Human-Understandable Inference of Causal Relationships

(Invited Paper)

Alva L. Couch
Department of Computer Science
Tufts University
Medford, Massachusetts 02155
Email: couch@cs.tufts.edu

Mark Burgess
Department of Informatics
Oslo University College
Email: Mark.Burgess@iu.hio.no

***Abstract*—We present a method for aiding humans in understanding causal relationships between entities in complex systems via a simplified calculus of facts and rules. Facts are human-readable subject-verb-object statements about system entities, interpreted as (entity-relationship-entity) triples. Rules construct new facts via implication and "weak transitive" rules of the form "If X r Y and Y s Z then X t Z", where X, Y, and Z are entities and r, s, and t are relationships. Constraining facts and rules in this way allows one to treat abductive inference as a graph computation, to quickly answer queries about the most related entities to a chosen one, and to explain any derived fact as a shortest chain of base facts that were used to infer it. The resulting chain is easily understood by a human without the details of how it was inferred. This form of simplified reasoning has many applications in human understanding of knowledge bases, including causality analysis, troubleshooting, and documentation search, and can also be used to verify knowledge bases by examining the consequences of recorded facts.**

I. Introduction

There are many situations in which causal information about a complex system must be interpreted by a human being to aid in some critical task. For example, the speed of troubleshooting depends upon how quickly one can link observed symptoms with potential causes. Likewise, in scanning documentation, it is often useful to associate a behavioral goal for a managed system with the subsystems involved in assuring that goal, and associate each subsystem with documentation on how to modify the subsystem toward that goal.

We present a simplified reasoning system for inferring causal relationships between entities in a complex system, including the use cases of linking symptoms with causes and linking goals with documentation. Input to the method is a set of facts that represent what is known about the world, as well as a set of rules for inferring new facts. We assume that predetermined facts are *not* complete and, at best, are an approximate description of relationships in the world. Rules that allow us to construct new facts are limited to two simple forms in order to allow us to treat the inference algorithms as graph computations. This limitation allows us to answer simple queries efficiently, and explain each inference via a linear chain of relationships that is easily understandable to a human being.

A. Facts

Facts in our method are relationships between named entities, expressed as subject-verb-object triples of the form:

host01 is an instance of file server

Entities such as host01 are underlined with straight lines while relationships such as is an instance of are underlined with wavy lines. A set of facts represents an entity-relationship model of the kind used commonly in Software Engineering, as opposed to those utilized in database theory; the former describe *interactions and causal dependencies*, while the latter describe *functional dependencies*. Facts are positive statements about causal relationships and cannot express a lack of relationship. Negation of an existing fact is handled specially below.

B. Relationships

Relationships in our method are chosen to aid in solving problems in a particular problem domain. In analyzing causality, influences and related concepts form the basis for analysis, while in thinking about and analyzing document spaces, describes and documents are more relevant. In thinking about scientific papers, explains would be a relevant verb.

The central relationship for a problem domain (e.g., influences) represents a concept that is related to other concepts also expressed as relationships (e.g., determines, can influence, partly influences). Interactions between relationships are coded as rules that might be more accurately viewed as defining interactions between the concepts that relationships represent.

A key feature of our method is that modalities (e.g., "can", "might", etc) are part of the symbols and not part of the logic. In systems that can perform "modal logic", there are symbols in the system for the concepts of "can", "might", etc., while in our system these concepts are appended to each specific token. This actually simplifies what might be very complex computations otherwise, when we try to reason in the system.

C. Rules

Rules are used to construct derived facts from base facts known to be true at the outset. However, all rules in our system

are stated in terms of relationships, and not individual facts. Rules include implications, inverses, and weak transitive rules.

Implicative rules have the form "For every X and Y, if X r Y then X s Y" where X and Y are entities and r and s are relationships. In general, all variables in our rules are universally quantified, so we leave out the quantifiers and treat them as implicit. The implicative rule "For every X and Y, if X requires Y then 'X depends upon Y" is written as

$$\langle \text{requires} \rightarrow \text{depends upon} \rangle$$

Implicative rules distinguish between more specific relationships (in the antecedent) and more general relationships (in the consequent); requires is a more specific connection between two entities than the more "generic" depends upon. Invoking an implicative rule thus *raises the level of abstraction* in describing a relationship.

D. Inverses

While a fact "X r Y" is typically directional in the sense that X and Y cannot be interchanged, every fact "X r Y" has a corresponding fact "Y s X" where X and Y appear in reverse order. Such relationships r and s are called *inverses* to one another. The relationship between inverses is defined (by the user) through rules, rather than being inferred (through logic). E.g., the fact

$$\text{host01 is an instance of file server}$$

is equivalent with the fact

$$\text{file server has instance host01}$$

Their equivalence is notated via the rule

$$\langle \text{is an instance of} \bowtie \text{has instance} \rangle$$

An inverse rule is just a special kind of implication; the above rule means that "for every A and B, if A is an instance of B, then B has instance A, and vice-versa. We also write that inv(has instance) is is an instance of and vice versa, Some facts are self-inverse, e.g., if "X is a peer of Y", then "Y is a peer of X".

E. Weak transitive rules

Weak transitive rules have the form "for every X, Y, and Z, if X r Y and Y s Z then X t Z," where X, Y, Z are entities and r, s, t are relationships. The (strong) transitive rule "if X requires Y and Y requires Z, then X requires Z" is written as

$$\langle \text{requires}, \text{requires}, \text{requires} \rangle$$

while the rule "if X has part Y and Y controls Z then X controls Z" is written as

$$\langle \text{has part}, \text{controls}, \text{controls} \rangle$$

The latter is called a *weak transitive rule* because the consequent controls does not match at least one of the antecedents (has part). Weak transitive rules are the key to turning a logic computation into a graph computation when computing queries about facts.

