Sound and Efficient Closed-World Reasoning for Planning

Oren Etzioni Keith Golden Daniel Weld*
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{etzioni, kgolden, weld}@cs.washington.edu

April 2, 1996

Abstract

Closed-world inference is the process of determining that a logical sentence is false based on its absence from a knowledge base, or the inability to derive it. This process is essential for planning with incomplete information. We describe a novel method for closed-world inference and update over the first-order theories of action used by planning algorithms such as NONLIN, TWEAK, and UCPOP. We show the method to be sound and efficient, but incomplete. In our experiments, closed-world inference consistently averaged about 2 milliseconds, while updates averaged approximately 1.2 milliseconds. We incorporated the method into the XII planner, which supports our Internet Softbot (software robot). The method cut the number of actions executed by the Softbot by a factor of one hundred, and resulted in a corresponding speedup to XII.

1 Introduction and Motivation

Classical planners such as NONLIN [47], TWEAK [5], or UCPOP [41, 48] presuppose correct and complete information about the world. Having complete information facilitates planning, since the planning agent need not obtain information from the external world — information absent from the agent's theory of the world is assumed to be false (this is the infamous *closed world assumption* [44]). However, in many cases, an agent has incomplete information about its world. For instance, a robot may not know the size of a bolt or the location of an essential tool [38]. Similarly, a software agent,

*We thank Paul Beame, Tom Dean, Denise Draper, Marc Friedman, Steve Hanks, Rao Kambhampati, Craig Knoblock, Neal Lesh, Alon Levy, Martha Pollack, Rich Segal, Yoav Shoham, Mike Wellman, Mike Williamson, and anonymous referees for helpful comments on earlier drafts of this paper. Omid Madani contributed to the formal aspects of the paper, and Rob Spiger contributed to the implementation. This research was funded in part by Office of Naval Research Grants 90-J-1904 and 92-J-1946, by a grant from the University of Washington Royalty Research Fund, and by National Science Foundation Grants IRI-8957302, IRI-9211045, and IRI-9357772, by ARPA / Rome Labs grant F30602-95-1-0024, and by a gift from Rockwell International Palo Alto Research. Golden is supported in part by a UniForum Research Award and by a Microsoft Graduate Fellowship.

such as the Internet Softbot [19, 14, 17], cannot be familiar with the contents of *all* the bulletin boards, FTP sites, and files accessible through the Internet.[1]

What do we mean by incomplete information? In this paper, we focus on incomplete but correct information about the state of the external world (see Section 2.1 for a formal description). In contrast to work on relational database theory [26], we do not assume that all objects in the external world are known in advance; agents constantly encounter new objects. In addition, we do not assume that the world is static; agents constantly sense (or cause) changes to the world. However, we do assume that agents are informed about changes to the world, so that their information about the world remains correct.

Recent work has sketched a number of algorithms for planning with incomplete information (*e.g.*, [1, 38, 16, 31, 42]). These algorithms make the *open world assumption* — information not explicitly represented in the agent's theory of the world is unknown. Because they make the open world assumption, none of the above algorithms handle universally quantified goals. The planners cannot satisfy even a simple goal such as "Print all of Smith's postscript files in the `/kr94` directory" because they have no way to guarantee that they are familiar with *all* the relevant files. In addition, these planners are vulnerable to *redundant information gathering* when they plan to "sense" information that is already known to the agent [18]. Since satisfying the preconditions of an information-gathering action can involve arbitrary planning, the cost of redundant information gathering is unbounded in theory and large in practice [24].

We can still salvage a partial notion of complete information, even in the presence of unknown facts. Many sensing actions yield *local closed world* information (LCW). For example, scanning with a TV camera shows *all* objects in view, and the UNIX `ls -a` command lists *all* files in a given directory. After executing `ls -a`, it is not enough for the agent to record that the files `paper.tex` and `proofs.tex` are in `/kr94` because, in addition, the agent knows that *no other* files are in that directory. Note that the agent is not making a closed world *assumption*. Rather, the agent has executed an action that yields closed world *information*.

The agent stores the limited information it has about the external world in a database $\mathcal{M}$, which we refer to as its "incomplete world theory." To represent LCW, we utilize an explicit database of meta-level sentences such as "I know all the files in `/kr94`." The sentences describe the limited instances over which $\mathcal{M}$ is in fact a complete theory of the external world. The information in the LCW database is equivalent to the "closed roles" found in knowledge-representation systems such as CLASSIC [2] and LOOM [3], to predicate completion axioms [6, 30], and to circumscription axioms [36, 35].

1.1 Contributions

While the notion of closed world reasoning appears in previous work, our novel contributions include the following:

- A sound and incomplete calculus for answering queries based on the LCW database (Section 2). The calculus answers queries such as: if the agent knows all the files in the directory `/kr94`, and knows all group-readable files on the file system, does it follow that the agent knows all the group-readable files in `/kr94`?

[1]Because our work is motivated by the softbot, most of our examples are drawn from the Internet and UNIX domains. However, we emphasize that our results are general and corresponding examples are easily found in physical domains as well.

- A sound but incomplete calculus for updating the `LCW` database as the state of the world changes (Section 3). The update calculus answers questions such as: if the agent knows the lengths of all the files in `/kr94`, and a file is added to `/kr94`, does the agent still know the lengths of all files in that directory? What if a file is deleted from `/kr94`?
- Efficient algorithms, based on the above calculi, for querying and updating a locally complete database of domain propositions. We experimentally evaluate the performance of our query and update algorithms in the Softbot domain (Section 4). We show that the algorithms are fast and that their incompleteness results in missing less than 1 percent of the `LCW` inferences requested. Overall, the `LCW` mechanism speeds up XII by a factor of one hundred, demonstrating that the trade-off we have struck between completeness and tractability is a good one.

1.2 Previous Work

Below, we briefly review the large body of related work on circumscription, autoepistemic logic, and database theory. At the end of this section, we summarize the key differences between this body of work and ours.

The bulk of previous work has investigated the *logic* of closed world reasoning (*e.g.*, [29, 12, 45, 37, 32]), and the semantics of theory updates (*e.g.*, [23, 27, 8]). Results include logical axiomatizations of the closed world assumption (CWA), exploring the relationship between the CWA and circumscription, distinguishing between knowledge base revision and knowledge base update, and more. Although decidable computational procedures have been proposed in some cases (*e.g.*, [22], and the Minimality Maintenance System [43]), they remain intractable. Update procedures have been described that involve enumerating the possible logical models corresponding to a database (e.g., [49, 7]), or computing the disjunction of all possible results of an update [28]. In contrast, we adopt the WIDTIO (When In Doubt Throw It Out [50]) policy. As [9] points out, this method is easy to implement efficiently but has the disadvantage that, in the worst case, all knowledge in the database has to be retracted. We have developed novel rules that enable us to retain closed world information in the face of updates. We believe our rules satisfy the update postulates specified in [27] and generalized in [8], but have not attempted a proof. Instead, we prove that our update scheme has polynomial worst case running time (Section 3) and we demonstrate experimentally (Section 4) that our update scheme is effective in practice.

Levy [33] has pointed out a close relationship between closed-world reasoning and the problem of detecting the independence of queries from updates. However, the computational model in the database literature (Datalog programs) is different from our own. Furthermore, polynomial-time algorithms for this problem are rare in the database literature (*e.g.*, [34] merely reports on decidability). Notable exceptions include Elkan's [10] polynomial time algorithm for conjunctive query disjointness, and Elkan's [11] approach for handling *monotonic* updates.

Recently, some excellent analyses of the computational complexity of closed-world reasoning have emerged [4, 9], which show that the different approaches described in the literature are highly intractable in the general case. Stringent assumptions are required to make closed-world reasoning tractable. For example, Eiter and Gottlob [9, page 264] show that propositional Horn theories with updates and queries of bounded size yield polynomial-time algorithms. However, all positive computational tractability results reported in [4, 9] are restricted to *propositional* theories. Motivated by the need for closed-world reasoning in modern planning algorithms, we have formulated a rather different special case where the knowledge bases record first-order information, queries are first-order conjunctions, and updates are atomic.

In short, there are three fundamental differences between the results in this paper and previous work. First, most previous work has focused on the logic of closed-world reasoning, not on its computational tractability. Second, we have formulated efficient closed-world reasoning algorithms for first-order theories of the sort used by modern planners.[2] Previous work has only found tractable algorithms for restricted classes of propositional theories. Finally, we not only show that our algorithms run in polynomial time (for queries and updates of bounded size), but also carry out a host of experiments demonstrating them to be efficient in practice (Section 4).

Our own research has its roots in the SOCRATES planner, where the difficulties caused by the open world assumption were first noted [18]. In addition, SOCRATES supported a restricted representation of local closed world information, which enabled it to avoid redundant information gathering in many cases. Provably sound and tractable calculi for LCW inference and update were introduced in [15]. In this paper, we provide a generalized inference calculus, efficient algorithms for inference and update, a precise semantics for update, and a detailed analysis of the tractability and power of our LCW mechanism. In addition, we report on a comprehensive experimental study in the Softbot domain, which includes close to 400,000 LCW queries.

1.3 Organization

The paper is organized as follows. Section 2 introduces our calculus for answering LCW queries in a static universe. In Section 3 we present our calculus for updating LCW as the world changes. Although our approach will never derive an LCW formula that doesn't hold, it is not guaranteed to derive all true LCW statements. Section 4 uses empirical techniques to show that this incompleteness is not problematic in practice, and that our approach is fast.

After concluding with a discussion of future work, we present additional details in two appendices. Appendix A proves that our rules and inference algorithms are sound, and Appendix B describes the methodology underlying our experiments.

2 Incomplete Information about the World

We've argued that when acting in complex, real-world domains such as the Internet, no agent can have complete information. In this section we present a compact representation for a class of incomplete theories of the world and describe a set of sound and tractable inference rules for reasoning about local closed world information. Subsequent sections explain how to keep the representation consistent as the world changes.

2.1 Semantics

We begin by formalizing the notion of an incomplete world theory. In essence, we adopt the standard semantics of first-order logic (adding the truth value U for unknown facts). At every point in time, the world is in a unique state, w, which may be unknown to the agent. For any ground,

[2]Since we consider formulae with an essentially unbounded number of instances, it is impractical to translate our first-order theories into propositional correlates. Furthermore, as shown in Sections 2.3 and 3, local closed-world reasoning makes essential use of first-order constructs such as unification.

atomic sentence φ, either $\mathsf{w} \models \varphi$ or $\mathsf{w} \models \neg\varphi$ hence the set of ground facts entailed by the world forms a complete logical theory, which we denote $\mathcal{W}$. Following [39, 20] and many others, we formalize an agent's incomplete information with a set of possible world states, Σ, that are consistent with its information. Since we assume that what information the agent *does* have is correct, the current world state, w, is necessarily a member of Σ. We say that φ is known to the agent (written $\Sigma \models \varphi$) just in case $\forall \mathsf{s} \in \Sigma,\ \ \mathsf{s} \models \varphi$. We use $\mathcal{S}$ to denote the *incomplete* theory of ground literals known by the agent.

$$\mathcal{S} \equiv \{\varphi \ \mid \ \Sigma \models \varphi\}$$

We say that the agent possesses complete information when Σ and w entail exactly the same set of facts, *i.e.* when $\mathcal{S} = \mathcal{W}$. Incomplete information means that there are facts, φ, such that neither $\Sigma \models \varphi$ nor $\Sigma \models \neg\varphi$; in this case we say φ is unknown to the agent. Thus, we say that an atomic formula, φ, has truth value T if $\Sigma \models \varphi$, has truth value F if $\Sigma \models \neg\varphi$, or has truth value U otherwise.

2.2 Local Closed World Information

We say that an agent has *local closed world information* (LCW) relative to a logical formula Φ if every ground sentence that unifies with Φ is either entailed by Σ or is necessarily false:[3]

$$\texttt{LCW}(\Phi) \equiv (\Sigma \models \Phi\theta) \vee (\Sigma \models \neg\Phi\theta) \text{ for all ground substitutions } \theta \tag{1}$$

In essence, this definition specifies which *parts* of the logical theory, $\mathcal{S}$, are complete (cf. [13] and others). Note that since $\mathcal{S}$ is a subset of $\mathcal{W}$, the definition of LCW amounts to a limited correspondence between Σ and w. If LCW(Φ) holds, then all states in Σ (including w) agree on the variable assignments satisfying Φ. As a concrete example, suppose that parent.dir(f,d) means "The parent directory of file f is directory d;" then we can encode the fact that an agent knows all the files in the directory /kr94 with:

$$\texttt{LCW}(\texttt{parent.dir}(f, \texttt{/kr94}))$$

If the agent knows that paper.tex and proofs.tex are in /kr94 then this LCW formula is equivalent to the following implication:

$$\forall f\ \texttt{parent.dir}(f,\ \texttt{/kr94}) \rightarrow$$
$$(f = \texttt{paper.tex}) \vee (f = \texttt{proofs.tex})$$

An LCW formula can also be understood in terms of circumscription [35]. For the example above, one defines the predicate P(x) to be true exactly when parent.dir(x, /kr94) is true, and circumscribes P in the agent's theory.

While our work can be understood within the circumscriptive framework, our implemented agent requires the ability to infer and update[4] closed world information *quickly*. We have developed

[3]We use *italics* to denote free variables and write $\Phi\theta$ to denote the result of applying the substitution θ to the formula Φ.

[4]Following [27, 28] we distinguish between *updating* a database and *revising* it. We assume that our agent's knowledge is correct at any given time point, hence there is no need to revise it. When the world changes, however, the agent may need to update its theory to remain in agreement with the world.

computationally tractable closed-world reasoning and update methods, applicable to the restricted representation languages used by modern planning algorithms. The next section describes the theory underlying LCW inference. Then, in Section 2.4 we explain how to represent LCW knowledge in a manner that facilitates efficient inference. Section 2.5 develops and analyzes an algorithm for LCW inference using these syntactic structures.

