ABSTRACT

Title of Dissertation: Time-Situated Metacognitive Agency and Other Aspects of Commonsense Reasoning

Matthew David Goldberg
Doctor of Philosophy, 2022

Dissertation Directed by: Professor Donald Perlis
Department of Computer Science

Much research in commonsense reasoning (CSR) involves use of external representations of an agent's reasoning, based on compelling features of classical logic. However, these advantages come with severe costs, including: omniscience, consistency, static semantics, frozen deadlines, lack of self-knowledge, and lack of expressive power to represent the reasoning of others. Active logic was developed to address many of these, but work to date still leaves serious gaps. The present work focuses on major extensions of active logic to deal with self-knowledge, and their implementation into a newly-developed automated reasoner for commonsense active logic. Dealing with self-knowledge has been designed and implemented in the reasoner via a new treatment of quotation as a form of nesting. More sophisticated varieties of nesting, particularly quasi-quotation mechanisms, have also been developed to extend the basic form of quotation. Active logic and the reasoner are applied to classical issues in CSR, including a treatment of one agent having the knowledge and inferential mechanisms to reason about another's time-situated reasoning.

Time-Situated Metacognitive Agency and Other Aspects of Commonsense Reasoning

by

Matthew David Goldberg

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Acknowledgments

I'm tremendously thankful for all of the support, encouragement, guidance, and love that I've received throughout my years as a grad student.

I'm grateful for the insight, wisdom, and patience of my advisor, Dr. Don Perlis, as well as Dr. Darsana Josyula. Throughout countless meetings and discussions over these years, they taught me so much about the process of research, writing, and expressing ideas, and could always help guide a way forward when I was stuck.

I'm grateful for the support of my friends, especially Joe, Alan, Kevin, Alex, and Grayson — for numerous reasons, including bouncing around research ideas, as well as for keeping me sane outside of work.

I'm grateful for the love and encouragement of my family: Mom and Dad, and Dan — as well as Carl, Dionne, Sarah, Mecca, Mikey, Luke, and Hunter.

And above all, I'm forever grateful for the unwavering love, support, and faith of my fiancée, Imani. At all times, especially through long days of dissertation writing, you have been a source of inspiration, reassurance, and motivation — and always held fast in the belief of my ability to succeed. I couldn't have done this without you!

Table of Contents

Chapter 1: Introduction

Much of AI has focused on the abilities of a system to execute particular tasks or objectives with a narrow focus. Less work has gone into developing systems which might be described as having a proper sense of agency. In referring to *sense of agency* specifically, in part we refer to an existing but limited sense used both in AI and in human psychology, namely the process of tracking and controlling actions being undertaken by the actor [8]. We intend to go beyond this meaning of the term, in considering a larger sense of agency that facilitates development of a system that potentially could support a variety of abilities such as: general kinds of reasoning and processing, persistence beyond a particular task, and most crucially, reasoning about itself (especially its own specific beliefs, expectations, and reasoning history) and about representing the beliefs and reasoning of other agents.

Of that list of desiderata, a key item that we most emphasize is the need for an agent to support wide-ranging metacognitive behavior and autoepistemic meta-reasoning. We contend that, as part of a further-developed sense of self-agency as we have defined it, perhaps the most critical process for a system to be aware of is maintaining awareness of its reasoning processes and internal operations. These processes are depend on the capacity for metareasoning. Metareasoning is of tremendous importance for AI, due to playing a key role in many abilities that seem to be crucial for characterizing an agent as intelligent in a wide sense. A broadly capable system ought to be able to have abilities such as being able to recognize when it has made errors, to explain its actions and reasoning, to have rich information on what it is doing (in a variety of senses including inferring and acting) — among many other abilities that all require such an agent's reasoning *about* its reasoning. Thus, a primary focus of this dissertation is on issues related to metareasoning.

The form of reasoning we focus our attention on for an artificial agent, and thus also the form of metareasoning investigated, is symbolic logic-based reasoning. Logic has formed a useful basis for numerous AI systems. Yet, an issue with symbolic reasoning is that, by and large, logical approaches have a notable weakness: if they model processing which might be attributed to an agent, this is typically viewed from the perspective of looking in on a hypothetical agent's reasoning, as an outside observer – an external view by the logic. An external viewpoint for a logic that is supposed to be the core of a reasoning *agent* is not properly agentive. Reiter [79] contrasts two viewpoints as the "external design stance" versus the "self-reflective design stance" and this terminology alone is illuminating. Perlis et al. [67] use the terms external logic and internal logic, which we adopt here as well. The former stance characterizes knowledge and beliefs as seen by an external observer/designer that is separate from the agent being observed.

To better illustrate this idea, we consider an agent with a knowledge base, in which referring to the agent's knowledge base containing the belief `P` is done via the formula `InKB(P)`. There are three languages used in the preceding sentence: 1) the formal (internal agent) language which contains `P` as a well-formed formula (wff), 2) the designer language which contains `InKB(P)` as a wff that provides a designer's representation of the internal agent language formula `P`, and finally 3) English as a natural language that is used to discuss the languages and beliefs. The formula `InKB(P)` in the designer language is distinct from whether the agent can itself reflect on `P` in its knowledge base (KB); this latter reflection is instead represented in the agent's language using `Bel(P)` (i.e., the agent believes that it holds the belief `P`).

These languages and examples help to clarify the two design stances and the external-internal logic distinction. `InKB(P)` exemplifies the external design stance, and represents the fact that the agent being studied holds the belief `P`, and this belief might belong to an observer or designer of the agent. On the other hand, `P` exemplifies the internal and self-reflective design stance, indicating that `P` itself stands on its own as a belief of the agent. As described above, `Bel(P)` is also a belief of the agent language, expressing agent reflecting on the fact that it believes `P`, and so this further demonstrates the self-reflective case. The table below summarizes how the paired examples compare between the three languages:

Agent language	Designer language	English
`P`	`InKB(P)`	agent has `P` in its KB
`Bel(P)`	`InKB(Bel(P))`	agent believes that it has `P` in its KB

The bulk of the work in commonsense reasoning has involved external logic using designer languages. However, in both the current form of active logic and in this dissertation, we dispense with using a designer language and focus on an internal language of an agent, which we also discuss in natural language with English. In this way, once we concern ourselves with the internal language, English corresponds to the role that had formerly been played by formal external designer language, by serving as an outside way of observing the internal language.

As another motivating factor for internal languages, Perlis et al. [67] point to accounting for the process of inference itself taking time, and evolving over time, as a key characteristic that distinguishes internal logics; a sharp contrast is made between internal logics and external logics, the latter of which have a tendency to specify agent beliefs via consequential closure. A realistic agent situated in a world, whether real or simulated, inherently consumes time during the process of its reasoning. If the system abstracts away this temporal process, to focus on the idealized (possibly-infinite) closure of its belief set, it ignores a critical piece of its own reasoning. For example, we wish to consider agents that have the potential to reason in ways such as: to interrupt an ongoing thread of their reasoning, to notice that enough or too little time remains to achieve a planning goal, and to contrast a belief of the present moment with a belief that was revised or discarded some time ago. Further, it is desirable to be able to characterize formulas in a particular agent's knowledge base as being that agent's *own* beliefs and knowledge, which an external logic by its nature would not be able to do.

Additionally, external approaches to logical AI tend to have a number of common issues, which further affect the ability of a logic or its reasoner to support certain kinds of commonsense reasoning:

Omniscience — a common problem affecting logics is that of omniscience, where the beliefs held are based on the closure of inference rules applied to the set of axioms. Yet a practical system performing commonsense reasoning must take time to derive conclusions and apply inference, and otherwise assuming that all (infinitely many) logical consequences of the set of beliefs are known is highly unrealistic.

Consistency and paradox — a logical reasoner ought to be able to continue to reason in the presence of contradictions, as these are common occurrences in commonsense reasoning when interacting with the world. However, when the knowledge base becomes inconsistent, many logics will entail all formulas (i.e., *ex contradictione quodlibet*). Paraconsistent logics do not suffer from this, but often ignore inconsistent parts of knowledge. A reasoner ought to have the ability to manipulate inconsistent formulas and reason toward some form of solution.