F. Kinds of queries

The inference method supports two kinds of queries, both based upon a concept of *inference distance* between entities in the fact base. The inference distance between two entities X and Y with respect to a relationship r is the minimum number of weak transitive rules that must be applied to base facts to infer the fact "X r Y". For reasons that will become clear by example, implication rules and inverses are not used in our measure of distance.

The first kind of query is to locate the entities Y related to a given entity (or set of entities) X via some predetermined relationship r, i.e., to solve the query "X r Y?" where X and r are known and Y? is unknown. For each appropriate Y, the minimum number of weak transitive rules applied to infer "X r Y" from base facts is used as a measure of distance between X and Y as subsystems, so that potential candidates Y can be listed in order from "near to" to "far away from" X.

The second kind of query is to create a human-readable explanation of the shortest derivation of a given derived fact "X r Y" in terms of recorded facts and weak transitive rules applied in order to derive it. This derivation takes the form of a sequence of entities and relationships, for which X and Y are endpoints.

Many other queries are possible, but for the purposes of this paper, these two kinds are exemplary of the benefits of the method.

II. Two use cases

Two use cases motivate our choices above. In troubleshooting, one of the central problems is to relate symptoms to causes, where it is possible that causes are several entities distant from symptoms. In scanning documentation, one of the central problems is to relate one's personal goal to relevant documents describing how to accomplish the goal. Both of these are causal inference problems that can be addressed through the proposed inference method.

A. Troubleshooting

Suppose we have a very simple network with a fileserver host01, a dns server host02, and a client host03. We might code the relationships between these hosts as a set of abstract "sentences", like

$$\begin{gathered}\text{host01 is an instance of file server}\\ \text{file server provides file service}\\ \text{host02 is an instance of dns server}\\ \text{dns server provides dns service}\\ \text{host03 is an instance of client workstation}\\ \text{client workstation requires file service}\\ \text{client workstation requires dns service}\end{gathered}$$

In this case, rules include

$$\langle \text{is an instance of} \bowtie \text{has instance} \rangle$$
$$\langle \text{provides} \bowtie \text{is provided by} \rangle$$
$$\langle \text{requires} \bowtie \text{is required by} \rangle$$
$$\langle \text{depends upon} \bowtie \text{is depended upon by} \rangle$$
$$\langle \text{requires}, \text{requires}, \text{requires} \rangle$$
$$\langle \text{provides}, \text{provides}, \text{provides} \rangle$$
$$\langle \text{provides} \rightarrow \text{is depended upon by} \rangle$$
$$\langle \text{requires} \rightarrow \text{depends upon} \rangle$$

Suppose that host03 has a problem. A query of the entities that host03 depends upon returns

host03 depends upon host01
host03 depends upon host02

but this information *is not enough* to be useful to a human; we must consider the explanation of the dependence. One explanation of the dependence between host01 and host03 is

entity	relationship	entity
host03	is an instance of	client workstation
client workstation	requires	file service
file service	is provided by	file server
file server	has instance	host01

We call such an explanation a *story* of the dependence between two entities, and often leave out the repeated entity, as in

entity	relationship
host03	is an instance of
client workstation	requires
file service	is provided by
file server	has instance
host01	

Some stories are easy to compute, and others require more subtle techniques. Architectural descriptions are often incomplete and specified at *different levels of abstraction.* To cope with this, our method utilizes implication to "lift" facts to a *common level of abstraction* at which reasoning can occur, and then "re-grounds" that reasoning by expressing high-level (abstract) inferred facts in terms of the low-level (concrete) facts that were their basis.

Consider, e.g., the following quandary:

host02 is authoritative for zone eecs.tufts.edu
host03 is inside zone eecs.tufts.edu

What is the real relationship or dependency between host02 and host03? To answer this question, we must proceed to a higher level of abstraction, via implication:

$$\langle \text{is authoritative for zone} \rightarrow \text{influences} \rangle$$
$$\langle \text{is inside zone} \rightarrow \text{is influenced by} \rangle$$

and define appropriate inverses:

$$\langle \text{influences} \bowtie \text{is influenced by} \rangle$$
$$\langle \text{is authoritative for zone} \bowtie \text{has zone authority} \rangle$$
$$\langle \text{is inside zone} \bowtie \text{contains zone member} \rangle$$

after which the facts available also include:

host02 influences eecs.tufts.edu
host03 is influenced by eecs.tufts.edu

We invert the latter to its inverse:

eecs.tufts.edu influences host03

so that by the obvious transitive rule:

$$\langle \text{influences}, \text{influences}, \text{influences} \rangle$$

we infer the story that:

entity	relationship
host02	influences
eecs.tufts.edu	influences
host03	

but *this is not good enough* for human consumption. The relationships influences in the above are the result of two implications:

$$\langle \text{is authoritative for zone} \rightarrow \text{influences} \rangle$$
$$\langle \text{contains zone member} \rightarrow \text{influences} \rangle$$

To complete the picture, we "ground" the lifted relationships by replacing them with the concrete relationships that imply them, e.g.,

entity	relationship
host02	is authoritative for zone
eecs.tufts.edu	contains zone member
host03	

and this grounded explanation "explains" the abstract reasoning in concrete (and useful) terms.

In this example, implication is used to *vary the level of abstraction* used in reasoning, and is not a component of the reasoning itself. This is part of the reason for why it is not counted as an inference step in computing which explanations are most succinct.

B. Document browsing

A second problem that can be solved via this form of reasoning is to locate documents relevant to a task in document space. Locating documents is the purpose for which the method was originally designed. In this case, the verb describes replaces the verb influences as the central relationship of interest.