2.3 Laws of Local Closed World Information

Correctly answering LCW queries is not a simple matter of looking up assertions in a database. For instance, suppose the agent wants to establish whether it knows which files are in the /kr94 directory, and it finds that it has LCW on the contents of *every* directory. Then, *a fortiori*, it knows which files are in /kr94. That is:

$$\mathtt{LCW}(\mathtt{parent.dir}(f, d)) \models \mathtt{LCW}(\mathtt{parent.dir}(f, \mathtt{/kr94}))$$

In general, we have:

Theorem 1 (Instantiation) *If Φ is a logical formula and θ is a substitution, then* $\mathtt{LCW}(\Phi) \models \mathtt{LCW}(\Phi\theta)$.[5]

Moreover, LCW assertions can be combined to yield new ones. For example, if one knows all the group-readable files, and for each group-readable file, one knows whether that file is in /kr94, then one knows the set of group-readable files in /kr94. In general, if we know the contents of set A, and for each member of A, we know whether that member resides in another set B, then we know the intersection of sets A and B. More formally:

Theorem 2 (Composition) *If Φ and Ψ are logical formulae and* $\mathtt{LCW}(\Phi)$ *and for all substitutions σ, we have that $\Sigma \models \Phi\sigma$ implies* $\mathtt{LCW}(\Psi\sigma)$*, then we can conclude* $\mathtt{LCW}(\Phi \wedge \Psi)$.

Note that if the agent knows all the group-readable files, and it knows which files are located in /kr94, it follows that it knows the set of group-readable files in /kr94. This is a special case of the Law of Composition, in which $\mathtt{LCW}(\Psi)$ holds for all σ, but it's interesting in it's own right. In general, we have:

Corollary 3 (Conjunction) *If Φ and Ψ are logical formulae then* $\mathtt{LCW}(\Phi) \wedge \mathtt{LCW}(\Psi) \models \mathtt{LCW}(\Phi \wedge \Psi)$

The intuition behind this corollary is simple — if one knows the contents of two sets then one knows their intersection. Note that the converse is invalid. If one knows the group-readable files in /kr94, it does not follow that one knows *all* group-readable files. The rule $\mathtt{LCW}(\Phi) \models \mathtt{LCW}(\Phi \wedge \Psi)$ is also invalid. For instance, if one knows all the group-readable files, it does not follow that one knows exactly which of these files reside in /kr94.

Two additional observations regarding LCW are worth noting. If one knows the contents of two sets then one knows their union. More formally:

[5]Proofs of the theorems are sequestered in Appendix A.

Theorem 4 (Disjunction) *If Φ and Ψ are logical formulae then* $\texttt{LCW}(\Phi) \wedge \texttt{LCW}(\Psi) \models \texttt{LCW}(\Phi \vee \Psi)$

Finally, knowing whether an element is in a set is equivalent to knowing whether an element is *not* in the set:

Theorem 5 (Negation) *If Φ is a logical formula then* $\texttt{LCW}(\Phi) \models \texttt{LCW}(\neg\Phi)$.

As we explain in the next section, our representation of `LCW` sentences is restricted to positive conjunctions. As a result, the observations regarding disjunction and negation are of purely theoretical interest.

2.4 Representing Closed World Information

In this section, we explain how our agent represents its incomplete information about the world, and how it represents `LCW` in this context. Clearly an agent cannot represent Σ (a potentially infinite set of large structures) explicitly. Nor can one represent $\mathcal{S}$ explicitly, since this theory contains an infinite number of sentences.

Instead we represent the facts known by the agent with a partial database, $\mathcal{M}$, of ground literals. Formally, $\mathcal{M}$ is a subset of $\mathcal{S}$; if $\varphi \in \mathcal{M}$ then $\Sigma \models \varphi$. Since $\mathcal{S}$ is incomplete, the Closed World Assumption (CWA) cannot be applied to $\mathcal{M}$. The agent cannot automatically infer that any atomic formula absent from $\mathcal{M}$ is false. Thus, the agent is forced to represent false facts in $\mathcal{M}$, explicitly, as sentences tagged with the truth value `F`.

This observation leads to a minor paradox: the agent cannot explicitly represent in $\mathcal{M}$ *every* sentence it knows to be false (there is an infinite number of files *not* in the directory `/kr94`). Yet the agent cannot make the CWA. We adopt a simple solution: we represent local closed world information explicitly as a meta-level database, $\mathcal{L}$, containing localized closure axioms of the form $\texttt{LCW}(\Phi)$; these record *where* the agent has closed world information. Together, the $\mathcal{M}$ and $\mathcal{L}$ databases specify an agent's state of incomplete information about the world (*i.e.*, they constitute a partial representation of $\mathcal{S}$).

When asked whether it believes an atomic sentence φ, the agent first checks to see if φ is in $\mathcal{M}$. If it is, then the agent responds with the truth value (`T` or `F`) associated with the sentence. However, if $\varphi \notin \mathcal{M}$ then φ could be either `F` or unknown (truth value `U`). To resolve this ambiguity, the agent checks whether $\mathcal{L}$ entails $\texttt{LCW}(\varphi)$. If so, the fact is `F`; otherwise it is `U`. Figure 1 formalizes this intuitive procedure by providing pseudo code for the `Query` algorithm.

Note that the agent need not perform inference on $\mathcal{M}$ since it contains only ground literals, but it *may* need to perform some deduction on its `LCW` sentences. To make `LCW` inference and update tractable, we restrict the formulae in $\mathcal{L}$ to conjunctions of positive literals. As a result, we lose the ability to represent `LCW` statements that contain negation or disjunction such as "I know the protection of all files in `/kr94` *except* the files with a `.dvi` extension." Thus, for any consistent $\mathcal{M}$, $\mathcal{L}$ pair, there exists an $\mathcal{S}$ that entails the same set of `LCW` sentences, but the converse is false. On the other hand, for any theory $\mathcal{S}$, there exists (at least one) pair of databases $\mathcal{M}$, $\mathcal{L}$ which represent a strict subset of the sentences in $\mathcal{S}$. We call such a $\mathcal{M}$, $\mathcal{L}$ pair a *conservative representation* of $\mathcal{S}$.

Restricting the expressiveness of $\mathcal{L}$ provides significant efficiency gains. To see this, consider a singleton `LCW` query such as $\texttt{LCW}(\texttt{parent.dir}(f, \texttt{/kr94}))$. If $\mathcal{L}$ contains only positive conjunctions, the query can be answered in by examining only singleton `LCW` assertions indexed under the predicate

```
function Query($\Phi$, $\mathcal{M}$, $\mathcal{L}$): 3-Boolean
1    let Result := T
2    let LCW := QueryLCW($\Phi$, $\mathcal{M}$, $\mathcal{L}$)
3    for each atomic conjunct $\varphi \in \Phi$ do begin
4        if $\neg\varphi \in \mathcal{M}$ then return F
5        else if $\varphi \notin \mathcal{M}$ then
6            if LCW then return F
7            else let Result := U
8    end(* for *)
9    return Result
```

Figure 1: `Query`, a fast algorithm for determining the agent's belief in a ground conjunction. `Query` returns the truth value of Φ, if Φ can be deduced from $\mathcal{M}$. Otherwise it returns either `F`, if `QueryLCW`(Φ) succeeds, or `U` if `QueryLCW` fails (`QueryLCW` is defined in Figure 2).

`parent.dir`. If negation is allowed, however, then a combination of multiple `LCW` sentences has to be explored. For instance, $\texttt{LCW}(\Phi \wedge \Psi) \wedge \texttt{LCW}(\Phi \wedge \neg\Psi) \models \texttt{LCW}(\Phi)$. Introducing disjunctive `LCW` sentences into $\mathcal{L}$ would make matters even worse. In general, answering a singleton `LCW` query, in the presence of negation and disjunction, is NP-hard.

Theorem 6 (NP-hardness of `LCW` queries for unrestricted $\mathcal{L}$) *If $\mathcal{L}$ contains unrestricted* `LCW` *formulae, including disjunction and negation, and p is a single literal, then answering a query* `LCW`(p) *is NP-hard.*

Since our planner makes numerous singleton queries, we chose to sacrifice completeness in the interest of speed and restrict $\mathcal{L}$ to positive conjunctions.

2.5 Inference Method

We have discussed the semantics of `LCW` entailment in terms of Σ and $\mathcal{S}$, but since the agent has access only to the syntactic representations $\mathcal{L}$ and $\mathcal{M}$, we must describe inference in terms of these databases. As it turns out, the actual inference procedure directly corresponds to the laws of Instantiation and Composition described in Theorems 1 and 2. We can define the transitive closure of $\mathcal{L}$ using the following two rules.

1. **Instantiation Rule** If $\texttt{LCW}(\Phi) \in \mathcal{L}$ and θ is a substitution, then $\mathcal{L}' := \mathcal{L} \cup \{\texttt{LCW}(\Phi\theta)\}$.
2. **Composition Rule** If $\texttt{LCW}(\Phi) \in \mathcal{L}$ and for all ground substitutions θ ($\Phi\theta \in \mathcal{M} \Rightarrow \texttt{LCW}(\Psi\theta) \in \mathcal{L}$) then $\mathcal{L}' := \mathcal{L} \cup \{\texttt{LCW}(\Phi \wedge \Psi)\}$

Given the direct correspondence between these two rules and Theorems 1 and 2, this inference process (denoted $\vdash$) is clearly sound. Unfortunately, however, the inference rules are incomplete.

Theorem 7 (Incompleteness) *Let $\mathcal{M}$ be a set of consistent ground literals and let $\mathcal{L}$ be a set of positive conjunctive* `LCW` *formulae. There may exist an* `LCW` *formula* `LCW`(Φ) *that logically follows from $\mathcal{L}$ and $\mathcal{M}$, but which is not in the transitive closure of $\mathcal{L}$ given the Instantiation and Composition rules.*

```
function QueryLCW($\Phi$, $\mathcal{M}$, $\mathcal{L}$): Boolean
1    QLCW*($\Phi$, {}, $\mathcal{M}$, $\mathcal{L}$)
function QLCW*($\Phi$, Matches, $\mathcal{M}$, $\mathcal{L}$): Boolean
1    if $\Phi = \{\}$ then return T
2    else if $\Phi$ is ground and $\exists\varphi \in \Phi, \neg\varphi \in \mathcal{M}$ or $\forall\varphi \in \Phi, \varphi \in \mathcal{M}$
             then return T
3    else for $C$ such that LCW($C$) $\in \mathcal{L}$ do
4        for $\Phi' \subseteq (\Phi \cup Matches)$ such that $\exists\theta, \Phi' = C\theta$ do
5            if $\Phi' - Matches \neq \{\}$ then
6                if $\forall\sigma \in$ ConjMatch($Matches \cup \Phi'$, $\mathcal{M}$)
                         QLCW*($(\Phi - \Phi')\sigma$, $(Matches \cup \Phi')\sigma$, $\mathcal{M}$, $\mathcal{L}$)
                         then return T
     return F
```

Figure 2: The QueryLCW algorithm determines whether a conjunctive LCW statement follows from the agent's beliefs as encoded in terms of the $\mathcal{M}$ and $\mathcal{L}$ databases. Since $\mathcal{L}$ is restricted to positive conjunctions, LCW inference is reduced to the problem of matching a conjunctive LCW query against a database of conjunctive LCW assertions. A successful match occurs when repeated applications of the Composition Rule (line 6) decompose the query into sub-conjunctions, which are directly satisfied by the Instantiation Rule applied to $\mathcal{L}$ (line 4) or reduced to ground formulae and found in $\mathcal{M}$ (line 2). Note that unlike Query, the QueryLCW algorithm allows variables in its Φ input. QueryLCW calls the QLCW* helper function which calls ConjMatch in turn. ConjMatch(C, $\mathcal{M}$) performs a standard conjunctive match, returning all bindings θ, such that $\mathcal{M} \models C\theta$. The variable *Matches* represents all conjuncts of the original query that have so far been matched by some LCW formula. The query is satisfied when all conjuncts have been matched. Matching against a conjunct multiple times is permitted, which is why $\Phi' \subseteq (\Phi \cup Matches)$ in line 4. Line 5 guarantees that progress is made in each recursive invocation, so the depth of the recursion is bounded by the number of conjuncts in Φ.

Fortunately, the incompleteness of these inference rules is not a problem in practice. Section 4.2 provides an empirical demonstration that they miss substantially less than 1% of the LCW inferences requested during Softbot operation.

Note, however, that maintaining an explicit transitive closure of $\mathcal{L}$ is impractical. For each LCW formula in $\mathcal{L}$ the Instantiation Rule alone generates a number of new LCW formulae that is polynomial in the number of objects in the universe. Given finite memory resources, we choose instead to compute the closure lazily, by performing the necessary inference during queries. Figure 2 shows the inference algorithm. Since the correctness of this backward chaining algorithm is less obvious than that of the inference rules used to define the transitive closure, we prove soundness formally.

Theorem 8 (Soundness) *Let $\mathcal{M}$ be a set of consistent ground literals and let $\mathcal{L}$ be a set of positive, conjunctive* LCW *formulae such that $\mathcal{M}$ and $\mathcal{L}$ form a conservative representation of $\mathcal{S}$. If* QueryLCW(Φ, $\mathcal{M}$, $\mathcal{L}$) *returns* T *then* LCW(Φ).

In the worst case, QueryLCW has to consider all possible decompositions, which is exponential in the number of conjuncts in the query.

Theorem 9 (Complexity of QueryLCW) *Let Φ be a positive conjunction with c conjuncts, let $\mathcal{M}$ be a set of ground literals, and let $\mathcal{L}$ be a set of positive conjunctive* LCW *sentences. In the worst case,* QueryLCW(Φ, $\mathcal{M}$, $\mathcal{L}$) *may require $O((|\mathcal{L}| + |\mathcal{M}|)^{c+1})$ time.*

Fortunately, for our purposes, the number conjuncts in an LCW query is bounded by the planning domain theory used by the agent. In the UNIX and Internet domains [17], for example LCW queries are typically short ($c \leq 2$) and never greater than 4. As a result, the worst case complexity is polynomial in the size of $\mathcal{L}$ and $\mathcal{M}$. With the aid of standard indexing techniques, this yields extremely fast LCW inference in practice. In our experiments, LCW queries averaged about 2 milliseconds (see Section 4 for the details).