Static semantics — a commonsense logical agent should have the ability to reason about the meaning of an expression, but also be able to mention expressions, to be able to reason about the syntactic form. This is an often-overlooked ability. As an example, an agent may mention a formula that contributes to inconsistency, as part of reasoning about this. Or, it may express in the syntax of its language that "John" is the name of the person John and refers to him. Symbols used as references in this way can also be free to change their use over time, avoiding the issue of references that are static, unchanging through inference. In this sense we do not mean formal semantics, but an intuitive semantics based on the meaning of words or names.

Frozen deadlines — reasoning takes time, and during inference toward solving an urgent goal the deadline continues to approach. The fact that reasoning itself happens over time and affects that resource should be accounted for in the logic, yet that accounting is rarely done. When this is not dealt with, the distance to a deadline seems frozen, during reasoning.

In light of these issues, external logics thus impose a significant barrier to internal reasoning and a variety of abilities identified as agency-related. Internal logics, conversely, offer an opportunity to act as a framework to meaningfully pull together both the wider sense of agency and the more narrow ability of metacognitive self-reference. Active logic is one form of internal logic, particularly because it centers around inference evolving in time, and was developed to address many of the limitations of external approaches that we have identified. Active logic has been developed in a rich body of work (discussed in detail in section 2.2), for which metareasoning has been an area of interest since active logic work began. The history of this work has investigated metacognitive aspects such as artificial efference copy in speech, planning within real-time deadlines, and object reference revision. But active logic as it has been developed until now is not ideal for supporting a sense of agency, primarily due to its inability to properly reason about active logic formulas themselves as objects of its beliefs. In the present work, a primary area of focus is extending the logic, and its formalism and inference rules, to incorporate a syntactic theory of quotation, for nesting formulas within other active logic formulas. This enhances some key qualities desired for a logical agent, to enable it to have a better basis for agency through the improved ability for self-reference, which we have indicated to be a component added into a richer conception of a sense of agency.

We provide the following sample formula which provides an example of a formula as an object of belief for internal reasoning, rather than using the design language approach. Additionally, this formula touches upon many of the aspects which are developed in the present research:

$$\mathtt{now(t)} \land \mathtt{t} > \mathtt{t_0} \land \mathtt{bel(P,t)} \land \mathtt{bel(\neg P, t_0)}$$

This formula is one which might appear in an agent's internal knowledge base, and it expresses an agent reflecting on the current point in time as `t`, that the agent previously had the belief $\neg$`P` at an earlier time, and that the agent had changed its mind so that `P` is a current belief at `t`. Hence, the example incorporates aspects of inference in time and evolving over time, reflection on internal beliefs and reasoning about nested formulas as objects, and the agent's change in belief. We thus illustrate skipping using a designer language, by elucidating the meaning of this internal belief without relying on an external reference to the agent, such as in an `InKB` formula would use.

1.1 Contributions

As an overarching goal, the present research pulls together the wide swath of active logic research, develops the logic toward making aspects of it more agent-centric, implements these developments to active logic in an automated reasoner, and applies this reasoner to a series applications of commonsense reasoning. More specifically, the contributions of the present work are thus:

1. *The development of a more agent-centric active logic*, through the expansion of the logic to express and draw inferences about its own reasoning and ac-

tions. This is achieved through active logic being extended to support nested formulas via a novel type of term, the quotation term. Quotation is developed first with a simple form, which is then extended to more sophisticated varieties of nesting via a quasi-quotation mechanism that provides the means to quantify into quotation terms. We ultimately present a full unification algorithm for quotation and quasi-quotation inference, which is exhaustive in accounting for how to unify terms in the extended active logic system. Quotation and quasi-quotation are described in chapter 3.

2. *The implementation of active logic reasoning*, both existing basic aspects of the logic as well as new developments for quotation, in a reasoning engine, ALMA 2.0, newly developed as part of the present work. In specifying the language of wffs for ALMA, we also provide a grammar for agency-based reasoning with active logic. ALMA itself reflects a range of reasoning capabilities for the existing active logic formalism. These include: basic inference abilities of forward-chaining and backward-chaining resolution as well as extended modus ponens reasoning; the option of implication premises that execute a body of procedures providing useful functions such as introspective lookups and searches through formula derivations; specialized inference rules such as the detection and distrusting of direct contradictions; and finally, the ability to model the beliefs of other agents in specialized partitions, which supports a category of commonsense reasoning development identified below. ALMA is described in chapter 4.

3. *The application of the implemented ALMA reasoner to commonsense reasoning problems.* ALMA has been designed to be a platform that is capable of supporting relatively general commonsense reasoning through active logic, and its application to a series of commonsense problems demonstrates this. The first of these applications consists of two groups of scenarios for commonsense reasoning with default formulas, in which variations of traditional problems involving interacting defaults, both nesting and otherwise overlapping, are solved in a novel and much more powerful and time-sensitive way — namely, by updating and revising their conclusions in unfolding time due to active logic and the fine-grained metareasoning of quotation. In doing so, we also develop a form of abnormality for default formulas that generalizes that of McCarthy [51, 52]. The second application is the development of axioms for question-answering regarding an agent's beliefs, on the basis of the self-reflective abilities that the ALMA reasoner has been endowed with by use of quotation terms. Finally, an example scenario demonstrates an ALMA-based agent reasoning about both self *and other agents*, through a case study regarding modeling the beliefs and inferences of fellow agents — which pushes the use of formula nesting and quotation beyond a focus mostly on self and the agent's own beliefs. All of these applications are described in chapter 5.

It is important to note that, beginning from the mention above of an agent holding internal beliefs, in the current work there is almost everywhere the idea of an implicit agent in our treatment of active-logic reasoning. Hence, any active logic formula is meant to represent a belief of some active-logic agent, in its own internal representation. However, none of the present work focuses on multi-agent reasoning; our focus is on a singular active-logic agent that may represent what beliefs *this agent* attributes to others. We refer to this singular agent of focus as *Alma* (which also appears in lowercase as a logical constant present in a formula, due to the grammar we define), and note the contrast between this name for the agent and *ALMA*, which signifies the reasoner itself.

We intend the contributions of the present work to make steps in areas which in the long term we hope will provide advantages in terms of futurecapabilities for artificially intelligent agents. That is, agents which would be able to know what they are doing, and why, and thus have a capacity to find and correct errors, and to explain this to themselves and others — and, more generally, to be much closer to human-level artificial intelligence.

1.2 Publications

Several earlier papers formed a progression toward arguing for agents having representational mechanisms which included beliefs about beliefs, such as by Goldberg et al. [24] (presented at a conference by the author) and Perlis et al. [66]. The actual quotation work of this dissertation itself is an attempt to grapple with this, to have nested representations of formulas and have this implemented in active logic systems; the quotation work was motivated and a roadmap for its development sketched out in the publication by Goldberg et al. [25] (also a conference presentation by the author).

Much of the work in this dissertation is also related to — and was supported by — a recent DARPA project, on the foundations of knowledge of cooperative agency. Key to that project were issues about an agent's evolving knowledge of its own beliefs and actions as well as those of a cooperating agent. While agent-cooperation is not a topic of the dissertation, the underlying knowledge-representation issues crucially involve much of the dissertation research. For example, knowledge that making an utterance is expected to alter a listener's beliefs — while seemingly so obvious as to need no special treatment — is far from trivial to represent for an automated system in a general (non-task-specific) manner; and similarly for an agent's evolving knowledge of its own activity. Both of these in turn require the methods developed here.

Thus, research reported in this dissertation (which was begun before the DARPA project was conceived) served as both a stimulus for that project, and also an outgrowth of it, as new issues arose during the course of the project. In particular, quotation and quasi-quotation in various ways figured substantially into the project work, as did time-sensitive nonmonotonicity. A variety of papers have been undertaken as a result of the dissertation work, as well as of the project. Three have appeared in workshops or conferences [12, 14, 37], and several others are in various stages of completion.

