Base(D) \not\models \phi_1$ implies $Base(D) \not\models \phi_2$. □

Proof of A6. This follows immediately from the definitions. □

Proof of A7. Suppose $SI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge SI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$ holds, and let $\Delta \in \Sigma$, and D be a derivation in $\mathcal{D}_0(\Delta)$. Since both $DI(\Phi_1, D)$ and $DI(\Phi_2, D)$ hold, also $DI(\Phi_1 \cup \Phi_2, D)$ holds. Because this holds for any $D \in \mathcal{D}_0(\Delta)$ and $\Delta \in \Sigma$, then $SI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$ holds. The proof for the second claim is similar. We simply consider the derivation D for which $DI(\Phi_2, D)$ holds, and $DI(\Phi_1 \cup \Phi_2, D)$ will hold. □

Proof of A8. The implication from right to left is immediate. For the other direction, suppose that $SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}^{\psi}(\Delta))$ holds and suppose in contradiction that D is a derivation of ψ from $\Delta \cup \tau$ such that $\phi \in D$ and $\phi \in \Phi$. We create a derivation D' of ψ from Δ such that $\tau \in D'$. The only modification to D is to replace every appearance of τ as a leaf in the proof tree by the derivation of τ from Δ. The result is a derivation of ψ from Δ that includes ϕ. Consequently, $SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}^{\psi}(\Delta))$ does not hold. □

A.2. Decidability of irrelevance

The following shows that weak irrelevance is undecidable even for function-free Horn rules (i.e., datalog):

Lemma A.1. *Let $\mathcal{P}$ be a set of function-free Horn rules and ψ be an atomic query. Determining whether $WI(\phi, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{\psi})$ holds when ϕ is either a rule or a ground atomic fact is undecidable even if the rules have no interpreted predicates.*

Proof. We say that a rule r is redundant in $\mathcal{P}$ if, for any set of ground facts G, the set of ground atomic facts derivable from $\mathcal{P} \cup G$ is the the same as the set derivable from $\{\mathcal{P} - r\} \cup G$. Let $r \in \mathcal{P}$ and ψ be a query. We prove the lemma by showing that the claim $WI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{\psi})$, when r is a rule, holds if and only if r is redundant in $\mathcal{P}$. An analogous proof can be given for the case that ϕ is a ground fact. In proof, suppose that $WI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{\psi})$ holds, then for any knowledge base $\Delta \in \Sigma_{\mathcal{P}}$, $WI(r, \psi, \Delta, DI_2, \mathcal{D}^{\psi})$ holds. Therefore, if there is a derivation of ψ, then there is one that does not use r. Consequently, r can be removed from $\mathcal{P}$ without changing the answer to ψ, regardless of the ground facts in the KB, and therefore, r is redundant. Conversely, if r is redundant, that means that for every $\Delta \in \Sigma_{\mathcal{P}}$, if ψ is derivable, there is a derivation that doesn't contain r. Therefore, $WI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{\psi})$ holds.

However, it follows from [82] that determining redundancy of rules is undecidable for datalog theories even without interpreted predicates. Therefore, weak irrelevance is undecidable. □

The next lemma shows that strong irrelevance is undecidable when we allow the rules to have stratified negation. In our discussion, we assume perfect model semantics of the rules (cf. [89]).[17]

Lemma A.2. *Let $\mathcal{P}$ be a set of function-free Horn rules with stratified negation and $r \in \mathcal{P}$. Determining whether $SI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}^{\psi})$ is undecidable, even if $\mathcal{P}$ has no interpreted predicates.*

Proof. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be sets of function-free Horn rules and let p_1 and p_2 be distinguished query predicates in $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. A set of rules $\mathcal{P}_1$ is said to *contain* a set of rules $\mathcal{P}_2$ (denoted by $\mathcal{P}_1 \supseteq \mathcal{P}_2$) if, for any given set of ground atomic facts G for the base predicates, the set of tuples derived in the relation p_1 from $\mathcal{P}_1 \cup G$ contains the set of tuples derived in the relation for p_2 from $\mathcal{P}_2 \cup G$. The two sets of rules are said to *equivalent* if $\mathcal{P}_1 \supseteq \mathcal{P}_2$ and $\mathcal{P}_2 \supseteq \mathcal{P}_1$. Testing equivalence of two function-free sets of rules is undecidable [82]. We will reduce the equivalence problem to the irrelevance problem of a rule in a set of rules with stratified negation, i.e., we show that if there is an algorithm for deciding whether a rule r in a function-free set of rules $\mathcal{P}$ with stratified negation is strongly irrelevant, then we can design an algorithm for testing equivalence of two sets of function-free rules.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two sets of rules with query-predicates p_1 and p_2. Without loss of generality we can assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ have distinct sets of derived predicates. To test equivalence, it is enough to test whether $\mathcal{P}_1 \supseteq \mathcal{P}_2$ and $\mathcal{P}_2 \supseteq \mathcal{P}_1$. Let Q be the set of rules containing the rules of $\mathcal{P}_1$ and $\mathcal{P}_2$ and the rule

$$r : p_1(\bar{X}) \wedge \neg p_2(\bar{X}) \Rightarrow q(\bar{X})$$

where q is the query predicate of Q and it appears nowhere in $\mathcal{P}_1$ or $\mathcal{P}_2$. Note that Q is a stratified set of rules, since r is the only rule containing negation. Clearly, r is strongly irrelevant to q if and only if $\mathcal{P}_2 \supseteq \mathcal{P}_1$, since r will be used in a derivation if and only if there is some set of ground facts in which some ground fact is a member of p_1 and not of p_2. In a similar fashion, we can create a set of rules with a rule r' which will be strongly irrelevant if and only if $\mathcal{P}_1 \supseteq \mathcal{P}_2$. Consequently, if rule irrelevance is decidable for rules with stratified negation, then equivalence of sets of function-free rules will be decidable. □

A.3. Proof of Theorem 11

In the proof we assume without loss of generality that the query is of the form $q(\bar{X})$, where $\bar{X}$ is a tuple of distinct variables. If the query originally contained a constraint formula c_q, we simply add a rule:

$$q(\bar{X}) \wedge c_q \Rightarrow q'(\bar{X})$$

and query $q'(\bar{X})$.