3 Updating Closed World Information

As the agent is informed of the changes to the external world — through its own actions or through the actions of other agents — it can gain and lose information about the world. For example, after executing the UNIX command `finger weld@june`, the agent should update $\mathcal{M}$ with the newly observed truth value for `active.on(weld, june)`. Similarly, an agent's actions can cause it to gain or lose LCW. When a file is compressed, for example, the agent loses information about its size; when all postscript files are deleted from a directory, the agent gains the information that the directory contains no such files.

This section presents a sound and efficient method for updating $\mathcal{L}$, the agent's store of LCW sentences. In Section 3.1, we start by defining the class of updates handled in terms of atomic components. The next four subsections (3.2–3.4) present policies for handling the four different types of atomic updates (see Table 1 for a summary). Section 3.5 provides an example illustrating the update mechanism. Then, in Section 3.6, we show that the updates can be performed in polynomial time. The final two sections discuss the optimality of our policies: Section 3.7 demonstrates that no valid LCW sentences are discarded by the atomic update policies, and Section 3.8 presents an optimal order for handling the atomic components of a complex update.

3.1 Representation of Change

We are motivated by the representation of dynamic change explored by planning researchers, *e.g.* ADL [40], UWL [16], and especially the action description language used by the planner guiding the Internet Softbot [17, 24, 25], which combines ideas from ADL and UWL. We assume that changes can be decomposed into a set of atomic updates, each concerning the truth value of a set of literals matching a pattern. For example, suppose that initially the agent doesn't know whether `weld` is active on the machine called `june`, so it executes a UNIX `finger` action which observes that `weld` *is* active. We can describe the resulting change in the agent's information with a single atomic update: Δ(`active.on(weld, june)`,`U` $\rightarrow$ `T`).

Below, we define this Δ notation formally, but before delving into the technical details note that the description of more complex changes may require multiple atomic components. For example, consider the UNIX action `mv /papers/kr94.tex /archive/kr94.tex` which has the effect of moving a file from one directory to another. The change due to the execution of this action can't be represented as a single update by our definitions, but it can be expressed as the following set of atomic updates:

- Δ(`parent.dir(kr94.tex, /papers)`,`T` $\rightarrow$ `F`)
- Δ(`parent.dir(kr94.tex, /archive)`,`F` $\rightarrow$ `T`)
- Δ(`parent.dir(kr94.tex, /archive)`,`U` $\rightarrow$ `T`)

The last two atomic components capture the fact that regardless of whether the agent knew whether a file named `kr94.tex` was present in `/archive` before the `mv`, the agent knows that such a file is present afterward. Informally, Δ(`parent.dir(kr94.tex, /archive)`,`F` $\rightarrow$ `T`) should be read as "If the agent knew (before the `mv`) that no file named `kr94.tex` was in `/archive`, then the agent now knows (after the `mv`) that there is such a file in `/archive`."

We assume that the atomic updates corresponding to a compound change are consistent, *i.e.* at most one atomic update changes the truth value of any single ground formula. Given this assumption, our update policy is free to process the atomic components in any order.[6] Furthermore, we assume that these atomic updates constitute a complete list of changes in the world, thus sidestepping the ramification problem [23].[7] In the example above, each of the three atomic updates changed the truth value of at most one ground literal, but in general an atomic update need not be ground; in other words, a single atomic update can affect the truth value of an unbounded number of ground literals. For example, suppose that `size(paper.tex, 14713)` $\in \mathcal{S}$ before the agent executes the UNIX command `compress paper.tex`. In this case, numerous literals change their truth value when the size of `paper.tex` becomes unknown: `size(paper.tex, 14713)` changes from `T` to `U`, while `size(paper.tex, 14712)` (and an unbounded number of similar literals) change from `F` to `U`. In this case, we summarize the change with the following pair of updates, the last of which affects the truth value of an infinite number of ground literals:

- Δ(`size(paper.tex,` x`)`,`T` $\rightarrow$ `U`)
- Δ(`size(paper.tex,` x`)`,`F` $\rightarrow$ `U`)

So far our discussion of atomic updates has been informal, but we now make the notion precise.[8] We model a change from an agent's incomplete theory, $\mathcal{S}$, to a new theory, $\mathcal{S}'$, as follows. Let φ be a positive literal possibly containing free variables, for example φ=`size(paper.tex,` x`)`. We define the sets $\mathcal{T}(\varphi, \mathcal{S})$, $\mathcal{F}(\varphi, \mathcal{S})$, and $\mathcal{U}(\varphi, \mathcal{S})$ as the ground instances of φ that are true, false or unknown, respectively:

$$\begin{aligned}
\mathcal{U}(\varphi,\mathcal{S}) &\equiv \{\psi \mid \psi = \varphi\theta \wedge \psi \notin \mathcal{S} \wedge \neg\psi \notin \mathcal{S}\} \\
\mathcal{T}(\varphi,\mathcal{S}) &\equiv \{\psi \mid \psi = \varphi\theta \wedge \psi \in \mathcal{S}\} \\
\mathcal{F}(\varphi,\mathcal{S}) &\equiv \{\psi \mid \psi = \varphi\theta \wedge \neg\psi \in \mathcal{S}\} \\
\mathcal{TF}(\varphi,\mathcal{S}) &\equiv \mathcal{T}(\varphi,\mathcal{S}) \cup \mathcal{F}(\varphi,\mathcal{S})
\end{aligned}$$

[6]Section 3.8 explains how transformations exploiting this commutativity can lead to improved performance.

[7]This is standard in the planning literature. For example, a STRIPS operator that moves block `A` from `B` to `C` must delete `on(A, B)` and also add `clear(B)` even though `clear(B)` can be defined as $\forall y\ \neg$`on(`y`,B)`.

[8]Readers satisfied with this informal explanation may wish to skip to Section 3.2.

Note that for any value of x, `size(paper.tex, `x`)` will be in exactly one of the three sets. Finally, $\Delta(\varphi, \mathtt{F} \rightarrow \mathtt{U})$ means that all elements of $\mathcal{F}(\varphi, \mathcal{S})$ become elements of $\mathcal{U}(\varphi, \mathcal{S}')$.

To define Δ precisely, we need one more notational device. For convenience in representing those literals that remain unchanged from $\mathcal{S}$ to $\mathcal{S}'$, we define the operator $\ominus$ which, given a theory $\mathcal{S}$ and a set of positive, ground literals $\mathcal{N}$, returns $\mathcal{S}$ with all positive *and negated* members of $\mathcal{N}$ removed:

$$\mathcal{S} \ominus \mathcal{N} = \{\psi | \psi \in \mathcal{S} \wedge \psi \notin \mathcal{N} \wedge \neg\psi \notin \mathcal{N}\} \tag{2}$$

To understand the intuition behind $\ominus$, consider the previous example in which φ=`size(paper.tex, `x`)`. $\mathcal{T}(\varphi,\mathcal{S}) = \{$`size(paper.tex, 14713)`$\}$, so $\mathcal{S}\ominus\mathcal{T}(\varphi, \mathcal{S})$ is equivalent to $\mathcal{S}$ with the information `size(paper.tex, 14713)` removed. So $\mathcal{S}'\ominus\mathcal{T}(\varphi, \mathcal{S}) = \mathcal{S}\ominus\mathcal{T}(\varphi, \mathcal{S})$ is simply a concise way of saying that the only change from $\mathcal{S}$ to $\mathcal{S}'$ concerns the belief that `paper.tex` has the size 14713. Neither the belief that `paper.tex` is in directory `/papers`, nor the belief that `paper.tex` *doesn't* have size 14712 has changed. In other words, $\mathcal{S}'\ominus\mathcal{T}(\varphi, \mathcal{S}) = \mathcal{S}\ominus\mathcal{T}(\varphi, \mathcal{S})$ is a frame axiom stating that nothing has changed aside from the truth value of literals contained in $\mathcal{T}(\varphi,\mathcal{S})$.

The formal definition of $\Delta(\varphi, \mathtt{T} \rightarrow \mathtt{F})$ appears below.

$$\begin{array}{ll} \Delta(\varphi, \mathtt{T} \rightarrow \mathtt{F}) & \mathcal{T}(\varphi, \mathcal{S}') = \emptyset \wedge \\ & \mathcal{F}(\varphi, \mathcal{S}') = \mathcal{F}(\varphi, \mathcal{S}) \cup \mathcal{T}(\varphi, \mathcal{S}) \wedge \\ & \mathcal{S}'\ominus\mathcal{T}(\varphi, \mathcal{S}) = \mathcal{S}\ominus\mathcal{T}(\varphi, \mathcal{S}) \end{array} \tag{3}$$

Definitions for most other truth values are similar, but one bears discussion: $\Delta(\varphi, \mathtt{U} \rightarrow (\mathtt{T} \vee \mathtt{F}))$. There is no need to specify the change *from* a disjunction of truth values because such a change can be decomposed into a pair of simpler updates. Specifically, there is no need to define $\Delta(\varphi, (\mathtt{T} \vee \mathtt{F}) \rightarrow \mathtt{U})$ because it would be equivalent to the set containing both $\Delta(\varphi, \mathtt{T} \rightarrow \mathtt{U})$ and $\Delta(\varphi, \mathtt{F} \rightarrow \mathtt{U})$. However, some useful changes can not be modeled without using a disjunction on the right hand side of the arrow. For example, the UNIX `ls -a` command observes the name of all files in a directory argument; when applied to the `/tex` directory, the command induces the following update: $\Delta(\mathtt{parent.dir}(o, \mathtt{/tex}), \mathtt{U} \rightarrow (\mathtt{T} \vee \mathtt{F}))$ because some files are observed to be present while all others are now known to be absent. We define the update formally as follows:

$$\begin{array}{lll} \Delta(\varphi, \mathtt{U} \rightarrow (\mathtt{T} \vee \mathtt{F})) & \equiv & \mathcal{U}(\varphi, \mathcal{S}') = \{\} \wedge \\ & & \mathcal{T}(\varphi, \mathcal{S}') \subseteq \mathcal{T}(\varphi, \mathcal{S}) \wedge \\ & & \mathcal{F}(\varphi, \mathcal{S}') \subseteq \mathcal{F}(\varphi, \mathcal{S}) \wedge \\ & & \mathcal{TF}(\varphi, \mathcal{S}') = \mathcal{TF}(\varphi, \mathcal{S}) \cup \mathcal{U}(\varphi, \mathcal{S}) \wedge \\ & & \mathcal{S}'\ominus\mathcal{U}(\varphi, \mathcal{S}) = \mathcal{S}\ominus\mathcal{U}(\varphi, \mathcal{S}) \end{array} \tag{4}$$

In subsequent sections we describe a process for handling these updates. Specifically, we assume that the agent starts with a $\mathcal{M}$, $\mathcal{L}$ pair that forms a conservative representation of an incomplete theory $\mathcal{S}$. When informed of a change, *i.e.* a set of atomic updates described using the Δ notation defined above, the agent must create a new $\mathcal{M}'$ and $\mathcal{L}'$, ensuring that these databases are still conservative representations, yet retain as much information as possible. We present our method for processing updates as a set of rules and state them as theorems since they are sound: *i.e.*, they preserve conservative representations.

By distinguishing between transitions to and from U truth values, $\mathcal{L}$ updates can be divided into four mutually exclusive and exhaustive cases which we call Information Gain, Information Loss, Domain Growth, and Domain Contraction. In the next four sections, we consider each case in turn.

3.2 Information Gain

An agent gains information when it executes an information-gathering action (*e.g.*, UNIX `wc` or `ls`), or when a change to the world results in information gain. In general, if an agent gains information, it cannot lose `LCW`, and will gain `LCW` in some cases as explained below.

Theorem 10 (Information Gain Rule) *Let $\mathcal{L}$ be part of a conservative representation. If an atomic change is of the form $\Delta(\varphi, \mathtt{U} \rightarrow (\mathtt{T} \vee \mathtt{F}))$, then $\mathcal{L}' := \mathcal{L} \cup \{\mathtt{LCW}(\varphi)\}$ yields a conservative representation.*

The Information Gain Rule is obviously true when φ is ground, in which case this `LCW` update would be vacuous. However, the rule can also apply when φ contains free variables. For example, execution of the UNIX command `ls -a /tex` produces a $\Delta(\mathtt{parent.dir}(f, \mathtt{/tex}), \mathtt{U} \rightarrow (\mathtt{T} \vee \mathtt{F}))$ update, where `f` is a free variable. As a result, the Information Gain Rule concludes that the agent now knows all files in the `/tex` directory: $\mathtt{LCW}(\mathtt{parent.dir}(f, \mathtt{/tex})$.

If the unique value of a function is determined, such as the word count of a file, then a ground update can lead to `LCW` of a lifted sentence. For example, if an agent discovers that `foo.tex` has length `5512` then it knows that the length is neither `5513` nor any other value. In other words, the agent knows $\mathtt{LCW}(\mathtt{word.count(foo.tex,}\ x))$.

In order to make this precise, we define the cardinality of a (lifted) literal, φ, in a set of sentences (*e.g.*, $\mathcal{M}$ or $\mathcal{W}$) to be the number of ground literals in the set that unify with φ.

$$\mathtt{Cardinality}(\varphi, \mathcal{M}) = |\{\phi \in \mathcal{M} \text{ such that } \phi \text{ is ground and } \exists\theta\phi = \varphi\theta\}$$

When an update causes the cardinality of φ to be the same in $\mathcal{M}$ as it is in $\mathcal{W}$, we can conclude that we have $\mathtt{LCW}(\varphi)$:

Theorem 11 (Counting Rule (after [46])) *Let $\mathcal{M}$, $\mathcal{L}$ be a conservative representation and let θ be a substitution. If an atomic change of the form $\Delta(\varphi\theta, \mathtt{U} \rightarrow \mathtt{T})$ causes $\mathtt{Cardinality}(\varphi, \mathcal{M}') = \mathtt{Cardinality}(\varphi, \mathcal{W})$ then $\mathcal{L}' := \mathcal{L} \cup \{\mathtt{LCW}(\varphi)\}$ yields a conservative representation.*

To utilize the Counting Rule in practice, our agent relies on a set of explicit axioms that define the cardinality of predicates in $\mathcal{W}$. For example, we tell our agent that `word.count` is functional in its first argument as is `file.size` *etc.*.