1.3 Outline

The outline of this dissertation is as follows: Chapter 2 surveys relevant background literature for logicism and commonsense reasoning that gives context to the logic-based approach, and pulls together past research on active logic, which informs and motivates the present state of the logic to be developed. It also introduces nesting of logical formulas, reviews literature on nesting methods — particularly syntactic theories utilizing formula quotation, and describes the suitability for pairing active logic with quotation. Chapter 3 describes the need for quotation in active logic, details the quotation term mechanism for active logic quotation, and provides examples; this includes the development of quantifying into active logic quotation. Chapter 4 describes the structure, logic, and uses of ALMA 2.0, the commonsense active-logic reasoning engine implementation; ALMA provides a basis for wide-ranging automated reasoning with active logic. Chapter 5 describes the applications to commonsense reasoning that utilize ALMA and the extended form of active logic with quotation. Chapter 6 describes the conclusions of the present work and some limitations which might be addressed in future research.

Chapter 2: Related work

2.1 Logicism and commonsense reasoning

2.1.1 Introduction

Since the beginning of the field of AI, the logicist tradition has championed the use of logic-based methods as a basic tool and foundation for reasoning, where formulas in a formalized language correspond to beliefs within a knowledge base. Broadly speaking, logic-based systems would meet the following criteria: 1) use of syntactic rules for combining symbols, and semantic rules for interpreting them, 2) symbols such as predicates defined with respect to a model, 3) context-independence of the meaning of symbols, and 4) reasoning patterns generally matching description by logical inference [16].

2.1.2 Situation calculus

A foundational form of logical reasoning for commonsense is the situation calculus, developed first by John McCarthy [53]. The situation calculus is a system using a first-order language keeping with the principles of logicism, yet unlike standard first-order logic, is designed to allow dynamic reasoning and incorporate a kind of time through use of situations and their successors. A situation is an entity intended to describe an entire world state at a given point, and situations are objects in the language which can bind to situation variables and be used as terms. Fluents express time-varying relations about the world, and can either map to a truth value as fluent predicates (propositional fluents), or map to a novel situation (situational fluents). For example, $\texttt{at}(\texttt{John}, \texttt{home}(\texttt{John}))$ uses the propositional fluent `at`, with the truth value of this expression determined by whether John is at home. The expression $\texttt{result}(\texttt{p}, \texttt{opens}(\texttt{safe}, \texttt{key}), \texttt{s})$ uses the situational fluent `result`, a special situational fluent that maps to the situation that results from performing an action in the previous situation. In this case, from being in situation `s`, the `opens` action performed by agent `p` leads to the situation indicated by the entire `result` fluent, describing the state of the world with the safe opened by the key.

Even from the situation calculus' first introduction, it was acknowledged that it must address the frame problem. For expressing what information will not change from one situation transition to another, a large number of additional frame axioms are required. As an alternative, all formulas ought to be retained except for those altered by an action causing situation change. The notion that a property that is true will typically remain true, unless something changes it, is called the commonsense law of inertia. An exception to the law is in effect represented as atypical or abnormal. Formalizing abnormality and concerns such as which formulas it would be applied to, were refined in the literature introducing circumscription, described in the following section.

2.1.3 Nonmonotonic reasoning

In the usual first-order predicate logic, reasoning is monotonic: from a set of formulas B, if $B \vdash S$, then adding another belief b does not prevent any of the formulas in S from being proven, and so $B \cup \{b\} \vdash S$. Likewise, entailment is monotonic. For reasoning in commonsense domains, in contrast, monotonicity does not hold. Adding new information to the set of beliefs may prevent the proof of certain formulas that were consequences of a smaller belief set — for example, learning that a particular bird Tweety is a penguin should preclude concluding that he can fly, which would have been a reasonable conclusion to draw when the only belief about Tweety was that he is a bird. Hence, conclusions which might nonmonotonically be withdrawn on the basis of additional premises are known as defeasible. Three prominent systems for handling nonmonotonic evolution of logical consequence are circumscription, default logic, and autoepistemic logic.

Circumscription was first developed by McCarthy [51, 52], and attempts to minimize concepts of abnormality, as is defined with a set of abnormality predicates for different aspects of formulas. For instance, consider birds which can fly. A bird might be deemed "abnormal" in any number of different ways, including with respect to flying, size, weight, color, or other attributes. Supposing the particular abnormality with respect to flying is termed `aspect2`, the fact that birds which are not abnormal in this way can fly is formalized as $\texttt{bird(X)} \land \neg\texttt{ab(aspect2(X))} \rightarrow \texttt{flies(X)}$. McCarthy presented a schema that gave a syntactic approach to transforming the set of formulas circumscribed. This second-order schema is provided an argument of a predicate specified to be minimized alongside `ab`, and produces a set of stronger formulas that minimize the predicate's extension to abnormal cases. McCarthy also provided a semantic point of view on circumscription, that entailment should be restricted to models that are minimal with respect to the circumscribed predicates — i.e., that the most-preferred models have a minimal set of abnormal objects. Lifschitz later further developed this [46, 47], and also proved that this semantic characterization exactly matches the syntactic transformation described by McCarthy.

2.1.4 Other relevant logicist approaches

The event calculus, first introduced by [42] (and further extended and developed by [40] and [41]; also developed significantly by [60]), allows reasoning That directly represents events, their effects, and the periods of time or time points during which such effects might hold. Hence, addresses some shortcomings of the situation calculus in relation to temporal issues, and enables a better handling of events, time, and narratives. This includes the persistence of properties in the future until an event might terminate them, and into the past until initiated by an event. Importantly however, this does not guarantee a reasoning process about time to *itself* be situated in time and evolve. Further, event calculus logics tend to be external, and thus are usually not designed for an agent itself to use in reasoning.

NARS (Non-Axiomatic Reasoning System) is a term-logic based reasoning system that focuses on agents with finite abilities, real-time reasoning, and openness to data and problems [93]. In the system, all beliefs of an agent are revisable, and the logic is nonmonotonic, allows for certain types of self-reference, and designed for concepts to be situated rather than disembodied, and grounded in experience as part of an agent [95]. However, the system lacks a notion of time which explicitly evolves or is seen as an object of reasoning, although there is recognition of issues regarding an assumption in which "the reasoning system itself is outside the flow of time" [94]. Additionally, it does not have specific mechanisms for monitoring ongoing reasoning processes.

SNePS (Semantic Network Processing System) is a logic-based knowledge representation system designed for an underlying motivational purpose of developing intelligent agents which use natural language [83]. The system has in particular been applied to model a cognitive agent, Cassie [84, 85], and is also used as the basis for the GLAIR cognitive architecture [86]. In several ways SNePS shares key similarities with active logic, but also has notable limitations relevant to these areas. First, SNePS is a monotonic logic. Also, although SNePS can represent beliefs about beliefs, Cassie cannot formally retract beliefs, and temporal beliefs must instead be allowed to expire [87]. While SNePS is paraconsistent, contradictions obtained cannot be reasoned about in a way that allows an agent to make use of them. The system uses a meta-logical variable *Now*, which is notably not a logical term, and changes over time — although this is contingent as a result of explicit actions of the agent making a change in the world.

There is a rich history of epistemic logics, from classical logics [22, 33, 54], to more modern developments [80, 89, 92, 96]. Epistemic logics are primarily modal logics, and hence the problem of logical omniscience as identified in chapter 1 is generally relevant to these approaches. While there have been some attempts to circumvent this, frequently a narrower form of omniscience remains as a problem, in that knowledge of the consequences of an agent's axioms that are known to the agent are treated as available immediately — and thus the agent does not represent the reasoning process of deriving this knowledge. Furthermore, there remains an issue of these approaches as external, for which reasoning is done within the logic but not within the agent in an internal manner. We further discuss some attributes of epistemic logic as bear on desiderata for agency in section 2.3.1.

2.2 Active logic

We now turn to discussing active logic, and tracing the key developments of this body of work. The present work's contributions in later chapters come in the form of enhancements and extensions to active logic. As such, at this point we note that with our focus specializing toward active logic, we put aside other types of reasoning such as induction, abduction, and probabilistic reasoning — which although of great importance for AI and a hypothetical fully-featured intelligent agent, would overly broaden the scope of the present work if dealt with here. Active logic can be characterized as deductive in some respects (including usage of introspection and contexts where formulas are not disinherited, both features of which are described below), and in other ways frequently uses default reasoning and nonmonotonic reasoning.