[17] The perfect model of a set of rules is the one computed in a bottom-up fashion, stratum by stratum.

We will prove the theorem by considering the symbolic derivations encoded by the query-tree. Every derivation of the query can be viewed as a pair (d, θ), where d is a symbolic derivation and θ is an assignment to the variables of d. A symbolic derivation tree represents all the derivation trees that can be obtained by an assignment to its variables that satisfies (1) the interpreted literals in the rules, and (2) the integrity constraints on the base relations. Formally, these constraints are the following:

Let d be a symbolic derivation that includes the rule-nodes $r_1, \ldots, r_m$, and let $c_i(\bar{U}_i)$ be the conjunction of the literals of interpreted predicates in the rule-node r_i. Let $l_1(\bar{X}_1), \ldots, l_k(\bar{X}_k)$ be the leaves of d, and ic_i be the integrity constraints on the relation in l_i. A symbolic derivation tree d therefore represents the set of derivations obtained by assignments to its variables satisfying the following constraint formula, c_d:

$$c_d = c_1(\bar{U}_1) \wedge \cdots \wedge c_m(\bar{U}_m) \wedge ic_1(\bar{X}_1) \wedge \cdots \wedge ic_k(\bar{X}_k).$$

We denote the relation represented by the formula c_d by R_d. We can define a label for every node $g \in d$ by a formula expressing the projection of R_d onto the variables in g. We denote this label by $L_{sat}(g)$. Note that given an instance of d, (d, θ), the restriction of θ to the variables of a node g in d will satisfy $L_{sat}(g)$. We denote the set of symbolic derivation trees of $q(\bar{X})$ such that c_d is satisfiable by Π_{sat}.

Our proof will be based on showing that the query-tree encodes exactly the symbolic derivations of Π_{sat}. The correspondence between the derivations of the query and the symbolic derivations in Π_{sat}, shown in the lemma below, entails that it suffices to show that T encodes precisely Π_{sat}.

Lemma A.3.

(1) *A ground fact $p(a_1, \ldots, a_n)$ can appear in a derivation of an instance of the query $q(\bar{X})$ (from some set of ground facts for the base predicates) if and only if there is some node $g = p(X_1, \ldots, X_n)$ in a symbolic derivation $d \in \Pi_{sat}$ such that $a_1, \ldots, a_n$ satisfies $L_{sat}(g)$.*

(2) *A rule $r \in \mathcal{P}$ can appear in a derivation of an instance of the query $q(\bar{X})$ if and only if some symbolic derivation $d \in \Pi_{sat}$ includes a rule-node containing the rule r.*

Proof. To prove (1), suppose there exists a symbolic derivation tree $d \in \Pi_{sat}$ and $a_1, \ldots, a_n$ satisfies $L_{sat}(g)$, i.e., the projection of c_d on the variables $X_1, \ldots, X_n$ of the goal-node g. Therefore, there is some mapping θ of the variables of d to constants such that $\theta(X_i) = a_i$, for $1 \leqslant i \leqslant n$, and such that applying θ to d will satisfy the constraint c_d. Consider the set of ground facts for the base predicates, G_0, obtained by applying θ to the leaves of d. Applying θ to d will yield a derivation of an instance of $q(\bar{X})$ that uses $p(a_1, \ldots, a_n)$ from the ground facts G_0. Conversely, suppose there is a derivation d_0 of an instance of $q(\bar{X})$ that uses $p(a_1, \ldots, a_n)$ from a set of ground facts G. The derivation d_0 can be represented as a pair (d, θ), and if g is the node in d corresponding to $p(a_1, \ldots, a_n)$, then $a_1, \ldots, a_n$ will satisfy $L_{sat}(g)$.

(2) follows from the observation that a symbolic derivation will have exactly the same rules as its ground derivation instances. □

To show that T encodes precisely the derivations Π_{sat}, we proceed in two steps. First, we show that given a symbolic derivation tree d, the labels $L_{sat}(g)$ of a node g in d can be computed in a two phase fashion (bottom-up followed by top-down) (Lemma A.4). We then show that the construction of the query-tree exactly mimics this computation for all symbolic derivation trees (Observation A.5 and Lemma A.6). Specifically, consider the following procedure applied to the nodes of a symbolic derivation tree d (the expressions $c_0(g)$, $c_b(g)$, and $c_f(g)$ denote constraint formulas on the variables appearing in the node g).

(1) Initialize: Compute $c_0(g)$ as follows:
$c_0(l_i) = ic_i(\bar{X}_i)$ for the leaves $l_1(\bar{X}_1), \ldots, l_k(\bar{X}_k)$ of d.
if g is an internal goal-node **then** $c_0(g) = TRUE$.
if r is the rule-node r_i **then** $c_0(r) = c_r$.

(2) Bottom-up: Compute $c_b(g)$ as follows:
if g is a leaf goal-node in d **then** $c_b(g) = c_0(g)$.
if r is a rule-node with children $g_1, \ldots, g_m$ **then**
$c_b(r) = c_0(r) \wedge c_b(g_1) \wedge \cdots \wedge c_b(g_m)$.
if g is an internal goal-node with child rule-node r **then**
$c_b(g)$ = Projection of $c_b(r)$ on the variables of g.

(3) Top-down: Compute $c_f(g)$ as follows:
$c_f(root(d)) = c_b(root(d))$.
if r is a rule-node with father goal-node g and children $g_1, \ldots, g_m$ **then**
$c_f(r) = c_f(g) \wedge c_b(r)$.
for $1 \leqslant i \leqslant m$, $c_f(g_i)$ = Projection of $c_f(r)$ on the variables of g_i.