In some cases the Information Gain and Counting Rules, used in conjunction with the Composition Rule, can lead to additional forms of local closed world information. For example, the UNIX command `ls -la /tex` detects the size of all files in the `/tex` directory. The $\Delta(\mathtt{parent.dir}(f, \mathtt{/tex}), \mathtt{U} \rightarrow (\mathtt{T} \vee \mathtt{F}$ update allows the Information Gain Rule to conclude `LCW(parent.dir(f, /tex))` as explained above. Suppose that there are two files in the directory, `foo.tex` and `bar.tex`, which are `55` and `66` bytes long respectively. The following two updates $\Delta(\mathtt{size(foo.tex,\ 55)}, \mathtt{U} \rightarrow \mathtt{T})$ and

Δ(size(bar.tex, 66),U $\rightarrow$ T) will yield LCW(size(foo.tex, l)) and LCW(size(bar.tex, l)) via the Counting Rule. The Composition Rule can now be used to conclude

$$\texttt{LCW(parent.dir(}f\texttt{,bin)} \wedge \texttt{size(}f,l\texttt{))}$$

Cases like this (*i.e.*, where LCW results from the execution of an action) are so common that we apply the Composition Rule proactively. In other words we add the LCW statement above at the time that the Δ updates are received rather than waiting for an LCW query. The details of this optimization are difficult to explain without a full description of the planner's action language [24, 25], but it is important to note that the policy does not add LCW sentences of arbitrary length to $\mathcal{L}$. For example, in the UNIX domain tested in Section 4, all LCW sentences added to $\mathcal{L}$ had fewer than 3 conjuncts.

3.3 Information Loss

An agent loses information when a literal, previously known to be true (or false), is asserted to be unknown. When a UNIX file is compressed, for example, information about its size is lost. In general, when information is lost about some literal, all LCW statements "relevant" to that literal are lost. To make our notion of relevance precise, we begin by defining the set PREL(φ) to denote the LCW assertions *potentially relevant* to a positive literal φ:[9]

$$\texttt{PREL}(\varphi) \equiv \{\Phi \in \mathcal{L} \quad \exists x \in \Phi, \exists\theta, \exists\alpha, x\theta = \varphi\alpha\}$$

For example, if an agent has complete information on the size of all files in /kr94, and a file lcw.tex in /kr94 is compressed (φ = size(lcw.tex, n)), then the sentence

$$\texttt{LCW(parent.dir(}f\texttt{, /kr94)} \wedge \texttt{size(}f,c\texttt{))} \tag{5}$$

is in PREL(φ) and should be removed from $\mathcal{L}$. Unfortunately, when a file in the directory /bin is compressed, the above LCW sentence is still in PREL(φ) (x = size(f,c)) even though the agent retains complete information about the files in /kr94. Clearly, LCW sentence 5 ought to remain in $\mathcal{L}$ in this case. To achieve this behavior, we check whether the agent has information indicating that the LCW sentence does not "match" the compressed file. If so, the LCW sentence remains in $\mathcal{L}$. In general, we define the set of LCW assertions *relevant* to a positive literal φ to be the following subset of PREL(φ):

$$\texttt{REL}(\varphi) \equiv \{\Phi \in \texttt{PREL}(\varphi) \quad \mathcal{L} \wedge \mathcal{M} \nvdash \neg(\Phi - x)\theta\}$$

where, from the definition of PREL(φ), $\exists x \in \Phi, \exists\theta, \exists\alpha$, such that $x\theta = \varphi\alpha$. If θ is not a complete mapping, then to exclude Φ from REL(φ), it is necessary that all possible ground instances of $(\Phi - x)\theta$ are known to be false, or equivalently, that LCW($(\Phi - x)\theta$), and so there is no match to $(\Phi - x)\theta$ in $\mathcal{M}$. We can now state our update policy for Information Loss:

Theorem 12 (Information Loss Rule) *Let $\mathcal{L}$ be part of a conservative representation. If an atomic change is either of the form $\Delta(\varphi,\texttt{T} \rightarrow \texttt{U})$ or $\Delta(\varphi,\texttt{F} \rightarrow \texttt{U})$, then $\mathcal{L}' := \mathcal{L} - \texttt{REL}(\varphi)$ yields a conservative representation.*

[9]Since the sentences in $\mathcal{L}$ are conjunctions of positive literals, we use the notation $\varphi \in \Phi$ to signify that φ is one of Φ's conjuncts, and the notation $\Phi - \varphi$ to denote the conjunction Φ with φ omitted.

Note that compressing a file `foo` in `/bin` does not remove LCW sentence 5. To see this, let $x = \texttt{size}(f, c)$, $\theta = (\texttt{foo}/f)$, and $\phi_i = \texttt{parent.dir}(f, \texttt{/kr94})$. Since `foo` is known to be in `/bin` (and `parent.dir` is a functional relation), from $\mathcal{L} \wedge \mathcal{M}$ one can prove that $\neg\phi_i\theta$. Hence, $(\mathcal{L} \wedge \mathcal{M} \nvdash \neg\phi_i\theta)$ is false and Φ is not included in $\mathtt{REL}(\varphi)$. Note also that, given our assumptions (correct information, *etc.*), information is only lost when the world's state changes.

3.4 Changes in Domain

Finally, we have the most subtle cases: an agent's theory changes without strictly losing or gaining information. For example, when the file `ai.sty` is moved from the `/tex` directory to `/kr94`, we have that the updated $\mathcal{M}' \neq \mathcal{M}$ but neither database is a superset of the other. When the theory changes in this way, the domain of sentences containing $\texttt{parent.dir}(f, \texttt{/kr94})$ grows whereas the domain of sentences containing $\texttt{parent.dir}(f, \texttt{/tex})$ contracts. LCW information may be lost in sentences whose domain grew. Suppose that, prior to the file move, the agent knows the word counts of all the files in `/kr94`; if it does not know the word count of `ai.sty`, then that LCW assertion is no longer true. As with Information Loss, we could update $\mathcal{L}$ by removing the set $\mathtt{REL}(\varphi)$. However, this policy is overly conservative. Suppose, in the file move described above, that the agent *does* know the word count of `ai.sty`. In this case, it retains complete information over the word counts of the files in `/kr94`, even after `ai.sty` is moved.

More generally, when the domain of an LCW sentence grows, but the agent has LCW on the new element of the domain, the LCW sentence can be retained. To make this intuition precise, we define the following "minimal" subset of $\mathtt{REL}(\varphi)$:

$$\mathtt{MREL}(\varphi) = \{\Phi \in \mathtt{REL}(\varphi) \quad \mathcal{M} \wedge \mathcal{L} \nvdash \mathtt{LCW}((\Phi - x)\theta)\}$$

where, from definition of $\mathtt{PREL}(\varphi)$, $\exists x \in \Phi, \exists\theta, \exists\alpha$, such that $x\theta = \varphi\alpha$. We can now state our update policy for Domain Growth:

Theorem 13 (Domain Growth Rule) *Let $\mathcal{L}$ be part of a conservative representation. If an atomic change is of the form $\Delta(\varphi, \mathtt{F} \rightarrow \mathtt{T})$, then $\mathcal{L}' := \mathcal{L} - \mathtt{MREL}(\varphi)$ yields a conservative representation.*

When the domain of a sentence contracts, no LCW information is lost. For instance, when a file is removed from the directory `/kr94`, we will still know the size of each file in that directory.

Theorem 14 (Domain Contraction Rule) *Let $\mathcal{L}$ be part of a conservative representation. If an atomic change is of the form $\Delta(\varphi, \mathtt{T} \rightarrow \mathtt{F})$, then $\mathcal{L}' := \mathcal{L}$ yields a conservative representation.*

The rules guarantee that $\mathcal{L}$ does not contain invalid LCW assertions, so long as the agent is apprised of any changes to the world state. However, for the sake of tractability, the rules are conservative — $\mathcal{L}'$ may be incomplete. For example, if `ai.sty` is moved into `/kr94` as in the earlier example, but the word count of `ai.sty` is unknown, we might wish to say that we know the word counts of all the files in `/kr94` *except* `ai.sty`. However, we refrain from storing negated sentences in $\mathcal{L}$ because such sentences would slow down LCW inference, as shown in Section 2.4.

$\Delta(\varphi, \rightarrow)$	Update Rule	$\mathcal{L} \rightarrow \mathcal{L}'$	UNIX Examples
U $\rightarrow$ (F $\vee$ T)	Information gain	$\mathcal{L}'$:=$\mathcal{L} \cup$ LCW(φ)	`ls`, `wc`
(F $\vee$ T) $\rightarrow$ U	Information loss	$\mathcal{L}'$:=$\mathcal{L} -$ REL(φ)	`compress`
T $\rightarrow$ F	Domain contraction	$\mathcal{L}' = \mathcal{L}$	`rm`
F $\rightarrow$ T	Domain growth	$\mathcal{L}'$:=$\mathcal{L} -$ MREL(φ)	`cp`

Table 1: A summary of the mutually exclusive and exhaustive atomic update rules for the LCW database $\mathcal{L}$. All update rules except Domain Contraction have the potential to introduce incompleteness into $\mathcal{L}$.

3.5 Example

Table 1 provides a capsule summary of our LCW update rules discussed in the previous four sections. Below, we provide an extended example of the update machinery in action. Specifically, we illustrate how the update rules affect $\mathcal{L}$ as the following command sequence is executed:

```
ls -al /kr94
ls -al /papers
mv /kr94/kr.ps /papers
compress /papers/kr.ps
```

Initially, both the databases representing ground formulae ($\mathcal{M}$) and LCW formulae ($\mathcal{L}$) are empty. The execution of `ls -al` in the directory `/kr94` reveals the files in the directory and their size in bytes. For brevity, we will ignore other effects. Suppose that the files are `kr.tex` and `kr.ps`, and their sizes are 100 and 300 respectively. In this case, $\mathcal{M}$ is updated as follows:

$\mathcal{M}$ = {parent.dir(kr.tex, /kr94),
size(kr.tex, 100),
parent.dir(kr.ps, /kr94),
size(kr.ps, 300)}

The agent knows the contents of /kr94, and the sizes of all the files therein. In addition, because the parent directory and size of each file are unique, the Counting Rule implies that the agent has LCW on the size and parent directory of each file. This information is recorded in the LCW database as follows:

$\mathcal{L}$ = {LCW(parent.dir(*f*, /kr94)),
LCW(parent.dir(*f*, /kr94) $\wedge$ size(*f*, *l*))
LCW(parent.dir(kr.tex, *d*)),
LCW(size(kr.tex, *l*)),
LCW(parent.dir(kr.ps, *d*)),
LCW(size(kr.ps, *l*))}

The directory /papers is initially empty. Thus, after executing `ls -al` in the directory /papers, the agent records LCW information for the directory /papers, but no updates are made to $\mathcal{M}$.

$\mathcal{L}$ = {LCW(parent.dir(f, /kr94)),
LCW(parent.dir(f, /kr94) $\wedge$ size(f, l)),
LCW(parent.dir(kr.tex, d)),
LCW(size(kr.tex, l)),
LCW(parent.dir(kr.ps, d)),
LCW(size(kr.ps, l)),
LCW(parent.dir(f, /papers)),
LCW(parent.dir(f, /papers) $\wedge$ size(f, l))}

Moving the file kr.ps from the directory /kr94 to the directory /papers results in both Domain Contraction to the directory /kr94, and Domain Growth to the directory /papers. $\mathcal{M}$is updates as follows:

$\mathcal{M}$ = {parent.dir(kr.tex, /papers),
size(kr.tex, 100),
parent.dir(kr.ps, /kr94),
size(kr.ps, 300)}

There is no change to $\mathcal{L}$ due to Domain Contraction. However, Domain Growth could potentially result in statements being retracted from $\mathcal{L}$. This example illustrates the advantage of having the Domain Growth Rule retract the set of MREL sentences from $\mathcal{L}$, rather than naively retracting the set of REL sentences. There are three statements in REL:

REL(parent.dir(kr.ps, /papers)) = {LCW(parent.dir(f, /papers)),
LCW(parent.dir(kr.ps, d)),
LCW(parent.dir(f, /papers) $\wedge$ size(f, l))}

However, the MREL of the update is empty, due to the fact that we have LCW on the size of kr.ps

MREL(parent.dir(kr.ps, /papers)) = {}

As a result, $\mathcal{L}$ remains unchanged after the mv command is executed. However, if we did not know the size of kr.ps when it was moved, we would have lost LCW on the size of the files in the directory /papers.