2.2.1 An active and temporal nature

Active logic is most essentially a logic situated in time, with an evolving reasoning process. The set of beliefs attributed to an agent in active logic is not a long-lasting entity that persists over a duration of time; there are many belief sets that are fundamentally parameterized by time. At each discrete point in time, a different set of formulas is believed, and the evolution of one set of beliefs into another is key. The discrete points in time are termed *timesteps*, and active logic timesteps are numbered with increasing natural numbers that abstract away any particular unit of time. In an implementation, timesteps may be grounded so that a specific interval of real time passes for each step, but this grounding is not required.

Inferences occur across the boundary of timesteps. Therefore, inferences in active logic are associated with a transition from a timestep to its successor; when a formula's premises are satisfied at timestep t, the conclusion is newly obtained at $t + 1$ and belongs to the belief set for that timestep. This may be viewed as part of a transition process from the current state of beliefs. For example, the following table shows some inferences across timesteps, from an initial KB of two implications at the first timestep, and a formula `p` being added in at time `t`:

Timestep	Inserted formulas	Inferred formulas
0	$\mathtt{p} \rightarrow \mathtt{q}$, $\mathtt{q} \rightarrow \mathtt{r}$	
1		$\mathtt{p} \rightarrow \mathtt{r}$
2		
...	...	...
t	p	
t+1		q, r

Inheritance of formulas is a notable property of active logic, by which formulas are typically present at the successor to a given timestep in which they were previously present.[1] Some information may conflict with inheritance, and cause formulas to be retracted. Along with providing the new formulas, inference rules may cause beliefs to be retracted as disinherited, which happens during the next timestep transition. In this sense, the logic is *active* as the inference rules are applied to the belief set to derive new inferences or remove old ones, from one discrete timestep to the succeeding one, rather than using an omniscient view of closure under consequence. It is a great strength of active logic that omniscience is dealt with rather simply, given the problem it poses for practical applications of many other kinds of logic.

Reasoning in active logic is situated in time by the evolution of its present timestep. An agent that reasons with active logic, moreover, can have knowledge of time passing due to the tracking of timestep number occurring within active logic formulas as well. A dedicated inference rule, the clock rule, maintains a special

[1] Inheritance also helps active logic address some aspects of the frame problem, in that the standard behavior of the logic is for formulas which are held as beliefs at a given timestep persist to subsequent timesteps.

atomic formula, which uses the predicate symbol `now` to express the timestep value. Each old `now` formula is retracted, and a new one with the incremented timestep value asserted, over each transition in timestep. When the present `now` is used in inference, active logic is accounting for the actual process of reasoning in time and taking time; more details on the rule are given in section 4.6.1.

Temporal logic is a term used to cover a range of logics involving time, which arguably encompasses active logic. However vast majority of temporal logics, from the tense logic used by Pnueli for analysis of program behavior [73] to the formalism for qualitative interval relations by Allen [2], are frozen in time in that they have no evolving now or accounting for agent reasoning in time.

The time-varying nature of active logic belief also makes the logic well-suited to handling two important aspects of commonsense reasoning: the presence of contradictions, and the nonmonotonic nature of commonsense inference.

2.2.2 Paraconsistency

Active logic is a paraconsistent logic that is able to continue to reason when inconsistencies and contradictions arise. Anderson et al. [4] note this as tolerance of *local* inconsistency (or *direct* inconsistency) between A and $\neg A$, in contrast to global inconsistency where the negation of all formulas hold; they further discuss active logic as situated in literature of paraconsistent logic. By virtue of the reasoning evolving in time, if the closure of the knowledge base under consequence is inconsistent, but a direct contradiction between a formula and its negation has not been derived, active logic is not yet affected by inconsistency. When reasoning continues onward over timesteps, and eventually reaches a point where an inconsistency manifests as a direct contradiction, then the inconsistency must be dealt with. Here again, active logic's progression across timesteps allows a convenient way to disarm the contradiction, and avoid erroneous derivations that would follow from an unresolved direct inconsistency — what has been referred to as the swamping problem. A specialized inference rule for contradictions exists:

$$\frac{t : P \quad \neg P}{t+1 : \texttt{contra}(P, \neg P, \texttt{t})}$$

This rule also applies the effect of disinheriting the two contradictands P and $\neg P$. All formulas derived from the contradictands also can no longer be trusted, as the result of being based on information found to be inconsistent. These contradictand descendants are thus likewise disinherited, and new formulas indicating that they became distrusted are derived.

The syntax in the above rule does not respect the active logic formalism, due to the use of entire formulas P and $\neg P$ as arguments to `contra`. The ALMA 1.0 implementation, described further in section 2.2.5, used a practical means to express contradictions within a first-order grammar through indirection in the first two arguments of `contra`. This approach replaced contradictands P and $\neg P$ by the integer index metadata attached to their respective ALMA formula records (x and y, respectively):

$$\frac{t : P \quad \neg P}{t+1 : \texttt{contra}(\texttt{x}, \texttt{y}, \texttt{t})}$$

An approach extending active logic in a way enabling nested formulas is developed in chapter 3, which supports a formalism where arguments to the contradiction rule can represented similarly to the above notation with nested wffs. For now, we sidestep this issue of the syntax and leave `contra` presented as it has traditionally been in active logic literature.

Active logic thus enables a prompt recovery from direct inconsistency with this rule. The acquisition of a `contra` formula in the contradiction-detection inference rule also provides a means to reason directly about the contradiction, which is advantageous for enabling the reinstatement of some formulas that were distrusted as a result of it. Formulas which have been distrusted are not accessible in regular reasoning, and another method of accessing them is needed. Active logic provides for this retrieval with introspection; the special introspective predicates `pos_int` and `neg_int` have their truth value determined by whether their argument is a formula in the present belief set.

Active logic's reasoning about the nature of a contradiction in the belief set is also a kind of metareasoning. Further details of metacognition and metareasoning that can potentially be directed by active-logic reasoning are addressed in section 2.3, following introduction of metacognitive and autoepistemic desiderata for agents.

2.2.3 Nonmonotonicity

Active logic naturally supports nonmonotonic reasoning. This is in large part due to how any formula may be disinherited at a future timestep; retraction of a previous formula is inherently a nonmonotonic change to the knowledge base. Of the different ways formulas may be removed, naturally there is the previously-mentioned contradiction-handling mechanism, that disinherits formulas participating in or descending from a contradiction. The tracking of a present `now` point in time via the clock rule, once again as indicated above, involves retraction due to the ephemeral nature of each `now` formula that has a one-timestep duration and is not be inherited to later timesteps. Other instances of retraction may be built into reasoning, and be further motivated by novel information overturning what was known before, in the classic nonmonotonic sense; active logic gives broad latitude to support such nonmonotonic reasoning.

In a more minor sense, active logic can also represent formulas in the style of theories of default reasoning, in which the lack of a formula in the knowledge base can be checked as a premise to be satisfied for the conclusion. This slightly resembles the style of default nonmonotonic rules developed by Reiter [78] especially if the target of negative introspection is itself a negated formula. However, active logic's timestep-stratified reasoning differs quite significantly from default logics, and moreover if the argument of a negative introspection later is derived in the knowledge base, it does not necessarily threaten formulas derived already.

2.2.4 Older approaches

Research into active logic grew out of older work on kinds of logic called step-logic. Features of certain step-logics were quite close to active logic, sharing the core features. The terminology gradually blended from step-logic to active logic over time; indeed, in some cases Nirkhe developed step-logic, but then referred back in later publications to that same work as using active logic.

2.2.4.1 Step-logic development

Elgot-Drapkin and Perlis began pioneering work toward active logic in developing the family of step-logics [18, 19, 20, 21]. The need for the passage of time during belief derivation, deduction with negative introspection, management of the swamping problem, and the need to track a “now” were all identified as motivating issues in step-logic. Step-logic characteristics were progressively developed through a series of seven numbered logics with increasing sophistication, where the last, SL_7, came closest to the abilities of active logic.

SL_7 was used to solve several interesting commonsense reasoning problems that also involved metacognitive issues. Most notably among these was the Three Wise Men problem, which includes as a central part that the reasoning of agents modeled in the problem takes time to unfold. Hence, the step-logic/active logic temporal approach was a natural fit to realistically tackling the problem. However, the method of representing the beliefs of other agents (the two other Wise Men who the third must model) that is used here we consider inadequate, and discuss further the weaknesses of this in section 3.2.