Suppose we want to understand how to set up a service switch in a network. We might start with the facts:

service switch requires service URL
service has attribute service URL
service URL is an instance of URL
URL described by wikipedia for URL
user service is an instance of service
user service described by user service manual

(and many others). Rules include:

$$\langle \text{is described by} \bowtie \text{describes} \rangle$$
$$\langle \text{is partly described by} \bowtie \text{partly describes} \rangle$$
$$\langle \text{has attribute} \bowtie \text{is an attribute of} \rangle$$
$$\langle \text{has instance} \bowtie \text{is an instance of} \rangle$$
$$\langle \text{describes} \rightarrow \text{partly describes} \rangle$$
$$\langle \text{describes}, \text{has attribute}, \text{describes} \rangle$$
$$\langle \text{has attribute}, \text{described by}, \text{partly described by} \rangle$$
$$\langle \text{describes}, \text{has instance}, \text{describes} \rangle$$
$$\langle \text{describes}, \text{is an instance of}, \text{partly describes} \rangle$$

In this case, we do not care about causal relationships as much as relevant documentation. If we query our reasoning system for X such that "user service is partly described by X", we obtain, among other responses:

user service is described by user service manual
user service is partly described by wikipedia for URL

etc. An explanation of the latter fact is:

entity	relationship
user service	is instance of
service	requires
service URL	is an instance of
URL	is described by
wikipedia for URL	

which demonstrates exactly how the Wikipedia documentation is relevant. Note that we asked for partly describes and actually received feedback on facts for the more specific relationship is described by; this is a result of the lifting and grounding technique discussed in the previous section.

Note the carefully crafted rules in the preceding example; if something is an instance of something else, and something describes the instance, it might not describe the whole scheme of things. If we describe the whole scheme, we do describe the instance. Also, describing an attribute of a thing does not describe the whole thing, but describing a thing does describe its attributes. These rules might be considered as part of the *definition* of the describes relationship, via its interaction with is an instance of and is an attribute of.

III. Implementation

The examples above assume the existence of both a knowledge base and a set of rules with specific properties. These properties lead both to the kinds of inferences that can be done, as well as the speed with which they can be accomplished.

A. Specifying facts

Facts in our knowledge base represent invariant properties of entities. Variation over time is not supported. Thus not every kind of fact can be represented. Kinds of facts that can be represented are mostly "architectural" in nature, in the sense that they do not vary for the lifetime of the entity.

Facts cannot express contradictions. Even if two classes A and B of entities are mutually exclusive, there is no way to express that in a fact. Mutual exclusivity is instead a result of reasoning, in the sense that if an object is an instance of A, then it enjoys all properties of an instance of A, while if it is not an instance of A, properties of A are not assumed to be either present or absent. Thus, in constructing a base of facts, it is important to eliminate seeming contradictions from the fact base, because contradictions cannot be detected by the reasoning method.

In a fact, entities and relationships are formal symbols devoid of meaning. The statement host01 is an instance of file server is a sequence of three tokens, as opposed to the English sentence "host01 is an instance of file server." The meaning of an entity token (as a mapping between the token and the real world) is implicit in the set of facts that describe the entity, just as the meaning of a relationship token is implicit in the rules that describe how it interacts with other relationships.

B. Retracting a fact

In our information store, facts cannot be deleted. They can, however, become outmoded, in the sense that they describe old information that is no longer of interest. Suppose for example that we record a fact about the authors, e.g.,

alva eats cornflakes

and then it turns out that this is inaccurate. There is no way to retract that fact, but in fact, our new information describes a "different alva" than before. So instead, we issue a new token alva' with no facts listed, and then duplicate all of the facts from alva to alva' except the fact to be deleted. Then we can incorporate new facts about alva' that do not apply to the old alva, e.g.,

alva' eats oatmeal

and computation proceeds as defined below. Thus *we delete facts by re-versioning the entities that they describe.* This can be done automatically through a machine-learning importance ranking, such as used by Cfengine [1].

C. Constructing rules

Constructing a rule requires more than considering how it acts on facts. At a superficial level, derived facts are computed from base facts by repeated use of rules. The rules themselves, however, can be combined to form new rules. There are several finer points of describing rules, including describing modal relationships and partial knowledge.

1) Modal and partial knowledge: There is no special mechanism for separately handling modal constructions such as might determine and can influence in our method. Instead, these are defined via their interactions with other tokens. The qualifiers "can" and "might", in a relationship, are weaker than the unqualified relationship; "can" indicates capability while "might" indicates possibility (this assumes a standard partial ordering of the terms in the ontology). Thus we write:

$$\langle \text{determines} \rightarrow \text{can determine} \rangle$$
$$\langle \text{can determine} \rightarrow \text{might determine} \rangle$$

from which we can immediately derive by transitivity of implication that:

$$\langle \text{determines} \rightarrow \text{might determine} \rangle$$

Likewise, one can distinguish between complete and partial determination via:

$$\langle \text{determines} \rightarrow \text{partly determines} \rangle$$

Influence is another way to describe partial determination:

$$\langle \text{determines} \rightarrow \text{influences} \rangle$$

The relationship between influences and can determine can be obtained from:

$$\langle \text{determines} \rightarrow \text{can determine} \rangle$$
$$\langle \text{can determine} \rightarrow \text{can influence} \rangle$$

In general, however, two abstract concepts may not enjoy any relationship whatever.

The above describe only one facet of the meaning of influences. Several more facets include:

$$\langle \text{is a part of}, \text{determines}, \text{determines} \rangle$$
$$\langle \text{determines}, \text{is a part of}, \text{influences} \rangle$$
$$\langle \text{is an instance of}, \text{determines}, \text{determines} \rangle$$
$$\langle \text{determines}, \text{is an instance of}, \text{influences} \rangle$$

In other words, if a thing is part of a determined thing, the part is likewise determined, but determining a part of a thing only influences the thing. Likewise, if a set (class) of things is determined, so are its members, but determining a member does not determine the set of which it is a member. These rules might be considered facets of a "definition" of the (more abstract) relationship influences, in terms of the (more concrete) relationship determines.