The last action in our example is compressing the file kr.ps. This action illustrates the advantage of retracting REL rather than PREL in the Information Loss Rule. After the file kr.ps is compressed, its size becomes unknown. Thus, $\mathcal{M}$ shrinks to:

{parent.dir(kr.tex, /papers),
size(kr.tex, 100),
parent.dir(kr.ps, /kr94)}

The set of PREL statements is:

$$\text{PREL}(\texttt{size}(\texttt{kr.ps}, l)) = \{\text{LCW}(\texttt{parent.dir}(f, \texttt{/kr94}) \wedge \texttt{size}(f, l)), \text{LCW}(\texttt{parent.dir}(f, \texttt{/papers}) \wedge \texttt{size}(f, l)), \text{LCW}(\texttt{size}(\texttt{kr.ps}, l))\}$$

In contrast, because we know that kr.ps is now in the directory /papers, the set of REL statements contains only the following:

$$\text{REL}(\texttt{size}(\texttt{kr.ps}, l)) = \{\text{LCW}(\texttt{parent.dir}(f, \texttt{/papers}) \wedge \texttt{size}(f, l)), \text{LCW}(\texttt{size}(\texttt{kr.ps}, l))\}$$

Thus after the compress action is executed, we remove the REL statement from $\mathcal{L}$, obtaining:

$$\mathcal{L} = \{\text{LCW}(\texttt{parent.dir}(f, \texttt{/kr94})), \text{LCW}(\texttt{parent.dir}(f, \texttt{/kr94}) \wedge \texttt{size}(f, l)), \text{LCW}(\texttt{parent.dir}(\texttt{kr.tex}, d)), \text{LCW}(\texttt{size}(\texttt{kr.tex}, l)), \text{LCW}(\texttt{parent.dir}(\texttt{kr.ps}, d)), \text{LCW}(\texttt{parent.dir}(f, \texttt{/papers}))\}$$

3.6 Computational Complexity of Updates

As stated earlier, our motivation for formulating conservative update rules has been to keep LCW update tractable. We make good on this promise below by considering the complexity of applying each update rule.

- **Information Gain:** The Information Gain Rule implies that no sentences have to be retracted from $\mathcal{L}$. LCW sentences may be added by the Information Gain Rule in time that is independent of the size of $\mathcal{L}$. The Counting Rule requires counting ground instances of a literal in $\mathcal{M}$, which requires time that is at most linear in the size of the database. For the most part, the Composition Rule is applied only in response to LCW queries; when applied proactively after action execution, it requires time that is linear in the number of atomic Δ updates, which correspond to the action's effects.

- **Information Loss:** First, the agent computes the set PREL(φ), which takes time linear in the size of $\mathcal{L}$ in the worst case. Next, the agent computes REL(φ) from PREL(φ). Since this means determining whether $\mathcal{L} \wedge \mathcal{M} \vdash \neg(\Phi - \varphi)\theta$, the agent can incur an $O((|\mathcal{L}| + |\mathcal{M}|)^c)$ cost for an LCW query (where c denotes the maximum number of conjuncts in a sentence in $\mathcal{L}$) for each element in PREL (Theorem 9). Thus, to determine the worst case complexity of Information Loss, one must calculate the maximum length of the elements of $\mathcal{L}$. This is easy because there are only two ways that LCW sentences can be added to $\mathcal{L}$: via the Information Gain Rule or via proactive use of the Composition Rule. Since the first method only adds literals, $c = 1$. While the Composition Rule could (in theory) lead to an LCW sentence of arbitrary length, it is only used for forward chaining in a limited context (as explained in Section 3.2). As a result, it never adds sentences that are longer than a constant bound determined by the planning-domain. For the UNIX and Internet domains, this constant is 3. In summary, $|$PREL(φ)$|$ is potentially linear in the size of $\mathcal{L}$, so computing REL(φ) from PREL(φ) could take $O(|\mathcal{L}|(|\mathcal{L}| + |\mathcal{M}|)^c)$. This cost dominates the time for the entire update.

- **Domain Growth:** The agent has to compute the set REL(φ) which, as explained above, is polynomial in the size of $\mathcal{L}$ and $\mathcal{M}$. Computing MREL(φ) from REL(φ) is linear in the size of REL, but polynomial in the size of $\mathcal{L}$ and $\mathcal{M}$, since additional bounded-length LCW queries may be involved. The agent then removes each element of the set from $\mathcal{L}$, which takes time linear in the size of the set MREL(φ). Thus the whole operation is $O(|\mathcal{L}|(|\mathcal{L}| + |\mathcal{M}|)^c)$.

- **Domain Contraction:** $\mathcal{L}$ remains unchanged in this case

We summarize the preceding discussion with the following theorem. The use of standard indexing techniques (*e.g.*, hashing on the predicates in φ) renders the effective polynomial coefficient somewhat lower than the conservative bound we present.

Theorem 15 (Tractability of Updates) *Let $\mathcal{M}$ be a set of ground literals, and let $\mathcal{L}$ be a set of positive conjunctive* LCW *sentences such that no member of $\mathcal{L}$ has more than c conjuncts. Updating $\mathcal{L}'$ in response to an atomic change requires time that is at most $O(|\mathcal{L}|(|\mathcal{L}| + |\mathcal{M}|)^c)$.*

3.7 Optimality of Atomic Update Policies

Since sentences in $\mathcal{L}$ are restricted to positive conjunctions, and since our update rules are conservative, the update process is incomplete. Nevertheless, it is easy to see that our algorithm is better than the trivial update algorithm ($\mathcal{L}'$:=$\{\}$). In our softbot's domain, for example, the Information Gain and Counting Rules enable us to derive LCW from a wide range of "sensory" actions, including `pwd`, `wc`, `grep`, `ls`, `finger`, and many more. Furthermore, our update rules retain LCW in many cases. For example, changes to the state of one "locale" (such as a directory, a database, an archive, etc.) do not impact LCW on other locales. This feature of our update calculus applies to physical locales as well.

Below, we make a much stronger claim, that the sets of sentences retracted by theorems 12 through 14 are, in fact, minimal. Every sentence retracted is invalid and *must* be removed from $\mathcal{L}$ to maintain soundness. This statement is trivially true for Domain Contraction where no sentences are retracted. Clearly, we cannot do better than that. The following theorem asserts that each LCW sentence retracted due to Information Loss is, in fact, invalid.

Theorem 16 (Minimal Information Loss) *Let $\mathcal{M}$, $\mathcal{L}$ be a conservative representation, and let φ be a positive literal. Let A denote an atomic change of the form $\Delta(\varphi, \mathtt{T} \rightarrow \mathtt{U})$ or of the form $\Delta(\varphi, \mathtt{F} \rightarrow \mathtt{U})$. If $\Phi \in \mathtt{REL}(\varphi)$ then $\mathtt{LCW}(\Phi)$ does not hold after A has occurred.*

Remarkably, the corresponding result holds for Domain Growth.

Theorem 17 (Minimal Domain Growth) *Let $\mathcal{M}$, $\mathcal{L}$ be a conservative representation, and let φ be a positive literal. Let A denote an atomic change of the form $\Delta(\varphi, \mathtt{F} \rightarrow \mathtt{T})$. If $\Phi \in \mathtt{MREL}(\varphi)$ then $\mathtt{LCW}(\Phi)$ does not hold after A has occurred.*

Are the update rules for Information Loss and Domain Growth the best possible? At first blush, the answer to this question would seem to be yes, since the rules are sound and they retract the minimal set of sentences from $\mathcal{L}$. So what more could we want? However, this observation overlooks the key fact that inference in our framework is lazy, so that when the sentence φ is retracted we effectively also retract $\varphi\sigma$ for any variable substitution σ. Above, we claimed that the sentence φ really ought to be retracted, but we didn't claim that the sentence $\varphi\sigma$ (which is weaker!) is invalid. In fact, there are cases where such sentences are valid. For example, consider the case where we have `LCW` on the size of all the files in the directory `/bin`, but the file `a.out` in that directory is compressed. Our update rule for Information Loss would retract the `LCW` statement, when, in fact, a weaker statement that we know the size of all the files in `/bin` — except `a.out` — is true. Since the sentences in $\mathcal{L}$ are conjunctions of positive literals, we have no way of expressing the above statement.

3.8 Optimal Order of Atomic Updates

So far our discussion has been restricted to atomic updates, but many updates consist of a set of such atoms. While the order in which these atomic components are handled does not affect eventual contents of $\mathcal{M}$, this is not true for $\mathcal{L}$. Of course the $\mathcal{M}$, $\mathcal{L}$ pair will be a conservative representation of $\mathcal{S}$ regardless of the order chosen, but some orderings will discard more sentences from $\mathcal{L}$ than others. For example, consider the following imaginary UNIX command `gen-file` < *dir* > which creates a new, uniquely named file in directory < *dir* >, gives it zero size, and returns the name. The effects of executing `gen-file /tex` and having it return the name `G003` are as follows:

$\Delta(\mathtt{parent.dir(G003,\ /tex)}, \mathtt{F} \rightarrow \mathtt{T})$
$\Delta(\mathtt{size(G003,\ 0)}, \mathtt{U} \rightarrow \mathtt{T})$

Now, suppose that before executing `gen-file /tex` the agent knew the names and lengths of all files in `/tex`.

$$\mathtt{LCW}(\mathtt{parent.dir}(f,\ \mathtt{/tex}) \wedge \mathtt{size}(f, c))$$

If the atomic updates are processed in the order given, then the Domain Growth Rule will eliminate this `LCW` sentence from $\mathcal{L}$, but if the updates are processed in the reverse order then that retraction is unnecessary.

To obtain an optimal order, an agent must be sure that as many sentences are added to $\mathcal{L}$ as possible and that as few are removed as possible. We believe the following order suffices:

1. Process Domain Contraction updates.
2. Process Information Gain updates and apply the Counting Rule.
3. Process Domain Growth updates.
4. Process Information Loss updates.

The insight behind this order is as follows. The only type of updates that remove items from $\mathcal{L}$ are Domain Growth and Information Loss which remove the sets MREL(φ) and REL(φ) respectively. More information present in $\mathcal{M}$, $\mathcal{L}$ means that we'll have more LCW information and be able to prove more sentences are false. This in turn means that the REL and MREL sets will be as small as possible. So Information Gain and Domain Contraction updates should be processed first.[10] It can be useful to process Domain Contraction Updates before Information Gain because this improves the chance that the proactive application of the Composition Rule (Section 3.2) will result in new LCW sentences.

Ultimately, the test of any mechanism for closed-world reasoning — conservative or not — is its impact on the agent's performance. In the next section we describe preliminary experiments that suggest ours is extremely effective in practice, dramatically improving the Softbot's performance.

4 Experimental Results

In the previous sections we argued that our LCW mechanism is computationally tractable, but incomplete. However, asymptotic analysis is not always a good predictor of real performance, and incompleteness is a matter of degree. To evaluate our LCW machinery empirically, we incorporated LCW into the XII planner [24] and measured its impact on the performance of the Internet Softbot [17]. In this section, we address the following questions experimentally:

Speed: What is the speed of LCW queries and LCW updates as a function of the size of the LCW database and the size of LCW formulae?

As shown in Section 4.1, LCW inference is very fast, 2 milliseconds per query, and updates are even faster: 1.2 milliseconds. Times increase for longer queries, but are relatively unaffected by the size of $\mathcal{L}$ and $\mathcal{M}$.

- **Completeness:** Because our LCW database is incomplete, a query may result in the truth value U even though its "true" truth value is F (Figure 1). How often does this occur as the database processes a sequence of queries and updates issued by the XII planner?

Section 4.2 argues that the incompleteness of our LCW formulation is more of a theoretical concern than a practical one. In over 99% of the cases that occur in practice, the LCW mechanism deduces the correct answer.

[10]The only problem with this argument would arise if a Domain Growth or Information Loss update removed an LCW sentence that had been added by Information Gain, but this is impossible because we assume that every set of updates corresponding to a single action or event is mutually consistent.

- **Impact:** What is the effect of the LCW machinery on the speed with which the XII planner can control the Internet Softbot? In particular, does the use of LCW information improve the agent's performance enough to offset the cost of LCW inference and update?

Even though LCW inference is fast and effectively complete, it is still conceivable that its use might detract from an agent's overall performance. Section 4.3 shows that this is not the case; indeed, LCW's ability to focus search and eliminate redundant sensing operations yields a 100-fold improvement in overall performance.

The experiments in the remainder of this section differ from most empirical work reported in the planning literature along both the dimensions of realism and size. Since the XII planner controls the actual execution of the Internet Softbot, we know that its domain theory is realistic in different sense from simulated robot domains such as the Blocksworld or the Tireworld — each action description in the theory is an accurate description of an actual UNIX command. In addition, our experiments are noteworthy in their comprehensive aspect — we report on data collected from thousands of planning problems resulting in over 390,000 LCW queries.

4.1 Factors Influencing LCW Speed

The interesting questions regarding LCW speed are "How fast are queries and updates on average?" and "How does the time vary as a function of the length of the LCW formula and the size of $\mathcal{L}$ and $\mathcal{M}$?" To answer these questions we generated randomly several thousand goals as explained in Appendix B. In the course of solving these planning problems, the XII planner issued over 390,000 LCW queries and performed numerous updates. On average, answering an LCW query required 2 milliseconds while processing an update took 1.2 milliseconds.

In answer to the second question, Figure 3 shows query time as a function of the length of the query and the size of the $\mathcal{L}$ database.[11] The graph shows the results for query sizes of up to four conjuncts; larger queries don't occur in our UNIX domain. In fact, large queries are uncommon in our domain; even queries with four conjuncts occur only as a result of user-supplied $\forall$ goals. The slow growth of query time as a function of $|\mathcal{L}|$ is due to the use of hashing, as opposed to the more expensive linear-time search assumed in our complexity analysis (Section 3.6). As mentioned earlier, updates are even faster than queries on average.

4.2 Completeness

Because our LCW machinery is incomplete, QueryLCW(Φ) may return "No" when the agent does in fact have LCW(Φ). We refer to this event as an LCW *miss*. Below, we explain how we measured the percentage of LCW queries that result in LCW misses.

The problem of detecting LCW misses raises a thorny issue. LCW is a semantic notion defined in terms of Σ, the infinite set of possible world states that are consistent with the agent's observations. How can we measure, experimentally, the percentage of times when the agent ought to have LCW, but does not? Comprehending the answer to this question requires a deep understanding of the formal basis for LCW. The definition of LCW in Section 2.2, combined with the fact that if $\varphi \in \mathcal{M}$ then $\Sigma \models \varphi$, implies that if LCW(Φ) then there is a one-to-one correspondence between instances of Φ in $\mathcal{W}$ and in $\mathcal{M}$.

[11] The size of $\mathcal{M}$ is strongly correlated with the size of $\mathcal{L}$, resulting in a very similar graph of query time with respect to the size of $\mathcal{M}$.