2.2.4.2 Reasoned change in belief

Miller studied step-logic in contexts related to reasoned change in belief [55, 56], particularly for problems of mistaken beliefs on issues of reference. In these cases, the part of greatest relevance was the nonmonotonic nature of the logic, which was suitable for withdrawing prior meanings attached to language or references, to obtain new and revised meanings. Hence, reasoned change in belief leveraged inheritance/disinheritance.

Miller handled specific issues such as 1) conflating two distinct objects (e.g. having it pointed out that a car thought to be the agent's is another), 2) mistakenly using a reference (e.g. a person the agent had called John has another name), or 3) realizing two objects are referents of the same name (e.g. using the name "John" without realizing two different Johns are being discussed). Miller's work also provided a proof that active logic mechanisms for recovering from inconsistency eventually produce a consistent knowledge base state. The capacity for reinstatement (represented in reasoning with the predicate `reinstate`) of a distrusted formula, that formerly belonging to a contradiction, is also due to Miller. Reinstatement via its special predicate became a key part of broader contradiction responses in active logic; this is discussed further in chapter 4.

Finally, Miller expressed interest in indexicality, in relation to solving a logic puzzle dependent on the identity of an agent making utterances. Part of the solution depends on formulas with a significant external interpretation, such as a "true utterance" predicate. Yet, this is still of note as a very early approach to self-agency, anticipating some of the later goals for active logic.

2.2.4.3 Deadline planning

Nirkhe applied active logic to time-situated planning situations with deadlines [61, 62, 63, 64, 65]. Active logic being situated in time was critical for the approach to appropriately plan while maintaining an awareness that the reasoning consumes time resources, as the deadline draws closer. Nirkhe used somewhat different formalism, more adapted to planning domains: each formula was parameterized by the start and end of its time interval, and action primitives with a condition, action formula, and result were introduced. New inferences were held in a hypothetical plan context, as the logic reasoned about expected consequences and expected state of the world to result from actions in the plan. Rather than general beliefs expanding each timestep, each timestep would bring new operations done by the planner to expand the context set and the partial plan. Temporal projection was used to tentatively project predicates to hold over gaps in their intervals, where these projections could be revised following contradiction-detection if a temporal inconsistency arises.

Metacognition was invoked as a component of this planning, in the way that Nirkhe's system checked the time remaining while continuing to plan; this working estimate of time was an estimate used to determine if the plan still remains feasible. However, in practice the use of metareasoning to calculate the time estimate was quite limited, and was mostly restricted to calculating an estimated value using the worst-case times that were provided as annotations to actions the agent had available. This is a meta-level issue, but the reasoning only superficially concerns other reasoning.

2.2.5 ALMA 1.0

Purang developed active logic into a more standardized form that has been used consistently since, and implemented the first general active-logic reasoner: ALMA 1.0 [74, 75]. ALMA 1.0 is an automated reasoner built using the Prolog language to leverage its existing first-order reasoning abilities. It is run via top-level control in which active logic timestep transitions are enacted using a “step” command, inference is done exhaustively, and key active logic components such as contradiction-handling are performed. Although this version of ALMA has significant limitations to be described later in sections 3.2 and 4.8.1, this was a major advance and provided context for subsequent active logic research. Previously, separate pieces of software that were not well-documented or maintained were developed for each large effort in researching active logic; ALMA 1.0 gave a common platform with some documentation that obviated the need for ad-hoc active logic programs, and was used in every active logic research effort up to the present. ALMA 1.0’s influence of course also includes providing many aspects of the behavior specification, as well as inspiration, for the new reasoner developed in this work. Work on active logic following Purang’s developments relied upon ALMA 1.0, and with the key features of active logic being relatively stable, focus turned toward investigating hypotheses about active logic systems.

2.2.6 The metacognitive loop

The metacognitive loop, or MCL [3, 81], was developed to aid artificially intelligent systems by reducing brittleness in the face of anomalies. The metacognitive loop is based on the idea of a common cycle of noting an anomaly, assessing the issue that has arisen (possibly with some standard or well-defined set of anomaly responses), and then guiding a response to handle the problem. The application of the concept of MCL itself is not restricted to active logic, and was applied in some other disparate contexts, such as guiding the relearning process of a reinforcement-learning game player [5]. Active logic was also a notable test domain of the loop. For detecting anomalies in the system an MCL is built into, active logic and ALMA 1.0 give a very clear indication of anomalies, in the detection of a direct contradiction. An MCL component could then interact with active logic and assist with guidance for resolving the contradiction, toward reinstatement of a particular contradictand.

Later work with MCL explored development of general ontologies for abstract kinds of anomalies and responses [82]. For instance, an indication of an anomaly that might be domain-independent across learners is the lack of an expected reward, or a sensor failure independent across a wide class of robot systems; ontologies could offer refinement from abstract categories to system-specific details. There was also work exploring a version of the loop as a general process that could exchange communications with different kinds of systems, trading overhead of communication with the benefit of a more general metacognitive module [31]. The benefits of the two approaches were contrasted with respect to integration into ALMA and the

ALFRED system, which is described below.

2.2.7 ALFRED

Josyula developed the metacognitive architecture DIRECTOR [38] for the purpose of control for general-purpose interfacing agents — the layer of an abstract system to translate from a user to task-oriented components. A specific instance of an interfacing agent, ALFRED [35, 36], was also developed as an instantiation of such an interfacing agent based on DIRECTOR.

ALFRED is a dialogic agent-system that translates natural language instructions from users, and mapping to commands and instructions. The approach exhibits meta-linguistic skills as well, which are enabled by ALFRED's integration of an MCL component, and foundation of active logic with ALMA. Violations of expectations in user interaction, such as a user repeating a previous command verbatim in short succession, or introduction of words outside ALFRED's present vocabulary, trigger informed responses based on MCL guidance.

2.2.8 Need for a sense of agency

Active logic, as developed to date with the capabilities described above, has been recognized as lacking a strong sense of its own agency and actions. Recent active logic literature in the last few years has included position work advocating for extensions related to a sense of agency, and even the possibility of a concept of "self."

Perlis et al. [66] argued that contemporary AI research has focused on tasks to the detriment of developing systems with a broad sense of agency that may not be capable at tasks. Emphasis was placed on the benefits of developing a simpler agent that can learn by dialogue and observation over a lifetime of its own, maintaining awareness of what it attempts, and reasoning about actions before and after doing — even if it succeeds in few tasks attempted. Relevant advances which might contribute toward this sort of agent were commented on, such as anomaly-handling, robot self-knowledge, and other-knowledge. Along similar lines, Chong [13] emphasized the importance of reflective abilities for an agent, developed a formal reflective model of agency, and created a proof of concept prototype for an agent based on this model, yet had not synthesized the stages into an integrated system.

Work on artificial efference copy [11] compared an internally captured waveform to a robot's microphone recording of its speech, enabling recognition of its own speech suggestive of how humans seem to gain a sense of ownership of their actions. The promising initial application of efference copy indicates it might be a fruitful approach toward realizing further aspects of the processual self, warranting studies of applying it widely. Brody et al. also discussed the concept of an immediate processual self [9, 10], in which an artificial process observes itself and responds to its ongoing processing in a tight loop, perhaps even the same "time slice" (wording that comes quite close to the active logic timestep). This concept is raised as one that may unite some varied ideas of what might be described as "self," particularly if acting and observing unify in time. Active logic was discussed in this context, as seeming well-suited to the further pursuit of a processual self, particularly if it might be able to support a "thick moment" of time, corresponding to a kind of interval, and if it can self-refer during thick moments. The ability to self-refer has also been suggested to be essential in aspiring toward a greater sense of "self" in a holistic sense, or even in contributing toward consciousness. Brody et al. [9] term self to be a "cognitive unifier" based in part on self-reference in a tight loop of ongoing processing, that might "perhaps sow the seeds of consciousness." Perlis similarly connects the notion of self via "strong self-reference" [72] to the development of consciousness.

2.2.9 Other active logic work

Asker and Malec explored the use of active logic modified for resource-bounded agents, such as in embedded systems [6]. Malec took steps toward a reasoner for resource-bounded active logic [49], with inspiration from work on a possible limited memory model to use with active logic [17]. Heins employed an active logic theorem-prover for a nontrivial case of a robot navigating a daily delivery schedule [32].