D. Deleting a rule

As for facts, there is no well-defined notion of deleting a rule. However, one can proceed exactly as one does for deleting facts, by creating a new version of the *consequent relationship* of the rule to invalidate current inferences via the rule. E.g., if one specifies:

$$\langle \text{is a part of}, \text{is an instance of}, \text{is a part of} \rangle$$

(in error), the solution is to create a new version of the consequent is a part of', instantiate all rules for the new relationship except the one to be deleted, and then resume computation of the consequences of the new rules. Eventually, the original is a part of can be removed, when the consequences of the new is a part of' are known and the original relationship is no longer needed.

E. Inferring new rules

So far, we have emphasized deriving new facts from base facts and base rules, but there is an equivalent calculus for deriving rules from other rules. Each relationship r can be viewed as representing the set of ordered pairs (X, Y) where "$X\ r\ Y$" is either a base fact about the world or can be inferred. For example, the relationship is a part of can be thought of as representing the set

$$\text{is a part of} \ \equiv \ \{(X, Y) \mid X \text{ is a part of } Y\} \tag{1}$$

Likewise, each rule is a statement about sets represented by relationships. Implication is a subset relationship:

$$\langle r \rightarrow s \rangle \equiv r \subset s \tag{2}$$

i.e., the set of facts r is a subset of the set of facts s. In like manner, a weak transitive rule is also a subset relationship of a different kind:

$$\langle r, s, t \rangle \equiv r \otimes s \subset t \tag{3}$$

i.e., t is a superset of the product $r \otimes s$ resulting from combining sets r and s , where $r \otimes s$ is defined by

$$r \otimes s \equiv \{(X, Z) \mid (X, Y) \in r, (Y, Z) \in s\} \tag{4}$$

Note that every rule is inclusive of *other* kinds of meaning, and that one never *limits* set contents with a rule.

Creating rules from others is most easily explained by treating relationships as sets. The obvious relationship between sets:

$$r \subset s \text{ and } s \subset t \Rightarrow r \subset t \tag{5}$$

can be trivially restated as a relationship between rules:

$$\langle r \rightarrow s \rangle \text{ and } \langle s \rightarrow t \rangle \Rightarrow \langle r \rightarrow t \rangle \tag{6}$$

in the sense that there is no problem with instantiating a new rule $\langle r \rightarrow t \rangle$ and using it instead of the other two.

The subset relationships for weak transitivity are described in the following diagram:

$$\begin{array}{ccccccccccccc} r' & & & & s' & & & & & & & & \\ \cap & & & & \cap & & & & & & & & \\ r & \otimes & s & \subset & t & \Rightarrow & r' & \otimes & s' & \subset & t' \\ & & & & \cap & & & & & & \\ & & & & t' & & & & & & \end{array} \tag{7}$$

(where $\cap$ indicates vertical subsetting). In the diagram, $r' \rightarrow r$ (as relationships) or equivalently $r' \subset r$ (as sets), $s' \rightarrow s$ (as relationships) or equivalently $s' \subset s$ (as sets), and $t \rightarrow t'$ (as relationships) or equivalently $t \subset t'$ (as sets). Because the subset relationship is transitive, for any rule $\langle r, s, t \rangle$ and any appropriate r', s', and t', the rule $\langle r', s', t' \rangle$ also applies by set containment. We might more concisely represent this set of relationships by substituting rules for products and implications for subsets, e.g.,

$$\begin{array}{ccccccccccc} & r' & & s' & & & & & & & \\ & \downarrow & & \downarrow & & & & & & & \\ \langle & r, & & s, & t & \rangle & \Rightarrow & \langle & r', & s', & t' & \rangle \\ & & & & \downarrow & & & & & & & \\ & & & & t' & & & & & & & \end{array} \tag{8}$$

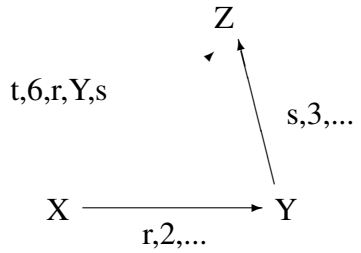


Fig. 1. Recording results of a best known inference of "X t Z" from "X r Y", "Y s Z", and ⟨r,s,t⟩. Input edges "X r Y" and "Y s Z" are marked with the number of inferences (2 and 3) needed to create them. The edge "X t Z" is labeled with the least inference distance + 1 and information about how to achieve that least distance (r,Y,s).

Finally, there are obvious rules for handling inverses.

$$\langle r, s, t\rangle \Rightarrow \langle inv(s), inv(r), inv(t)\rangle \tag{9}$$

and

$$\langle r \rightarrow s\rangle \Rightarrow \langle inv(r) \rightarrow inv(s)\rangle \tag{10}$$

These "meta-rules" can be used to compute unknown rules from known ones. More important, they account for all possible uses of implication and inverses in subsequent applications of weak transitive rules.

The key to everything that follows is that applying rules only adds facts, so that if one simply tries all rules until no new facts are added, one has all of the current available facts. One naive way to accomplish this is:

repeat
  for all $\langle r \rightarrow s\rangle$ **do**
    Set $s = s \cup r$.
  end for
  for all $\langle r, s, t\rangle$ **do**
    Set $t = t \cup r \otimes s$.
  end for
until there is no change in any relationship set.

In doing this, we keep track of which relationships are ground and which are derived, which allows us to construct a "shortest path" between two given entities as a chain of relationships.

F. Computing queries

Queries are computed efficiently by relying upon mathematical properties of our choice of facts and rules. By construction, any derived fact or rule cannot be invalidated over time and may safely be cached for later use. Adding a new fact does not require recomputation of existing cached facts. Changing a fact is a matter of creating a new version of its subject and/or object. Changing a rule is a matter of creating a new version of its consequent relationship. No backtracking is necessary to compute new facts.