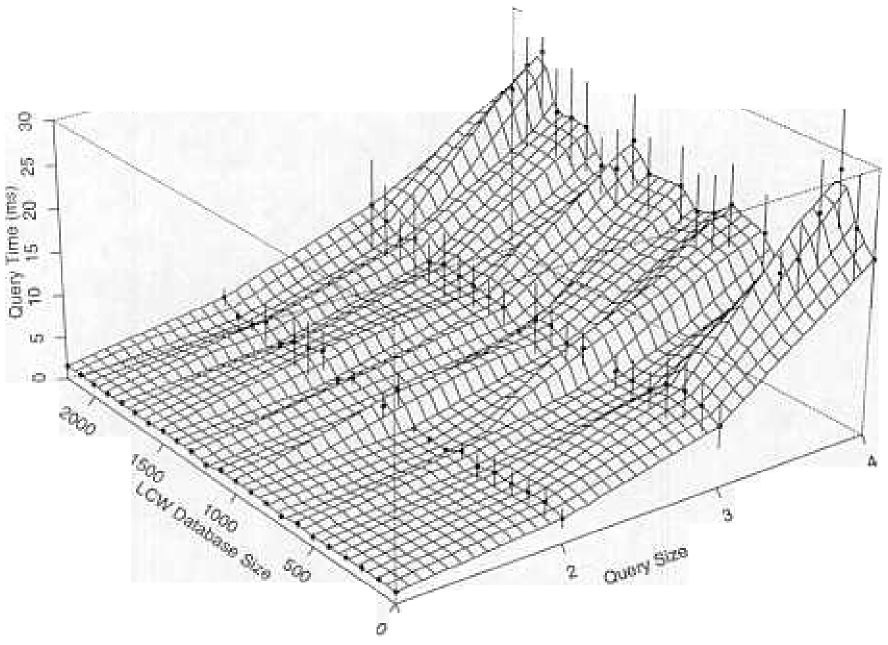

Figure 3: CPU Time for LCW queries as a function of the size of the LCW database $\mathcal{L}$, and the number of conjuncts in the query. Experiments were run on a Sun SPARCstation 20; vertical bars indicate 95% confidence intervals. Note that even as $\mathcal{L}$ grows large, the average query time is approximately 2 milliseconds. Over 90% of the 390,000 queries contained fewer than three conjuncts. Because the sizes of $\mathcal{L}$ and $\mathcal{M}$ are strongly correlated in all of our experiments, the graph of query time with respect to the size of $\mathcal{M}$ is similar, and thus omitted.

This one-to-one correspondence is important because it can be tested experimentally via simulation. Appendix B describes the methodology in more detail, but the idea is simple. We replace the agent's effectors (which normally manipulate an actual UNIX shell), with new procedures that update and sense a simulation of a UNIX computer. Although the simulated environment doesn't model *every* aspect of UNIX, it is complete relative to every action that could be executed in service of the test suite.

Thus, to check whether QueryLCW(Φ) has resulted in an LCW miss, we do the following: When QueryLCW returns "No," we check whether every instance of Φ in the simulation in fact appears in $\mathcal{M}$. If so, LCW is possible, and we report that an LCW miss has occurred. Of course, this mechanism can over-report LCW misses. Although LCW(Φ) is *possible*, and QueryLCW(Φ) failed, it may be that no sensing of Φ has taken place and we could not expect any agent to deduce LCW(Φ).

For example, if directory dir1 is empty, then both $\mathcal{M}$ and the simulation database will agree on the extension of parent.dir(dir1, f), even if the agent has never executed a command such as ls dir1. But not knowing whether there are any files in a directory that happens to be empty is not the same as knowing that there aren't any, so this case would be a *false miss.* We are able to eliminate some of these false misses, but not all of them. However, since we are trying to demonstrate the success of our LCW machinery, we are content to be conservative and overstate the number of LCW misses.

In our experiments, fewer than 1% of the LCW queries generated by XII result in misses. The percentage of misses does not vary significantly with the amount of dynamism, or with the percentage of Domain Growth or Information Loss updates that occur.

Answering the question of how often misses occur independent of the XII planner and the Softbot domain is problematic, since we could construct cases in which all LCW queries are misses, or none are. For example, suppose we have a directory containing only postscript and $\TeX$ files, and we have LCW on the size of all files in that directory. Suppose we then compress one of the postscript files. By the Information Loss Rule, the LCW we had on the size of all the files will be removed from $\mathcal{L}$, whereas if our LCW machinery were *complete*, it would *retain* LCW on the size of all $\TeX$ files in the directory. Now if all queries are of the form "Do I know the size of all $\TeX$ files in this directory?" then every query will be a miss. Perverse cases like this one in practice are highly unlikely. This is due, in part, to the fact that failed LCW queries are likely to be followed by actions that achieve the desired LCW.

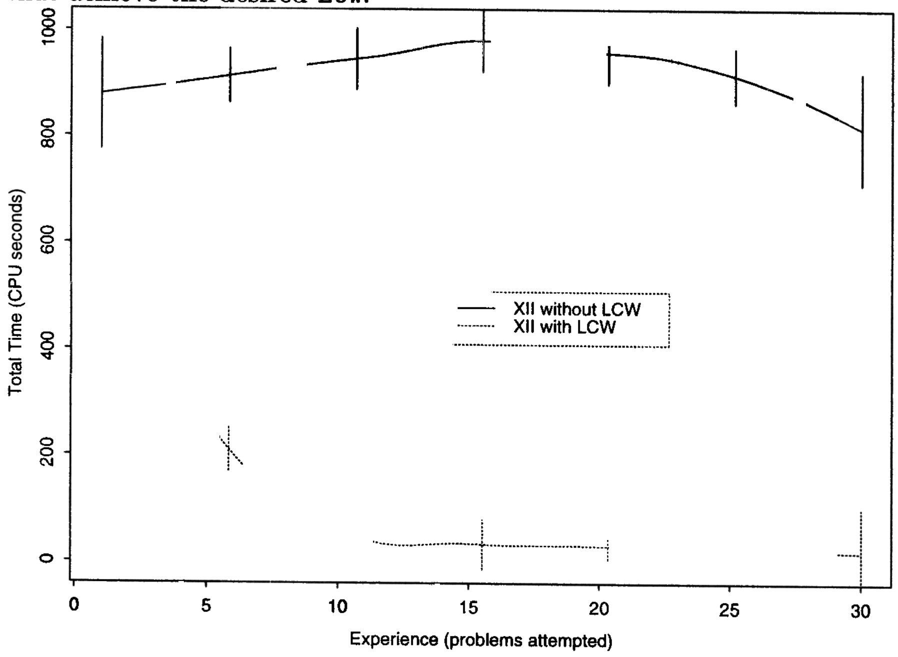

Figure 4: The use of LCW reasoning yields dramatic performance improvements to the XII planner. Times indicated are CPU seconds on a Sun SPARCstation 20; vertical bars indicate 95% confidence intervals. The experiment was repeated 10 times on randomly generated initial worlds. Thus, each of 30 distinct points on the X-axis represents the average of 10 planning sessions on randomly generated goals. The databases $\mathcal{M}$ and $\mathcal{L}$ were left intact between goals to measure the impact of increasing knowledge on planner performance. Thus, $\mathcal{M}$ and $\mathcal{L}$ tend to increase along the X-axis. The curves show a best-fit to each set of 300 data points.

4.3 Impact on Planning

We have shown that individual LCW queries are fast and that the reasoning mechanism is effectively complete, but given that a significant number of LCW queries are performed during planning, it is still conceivable that LCW might slow the planner down. We show that this is not the case; in fact LCW inference speeds planning considerably by reducing redundant sensing operations. Figure 4 shows the performance of the XII planner with and without LCW, solving a sequence of randomly generated goals, with $\mathcal{M}$ and $\mathcal{L}$ initially empty. The planner runs faster with LCW even on the first goal, since it leverages the LCW information which it gains in the course of planning. In subsequent goals,

LCW?	% probs solved	Total Number of executed actions	Time per plan (sec)
yes	94%	34,865	0.26
no	8%	93,050	1.16

Table 2: The number of executions performed by XII with and without LCW on 300 randomly generated problems. The number of executions for XII without LCW are drastically under-reported, because without LCW the planner could only solve 8% of the problems within the 1000 CPU second time bound. In contrast, with LCW reasoning the planner solved 94% of the problems. Had both versions of the planner been run until every problem was solved, we would expect a much larger difference in favor of XII with LCW. Surprisingly, LCW also reduces the amount of time XII spends per plan on average. This is because the non-LCW planner tends to consider more complicated plans, which require more CPU time to evaluate.

XII can take advantage of LCW gained in previous planning sessions for an even more pronounced speedup. Without LCW, the planner wastes an enormous amount of time doing redundant sensing. The version of XII without LCW completed only 8% of the goals before hitting a fixed time bound of 1000 CPU seconds. In contrast, the version with LCW completed 94% of the goals in the allotted time.

5 Future Work

Although we have relaxed the assumption of complete information, we still assume correct information. Since we want our agents to cope with exogenous events, we are in the process of relaxing this assumption as well. We are investigating two complementary mechanisms to solve this problem. The first mechanism associates *expiration times* with beliefs. If an agent has a belief regarding φ, which describes a highly dynamic situation (*e.g.*, the idle time of a user on a given machine), then the agent should not keep that belief in $\mathcal{M}$ for very long. Thus, after an appropriate amount of time has elapsed, the update $\Delta(\varphi, \mathtt{T} \vee \mathtt{F} \rightarrow \mathtt{U})$ occurs automatically. Note that by the Information Loss Rule, this update will cause LCW to be retracted as well. This mechanism is effective when the belief about φ expires before φ changes in the world. However, unless we have extremely short expiration times, we cannot guarantee this to be the case in general.

Thus, an additional mechanism is required that enables the agent to detect and recover from out-of-date beliefs. This is a harder problem, because it involves belief revision, rather than mere update. If executing an action fails, and the action's preconditions are known, it follows that one or more of the preconditions of the action were not satisfied — but which ones? A conservative mechanism would retract the ground literals satisfying the action's preconditions from the agent's theory. However, this mechanism could discard a great deal of valuable information. We are investigating less conservative mechanisms.

Finally, we need to investigate increasing the expressive power of $\mathcal{M}$ and $\mathcal{L}$. First, the introduction of negation into $\mathcal{L}$ would enable us to express sentences such as "I know the size of each file in /kr94 except paper.tex," which would make LCW update less conservative. Second, suppose that an agent was unfamiliar with the contents of the /kr94 directory, yet executed chmod g+r * while in that directory. The reasoning mechanism described in this paper is incapable of inferring that all the files in /kr94 are group-readable.[12] The LCW sentence

$$\texttt{LCW}(\texttt{parent.dir}(f, \texttt{/kr94}) \land \texttt{group.protection}(f, \texttt{readable}))$$

is not warranted because it implies that the agent is familiar with all the group-readable files in /kr94, which is false by assumption.

We *could* represent the information gained from the execution of `chmod g+r *` in /kr94 by introducing the following Horn clause into $\mathcal{M}$:

$$\texttt{parent.dir}(f, \texttt{/kr94}) \rightarrow \texttt{group.protection}(f, \texttt{readable})$$

The Horn clause represents the fact that all the files in the directory /kr94 are group-readable, even though the agent may be unfamiliar with the files in /kr94. Although the mechanisms described in this paper do not allow Horn clauses in $\mathcal{M}$, this example demonstrates that such an extension would provide increased expressiveness. Future work should determine whether this increase in expressive power is worthwhile.

6 Conclusion

This paper described a sound and efficient method for representing, inferring, and updating *local closed world information* (LCW) (*e.g.*, "I know the size of each file in /kr94") over a restricted first-order theory of the sort used by planning algorithms such as NONLIN, TWEAK, and UCPOP. To evaluate our LCW machinery empirically, we incorporated LCW into the XII planner [24, 25] and measured its impact on the performance of the Internet Softbot [17] under a wide range of experimental settings. As our experiments in the Softbot domain show, LCW queries require approximately 2 milliseconds, while LCW updates require only 1.2 milliseconds on average. Although our method is incomplete, our experiments show that inference fails to reach a conclusion in less than 1% of the queries posed. We hope that the dramatic performance improvements engendered by LCW reasoning will lead planning researchers to incorporate the technology into other planning systems as well.

[12] Indeed, this inference is only licensed when the agent is authorized to change the protection on each of the files in /kr94; suppose this is the case.

A Proofs

In many of the following proofs we rely on the following two facts:

- $\mathcal{L}$ contains only positive sentences.
- The variable substitution θ maps a sentence Φ to a ground sentence $\Phi\theta$. Thus, once the truth value of $\Phi\theta$ is known, we have $\texttt{LCW}(\Phi\theta)$.