Vinkov and Fominykh presented a more granular system of time for active logic than integer timesteps, based on an external clock entity associated per agent [90]. Their approach also developed a version of active logic with semantics for an argumentation framework, and a variant with two kinds of negation, including a notion of subjective negation based on inability to derive before the current timestep [91], much like negative introspection.

2.2.10 An active logic grammar

Research contributions in subsequent chapters extend active logic beyond the language that has characterized the work surveyed in this section. As a point of contrast with those developments, a context-free grammar applicable for previous work in active logic is provided here. The grammar is first-order, and includes the usual operators of first-order predicate logic, with inspiration from the syntax used by Prolog. Notably, and also similar to Prolog, quantifiers do not appear explicitly, and all variables are considered to be implicitly universally quantified, with the scope of the quantifier over the entire formula. Existential quantifiers are not a part of the grammar, and must be accommodated by means such as Skolemization over the formulas specified, to translate existentially-quantified formulas into the language. This restriction was motivated by design decisions for the reasoner developed to implement active-logic reasoning, described further in chapter 4.

All of the usual first-order logic term categories — functions, term variables, and constants — are represented in the typical representation, with functions written using prefix notation. Logical operators connecting terms — negation, conjunction, disjunction, and implication — are also represented using prefix notation, much like functions, to be specified unambiguously and avoid defining the precedence of operators in user-defined formulas. For the purposes of simpler unification and reasoning, variable names are assumed to be unique across distinct formulas.

$\langle \mathit{Formula} \rangle \rightarrow$ `and(` $\langle \mathit{Formula} \rangle$ `,` $\langle \mathit{Formula} \rangle$ `)`

$\mid$ `or(` $\langle \mathit{Formula} \rangle$ `,` $\langle \mathit{Formula} \rangle$ `)`

$\mid$ `if(` $\langle \mathit{Formula} \rangle$ `,` $\langle \mathit{Formula} \rangle$ `)`

$\mid$ `not(` $\langle \mathit{Formula} \rangle$ `)`

$\mid$ $\langle \mathit{Literal} \rangle$

$\langle \mathit{Literal} \rangle \rightarrow \langle \mathit{Predname} \rangle$ `(` $\langle \mathit{Listofterms} \rangle$ `)`

$\mid$ $\langle \mathit{Predname} \rangle$

$\langle \mathit{Listofterms} \rangle \rightarrow \langle \mathit{Term} \rangle$ (`,` $\langle \mathit{Term} \rangle$)*

$\langle \mathit{Term} \rangle \rightarrow \langle \mathit{Funcname} \rangle$ `(` $\langle \mathit{Listofterms} \rangle$ `)`

$\mid$ $\langle \mathit{Variable} \rangle$

$\mid$ $\langle \mathit{Constant} \rangle$

$\langle \mathit{Predname} \rangle \rightarrow \langle \mathit{Const} \rangle$

$\langle \mathit{Constant} \rangle \rightarrow \langle \mathit{Const} \rangle$

$\langle \mathit{Funcname} \rangle \rightarrow \langle \mathit{Const} \rangle$

$\langle \mathit{Variable} \rangle \rightarrow$ [A-Z][a-zA-Z0-9_]*

$\langle \mathit{Const} \rangle \rightarrow$ [a-z0-9_][a-zA-Z0-9_]*

2.3 Agency and internal reasoning with nested formulas

In this section, we discuss a logic's ability to represent another formula in the logic, that nests within the outer formula. The most common use of this nesting is to represent propositional attitudes, matters such as knowledge and belief, for agents. As an example, to represent that the agent John has the belief that snow is white, one method to nest this formula using a syntactic theory of belief is as follows:

$$\texttt{Bel}(\texttt{John}, \text{``}\texttt{white}(\texttt{snow})\text{''})$$

Recall that this entire wff is in the KB of the underlying active logic agent Alma, and does not represent a belief of the agent John — and thus, our main interest is this formula being used in the agent Alma's own internal language, as its way of considering that John has the belief about snow. Naturally, beyond nesting a belief, we see that an agent can have reason to have more arbitrary kinds of predicates nested in formulas.

It is useful to be able to use an abstract notion of formula nesting, particularly for the discussion of autoepistemic desiderata for agents, where an arbitrary agent can be considered in a way that is agnostic to specific mechanisms, for detailing what it ought to be able to reason about. Major methods of nesting are reviewed, and ultimately we advocate a specific form in the context of merging it with active logic. However, the terminology for nesting is convenient to continue using, for clarity of the language concept.

2.3.1 Autoepistemic desiderata for agency

To enable wide-ranging metacognitive behavior, formulas from an agent's own knowledge base must be accessible to its reasoning through nesting. A knowledge-based agent which cannot refer to formulas of its own beliefs via nesting would certainly have severe restrictions on its ability to do meta-reasoning; this seems necessary for attaining generality in reasoning about reasoning. The accessibility of beliefs to reasoning through a nesting approach must then be foundational. The hypothetical agent under consideration should be able not only to *use* beliefs (e.g., to infer new beliefs), but also to *mention* (and make assertions about) those beliefs (e.g., that one belief contradicts another, or that a belief was held until a given time). Thus it should be able to nest formulas within beliefs and to perform internal reasoning actions revising these beliefs. This in turn includes as a key part the ability to perform introspection on general formula patterns, determining their presence or absence among its beliefs over time. For the reasoning to be explored here, introspection is an indispensable operation in metacognitive chains of reasoning.

A general-purpose agent should also be able to continue reasoning in the presence of direct contradictions, without the "usual" *ex contradictione quodlibet* — from a contradiction everything follows (also called the swamping problem). Contradictions commonly arise in a changing world, as fluents are updated, and information is inevitably found out to have been hasty, mistaken, or otherwise different than it had been viewed before. The appearance of a contradiction is therefore a natural development in the course of inference which may be a frequent occurrence, and not a disaster indicative of a badly designed formalism. Indeed, when a direct contradiction arises a computational reasoner should represent the presence of the contradiction, and enact an appropriate response (e.g., removing the inconsistency, and then to possibly reinstate one of the two contradictands). The precise determination of what form of response is suitable is context-dependent; it is more essential that the agent *have* some kind of response. It should be able to handle contradictory information through abilities such as: to know where the contradiction lies, to identify each formula that contributes to inconsistency, and to consider which assumption(s) involved should no longer be believed.

Moreover, an agent that resolves a contradiction should be able to recount that process. There is utility in an agent being able to note how it came about the new beliefs of its knowledge base in the current moment: that at a previous point in time it was grappling with a contradiction, and that it held different beliefs, some of which were revised into current ones. In addition to involving reference to expressions of the agent's beliefs, this reasoning is also inherently time-based. That is, it is about the change in beliefs over time as the knowledge base evolves: reasoning that there was once a contradiction, but this belief is no longer presently held, and instead is recalled as one that was believed in the past. Deadline reasoning also reflects both of these aspects. An agent with a deadline must focus on producing and enacting a plan while also taking into account that all reasoning requires time, and that both its planning and metacognition consume that resource.

There is more required beyond contradiction-oriented processing as well. An agent must act, but cannot neglect maintaining beliefs that keep it aware of its actions. A system performing actions without recollection of the nature of these actions, their timing, duration, or general properties would be operating almost totally in the dark. Yet to support reasoning involving actions in this way again requires nesting: if actions undertaken are represented at the reasoning level with predicates, then their properties cannot be expressed without being able to nest beliefs about action formulas. For example, an agent that uses the presence in its knowledge base of the atomic formula `move(left)` as a command for leftward motion might also express a fact about this action using nesting (e.g. that it is an action which can readily be performed in the world: `canDo`(“`move(left)`”)).

Many of these kinds of reasoning deal with evolving time, as well as some mechanism for expressing properties of beliefs via nesting within other formulas, such as the `canDo` formula above. There are two standard approaches to the latter: syntactic and modal. Modal treatments to nesting use possible-world semantics and modal operators of knowledge, belief, or related notions [43, 45, 79]. However, modal approaches tend to assume logical omniscience (although some approaches address this, e.g., [97]). Perlis also described how modal logics suffer from the same potential paradoxes as first-order logic for self-reference [70]. More serious still for our interests is the fact that modal logics are generally external logics in the sense we have defined in chapter 1; or, if an attempt is made to adjust a modal language to be an internal logic, this requires constraints on the semantics of modalities such as belief and truth. With these problems, it seems more effective to work with a syntactic theory of quotation instead, and remain working with first-order logic. Following discussion of some practical constraints on representation for an agent's beliefs, we discuss syntactic theories of belief.