The steps in satisfying a query ""X r Y?"" are as follows. Let $\mathcal{E}$ be the set of entities described by facts, and let $\mathcal{F}$ represent the a priori facts we have about entities in $\mathcal{E}$. Note that facts $\mathcal{F}$ are edges in a graph $\mathcal{G} = (\mathcal{E}, \mathcal{F})$ where $\mathcal{E}$ is the set of entities that facts describe. Let $\mathcal{R}$ represent our a priori rules. These can be separated into weak transitive rules $\mathcal{W}$ and implicative and inverse rules $\mathcal{I}$, where $\mathcal{R} = \mathcal{W} \cup \mathcal{I}$.

First we utilize all implication and inverse rules in $\mathcal{I}$ to generate the derived facts and rules that result from implication:

1) Apply all implications and inverse rules in $\mathcal{I}$ to all facts $\mathcal{F}$ to create a complete list of implied facts $\mathcal{F}'$.
2) Use implication and inverse rules in $\mathcal{I}$ to generate a complete set of weak transitive rules $\mathcal{W}'$ from $\mathcal{W}$, by using equations 6, 8, 9, and 10.

These two steps account for all implication and inverse rules in $\mathcal{I}$, both for facts and rules. Thus *we need not account for these in further computations.* This step results in what might be called "weaker rules" than the originals, e.g., combining the rules:

$$\langle \text{requires}, \text{requires}, \text{requires}\rangle$$
$$\langle \text{requires} \rightarrow \text{may require}\rangle$$

results in the derived rule:

$$\langle \text{requires}, \text{requires}, \text{may require}\rangle$$

that is "weaker" than the original.

Our algorithm is based upon the observation that "X r Y" is a consequence of facts $\mathcal{F}$ and rules $\mathcal{R}$ if and only if it is a consequence of facts $\mathcal{F}'$ and rules $\mathcal{W}'$. Thus the problem is no longer an inference problem, but rather, a graph problem as to whether "X r Y" is in the transitive closure $\mathcal{G}''$ of the derived labeled graph $\mathcal{G}' = (\mathcal{E}, \mathcal{F}')$ with respect to the weak transitive laws in $\mathcal{W}'$.

To compute $\mathcal{G}''$, let dist("X r Y") represent the current minimum known inference distance between base facts and the fact "X r Y", or infinity if there is no a priori relationship between X and Y. We will compute $\mathcal{G}''$ by a simple variation of the transitive closure algorithm, to wit:

Set $\mathcal{G}'' = \mathcal{G}'$.
for all existing edges "X r Y" $\in \mathcal{F}'$ **do**
  Set dist("X r Y")=0.
end for
while some edges are updated or added to $\mathcal{F}''$ **do**
  for all facts "X r Y" and "Y s Z" in $\mathcal{F}''$ **do**
    if there is a rule ⟨r,s,t⟩ **then**
      if "X t Z" $\notin \mathcal{F}''$ **then**
        Put "X t Z" into $\mathcal{F}''$.
        Label dist("X t Z")
          =dist("X r Y") + dist("Y r Z") + 1.
      else if dist("X t Z")
        > dist("X r Y") + dist("Y r Z") + 1 **then**
        Label dist("X t Z")
          =dist("X r Y") + dist("Y r Z") + 1.
      end if
    end if
  end for
end while

The result of this process is a graph $\mathcal{G}'' = (\mathcal{E}, \mathcal{F}'')$ in which X is connected to every candidate Y that has a relationship with it, and every edge between X and Y is labeled with the minimum number of rule applications needed to infer the edge.

From this computation, one can list the connected entities in order of inference distance from X.

The general pattern of computation can be summarized as follows:

$$\begin{array}{ccc} \mathcal{F} & \mathcal{W} & \\ \downarrow \mathcal{I} & \downarrow \mathcal{I} & \\ \mathcal{F}' & \xrightarrow{\mathcal{W}'} & \mathcal{F}'' \end{array}$$

where arrows represent (implicative and weak transitive) closure computations and, at the end of this, "X r Y" $\in \mathcal{F}''$.

G. Improving runtime

The above "brute force" algorithm captures the idea of the algorithm succinctly, but there are many simple optimizations that improve its runtime. First, it is always safe to cache prior computations of derived rules and facts, and use them later. All that is necessary is to incorporate the effects of new rules into the cache.

Second, one can safely *restrict the domain* of the computation to include only relationships of interest or that result in relationships of interest. One need not consider, e.g., describes, when one is interested only in determines, because there is no relationship between documentation and causality.

Third, one can limit inference to facts that involve X and Y. One can proceed *breadth-first* from the facts involving X, to include facts that connect that set with others. This reduces the iteration above from "for all facts "X r Y" and "Y r Z" in $\mathcal{F}''$, to "for all facts "X r Y" previously derived about X and all related facts "Y r Z" in $\mathcal{F}''$, which converts iteration over all facts to a breadth-first traversal similar to that in Dijkstra's "single-destination shortest path" algorithm [2].

H. Inferring stories

To compute the sequence of inferences that connect two entities A and B, we repeat the computation above, but this time, record the midpoint entity *and* relationships used to create each minimum-distance edge, on the edge. E.g., in Figure 1, as a result of applying ⟨r,s,t⟩ to "X r Y" and "Y s Z", we get "X t Z" labeled with both the number of rules required (6 = 1+2+3) *and* the prior inference details "r Y s" that resulted in that inference count.

To produce an explanation of "X t Z", we use the result of the transitive closure calculation to proceed in reverse, replacing each edge with the antecedent of the weak transitive inference that produced the edge. First we replace "X t Z" with "X r Y s Z", where we applied ⟨r,s,t⟩ to "X r Y" and "Y s Z" to produce "X t Z" via the minimum number of rules. We continue to replace one adjacent pair with a triple at a time, until all that remains is a sequence of base facts, all labeled with distance 0.

The result of this sequence of substitutions is a derivation tree for the relationship "X t Z". The leaves of the derivation tree are the entities on the path between X and Z, while the relationships in the tree describe the relationships between leaves.

Finally, for each of the leaf facts, referring to the implicative rules $\mathcal{I}$, we choose the base fact that is most specific and implies it, thus "grounding" the sequence in low-level terms. This gives the outputs discussed in the use cases above.