Proof of Theorem 1 (Instantiation) Let Φ be a logical sentence and suppose $\texttt{LCW}(\Phi)$ holds. Let θ be an arbitrary substitution; we need show that $\texttt{LCW}(\Phi\theta)$ holds. *I.e.*, by definition of LCW (Equation 1) we need show that for all substitutions, σ, either $\Sigma \models \Phi\theta\sigma$ or $\Sigma \models \neg\Phi\theta\sigma$. But since the composition $\theta\sigma$ of substitutions is a substitution, and since $\texttt{LCW}(\Phi)$ we conclude $\texttt{LCW}(\Phi\theta)$. □

Proof of Theorem 2 (Composition) Let Φ and Ψ be logical formulae and suppose $\texttt{LCW}(\Phi)$ and $\forall\sigma, \Sigma \not\models \Phi\sigma \vee \texttt{LCW}(\Psi\sigma)$. Let θ be an arbitrary substitution. We need to show $[\Sigma \models (\Phi \wedge \Psi)\theta] \vee [\Sigma \models \neg(\Phi \wedge \Psi)\theta]$. If $\Sigma \models (\Phi \wedge \Psi)\theta$, then the proof is complete; so instead assume that $\Sigma \not\models (\Phi \wedge \Psi)\theta$. Since $\texttt{LCW}(\Phi)$, either $\Sigma \models \Phi\theta$ or $\Sigma \models \neg\Phi\theta$. If $\Sigma \models \neg\Phi\theta$, then clearly $\Sigma \models \neg\Phi\theta \vee \neg\Psi\theta$, and the proof is complete. If $\Sigma \models \Phi\theta$ then $\Sigma \not\models \Psi\theta$ (otherwise, $\Sigma \models (\Phi \wedge \Psi)\theta)$. Furthermore, $\Sigma \models \Phi\theta$ implies $\texttt{LCW}(\Psi\theta)$ (given), so $\Sigma \models \neg\Psi\theta$. Thus $\Sigma \models \neg\Phi\theta \vee \neg\Psi\theta$, which means that $\texttt{LCW}(\Phi \wedge \Psi)$. □

Proof of Theorem 3 (Conjunction) Let Φ and Ψ be logical sentences and suppose $\texttt{LCW}(\Phi)$ and $\texttt{LCW}(\Psi)$. By the Instantiation Rule, we have $\forall\sigma\ \texttt{LCW}(\Psi\sigma)$, so the condition $\forall\sigma, \Sigma \not\models \Phi\sigma \vee \texttt{LCW}(\Psi\sigma)$ is trivially true. Thus the Composition Rule applies, and we have $\texttt{LCW}(\Phi \wedge \Psi)$.
□

Proof of Theorem 4 (Disjunction) Follows trivially from the definition of LCW and Theorem 3. □

Proof of Theorem 5 (Negation) $\texttt{LCW}(\Phi) \equiv \forall\theta\ [\Sigma \models \Phi\theta] \vee [\Sigma \models \neg\Phi\theta] \equiv \forall\theta\ [\Sigma \models \neg(\neg\Phi\theta)] \vee [\Sigma \models (\neg\Phi\theta)] \equiv \texttt{LCW}(\neg\Phi)$. □

Proof of Theorem 6 (NP-hardness of LCW queries, unrestricted $\mathcal{L}$) We reduce formula satisfiability (SAT) to the problem of answering a singleton LCW query. Let π be an arbitrary propositional boolean formula. Let $\mathcal{L} = \{\texttt{LCW}(p \vee \pi)\}$, where p is a proposition not appearing in π, and let $\mathcal{M}$ be empty. We will show that the query LCW(p) *fails* iff there is a truth assignment to propositions in π such that π is true. Thus answering LCW(p) in the negative can be used to determine whether π is satisfiable.

1. Say there is no assignment such that π is true (i.e. π is provably false). Thus $p \vee \pi$ has the same truth value as p, so it follows from the definition of LCW that LCW($p \vee \pi$) implies LCW(p). Therefore, the query must succeed.

2. Conversely, if π is satisfiable, then either π is provably true (a tautology), or neither π nor $\neg\pi$ is provable. If π is a tautology, then so is $p \vee \pi$, which is completely independent of the truth value of p. Since we have no other information about p, LCW(p) does not follow from anything we know, and thus the query must fail. If neither π nor $\neg\pi$ is provable, then $p \vee \pi$ is irreducible. Since there are some truth assignments under which LCW(p) follows and others under which it doesn't, it's impossible to conclude LCW(p) in general, so again the query must fail.

The above two cases are exhaustive, so we have shown a (linear time) reduction of SAT to LCW inference. Since formula satisfiability is NP-hard, it follows that (unrestricted) LCW inference is also NP-hard. □

Proof of Theorem 7 Incompleteness of LCW Inference Rules We provide a simple counter-example. Consider the case in which we know LCW(parent.dir(bak, f) $\wedge$ is.backup(bak)), and we also know that is.backup(bak) is true. Since is.backup(bak) is ground and true, parent.dir(bak, f) $\wedge$ is.backup(bak) always has the same truth value as parent.dir(bak, f). It follows then from the definition of LCW that LCW(parent.dir(bak, f)). Since our inference rules won't derive this formula, they are incomplete. The more general problem is that whenever all possible instances of a formula A are both known and true, LCW(A$\wedge$B) implies LCW(B). For unbounded universes, and a finite knowledge base of positive ground facts, the formula must be ground for all instances to be known true. □

Proof of Theorem 8 Soundness of QueryLCW We use induction on the number of conjuncts in Φ.

Case $|\Phi| = 0$: An invocation of QueryLCW induces a call to QLCW* where line 1 returns T. This is correct, because the null clause (*i.e.*, a ground query with zero conjuncts) is unsatisfiable by definition. Since every state in Σ agrees that the null clause is false, $\Sigma \models \neg\Phi$ and hence LCW(Φ).

Case $|\Phi| = k \geq 1$: If QLCW* returns T, it must have terminated on line 2 or 6.

But line 2 only returns true when all ground instantiations, namely Φ itself, are entailed by $\mathcal{M}$. This corresponds directly to the definition of LCW.

Line 6 will only return T under conditions matched by Composition which is sound by Theorem 2, or Instantiation (line 4, $\Phi - \Phi' = \{\}$), which is sound by Theorem 1. Since these are the only termination points for QueryLCW, the algorithm is sound. □

Proof of Theorem 9 Complexity of `QueryLCW` Suppose Φ has c conjuncts, let L denote $|\mathcal{L}|$, and let $M = |\mathcal{M}|$. In the worst case, control falls through to line 3 entering a loop over the elements of $\mathcal{L}$. In line 6 the loop body performs a conjunctive match on $\mathcal{M}$ with a pattern whose length is at most c (giving a complexity $O(M^c)$), and then possibly makes a recursive call. Thus the following recurrence relation defines the time required by `QueryLCW`:

$$t[c] = L(M^c + t[c-1])$$

Unrolling the recursion yields

$$t[c] = LM^c + L^2M^{c-1} + L^3M^{c-2} + \ldots + L^cM$$

But checking the binomial expansion shows that this is bounded above by $(L+M)^{c+1}$ so `QueryLCW` requires at most $O((|\mathcal{L}| + |\mathcal{M}|)^{c+1})$ time. Given that c is bounded by the domain theory, the complexity is polynomial in the size of $\mathcal{L}$ and $\mathcal{M}$. □

Proof of Theorem 10 (Information Gain Rule) First we prove that for any formula, Φ, and literal, φ, if `LCW`(Φ) holds before action A is executed and the sole effect of A is $\Delta(\varphi, \mathtt{U} \rightarrow \mathtt{T} \vee \mathtt{F})$, then `LCW`$(\Phi)$ still holds. Suppose `LCW`(Φ) holds and let θ be an arbitrary substitution. By Equation 1, we know that $[\Sigma \models \Phi\theta] \vee [\Sigma \models \neg\Phi\theta]$. Since, by the definition of $\Delta(\varphi, \mathtt{U} \rightarrow \mathtt{T} \vee \mathtt{F})$, $\Sigma'\ominus\mathcal{U}(\varphi, \Sigma)$ $= \Sigma\ominus\mathcal{U}(\varphi, \Sigma)$, and by definition of $\mathcal{U}$, $\Sigma\ominus\mathcal{U}(\varphi, \Sigma) = \Sigma$, $\Sigma \subseteq \Sigma'$. As a result, for any formula Ψ if $\Sigma \models \Psi$ then $\Sigma' \models \Psi$. Thus, clearly $[\Sigma' \models \Phi\theta] \vee [\Sigma' \models \neg\Phi\theta]$. Next we prove that if sole effect of A is $\Delta(\varphi, \mathtt{U} \rightarrow \mathtt{T} \vee \mathtt{F})$, then `LCW`$(\varphi)$ holds. By the definition of $\Delta(\varphi, \mathtt{U} \rightarrow \mathtt{T} \vee \mathtt{F})$, $\mathcal{U}(\varphi, \Sigma') = \{\}$. By the definition of $\mathcal{U}$, $\nexists\theta(\Sigma \not\models \varphi\theta \wedge \Sigma \not\models \neg\varphi\theta)$, or equivalently, $\forall\theta(\Sigma \models \varphi\theta \vee \Sigma \models \neg\varphi\theta)$. But that is exactly the definition of `LCW`(φ). □

Proof of Theorem 11 (Counting Rule) Let φ be a literal and suppose that `Cardinality`$(\varphi, \mathcal{M})$ = `Cardinality`$(\varphi, \mathcal{W})$. We need show that `LCW`(φ); in other words, we need show that for an arbitrary substitution θ, $[\Sigma \models \varphi\theta] \vee [\Sigma \models \neg\varphi\theta]$. Let M denote the $\{\phi \in \mathcal{M}$ such that ϕ is ground and $\exists\sigma\ \phi = \varphi\sigma\}$. If $\varphi\theta \in \mathcal{M}$ then $\Sigma \models \varphi\theta$ and the proof is complete, so assume that $\varphi\theta \notin \mathcal{M}$. In other words, $\varphi\theta$ is not in M. Let W denote the set $\{\phi \in \mathcal{W}$ such that ϕ is ground and $\exists\sigma\ \phi = \varphi\sigma\}$. Since we assume correct information, $M \subseteq M$, and so by our assumption of cardinality $M = W$. So $\varphi\theta \notin \mathcal{W}$. So $\Sigma \models \neg\varphi\theta$. We conclude `LCW`$(\varphi)$. □

Proof of Theorem 12 (Information Loss Rule) Let Φ be a conjunction of positive literals and suppose that $\texttt{LCW}(\Phi)$. Let φ be a positive literal and let A be an action whose execution leads solely to an update of the form $\Delta(\varphi, \texttt{T} \vee \texttt{F} \rightarrow \texttt{U})$. To prove that the Information Loss Rule is sound in this case, we need to show that if $\texttt{LCW}(\Phi)$ no longer holds after executing A then $\Phi \in \texttt{REL}(\varphi)$ (the set of beliefs removed from $\mathcal{L}$), hence the update correctly recognizes that $\texttt{LCW}$ has been lost, and $\mathcal{L}$ remains conservative. Suppose that $\texttt{LCW}(\Phi)$ *doesn't* hold after executing A; then there exists a substitution, θ such that $[\Sigma' \not\models \Phi\theta] \wedge [\Sigma' \not\models \neg\Phi\theta]$ even though $[\Sigma \models \Phi\theta] \vee [\Sigma \models \neg\Phi\theta]$. Note that since Φ is conjunctive, $\Phi = \phi_1 \wedge \ldots \wedge \phi_n$. There are two cases:

1. $(\Sigma \models \Phi\theta)$. So forall $\phi_i \in \Phi$ we know that $\Sigma \models \phi_i\theta$. But since $\Sigma' \not\models \Phi\theta$ there exists ϕ_j such that $\Sigma' \not\models \phi_j\theta$. Hence execution of A caused $\Delta(\phi_j\theta, \texttt{T} \rightarrow \texttt{U})$. But by the definition of Δ, the only updates produced by A were of the specific form, $\varphi\alpha = \phi_j\theta$. We conclude that $\Phi \in \texttt{PREL}(\varphi)$.

2. $(\Sigma \models \neg\Phi\theta)$. In this case we know that $\exists\phi_j \in \Phi$ such that $\Sigma \models \neg\phi_j\theta$ yet $\Sigma' \not\models \neg\phi_j\theta$. As above, the restriction on Δ allows us to conclude that $\varphi\alpha = \phi_j\theta =$ and $\Phi \in \texttt{PREL}(\varphi)$.

To show $\Phi \in \texttt{REL}(\varphi)$, we now need argue that $\mathcal{M} \cup \mathcal{L} \not\models \neg(\Phi - \phi_j)\theta$. Suppose that this is *not* the case. Since $\mathcal{M}$ and $\mathcal{L}$ are conservative, $\Sigma \models \neg(\Phi - \phi_j)\theta$ as well. Furthermore, since the only change affected by action A had Δ restricted to $\phi_j\theta$, we know that $\Sigma' \models \neg(\Phi - \phi_j)\theta$. But since the falsity of a single conjunct entails the falsity of the whole conjunction (and $\Phi\theta = \phi_j\theta \wedge (\Phi - \phi_j)\theta)$, we conclude that $\Sigma' \models \neg\Phi\theta$. But this contradicts our assumption that A destroyed $\texttt{LCW}(\Phi)$. So it must be the case that $\mathcal{M} \cup \mathcal{L} \not\models \neg(\Phi - \phi_j)\theta$. Thus $\Phi \in \texttt{REL}(\varphi)$. $\square$

Proof of Theorem 13 (Domain Growth Rule) Let Φ be a conjunction of positive literals and suppose that $\texttt{LCW}(\Phi)$. Let φ be a positive literal and suppose A is an atomic action whose only effect is $\Delta(\varphi, \texttt{F} \rightarrow \texttt{T})$. Suppose that $\texttt{LCW}(\Phi)$ no longer holds after executing A; then there exists a substitution, θ such that $[\Sigma' \not\models \Phi\theta] \wedge [\Sigma' \not\models \neg\Phi\theta]$ even though $[\Sigma \models \Phi\theta] \vee [\Sigma \models \neg\Phi\theta]$. A case analysis on the these disjuncts (as in the proof of Theorem 12) yields that $\exists\phi_j \in \Phi$ such that $\phi_j\theta = \varphi\alpha$ and that $\Phi \in \texttt{PREL}(\varphi)$. The contradiction argument from that proof also extends to show that $\Phi \in \texttt{REL}(\varphi)$. Now note that after execution of A, we have $\texttt{LCW}(\phi_j\theta)$ (since we know that φ changed to $\texttt{T}$), but by assumption not $\texttt{LCW}(\Phi\theta)$. Therefore, by the contrapositive of Theorem 3 (Conjunction), $\neg\texttt{LCW}((\Phi - \phi_j)\theta)$. This leads to $\Phi \in \texttt{MREL}(\varphi)$. $\square$

Proof of Theorem 14 (Domain Contraction Rule) Let φ be a positive literal and suppose A is an action whose only effect is $\Delta(\varphi, \mathtt{T} \rightarrow \mathtt{F})$. To show that the update rule is sound, it is sufficient to prove that for any conjunction of positive literals, $\Phi = \phi_1 \wedge \ldots \wedge \phi_n$, if $\mathtt{LCW}(\Phi)$ holds before executing A then $\mathtt{LCW}(\Phi)$ holds after executing A. If $\mathtt{LCW}(\Phi)$ holds before execution then, for arbitrary θ, we know that $[\Sigma \models \Phi\theta] \vee [\Sigma \models \neg\Phi\theta]$. We need to show that after executing A $[\Sigma' \models \Phi\theta] \vee [\Sigma' \models \neg\Phi\theta]$. Suppose, on the other hand, that $[\Sigma' \not\models \Phi\theta] \wedge [\Sigma' \not\models \neg\Phi\theta]$. But since the Δ effected by A only made *more* atomic formulae false, $\Sigma' \not\models \neg\Phi\theta$ implies $\Sigma \not\models \neg\Phi\theta$. Since $\mathtt{LCW}(\Phi)$ holds before executing A, it follows that $\Sigma \models \Phi\theta$ which means that $\Sigma \models \phi_i\theta$ for all $\phi_i \in \Phi$. Now consider the literal φ that has become false.