Lemma A.4. *Let d be a symbolic derivation tree. For every node $g \in d$, $c_f(g) = L_{sat}(g)$.*

Proof. The relation R_c representing the tuples satisfying the constraint formula c has one attribute for every variable in c. In the proof it is more convenient to refer to the relations represented by the constraint formulas, rather than to the formulas themselves. The conjunction of two constraints c_1 and c_2 is represented by the *join* of their corresponding relations, denoted by $R_{c_1} \bowtie R_{c_2}$. Recall that since the constraint language $\mathcal{L}$ satisfies the Closure property, we can express a join of two relations, and a projection of a relation on a subset of its variables as a sentence in $\mathcal{L}$. We denote the relations represented by $c_0(g), c_b(g)$ and $c_f(g)$ by $R_0(g), R_b(g)$ and $R_f(g)$ respectively. We denote the projection of a relation R on a subset of its attributes $\bar{X}$ by $R|_{\bar{X}}$. $R|_g$ denotes the projection of R on the attributes appearing in the node g.

Let $r_1, \ldots, r_l$ be the top-down ordering of the rule-nodes in d that was used in the top-down phase of the computation of the c_f labels, and $l_1, \ldots, l_k$ be the leaves of d. Recall that the global constraint on d is $c_d = c_0(r_1) \wedge \cdots \wedge c_0(r_l) \wedge c_0(l_1) \wedge \cdots \wedge c_0(l_k)$, or in notation of relations, $R_d = R_0(r_1) \bowtie \cdots \bowtie R_0(r_l) \bowtie R_0(l_1) \bowtie \cdots \bowtie R_0(l_k)$. We define a sequence of relations $S_0, \ldots, S_l$ as follows:

- $S_0 = R_b(root(d))$
- $S_i = S_{i-1} \bowtie R_b(r_i)$

We prove that the following properties hold for the sequence:

D1. $S_l = R_d$, i.e., the final relation in the sequence is the same as the global constraint on d.

D2. If $\bar{X}_i$ is the set of variables that appear in S_i, then

D2.1. $S_{i+1}|_{\bar{X}_i} = S_i$, and

D2.2. $S_i|_{r_i} = R_f(r_i)$.

This means that once a set of attributes appears in an intermediate relation in the sequence, the projection of the subsequent relations in the sequence on these attributes does not change.

The lemma follows from properties D1 and D2 as follows. Let $\bar{X}_i$ be the variables that appear in the rule-node $r_i \in d$. It follows from D2 that $R_f(r_i) = S_l|_{\bar{X}_i}$, and by D1, that $S_l = R_d$. Therefore, $R_f(r_i) = R_d|_{\bar{X}_i}$ holds which is the way $L_{sat}(r_i)$ is defined.

Proof of D1. Note that $S_l = R_b(r_1) \bowtie \cdots \bowtie R_b(r_l)$ (i.e., the join of the constraints obtained in the bottom-up phase). Therefore, it is enough to show that $R_d = R_b(r_1) \bowtie \cdots \bowtie R_b(r_l)$. To show this we prove that for every rule-node r in the tree and goal-node g the following hold:

(a) $R_b(r) \subseteq R_0(r)$, $R_b(g) \subseteq R_0(g)$, and

(b) $R_b(r) \supseteq R_d|_r$, $R_b(g) \supseteq R_d|_g$.

For each j, $1 \leqslant j \leqslant k$ there exists an i, $1 \leqslant i \leqslant l$ such that r_i is the father of l_j and therefore $R_b(r_i)|_{l_j} \subseteq R_0(l_j)$. Therefore, since $R_d = R_0(r_1) \bowtie \cdots \bowtie R_0(r_l) \bowtie R_0(l_1) \bowtie \cdots \bowtie R_0(l_k)$, (a) gives us that $S_l \subseteq R_d$. From the properties of the join operation and from (b) we get $S_l \supseteq R_d$. Hence, $S_l = R_d$.

Note that (a) and (b) hold trivially for the leaves of d. Furthermore, if g is the father goal-node of r, then the second parts of (a) and (b) follow from their respective first parts, since the relation for g is the projection of the corresponding relation for r on the variables in g. Therefore, it is enough to prove the claim for the rule-nodes in d, and we do so by induction on the elements of $r_1, \ldots, r_l$ in reverse order (i.e., the bottom-up order).

The base case, including all the rule nodes whose children are all leaves, follows immediately from the definitions.

Assume (a) and (b) hold for all rule nodes $r_{i+1}, \ldots, r_l$. We need to show that it holds for r_i. Claim (a) holds because $R_b(r_i)$ is the join of $R_0(r_i)$ with other relations (specifically, the bottom-up labels of its children). To prove (b), let $g_1, \ldots, g_m$ be the children of r_i. By the induction assumption, $R_b(g_i) \supseteq R_d|_{g_i}$ for each g_i. Clearly, by the definition of R_d, $R_0(r_i) \supseteq R_d|_{r_i}$. Therefore, since

$$R_b(r_i) = R_0(r_i) \bowtie R_b(g_1) \bowtie \cdots \bowtie R_m(g_m),$$

it follows that $R_b(r_i) \supseteq R_d|_{r_i}$. □

Proof of D2. By induction on i.

For the base case $i = 0$, we note that R_1 is simply $R_b(r_1)$. Therefore, since the $R_b(root(d))$ is the projection of $R_b(r_1)$ on the variables of $root(d)$, D2.1 holds. Moreover, since $R_f(r_1) = R_b(r_1)$, D2.2 holds too.