IV. Related work

This work arose over time from ideas on utilizing logic programming in system administration [3], but approaches the problem of applying logic to system administration from a new angle based upon ideas in topic maps [4], [5]. We initially intended to solve problems in navigating in a specific topic map – Copernicus [6], [7] – that documents and tracks use of the Cfengine suite of autonomic management tools [8]–[11]. We realized that our method for navigating Copernicus has application to troubleshooting, by giving human operators more detailed causal information than is available by other methods.

Surprisingly, this work did not arise out of any tradition of formal reasoning or knowledge representation, but instead, from traditions of library science, graph algorithms, autonomic system management, computer immunology, and troubleshooting. We asked ourselves which graph algorithms would provide meaningful connections between objects in a topic map, and the answer was a form of abductive reasoning. Our method represents a limited form of abduction, via a limited information model, where our simplifications avoid computational quandaries and make our specific problem of reporting causal chains easy to solve.

A. Topic maps

Our entities and relationships bear strong resemblance to "topics" and "associations" between topics in a topic map [4], [5]. A topic map is a kind of generalized entity-relationship (ER) model utilized in library science:

1) *Topics* (entities) are analogous to entries in an index of a book.
2) *Associations* (relationships) are analogous to "See also" in a book index.
3) *Occurrences* are analogous to page numbers in an index, and specify "where" a topic is mentioned.

The most important thing we draw from topic maps is the limitation to positive relationships, and the lack of negatives. Our ER-diagrams, like topic maps, are intended to *define* entities through their relationships with other entities. Our facts and rules have "definition-like" qualities. Notably:

1) Entities are static and do not change over time (from the point of view of the reasoning method, inside the formal model).
2) Relationships are static and do not change over time.
3) Rules are additive and define *facets* of the definition of a relationship.

Topic map associations are slightly more expressive than our relationships; unlike our triples, an association is a quintuple

$$\underline{\text{topic1}}_{\text{role1}}\underline{\text{assocname}}_{\text{role2}}\underline{\text{topic2}}$$

where role1 serves to disambiguate the scope of the name topic1 while role2 serves to disambiguate the scope of the

name topic2. The scope of a name is the context in which it has meaning. E.g., "Charlie" could be a host name, or a pet's name, or even a software package. The scope of "Charlie" determines the one of these to which it refers. E.g., our fact

host01 is an instance of file server

might be written in a topic map as

host01$_{\text{hostname}}$is an instance of$_{\text{host type}}$file server

Its inverse association would be

file server$_{\text{host type}}$is an instance of$_{\text{hostname}}$host01

because roles disambiguate direction and allow, e.g., use of languages that read right-to-left as topics, associations, and roles. The association itself is viewed as a triple

hostname$^{\text{assocname}}$host type

Our reasoning methods are easily adapted to handle roles, but we left that adaptation out of this paper for simplicity.

In using topic maps to index Copernicus, we found that a particular way of thinking about the map led to more efficient use of documentation. If we view the map as a set of *links* between topics, it is easy to get lost in the map, while if we view it as a set of chains of *reasoning*, the relationships become more clear and the map becomes more useful. This led to our algorithms for computing chains, which serve as "explanations" of relationships between topics.

B. Cfengine

We were also inspired by the philosophy of the configuration management suite Cfengine that Cfknowledge documents. Cfengine distributes complex problems of configuration management among cooperating autonomous agents. Cooperation between agents is based upon *promises* that in their simplest form are assertions of behavior presented by one agent to another. An agent's promise – e.g., the promise to provide a particular kind of service – maps to a base fact in our method. This allows one to link the configuration of Cfengine (a set of promises) with the documentation for Cfengine (a set of other facts and relationships that explain the former).

C. Troubleshooting

There are many other approaches to troubleshooting. Snitch [12] applies a maximum-entropy approach to creating dynamic decision trees for troubleshooting support, using a probabilistic model. The Maelstrom approach [13] exploits self-organization in troubleshooting procedures to allow use of less effective procedure. STRIDER [14] employs a state-based approach and knowledge of behavior of peer stations to infer possible trouble points. Outside the system administration domain, SASCO [15] guides troubleshooting by heuristics, using what it calls a "greedy approach" to pick most likely paths to a solution. Troubleshooting has an intimate relationship with cost of operations [16], which justifies use of decision trees and other probabilistic tools to minimize cost and maximize value.

There are several differences between our work and the above approaches to troubleshooting. We base our troubleshooting upon a partial description of the *architecture* of the system under test; it is partial because any sufficiently detailed description of architecture lacks some details, and details change over time so that no snapshot of architecture can be completely accurate. . We use architectural reasoning to infer the nature of dependencies in the system, and utilize those inferences to guide troubleshooting. The net result is that we show how to apply something we already need to have – a global map of architecture – to the troubleshooting process.

D. Abduction

The problem of determining a set of rules to apply to achieve a desired relationship is a simple form of logic-based abduction [17]–[19], i.e., deriving an "explanation" from a logical description of a problem and an observed symptom. Unlike prior work, we limit the problem structure to gain substantive performance advantages. The general abduction problem is to support some conclusion C from a base of facts $\mathcal{B}$, via logical reasoning. In our case, C is a single fact "X r Y", while $\mathcal{B}$ is limited to the facts and rules as specified above. The output of our abduction calculation is limited to linear chains of reasoning that a human can interpret quickly. Thus we do not solve a general abduction problem, but rather, a very specific one, constructed so that only deduction from known facts is required.

E. Information Modeling

Our facts and rules are a (very limited) form of information modeling as proposed by Parsons [20]. Whereas early information modeling tried to express models by classifying objects in a form of object-oriented modeling, this mechanism quickly proved vulnerable to a problem Parsons calls the "tyranny of classification" [21] in which an instance *must* be a member of some class. Parsons proposes a separation of data into instances and separate classes, which mimics our design closely. The main difference between our data model and Parsons' is that it intentionally does not model certain kinds of relationships, e.g., ternary relationships such as"foo(X,Y,Z)".