1. If $\varphi \notin \Phi$ then $\Sigma' \models \Phi\theta$ (since the truth will be unchanged)
2. If $\varphi \in \Phi$ then $\Sigma' \models \neg\Phi\theta$.

Either way there is a contradiction. □

Proof of Theorem 15 (Tractability of Updates) The proof was sketched to such an extent in Section 3.6 that we will not repeat all details here. Note however, that the exponent c (maximum number of conjuncts in the longest element of $\mathcal{L}$) is the correct one for the following reason. Theorem 9 shows that a call to `QueryLCW` with an argument of b conjuncts requires $O((|\mathcal{L}| + |\mathcal{M}|)^{b+1})$ time. When computing $\mathtt{REL}(\varphi)$ or $\mathtt{MREL}(\varphi)$ however, the longest argument to `QueryLCW` has $c - 1$ conjuncts since the conjunct "x" is removed before the call to `QueryLCW`. (Refer to the definition of `REL` and `MREL`). □

Proof of Theorem 16 (Minimal Information Loss) Let φ be a positive literal and let A be an atomic change whose only effect is $\Delta(\varphi, \mathtt{T} \vee \mathtt{F} \rightarrow \mathtt{U})$. Suppose $\Phi \in \mathtt{REL}(\varphi)$. We need to show that $\mathtt{LCW}(\Phi)$ does not hold after A has occurred. Thus it suffices to show that there exists a θ such that $\Sigma' \not\models \Phi\theta$ and $\Sigma' \not\models \neg\Phi\theta$. Since Φ is conjunctive, the definition of $\mathtt{PREL}(\varphi)$ dictates that there exists $\phi \in \Phi$ such that $\phi\theta = \varphi\alpha$. Since the only change from w to w' is that all instances of φ changed their value to unknown, and since from the definition of $\mathtt{REL}(\varphi)$, we also have $\mathcal{L} \wedge \mathcal{M} \not\models \neg(\Phi - \phi)\theta$, i.e. all other conjuncts may be true in w, it follows that $\Phi\theta$ may be true in w'. Let $\mathcal{M}'$ denote the state of $\mathcal{M}$ after the update due to A, and let Σ' denote the possible states of the world after the update due to A. Since $\Phi\theta$ may be true in w', we have that $\Sigma' \not\models \neg\Phi\theta$. Furthermore, since $\mathcal{M}' \not\models \varphi\alpha$, $\mathcal{M}' \not\models \Phi\theta$, and thus $\Sigma' \not\models \Phi\theta$. Therefore, $\mathtt{LCW}(\Phi)$ does not hold. □

Proof of Theorem 17 (Minimal Domain Growth) Let φ be a positive literal and let A be an atomic change whose only effect is $\Delta(\varphi, \mathtt{F} \rightarrow \mathtt{T})$. We need show that if $\Phi \in \mathtt{MREL}(\varphi)$ then $\mathtt{LCW}(\Phi)$ does not hold after A has occurred. Since Φ is conjunctive, the definition of $\mathtt{PREL}(\varphi)$ dictates that there exists $\phi \in \Phi$ such that $\phi\theta = \varphi\alpha$. Since $\Phi \in \mathtt{REL}(\varphi)$, we know that $\Phi\theta$ *may* be true in w'. So, $\Sigma' \not\models \neg\Phi\theta$. Since $\Phi \in \mathtt{MREL}(\varphi)$, we conclude that $\neg\mathtt{LCW}((\Phi - \phi)\theta)$, meaning that for some $\psi \in \Phi$, $\psi\theta \notin \mathcal{M}'$. Hence $\mathcal{M}' \not\models \Phi\theta$, and since Φ contains only positive literals we can conclude that $\Sigma' \not\models \Phi\theta$. Therefore, $\mathtt{LCW}(\Phi)$ does not hold. □

B The Experimental Framework

The goal of our experiments was to measure the performance of our LCW machinery in a real-world setting. All of our evaluations of LCW are through queries and updates generated by the XII planner in the course of satisfying randomly-generated file manipulation goals in the Softbot domain. To make our experiments easier to control, vary, and replicate, we built a simulation environment that allows us to generate arbitrary UNIX world states, which behave exactly as UNIX behaves in response to actions executed by the softbot. Additionally, the simulation greatly simplifies the task of evaluating LCW, as we discuss in Section 4.2. Nearly all the of results we report using simulated UNIX worlds are identical to the results we would obtain if XII were executing in an equivalent, real UNIX environment. The one exception is the report of total time in Figure 4, which does not reflect the time required to execute actions in a UNIX shell. However, the purpose of Figure 4 is to evaluate the impact of LCW on planning, not to measure the performance of the Internet Softbot. Based on earlier experiments in this domain (see [24]), it seems likely that accurately reporting execution time would only make our results stronger, since, without LCW, XII spends a greater percentage of its time executing actions (see Table 2), and execution is expensive.

The Simulation Environment

The simulation environment consists of a current world state w_s, represented as a database, which completely specifies the state of all files and directories in the simulation, and an execution procedure that translates an action to be executed into the appropriate queries and updates on w_s. In our experiments, w_s contains up to 80 directories, each directory holding between 5 and 20 files. The topology of the directory tree is random, each directory containing at most five other directories. Filenames are all of the form `dir1`, `file2`, etc. The values of other file attributes, such as `size` and `file.type`, are chosen randomly. Although w_s doesn't model *every* aspect of UNIX, it is complete relative to every action that could be executed in service of the test suite.

The execution procedure simply computes a mapping from an action to database operations on w_s. This mapping is straightforward; all the required information is contained in the effects of the action. For example, `ls -la dir3` determines, among other things, the size of each file in `dir3`, so the execution procedure handles the execution of `ls -la dir3` by querying w_s for

$$\texttt{parent.dir}(f, \texttt{dir3}) \wedge \texttt{size}(f, n)$$

and updating $\mathcal{M}$ with the results. Similarly, since `cd dir11` has the effect `current.dir(dir11)`, this update is done to w_s as well as to $\mathcal{M}$.

The Goal Distribution

The test suite consists of a series of *runs*. At the beginning of each run, a simulated world w_s is randomly generated, and $\mathcal{M}$ and $\mathcal{L}$ are empty. A sequence of 30 goals is then randomly generated, and XII is given the goals to solve one by one. $\mathcal{M}$ and $\mathcal{L}$ are left intact between goals, so for each goal, XII has the benefit of knowledge obtained in solving the previous goals. After the 30 goals are completed, a new world is generated, $\mathcal{M}$ and $\mathcal{L}$ are emptied, and the process is repeated.

Our goal generator creates either universally-quantified or existentially-quantified goals. Quantification aside, the two sets of goals are essentially equivalent, and consist of finding files meeting certain properties, such as `filename`, `parent.dir`, `word.count` and `file.type`, and performing certain operations on them, such as compressing them, moving them to a different directory or finding out their size. A typical goal is "Compress all postscript files in the directory `/dir0/dir1/dir21`."

References

[1] J Ambros-Ingerson and S. Steel. Integrating planning, execution, and monitoring. In *Proc. 7th Nat. Conf. on A.I.*, pages 735–740, 1988.

[2] R. Brachman. "Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, October 1992.

[3] D. Brill. *LOOM Reference Manual.* USC-ISI, 4353 Park Terrace Drive, Westlake Village, CA 91361, version 1.4 edition, August 1991.

[4] M. Cadoli and M. Schaerf. A survey of complexity results for non-monotonic logics. *Journal of Logic Programming*, 17:127–160, November 1993.

[5] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[6] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Publishing Corporation, New York, NY, 1978.

[7] Alvaro del Val. Computing Knowledge Base Updates. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 740–750, 1992.

[8] Alvaro del Val and Yoav Shoham. Deriving Properties of Belief Update from Theories of Action (II). In *Proceedings of IJCAI-93*, pages 732–737, 1993.

[9] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57:227–270, October 1992.

[10] Charles Elkan. A decision procedure for conjunctive query disjointness. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 134–139, 1989.

[11] Charles Elkan. Independence of logic database queries and updates. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 154–160, 1990.

[12] D. Etherington. *Reasoning with Incomplete Information.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[13] D. Etherington, S. Kraus, and D. Perlis. Nonmonotonicity and the scope of reasoning: Preliminary report. In *Proc. 8th Nat. Conf. on A.I.*, pages 600–607, July 1990.

[14] O. Etzioni. Intelligence without robots (a reply to brooks). *AI Magazine*, 14(4), December 1993. Available via anonymous FTP from `pub/etzioni/softbots/` at `cs.washington.edu`.

[15] O. Etzioni, K. Golden, and D. Weld. Tractable closed-world reasoning with updates. In *Proc. 4th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 178–189, San Francisco, CA, June 1994. Morgan Kaufmann.

[16] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, San Francisco, CA, October 1992. Morgan Kaufmann. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[17] O. Etzioni and D. Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, July 1994. See `http://www.cs.washington.edu/research/softbots`.

[18] Oren Etzioni and Neal Lesh. Planning with incomplete information in the UNIX domain. In *Working Notes of the AAAI Spring Symposium: Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 24–28, Menlo Park, CA, 1993. AAAI Press.

[19] Oren Etzioni, Neal Lesh, and Richard Segal. Building softbots for UNIX (preliminary report). Technical Report 93-09-01, University of Washington, 1993. Available via anonymous FTP from `pub/etzioni/softbots/` at `cs.washington.edu`.

[20] M. Genesereth and I. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proc. 11th Nat. Conf. on A.I.*, pages 724–730, July 1993.

[21] M. Ginsberg, editor. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, San Mateo, CA, 1987.

[22] M. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39(2):209–230, June 1989.

[23] M. Ginsberg and D. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35(2):165–196, June 1988.

[24] K. Golden, O. Etzioni, and D. Weld. Omnipotence without omniscience: Sensor management in planning. In *Proc. 12th Nat. Conf. on A.I.*, pages 1048–1054, Menlo Park, CA, July 1994. AAAI Press.

[25] K. Golden, O. Etzioni, and D. Weld. Planning with execution and incomplete information. Technical Report 96-01-09, University of Washington, Department of Computer Science and Engineering, February 1996. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[26] G. Grahne. The problem of incomplete information in relational databases. In *Lecture Notes in Computer Science*, volume 554. Springer Verlag, New York, 1991.

[27] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 387–394, 1991.

[28] A. Keller and M. Wilkins. On the use of an extended relational model to handle changing incomplete information. *IEEE Transactions on Software Engineering*, SE-11(7):620–633, July 1985.

[29] K. Konolidge. Circumscriptive ignorance. In *Proc. 2nd Nat. Conf. on A.I.*, pages 202–204, 1982.

[30] R. Kowalski. Logic for data description. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 77–103. Plenum Publishing Corporation, New York, NY, 1978.

[31] K. Krebsbach, D. Olawsky, and M. Gini. An empirical study of sensing and defaulting in planning. In *Proc. 1st Intl. Conf. on A.I. Planning Systems*, pages 136–144, June 1992.

[32] H.J. Levesque. All I know: A study in autoepistemic logic. *Artificial Intelligence*, 42(2–3), 1990.

[33] A. Levy. Queries, updates, and LCW. Personal Communication, 1994.

[34] A. Levy and Y. Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference*, 1993.

[35] V. Lifschitz. Closed-World Databases and Circumscription. *Artificial Intelligence*, 27:229–235, 1985.

[36] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1,2):27–39, April 1980.

[37] R. Moore. A Formal Theory of Knowledge and Action. In J. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World.* Ablex, Norwood, NJ, 1985.

[38] D. Olawsky and M. Gini. Deferred planning and sensor use. In *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.* Morgan Kaufmann, 1990.

[39] C. Papadimitriou. Games against nature. *Journal of Computer and Systems Sciences*, 31:288–301, 1985.

[40] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[41] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–114, October 1992. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[42] M. Peot and D. Smith. Conditional Nonlinear Planning. In *Proc. 1st Intl. Conf. on A.I. Planning Systems*, pages 189–197, June 1992.

[43] O. Raiman and J. de Kleer. A Minimality Maintenance System. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 532–538, October 1992.

[44] R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978. Reprinted in [21].

[45] R. Reiter. Circumscription implies predicate completion (sometimes). *Proc. 2nd Nat. Conf. on A.I.*, pages 418–420, 1982.

[46] D. Smith. Finding all of the solutions to a problem. In *Proc. 3rd Nat. Conf. on A.I.*, pages 373–377, 1983.

[47] A. Tate. Generating project networks. In *Proc. 5th Int. Joint Conf. on A.I.*, pages 888–893, 1977.

[48] D. Weld. An introduction to least-commitment planning. *AI Magazine*, pages 27–61, Winter 1994. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[49] M. Winslett. Reasoning about action using a possible models approach. In *Proc. 7th Nat. Conf. on A.I.*, page 89, August 1988.

[50] M. Winslett. *Updating Logical Databases.* Cambridge University Press, 1990. Cambridge, England.