We assume D2 holds for all $j \leqslant i$, and prove that it holds for $i+1$. We first prove D2.2. Let g be the father of r_{i+1} and r_{i_0} be the father of g (where $i_0 \leqslant i$). By the inductive assumption, $S_{i_0}|_{r_{i_0}} = R_f(r_{i_0})$, and $S_i|_{r_{i_0}} = R_f(r_{i_0})$. Therefore, the same holds for the goal node g, i.e., $S_i|_g = R_f(g)$. Moreover,

$$R_f(r_{i+1}) = R_f(g) \bowtie R_b(r_{i+1}) = S_i|_g \bowtie R_b(r_{i+1}).$$

However, since the variables common to r_{i+1} and to S_i are only those in g, the join and the projection commute, i.e.,

$$R_f(r_{i+1}) = (S_i \bowtie R_b(r_{i+1}))|_{r_{i+1}} = S_{i+1}|_{r_{i+1}}.$$

To show D2.1, it is enough to show that $S_i|_g = S_{i+1}|_g$, since the variables appearing in g are the only ones common to S_i and $R_b(r_{i+1})$. Equivalently, because of D2.2, it suffices to show

$$R_f(r_{i+1})|_g = R_f(g). \tag{A.1}$$

The proof is based on the following property of relational algebra:

If $A = R|_{\bar{X}}$, *and* $B \subset A$ *then* $(R \bowtie B)|_{\bar{X}} = B$.

In our case,

$$R_b(g) = R_b(r_{i+1})|_g. \tag{A.2}$$

Clearly, $R_f(r) \subseteq R_b(r)$ and $R_b(r)|_g \subseteq R_b(g)$, and therefore, since $R_f(g) = R_f(r)|_g$ it follows that

$$R_f(g) \subseteq R_b(g). \tag{A.3}$$

Finally, recall that

$$R_f(r_{i+1}) = R_f(g) \bowtie R_b(r_{i+1}). \tag{A.4}$$

Therefore, the above observation together with Eqs. (A.2), (A.3) and (A.4) entails (A.1). □

The importance of Lemma A.4 is that we can now show that the construction of the query-tree exactly mimics the two phase computation of labels. The following observation shows that in the bottom-up phase of constructing the query-tree the adorned predicates we compute are all the possible labels of the form $c_b(g)$ in symbolic derivation trees. The observation follows by a simple bottom-up induction on the nodes in a symbolic derivation tree d:

Observation A.5. *Let d be a satisfiable symbolic derivation.*

- *For every goal-node $g \in d$, of the predicate p, where c is the standard form of $c_b(g)$ (using the arguments of p), p^c is an adorned predicate in $\mathcal{P}_1$.*

- *Suppose r is a rule-node in d containing the rule*

$$q_1(\bar{X}_1) \wedge \cdots \wedge q_m(\bar{X}_m) \wedge c_r \Rightarrow p(\bar{X}).$$

Suppose r's father is g and children are $g_1, \ldots, g_m$, then the following rule is in $\mathcal{P}_1$:

$$r' : q_1^{c_b(g_1)}(\bar{X}_1) \wedge \cdots \wedge q_m^{c_b(g_m)}(\bar{X}_m) \wedge c_b(r) \Rightarrow p^{c_b(g)}(\bar{X}).$$

Note that r and r' differ only in predicate names, but have the same variable pattern.

Given this observation, the following lemma will show that the query-tree encodes exactly the derivations in Π_{sat}. Theorem 11 follows from this lemma and Lemma A.3.

Lemma A.6. *A node g with label $L_{sat}(g)$ appears in some symbolic derivation tree in Π_{sat} if and only if there exists a node g_1 in the query-tree T and an isomorphism ϕ : Variables$(g) \rightarrow$ Variables(g_1), such that $\phi(g) = g_1$ and $\phi(L_{sat}(g))$ is equivalent to the label of g_1 in T.*

Proof. (*if*) To show the *if* direction, we show that every symbolic derivation tree $d \in \Pi_{sat}$ is encoded by T using an encoding mapping ψ, such that in addition to the conditions of Definition 9, ψ is also *label-preserving*, i.e.,

- If θ is the variable isomorphism between g and $\psi(g)$,[18] $L(\psi(g)) = \theta(L_{sat}(g))$ for every node $g \in d$.

Let d be a symbolic derivation tree in Π_{sat}. We show that d is encoded by the query-tree by mimicking the execution of procedure **build-forest**. We construct the encoding mappings ψ of the nodes as we go along.

We begin with the root of d and its child rule-node r. Let $c = c_b(root(d))$. Observation A.5 entails that q^c is one of the adorned predicates in $\mathcal{P}_1$, and that there is a rule r_1 in $\mathcal{P}_1$ that is a refinement of the rule in r, and its consequent is q^c. Therefore, procedure **build-forest** will create a node g of the form $q^c(\bar{X})$ with label c, and it will expand g with the rule r_1, creating the child goal-nodes $g_1, \ldots, g_m$. The label of the rule-node will be $c_b(r_1)$, and the labels of $g_1, \ldots, g_m$ will be the respective projections of $c_b(r_1)$ on the variables in g_i's.

We define ψ to map $root(d)$ to g, to map r to the child of g formed by the expansion with r_1, and to map the children of r to the respective children of $\psi(r)$, $g_1, \ldots, g_m$. Since $L_{sat}(root(d)) = c_b(root(d))$ and $L(g) = c$, the mapping ψ is label preserving for $root(d)$, and similarly for its child rule-node. By Lemma A.4 it follows that ψ is label preserving also for the children of r (because the label of its children is computed in exactly the same way as the label of their image in T).

We proceed by induction on a top-down ordering of rule-nodes in d, $r_1, \ldots, r_n$. We assume by induction that we have built a label-preserving encoding mapping ψ for all the rule-nodes $r_1, \ldots, r_{i-1}$, and their children goal-nodes. Furthermore, our induction

[18] Recall that we assumed that all literals in rules have distinct variables in every argument position. Therefore, there is an isomorphism θ between the variables in g and $\psi(g)$, such that $\psi(g) = \theta(g)$.

assumption also states that the goal-node $\psi(g)$ in the query-tree is a node of the predicate $p^{c_b(g)}$ (before the adornments are removed in the end of procedure **build-forest**). Note that this assumption holds for the base case.