2.3.1.1 Constraints on representation

Before introducing methods of nesting formulas, it is useful to discuss some constraints which should affect the representation to be used by a flexible commonsense reasoner. Nesting methods must deal with constraints such as omniscience, consistency, and semantics — several of the important issues for classic methods identified in chapter 1 as common problems for logicist methods. A suitable nesting approach should not be subject to the problem of omniscience, whether nesting is done in a logic prone to omniscience that has been adjusted to mitigate or address the problem, or if a more limited logic is used. Maintaining direct consistency is discussed above as a valued ability for metacognition, but it also has a place in the nesting method, which should be free of inconsistency and paradox troubles that have held back certain ways of nesting, as are described in the following section. A set of capabilities for handling expression semantics is also desired, in the need to support dual use and mention for formulas or formula fragments.

Metacognition about beliefs will frequently be coupled with reasoning unfolding in time. Incorporating temporal aspects and accommodating temporal evolution for a theory of nested belief is also an important constraint.

2.3.2 Syntactic theories of belief

Syntactic theories of belief tend to use a first-order syntax, but with the added ability to nest formulas via quotation. Given that the language remains first-order overall, a first-order formula is not truly nested, but there is a close relation in each syntactic theory between the quotation that is *to represent a formula* and that particular formula itself. All approaches to quotation surveyed below take the stance that formulas nested with quotation are instances of a kind of string which refer to the formulas spelled out — essentially a kind of new constant in theory's language.

In this sense, using quotation is similar to gödelization representing nested first-order formulas using integers, although the correspondence in syntax between a quotation and formula makes this far more human-readable than Gödel numbers. Again, we provide the simple motivating example of a nested formula for a belief of John, which utilizes quotation: $\mathtt{Bel}(\mathtt{John}, \text{“}\mathtt{white}(\mathtt{snow})\text{”})$. The formalisms described below also support other operations for constructing or manipulating quotation expressions.

2.3.2.1 Konolige

Konolige constructed a syntactic theory in which at least two languages are considered together for an agent: an object language reflecting the agent's beliefs about the world, and a metalanguage that uses quotation to describe beliefs in the object language [39]. This work was in response to prior work using a possible-worlds approach to propositional attitude reasoning and acting, and intended to address the same attitudes better through a syntactic approach. Rather than using a single modal language for expressing what an agent might believe or know, in Konolige's system belief and knowledge are expressed in a metalanguage with nested formulas exclusively from a “lower” object language. Syntactically, expressions denoting object-language expressions are constructed in the metalanguage with terms denoting syntactic elements in that language (e.g. the metalanguage term $\mathtt{and}(\mathtt{P}, \mathtt{Q})$ denoting $P \wedge Q$). But acknowledging that a representation of object language abstract syntax is cumbersome, especially as formulas grow in size, Konolige used quotation marks as an abbreviation around formulas written with the usual first-order operators. Konolige also referred to this syntax as sense quotes or Frege quotes, indicating the sense of a quoted expression is wanted and not its truth value. The specification of a sense quote expression is straightforward in Konolige's theory due to the defining of separate metalanguage terms that denote object language proposition letters, operators, and other constructs.

In this way, for an agent under consideration, metalanguage formulas can express what inference procedures are available that the agent might apply to formulas in its object language. Predicates existing exclusively in the metalanguage capture aspects of object language reasoning: `True` has the Tarskian definition of truth for its quoted object-language argument, `Fact` indicating an object-language axiom, `Pr` indicating provability in the object language based on axioms and inference rules, `Bel` for belief of formulas, and `Know` for formulas believed that also are true in the sense of the metalanguage predicate. These predicates are also used to overcome the lack of an inherent connection between the symbols of the metalanguage and object language. Predicates in the object language (e.g. a binary predicate `P`) are connected in the metalanguage indicating the equivalence of formulas using `P` with `True` applied to matching metalanguage quoted terms for `P`, e.g.:

$$\forall\alpha\beta[\mathtt{True}(\text{``}\mathtt{P}(\alpha,\beta)\text{''}) \equiv \mathtt{P}(\Delta(\alpha),\Delta(\beta))]$$

Where $\Delta(\alpha)$ gives the individual denoted by α.

For building a computational agent, although the object language is the one first identified with beliefs about the world, Konolige made the interesting suggestion that an agent be endowed with the pair both of metalanguage and object language (called ML and OL, respectively, in the agent context). In this case, the use of a predicate such as `Bel` indicates introspective access to what the agent reasoning with the object language knows; If the agent's knowledge base has an OL formula such as `white(snow)`, the next level can "introspect" to have an ML formula such as `Bel`("`white(snow)`").

Because Konolige's sense quotes cannot quote a formula in the same language, for an agent to represent more complex levels of belief, such as the beliefs of other agents or beliefs about its beliefs, additional logics must be added in (for example an OL′, for which OL becomes its metalanguage). Every additional level of nesting agent beliefs in a formula requires dealing with an additional embedded language, with accompanying machinery to be added into the metalanguages above it as new kinds of terms are required. This can impose a significant burden on reasoning with the extra terms to account for, and lead to very long proofs.

Konolige's resulting hierarchical system of first-order languages was used in conjunction with the situation calculus for planning actions and reasoning about knowledge gained by acting. The agent considering a plan to act or infer identifies with the object language hypothetical future world states after the current situation; the metalanguage represents what beliefs the agent would hold in a given situation.

2.3.2.2 Perlis

Unlike Konolige, Perlis [68, 69, 71] studied a single language representing its own formulas using quotation, where all syntactic elements of the language could be accessible within quotation. Rather than Konolige's motivation from planning and acting, Perlis' approach was motivated from a desire to further an understanding of an internal language in relation to external reality.[2] Perlis then developed a single highly-expressive first-order language with the ability to refer arbitrarily to its own sentences, rather than hierarchies of different meta- and object languages. This aims to capture some of the power of natural language, without using it directly. The single-language approach also allows self-referential formulas. Self-reference does lead to issues of consistency and paradox; a significant focus of Perlis' developments was addressing these problems.

Strings that refer to nested formulas are indicated in the theory as follows: for a formula spelled out as the string of n characters $a_1 ... a_n$, quoting this produces the string $n : a_1 ... a_n$ (called by Perlis a Hollerith quotation, or a *name* for the formula to which it refers). The prefix with string length n in some base, such as base-10, and a colon avoids ambiguity from nesting quotation marks, and allows arbitrary strings to be constructed (although, subsequently we use quotation marks for convenience once again, such as "$a_1 ... a_n$" for the above expression). For example, a Hollerith quotation of the formula $\mathtt{X} = \mathtt{Y}$ is $\mathtt{3} : \mathtt{X} = \mathtt{Y}$, and a Hollerith quotation of that string in turn is $\mathtt{5} : \mathtt{3} : \mathtt{X} = \mathtt{Y}$. Both examples above are considered constant terms. Concatenation of these strings is supported by the binary `concat` function, which denotes the appropriate Hollerith quotation of its arguments (e.g. $\mathtt{concat}(\mathtt{n} : \mathtt{a_1 ... a_n}, \mathtt{m} : \mathtt{b_1 ... b_m}) = \mathtt{n} + \mathtt{m} : \mathtt{a_1 ... a_n b_1 ... b_m}$). Note that the numeral value of $n + m$ in the relevant base is the prefix beginning the string.

[2] A language itself can be part of the reality it describes. For example, ""Snow is white" is an English sentence" and ""Snow is white" has three words" are cases where the English language refers to sentences, which are part of the reality it describes. Bits of the language can refer to the world, to some extent language parts of the world, and to some extent to non-language parts of it.