Our approach is quite different from information modeling regimens such as the Shared Information and Data model(SID) [22], mostly due to lack of structure (or even the need for structure) in our approach. While SID gains its strength from stratifying knowledge into domains, our approach invokes stratification by simple mechanisms such as the relationship is an instance of. There is no overall required hierarchical structure to our data, and any hierarchical relationships emerge from defined facts and rules (and potentially topic map roles).

Our approach is also quite distinct from prior approaches to "causal modeling," e.g., "revealed causal modeling" [23], [24], because we rely upon user knowledge of the details of causality, and do not try to infer it second-hand. Our model of causality is based upon coding simple English statements, rather than inferring probabilities of relationship between entities.

F. Ontology

Our relationship to ontological mapping is to propose a new *problem*. Like all other approaches to information representation, our approach requires ontological mapping to link concepts arising from different information domains. For example, the concept of a determines relationship may differ depending upon who is using it. However, ontological mapping in our system – like reasoning – is made simpler by the limitations we impose upon our logic. The rules that govern a relationship in our representation constitute – in some sense – its "meaning", and a set of relationships that satisfy the same rules may safely be considered as equivalent. Thus the ontological mapping problem is – for us – a matter of matching relationships across information domains in such a manner that the same rules apply to either side of a mapping. This is – again – a much simpler problem than general ontological mapping as embodied, e.g., in DEN-NG [25].

V. Theoretical Concerns

The proposed method solves some common problems but raises some deeper questions about the use of knowledge in systems management. The main cost of using the method is that its input knowledge must be carefully structured into a usable form, which often requires substantive transformation from its original form. During this transformation, some meaning is lost. The method is very sensitive to the choice of facts and rules to represent ideas, and one invalid rule or fact can render the output useless.

A. Ambiguity and uncertainty

Some statements one can make about architecture are certain, and others portray only partial information. In formal reasoning, certain kinds of uncertainty are "good", in the sense that they *enable* reasoning, while other kinds are "bad", in the sense that they *impede* reasoning. Acknowledging uncertainty in *knowledge* of architecture enables reasoning, while uncertainty in *interpreting* facts impedes reasoning.

Ambiguity based upon lack of assumptions is "good", in the sense that the reasoning method functions best when as little as possible is assumed. For example, if one is not absolutely sure about the nature of a dependence between entities, one uses the generic depends upon relationship to assert that there is some unspecified dependence. Likewise, if one is not absolutely sure that there is a dependence between two entities, one should code that relationship as might depend upon to remember that the dependence is not known to be a certainty. In both of these examples, we encode uncertainty in a set of facts as a *generic* relationship.

Another kind of ambiguity impedes reasoning, by making the user uncertain as to how to interpret a relationship. For example, the relationship is a is ambiguous about the domain in which similarity is invoked. Depending upon how one speaks in English, "X is a Y" could mean that X is an instance of class Y, is similar to an instance Y, or even that X is a variant of an instance Y. Thus in coding facts, we avoid contextually-defined verbs such as is a in favor of the disambiguated forms is an instance of, is a type of, and is a peer of.

B. Why we did not use logic programming

We implemented a prototype reasoning system entirely in the Perl programming language. The literate Prolog programmer may realize that our rules and facts fit very well into the logic programming language Prolog. E.g., if we code facts like:

```
fact(host01, instance_of, file_server).
```

and implication rules like:

```
fact(X,influences,Y)
  :-fact(X,determines,Y).
```

and transitive rules like:

```
fact(X,influences,Z)
  :-fact(X,influences,Y),
    fact(Y,influences,Z).
```

then the entire reasoning method is very easily coded in Prolog with one clause per fact and one clause per rule.

The reason we did not do this is that our rules are much simpler than Prolog supports, with the result that our Perl prototype executes a few orders of magnitude faster than an equivalent Prolog program. It was rather important to us to have the program function quickly, and direct access to data structures was useful in speeding up the search process. Unlike general, unconstrained logic programs, our careful choice of facts and rules allows us to eliminate backward chaining completely, which means we do not need Prolog at all.

VI. Limitations

The reasoning method remains extremely simple and several obvious quandaries remain in the work.

First, the desire to express reasoning as stories does not just simplify computation, but also limits the kind of reasoning that can be done. It is not clear exactly where the limits lie, though we know that there are some facts that cannot be inferred or even represented.

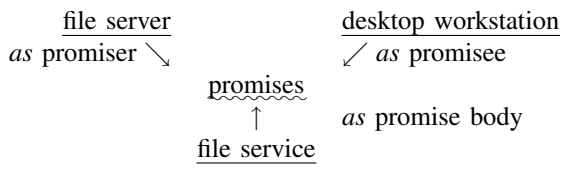
The simplest shortcoming arises when we try to use Burgess' promises [26], [27] as a description of architecture, which is part of what promises are intended to describe. A promise is a ternary relationship between three entities: a promiser, a promisee, and a promise body (or description). Our rules only act on binary relationships, so that so far, the full concept of a promise cannot be implemented or reasoned about in the current method. In particular, the relationship between promises and bindings might be described in Prolog as:

```
bound_to(Agent2,Agent1)
  :- promise(Agent1,Service,Agent2),
     uses(IsClient,Service),
     promise(Agent2,IsClient,Agent1).
```

where capitalized phrases represent variables while lowercase phrases are atoms (strings). This means that `Agent2` is bound

to `Agent1` if `Agent1` promises a service and `Agent2` promises to use that service. To our knowledge, this rule cannot be coded in our method, though bindings can be coded explicitly.

One potential solution to this kind of quandary is to invoke topic map roles to disambiguate roles in ternary facts, and to express these relationships in the manner of chemical bonds, e.g.,



to mean that a file server (in the role of promiser) promises file service (in the role of a promise body) to a desktop workstation (in the role of promisee). This has potential to encode more kinds of relationships, at the expense of depicting the results of inference as a two-dimensional graph rather than as a linear chain of reasoning.