Let g be the father of r_i in d, and assume that g is a node of the predicate p. By the inductive assumption, $\psi(g)$ is a node of the predicate $p^{c_b(g)}$. Consider first the case in which $\psi(g)$ was expanded in the query-tree. It would have been expanded with the rule

$$r' : q_1^{c_1}(\bar{X}_1) \wedge \cdots \wedge q_m^{c_m}(\bar{X}_m) \wedge c_b(r_i) \Rightarrow p^{c_b(g)}(\bar{X})$$

where $c_1, \ldots, c_m$ are the bottom-up labels of the children of g, and r' is a refinement of the rule in r_i. Denote the resulting rule-node in the query-tree by r. The mapping ψ will map r_i to the node r and the subgoals of r_i to the subgoals of r (therefore satisfying condition E0 of Definition 9). Note that after removing the adornments from nodes in the query-tree, the rule in r_i will be exactly the same as in $\psi(r_i)$ (as required by condition E1). Furthermore, condition E3 is also satisfied by ψ. As in the base case, ψ will be label-preserving on r_i and its children because the computation of $L(\psi(r_i))$ mimics the computation of $L_{sat}(r_i)$ and its children. Finally, note that if g is a child of r_i and is a node of the predicate p, then $\psi(g)$ will be a node of the predicate $p^{c_b(g)}$, as required by our inductive assumption.

In the case that $\psi(g)$ was not expanded in the query-tree, it would be because $Eq(\psi(g))$ is expanded. In that case, r would be a child of $Eq(\psi(g))$, and the children of r_i will be mapped to the children of r. Because $L(\psi(g))$ is isomorphic to $L(Eq(\psi(g)))$ (and they are goal-nodes of the same refined predicate), the proof follows in the same way as in the previous case, except that E3 is satisfied second clause in its definition.

(*only if*) We prove this part by considering the possible *embeddings* of symbolic derivations in the query-tree T. An embedding is a possible image of an encoding mapping for some symbolic derivations. Every embedding can be constructed as follows. We begin with one of the roots in the forest. For every goal-node of a derived predicate in the embedding, we choose either one of its child rule-nodes (if it was expanded) or one of the children of its expanded equivalent (if it was not expanded). For every rule-node, we choose its children. Only goal-nodes of base predicates remain without children in an embedding.

Therefore, to prove the claim, we show that for every embedding d' in T, there exists a symbolic derivation $d \in \Pi_{sat}$ such that d' is the image of a label-preserving encoding mapping ψ from d to T. To see that this holds, simply consider the symbolic derivation d that has exactly the same structure as d' (i.e., the same structure of rules). Following the proof of the if-direction will show that d is encoded by the query-tree, and that its image is precisely d'. Furthermore, since the query-tree does not contain nodes with unsatisfiable labels, d must be a symbolic derivation in Π_{sat}. □

A.4. *Proof of Theorem 12*

The theorem is proved by reducing the acceptance problem of a linear-space alternating Turing machine (ATM) [17] to the problem of finding irrelevance of rules.

The execution of an ATM is described by a sequence of *instantaneous descriptions*, id's, each describing the state of the machine at consecutive stages of the execution, i.e., the contents of the input tape, the location of the head and the state of the machine. An ATM is similar to a Turing machine, except that its transition function gives a pair of moves for each combination of state and symbol. Furthermore, every state is either an *and-state* or an *or-state*. If q is an and-state, then an id having state q leads to acceptance of the input if both its successors lead to acceptance. If q is an or-state, then an id having state q leads to acceptance if either one of its successors leads to acceptance. The states of the machine alternate in the sense that the successors of an and-state are or-states, and the successors of an or-state are and-states.

Instantaneous descriptions (id's) are represented by a symbol for every cell on the tape. The symbol can either be a symbol in the alphabet, or a composite symbol (s, b) including an alphabet symbol b and a state of the machine s. In a legal id, all cells but one contain the alphabet symbol that is on the tape in that state, and the cell on which the head is placed contains a composite symbol containing the alphabet symbol in that cell and the internal state of the machine. The union of the alphabet symbols and composite symbols is denoted by B.

The reduction is based on representing id's as tuples of a predicate *id*, whose arity is linear in the size of the input tape, n. Each cell in the id is represented by a block of variables of size $|B|$. The variable X appears in the block in the position corresponding to the symbol appearing in the cell (we assume some arbitrary ordering on the elements of B). All other columns contain the variable W. Thus the arity of the predicate *id* is $|B|n$. The tuples $\boldsymbol{X}_i$ are used to denote blocks of variables corresponding to one cell. The tuple $\boldsymbol{U}_i$ denotes a block of variables representing a cell with the symbol i, (i.e., X appears in the position of i in B and all other positions are W).

Intuitively, we construct a set of rules, whose bottom-up computation emulates the computation of M backward from the accepting id to the initial one. The atom $id(\bar{X})$ is derivable from the rules if and only if $\bar{X}$ describes a legal id and leads to acceptance. Given an ATM, M, and an input X_{init}, we construct a set of rules as follows. First we need rules representing transitions between consecutive states. For example, suppose $\delta(c, q) = \{(d_1, s_1, R), (d_2, s_2, L)\}$ is a transition of M, that is, if the machine is in state q and the symbol c is under the head of the tape, then one of the successor states is obtained by writing the symbol d_1 on the tape, moving to state s_1 and moving the head to the right. If q is an or-state, then for every i $(1 \leqslant i \leqslant n)$ and every input symbol b, we have the following rules:

$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-1}, \boldsymbol{U}_{d_1}, \boldsymbol{U}_{(s_1,b)}, \boldsymbol{X}_{i+2}, \ldots, \boldsymbol{X}_n) \Rightarrow$$
$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-1}, \boldsymbol{U}_{(c,q)}, \boldsymbol{U}_b, \boldsymbol{X}_{i+2}, \ldots, \boldsymbol{X}_n)$$
$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-2}, \boldsymbol{U}_{(s_2,b)}, \boldsymbol{U}_{d_2}, \boldsymbol{X}_{i+1}, \ldots, \boldsymbol{X}_n) \Rightarrow$$
$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-2}, \boldsymbol{U}_b, \boldsymbol{U}_{(c,q)}, \boldsymbol{X}_{i+1}, \ldots, \boldsymbol{X}_n)$$

If q is an and-state then for every i, $(1 \leqslant i \leqslant n)$ and every pair of input symbols b_1, b_2, the theory contains the following rule:

$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-2}, \boldsymbol{U}_{b_1}, \boldsymbol{U}_{d_1}, \boldsymbol{U}_{(s_1,b_2)}, \boldsymbol{X}_{i+2}, \ldots, \boldsymbol{X}_n) \wedge$$
$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-2}, \boldsymbol{U}_{(s_2,b_1)}, \boldsymbol{U}_{d_2}, \boldsymbol{U}_{b_2}, \boldsymbol{X}_{i+2}, \ldots, \boldsymbol{X}_n) \Rightarrow$$
$$id(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-2}, \boldsymbol{U}_{b_1}, \boldsymbol{U}_{(c,q)}, \boldsymbol{U}_{b_2}, \boldsymbol{X}_{i+2}, \ldots, \boldsymbol{X}_n).$$

To complete the set of rules, a few more rules are necessary. Denote by $\boldsymbol{X}_{final}$ the tuple representing M's accepting id (which we can assume, without loss of generality is unique), and by $\boldsymbol{X}_{init}$ the tuple representing the initial id. R1 places the initial state as a subgoal of the query:

R1: $id(\boldsymbol{X}_{init}) \Rightarrow p(X, W)$

R2 and R3 will lead from the accepting state to an EDB node. Note that e is the only EDB predicate.

R2: $a(X, W) \Rightarrow id(\boldsymbol{X}_{final})$
R3: $(X \neq W) \wedge e(X, W) \Rightarrow a(X, W)$

We denote the set of rules by $\mathcal{P}$. The following theorem establishes the correctness of the reduction.

Theorem A.7. *$SI(\mathrm{R1}, p, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_p)$ holds if and only if M does not accept its input with the initial state X_{init}.*

Proof. We first note that any derivation of $p(X, W)$ will contain only two constants. This follows from the fact that in all rules, all variables appearing in the body of the rule appear in the head too. Therefore, the only constants in the derivation will be those assigned to X and W. Furthermore, since a derivation must include the rule R3, these constants must be distinct. We will refer to them as x and w hereafter. Moreover, if x and w are distinct, a ground instance of an *id* subgoal can be unified with the head of a rule only if both the ground instance and the head represent the same id (i.e., the ground instance is obtained from the head by substituting x for X and w for W). This can be seen by considering each block in the *id*. If it differs from the ground instance in a position of the X variable (or if one of them does not contain exactly one occurrence of X), it will force X and W to unify.[19] Therefore, because of the way the transition rules are written, the subgoals of any *id* node are the instantaneous descriptions of its successor states. Consequently, if the top *id* goal-node in a derivation is the node describing the initial state, then every partial derivation of p describes a possible execution tree of the ATM M. Therefore, because the only way to get to an EDB subgoal is through rules R2 and R3, every derivation of p must describe an accepting execution trace of M. Therefore, if p has some derivation, then there then there is execution of M that will accept $\bar{X}_{init}$.

Conversely, suppose M accepts its input. A simple trace of the machine's execution will produce a symbolic-derivation of $p(X, W)$ which must contain R1. □

[19] This assumes each block is at least of size 3. If this does not hold, we simply add another dummy column to each block, and leave it unchanged in all the rules.

References

[1] S. Abiteboul and R. Hull, Data functions, datalog and negation, in: *Proceedings ACM SIGMOD 1988 International Conference on Management of Data* (1988).

[2] S. Addanki, R. Cremonini and J. Penberthy, Reasoning about assumptions in graphs of models, in: *Proceedings IJCAI-89*, Detroit, MI (1989).

[3] A.R. Anderson and N.D. Belnap, *Entailment* (Princeton University Press, Princeton, NJ, 1975).

[4] Y. Arens, C.A. Knoblock and W.-M. Shen, Query reformulation for dynamic information integration, *Internat. J. Intelligent and Cooperative Information Systems* 6 (2/3) (1996) 99–130.

[5] A. Avron, Whither relevance logic?, *J. Philos. Logic* 21 (1992) 243–281.

[6] F. Baader and B. Hollunder, A terminological knowledge representation system with complete inference algorithm, in: *Proceedings Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence (Springer, Berlin, 1991) 67–86.

[7] F. Bacchus, A.J. Grove, J.Y. Halpern and D. Koller, Statistical foundations for default reasoning, in: *Proceedings IJCAI-93*, Chambery, France (1993).

[8] F. Bancilhon, D. Maier, Y. Sagiv and J. Ullman, Magic sets and other strange ways to implement logic programs, in: *Proceedings 5th ACM Symposium on Principles of Database Systems* (1986) 1–15.

[9] C. Beeri and R. Ramakrishnan, On the power of magic, in: *Proceedings 6th ACM Symposium on Principles of Database Systems* (1987) 269–283.

[10] J.A. Blakeley, N. Coburn and P.A. Larson, Updating derived relations: Detecting irrelevant and autonomously computable updates, *Trans. Database Systems* 14 (3) (1989) 369–400.

[11] D. Bobrow, ed., *Qualitative Reasoning about Physical Systems* (North-Holland, Amsterdam, 1984).

[12] A. Borgida and P. Patel-Schneider, A semantics and complete algorithm for subsumption in the CLASSIC description logic, *J. Artif. Intell. Res.* 1 (1994) 277–308.

[13] A. Brodsky and Y. Sagiv, On termination of datalog programs, in: J.M. Nicolas W. Kim and S. Nishio, eds., *Deductive and Object-Oriented Databases* (North-Holland, Amsterdam, 1990) 47–64.

[14] M. Bruynooghe, D. De Schreye and B. Krekels, Compiling control, *J. Logic Programming* 6 (1989) 135–162.

[15] M. Bruynooghe, D. De Schreye and B. Martens, A general criterion for avoiding infinite unfolding during partial deduction of logic programs, in: *Proceedings International Symposium on Logic Programming*, Paris (1991) 117–131.

[16] R. Carnap, *Logical Foundations of Probability* (University of Chicago Press, Chicago, IL, 1950).