VII. Conclusions

In starting this work, we suffered from all of the preconceptions of regular computer algebra and knowledge representation: that computers can solve the worlds' problems and that it does not matter whether a human understands what the computer is doing or not. We learned that if reasoning is limited to the kind that is easily human-understandable, then it has more value and reasoning actually becomes simpler to implement. Knowledge domains that might have once been irrelevant to a task become relevant. Notions such as inference distance become a meaningful measure of the strength of relationships between entities, and one can express a very complex sequence of the proper kind of inferences in a very readable form.

Our method of reasoning is not "the solution" to troubleshooting or to documentation search, but rather, utilizes a part of available information that was previously ignored. It is not a replacement for current strategies, but rather, a synergistic addition to the toolbox of the system administrator, in the grand challenge of making the job more doable by integrating all available knowledge about each problem.

Acknowledgment

The authors would like to thank John Strassner for the opportunity to present this work, as well as years of thoughtful comments on the use of knowledge in management. We also thank Oslo University College for generously funding Prof. Couch's extended residence at the University, during which time this work was done.

References

[1] M. Burgess, "Probabilistic anomaly detection in distributed computer networks," *Science of Computer Programming*, vol. 60, no. 1, pp. 1–26, 2006.

[2] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik 1*, pp. 269–271, 1959.

[3] A. Couch and M. Gilfix, "It's elementary, dear watson: Applying logic programming to convergent system management processes," *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII) (USENIX Association: Berkeley, CA)*, p. 123, 1999.

[4] S. Pepper, *Encyclopedia of Library and Information Sciences*. CRC Press, ISBN 9780849397127, 2009, ch. Topic Maps.

[5] ——, "The tao of topic maps," in *Proceedings of XML Europe Conference*, 2000.

[6] M. Burgess, "Cfengine knowledge management," CFengine AS, Tech. Rep., 2009. [Online]. Available: http://cfengine.com/files/knowledge.pdf

[7] ——, "Knowledge management and promises," *Lecture Notes on Computer Science*, vol. 5637, pp. 95–107, 2009.

[8] ——, "A site configuration engine," *Computing systems (MIT Press: Cambridge MA)*, vol. 8, p. 309, 1995.

[9] M. Burgess and R. Ralston, "Distributed resource administration using cfengine," *Software practice and experience*, vol. 27, p. 1083, 1997.

[10] M. Burgess, "Automated system administration with feedback regulation," *Software practice and experience*, vol. 28, p. 1519, 1998.

[11] ——, "Cfengine as a component of computer immune-systems," *Proceedings of the Norwegian conference on Informatics*, 1998.

[12] J. Mickens, M. Szummer, and D. Narayanan, "Snitch: interactive decision trees for troubleshooting misconfigurations," in *SYSML'07: Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–6.

[13] A. Couch and N. Daniels, "The maelstrom: Network service debugging via "ineffective procedures"," *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, p. 63, 2001.

[14] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang, "Strider: A black-box, state-based approach to change and configuration management and support," in *LISA '03: Proceedings of the 17th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 2003, pp. 159–172.

[15] F. V. Jensen, U. Kjæ rulff, B. Kristiansen, H. Langseth, C. Skaanning, J. Vomlel, and M. Vomlelová, "The sacso methodology for troubleshooting complex systems," *Artif. Intell. Eng. Des. Anal. Manuf.*, vol. 15, no. 4, pp. 321–333, 2001.

[16] A. Couch, N. Wu, and H. Susanto, "Towards a cost model for system administration," *Proceedings of the Nineteenth Systems Administration Conference (LISA XIX) (USENIX Association: Berkeley, CA)*, pp. 125–141, 2005.

[17] T. Eiter and G. Gottlob, "The complexity of logic-based abduction," *J. ACM*, vol. 42, no. 1, pp. 3–42, 1995.

[18] P. Liberatore and M. Schaerf, "Compilability of propositional abduction," *ACM Trans. Comput. Logic*, vol. 8, no. 1, p. 2, 2007.

[19] G. Nordh and B. Zanuttini, "What makes propositional abduction tractable," *Artif. Intell.*, vol. 172, no. 10, pp. 1245–1284, 2008.

[20] J. Parsons, "An Information Model Based on Classification Theory," *MANAGEMENT SCIENCE*, vol. 42, no. 10, pp. 1437–1453, 1996. [Online]. Available: http://mansci.journal.informs.org/cgi/content/abstract/42/10/1437

[21] J. Parsons and Y. Wand, "Emancipating instances from the tyranny of classes in information modeling," *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 228–268, 2000.

[22] T. Forum, "Information framework (sid)," website. [Online]. Available: http://www.tmforum.org/InformationFramework/1684/home.html

[23] K. M. Nelson, H. J. Nelson, and D. Armstrong, "Revealed causal mapping as an evocative method for information systems research," in *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 7*. Washington, DC, USA: IEEE Computer Society, 2000, p. 7046.

[24] K. M. Nelson, S. Nadkarni, V. K. Narayanan, and M. Ghods, "Understanding software operations support expertise: a revealed causal mapping approach," *MIS Q.*, vol. 24, no. 3, pp. 475–507, 2000.

[25] J. Strassner, S. Meer, D. O'Sullivan, and S. Dobson, "The use of context-aware policies and ontologies to facilitate business-aware network management," *J. Netw. Syst. Manage.*, vol. 17, no. 3, pp. 255–284, 2009.

[26] M. Burgess and A. Couch, "Autonomic computing approximated by fixed point promises," *Proceedings of the 1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE); Multicon verlag 2006. ISBN 3-930736-05-5*, pp. 197–222, 2006.

[27] M. Burgess, "Promise you a rose garden," http://research.iu.hio.no/papers/rosegarden.pdf.