[17] A. Chandra, D. Kozen and L. Stockmeyer, Alternation, *J. ACM* 28 (1) (1981) 114–133.

[18] C.L. Chang, Resolution plans in theorem proving, in: *Proceedings IJCAI-79*, Tokyo (1979) 143–148.

[19] W.J. Clancey, The advantages of abstract control knowledge in expert system design, in: *Proceedings AAAI-83*, Washington, DC (Morgan Kaufmann, Los Altos, CA, 1983) 74–78.

[20] P. Cousot and R. Cousot, Abstract interpretation and application to logic programs, *J. Logic Programming* 13 (2–3) (1992) 103–179.

[21] A. Darwiche, A logical notion of conditional independence, *Artificial Intelligence* 97 (1997) 45–82 (this issue).

[22] R. Davis, Diagnostic reasoning based on structure and behavior, *Artificial Intelligence* 24 (1984) 347–410.

[23] D.L. Draper, Relevance measures for localized partial evaluation of belief networks, in: *Working Notes of the AAAI Fall Symposium on Relevance* (November 1994).

[24] M. Dunn, Relevance logic and entailment, in: D. Gabbay and F. Guenthner, eds., *Handbook of Philosophical Logic, Vol. III. Alternatives to Classical Logic* (Reidel, Dordrecht, The Netherlands, 1986) 117–224.

[25] C. Elkan, A decision procedure for conjunctive query disjointness, in: *Proceedings 8th ACM Symposium on Principles of Database Systems* (1989).

[26] C. Elkan, Independence of logic database queries and updates, in: *Proceedings 9th ACM Symposium on Principles of Database Systems* (1990) 154–160.

[27] T. Ellman, Abstraction via approximate symmetry, in: *Proceedings IJCAI-93*, Chambery, France (1993) 916–921.

[28] O. Etzioni, Why PRODIGY/EBL works, in: *Proceedings AAAI-90*, Boston, MA (1990).

[29] O. Etzioni, Acquiring search-control knowledge via static analysis, *Artificial Intelligence* 62 (1993).
[30] O. Etzioni and S. Minton, Why EBL produces overly-specific knowledge: a critique of the PRODIGY approaches, in: *Proceedings Machine Learning Conference*, Aberdeen, Scotland (1992).
[31] O. Etzioni and D. Weld, A softbot-based interface to the internet, *Comm. ACM* 37 (7) (1994) 72–76.
[32] B. Falkenhainer and K. Forbus, Compositional modeling: finding the right model for the job, *Artificial Intelligence* 51 (1991) 95–143.
[33] R. Fikes, M. Cutkosky, T. Gruber and J. Van Baalen, Knowledge sharing technology, project overview, Knowledge Systems Laboratory Tech. Rept. No. KSL 91-71, Stanford, CA (1991).
[34] P. Gärdenfors, On the logic of relevance, *Sythese* 37 (1978) 351–367.
[35] P. Gärdenfors, *Knowledge in Flux: Modeling the Dynamics of Epistemic States* (MIT Press, Cambridge, MA, 1988).
[36] H. Geffner and J. Pearl, A framework for reasoning with defaults, in: H.E. Kyburg, R. Loui and G Carlson, eds., *Knowledge Representation and Defeasible Reasoning* (Academic Press, Dordrecht, The Netherlands, 1990).
[37] M.R. Genesereth, An overview of metalevel architectures, in: *Proceedings AAAI-83*, Washington, DC (Morgan Kaufmann, Los Altos, CA, 1983) 119–123.
[38] M.R. Genesereth, An agent-based framework for software interoperability, in: *Proceedings Software Technology Conference*, Los Angeles, CA (1992).
[39] M.R. Genesereth and N.J. Nilsson, *Logical Foundations of Artificial Intelligence* (Morgan Kaufmann, Los Altos, CA, 1987).
[40] F. Giunchiglia and T. Walsh, A theory of abstraction, *Artificial Intelligence* 56 (3) (1992).
[41] R. Greiner, RLL-1: a representation language language, Stanford Heuristic Programming Project, HPP-80-9 (Working Paper), Stanford, CA (1980).
[42] R. Greiner, Finding optimal derivation strategies in a redundant knowledge base, *Artificial Intelligence* 50 (1) (1991) 95–116.
[43] R. Greiner, Learning efficient query processing strategies, in: *Proceedings 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Diego, CA (1992).
[44] R. Greiner and I. Jurišica, A statistical approach to solving the EBL utility problem, in: *Proceedings AAAI-92*, San Jose, CA (1992).
[45] P.J. Hayes, Computation and deduction, in: *Proceedings 1973 Mathematical Foundations of Computer Science Symposium* (Czechoslovakian Academy of Sciences, 1973).
[46] Y. Iwasaki and A.Y. Levy, Automated model selection for simulation, in: *Proceedings AAAI-94*, Seattle, WA (1994) 1183–1190.
[47] J.M. Keynes, *A Treatise on Probability* (Macmillan, London, 1921).
[48] M. Kifer, On safety, domain independence and capturability of database queries, in: *Proceedings International Conference on Data and Knowledge Bases*, Jerusalem (1988).
[49] A. Klug, On conjunctive queries containing inequalities, *J. ACM* 35 (1) (1988) 146–160.
[50] C.A. Knoblock, Learning abstraction hierarchies for problem solving, in: *Proceedings AAAI-90*, Boston, MA (Morgan Kaufmann, Los Altos, CA, 1990).
[51] C.A. Knoblock and A.Y. Levy, eds., *Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments* (American Association for Artificial Intelligence, 1995).
[52] R. Kowalski, A proof procedure using connection graphs, *J. ACM* 22 (4) (1975) 572–595.
[53] G. Lakemeyer, A logical account of relevance, in: *Proceedings IJCAI-95*, Montreal, Que. (1995) 853–859.
[54] D.B. Lenat, R. Davis, J. Doyle and M.R. Genesereth, Reasoning about reasoning, in: F. Hayes-Roth, D.A. Waterman and D.B. Lenat, eds., *Building Expert Systems* (Addison Wesley, Reading, MA, 1983).
[55] A.Y. Levy, Irrelevance reasoning in knowledge based systems, Ph.D. Thesis, Stanford University, Stanford, CA (1993).
[56] A.Y. Levy, Creating abstractions using relevance reasoning, in: *Proceedings AAAI-94*, Seattle, WA (1994) 588–594.
[57] A.Y. Levy, R.E. Fikes and Y. Sagiv, A proof-theoretic approach to irrelevance: foundations and applications, in: *Working Notes of the AAAI Fall Symposium on Relevance* (November 1994).

[58] A.Y. Levy, I. Singh Mumick and Y. Sagiv, Query optimization by predicate move-around, in: *Proceedings 20th VLDB Conference*, Santiago, Chile (1994) 96–107.

[59] A.Y. Levy, I. Singh Mumick, Y. Sagiv and O. Shmueli, Equivalence, query-reachability and satisfiability in datalog extensions, in: *Proceedings 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Washington, DC (1993) 109–122.

[60] A.Y. Levy, A. Rajaraman and J.J. Ordille, Query answering algorithms for information agents, in: *Proceedings AAAI-96*, Portland, OR (1996) 40–47.

[61] A.Y. Levy and Y. Sagiv, Constraints and redundancy in Datalog, in: *Proceedings 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Diego, CA (1992) 67–80.

[62] A.Y. Levy and Y. Sagiv, Exploiting irrelevance reasoning to guide problem solving, in: *Proceedings IJCAI-93*, Chambery, France (1993) 138–144.

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

[64] A.Y. Levy and Y. Sagiv, Semantic query optimization in datalog programs, in: *Proceedings 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, CA (1995) 163–173.

[65] A.Y. Levy, D. Srivastava and T. Kirk, Data model and query evaluation in global information systems, *J. Intelligent Information Systems* (Special Issue on Networked Information Discovery and Retrieval) 5 (2) (1995).

[66] N. Littlestone, Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm, *Machine Learning* 2 (1988) 285–318.

[67] J.W. Lloyd and J.C. Shepherdson, Partial evaluation in logic programming, *J. Logic Programming* 11 (1991) 217–242.

[68] S. Minton, Quantitative results concerning the utility of explanation based learning, in: *Proceedings AAAI-88*, St. Paul, MN (1988).

[69] S. Minton, J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni and Y. Gil, Explanation based learning: a problem solving perspective, *Artificial Intelligence* 40 (1989) 63–118.

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

[71] J. Pearl, System Z: a natural ordering of defaults with tractable applications to nonmonotic reasoning, in: M.Y. Vardi, ed., *Theoretical Aspects of Reasoning about Knowledge* (Morgan Kaufmann, Los Altos, CA, 1990) 121–135.

[72] C. Petalson, The BACK system: an overview, in: *Proc. SIGART Bull.* 2 (3) (1991) 114–119.

[73] D. Plaisted, Theorem proving with abstraction, *Artificial Intelligence* 16 (1981) 47–108.

[74] R.B. Rao, R. Greiner and T. Hancock, Exploiting the absence of irrelevant information: what you don't know can help you, in: *Working Notes of the AAAI Fall Symposium on Relevance* (November 1994).

[75] I.S. Rombauer and M. Rombauer-Becker, *Joy of Cooking* (Merrill, New York, 1975).

[76] S. Russell, The complete guide to MRS, Tech. Rept. KSL-85-12, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, CA (1985).

[77] E.D. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artificial Intelligence* 5 (1974) 115–135.

[78] Y. Sagiv and M. Yannakakis, Equivalence among relational expressions with the union and difference operators, *J. ACM* 27 (4) (1981) 633–655.

[79] Y. Sagiv, On testing effective computability of magic programs, in: C. Delobel, M. Kifer and Y. Masunaga, eds., *Proceedings 2nd International Conference on Deductive and Object-Oriented Databases*, Munich, Germany (1991) 244–262.

[80] Y. Sagiv, A termination test for logic programs, in: V. Saraswat and K. Ueda, eds., *Proceedings 1991 International Symposium on Logic Programming* (MIT Press, Cambridge, MA, 1991) 518–532.

[81] A. Segre and C. Elkan, A high-performance explanation-based learning algorithm, *Artificial Intelligence* 69 (1994) 1–50.

[82] O. Shmueli, Equivalence of datalog queries is undecidable, *J. Logic Programming* 15 (1993) 231–241.

[83] S. Sickel, A search technique for clause interconnectivity graphs, *IEEE Trans. Comput.* 25 (8) (1976) 823–835.

[84] D. Smith, Controlling inference, Ph.D. Thesis, Stanford University, Stanford, CA (1986).

[85] D.A. Smith and T.J. Hickey, Partial evaluation of a CLP language, in: *Proceedings International Symposium on Logic Programming* (1990) 119–138.
[86] D. Srivastava and R. Ramakrishnan, Pushing constraint selections, *J. Logic Programming* 16 (3–4) (1993) 361–414.
[87] D. Subramanian and M.R. Genesereth, The relevance of irrelevance, in: *Proceedings IJCAI-87*, Milan, Italy (Morgan Kaufmann, Los Altos, CA, 1987).
[88] D. Subramanian, A theory of justified reformulations, Ph.D. Thesis, Stanford University, Stanford, CA (1989).
[89] J.D. Ullman, *Principles of Database and Knowledge-base Systems, Vols. I and II* (Computer Science Press, Rockville, MD, 1989).
[90] J.D. Ullman, Information integration using logical views, in: *Proceedings International Conference on Database Theory* (1997).
[91] A. Van Gelder, A message passing framework for logical query evaluation, in: *Proceedings ACM SIGMOD International Conference on Management of Data* (1986) 155–165.
[92] M.Y. Vardi, Automata theory for database theoreticians, in: *Proceedings 8th Symposium on Principles of Database Systems (PODS)* (March 1989) 83–92.
[93] L. Vieille, Recursive query processing: the power of logic, *Theoret. Comput. Sci.* 69 (1989) 1–53.