An agent with such a highly expressive first-order language can make statements about the truth, falsity, or other properties of formulas in the language, much like is commonly done in natural language. This also serves as a mechanism for unquoting, in that the truth (once again expressed with a `True` predicate in the language, like used by Konolige) of a Hollerith quotation is related to the truth value of the formula it contains. For relating formulas in such a way, using the following axiom schema appears attractive: $\texttt{True}(\text{“}\texttt{A}\text{”}) \leftrightarrow \texttt{A}$. However, this leads to inconsistency. A liar sentence that asserts its own falsity, such as $\texttt{L} \leftrightarrow \neg\texttt{True}(\text{“}\texttt{L}\text{”})$ is grammatical in a first-order syntactic theory. Substituting `L` for the first occurrence of `A` in the schema and $\neg\texttt{True}(\text{“}\texttt{L}\text{”})$ for the second occurrence gives the inconsistent $\texttt{True}(\text{“}\texttt{L}\text{”}) \leftrightarrow \neg\texttt{True}(\text{“}\texttt{L}\text{”})$.

The solution to this problem that Perlis provided was a revision to the truth schema that restored consistency:

$$\texttt{True}(\text{“}\texttt{A}\text{”}) \leftrightarrow (\texttt{A})*$$

The $*$ is a rewriting operator that replaces the arguments of $\neg\texttt{True}(\text{“...”})$ with $\texttt{True}(\text{“}\neg\text{...”})$, pushing negation inside the quotation marks for truth. The distinction between the cases of negation outside or inside `True` that Perlis identified is that $\neg\texttt{True}(\text{“...”})$ denies that its argument can be codified using `True`, while $\texttt{True}(\text{“}\neg\text{...”})$ codifies the denial of its argument. These express two slightly different forms of negation; the former indicating lack of a decision on being able to represent the argument formula as true. Perlis claims it is justifiable to make the substitution in this case; regardless of whether this is true, proof is provided that modified rule utilizing the $*$ operator leads back to a consistent theory. The expressiveness of the language syntax remains extremely general, and able to support self-referential statements.

2.3.2.3 Haas

Haas followed Konolige in pursuing syntactic theories toward purposes of planning and acting [27, 28, 29, 30], although his developments are closer in spirit to the work by Perlis. Haas used a single logic for his syntactic theory, which is capable of self-reference and incorporates Perlis' solution to maintain consistency in use of the `True` predicate.[3] Instead of supporting string operators that assemble terms with concatenation, the syntax of Haas' language allows any syntactic object (operator, predicate symbol, functor, or term) to be marked as quoted with an apostrophe prefix.

A quoted syntactic object is a constant with the semantics that it denotes the object spelled out after the apostrophe; for example, ″`Superman` denotes the constant ′`Superman`, which denotes `Superman`, which is a constant without quotes that denotes the man from Krypton. The special function `quote` provides a way to translate from a term to another term with an additional level of quotation for each object in it; e.g. `quote(white(snow))` denotes ′`white(`′`snow)`, and `quote(`′`white(`′`snow))` denotes ″`white(`″`snow)`. `True` similarly connects quote terms to their unquoted versions using predicate expressions, rather than what is denoted by a function; an example instance of the truth schema for the same term in the `quote` example is

[3]Although in a later publication, Haas suggested an artificial agent need not necessarily be able to solve the Liar paradox, as humans do not demonstrate the capacity to do so formally in commonsense reasoning.

$\texttt{true}('\texttt{white}('\texttt{snow})) \leftrightarrow \texttt{white}(\texttt{snow})$.

Haas put this logic to use for an artificial agent that represented every formula corresponding to beliefs it held with quotation, regardless of a formula's origin from perception, introspection, planning, or other means. As a result, inference rules for such an agent could be encoded with quotation as well, giving the agent knowledge of its applicable inference rules, and enabling planning about which inference rules will be selected and applied. The downside to this rich control is a cumbersome means of representing even very basic beliefs of an agent, and that all inference steps require managing multiply-nested formulas. By using a `believe` predicate with one argument for another agent's name and another of a quoted formula, as well as a further set of beliefs indicating that agent's inference rules, Haas' system supports an agent simulating the inference of others, although likewise with the same downsides to inference and representation as affect individual reasoning.

2.3.2.4 Morgenstern

Morgenstern, similarly to Haas, motivated a syntactic theory much from the perspective of an agent needing to plan actions relating to its beliefs or lack thereof [57, 58]. For instance, an agent might plan to address its ignorance with knowledge acquisition; this requires an expressive language for reflecting knowledge and belief. With regard to resolving inconsistency due to the liar paradox and other paradoxes of truth, Morgenstern took a very different approach from Perlis, whose approach Morgenstern considered to unrealistically redefine the truth predicate. Morgenstern defined the truth values for the `True` predicate to be recursive based on hierarchically building up a language. The first level consists of the first-order language with `True` added undefined. Subsequent levels of the hierarchy apply `True` to the formulas of the previous level, and assign truth values. The fixed point at which no additional truth values can be assigned gives the final set of *grounded* formulas: those that have obtained a truth value at any point in the process. Paradoxical formulas such as the liar sentence remain ungrounded with respect to all models after this process. The `Know` and `Believe` predicates are similarly recursively defined. This approach comes with the notable problem that determining if an arbitrary formula is grounded is an undecidable problem, however.

Nested formulas are considered to be strings in a style similar to Perlis, although again written with quotation marks for convenience. Due to lacking the focus on expressiveness in language, formulas are also not written in a manner like Hollerith quotation that maximizes generality. `Concat` provides a concatenation function for joining its arguments together, for example `concat`(“`At(X, Y`”, “`S)`”) denoting “`At(X, Y, S)`”. Another function interacting with quotation is `name-of`, which maps from a term to a string acting as its name. For constants, this is always a quoted version of the constant, but in other cases the appropriate choice of the mapped string depends on information about the world (e.g. `name-of(president(US, 1988))` should denote “`Ronald-Reagan`”). Although a useful function in the logic, the reliance by `name-of` on semantic information from the world makes it difficult to properly accommodate in a reasoner doing inference as syntactic proof. Functions, called by Morgenstern `g` and `h`, are used to add or remove an extra level of quotation from an existing string, respectively, for the common cases of assembling or disassembling quotation strings. Morgenstern also provided several shorthand abbreviations defined in terms of the above functions; these abbreviations are relevant for expressing formulas with quantifying-in, and are described in the following section.

2.3.2.5 Quantifying-in

In all of the syntactic theories as discussed above, the languages are not as expressive as is desired. To solve general reasoning problems, there must be a way to allow (in certain cases) that a quantifier outside of a nested formula may reach into the formula (succinctly referred to as quantifying-in, and introduced by Quine [76]). Consider the following example, characteristic of the need for quantifying-in. Suppose in trying to formalize "There is someone whom John believes to be president," the following is attempted:

$(\exists \mathtt{X})[\mathtt{Bel}(\mathtt{John}, \text{“}\mathtt{President}(\mathtt{X})\text{”})]$

The goal is for the quantifier $\exists \mathtt{X}$ outside the scope of quotation marks to apply to the variable $\mathtt{X}$ inside the quoted expression. However, in the kinds of nested formulas for the four formalisms as they have been described thus far, this does not work. The issue is that the entire nested $\mathtt{President}(\mathtt{X})$ is a constant inside of quotation. Some other additional mechanism must allow the quantifier to reach into the quotation to quantify over $\mathtt{X}$. This problem also occurs with cases beyond quantifiers, where more broadly it is intended that the same variable(s) appear in a formula both outside quotation (e.g. $\mathtt{bel}(\mathtt{John}, \mathtt{Y})$) and inside quotation (e.g. $\mathtt{bel}(\mathtt{John}, \mathtt{p}(\text{“}\mathtt{Y}\text{”})))$.

Konolige achieved this through modification to the sense-quote translation rules, in which metalanguage variables may quantify into quoted object language formulas and be translated into the standard name (denoting the same individual in every object language model) of the variable. Perlis' method of constructing a Hollerith quotation using `concat` supports quantifying-in when variables are provided as arguments to the function instead of strings. `Concat` accepts these variable arguments, and uses the value the variable is bound to in concatenation, so long as the binding is for a string, and thus suitable to be concatenated. Haas had the simplest method of the four theories: since a nested formula is formed by quoting every part of its syntax with an apostrophe tag, quantifying-in is rather trivially achieved by not prepending the character before variables to be quantified into quotation.

Finally, Morgenstern's system supports quantifying-in similarly to Perlis', in that arbitrary arguments to `concat` may quantify into quotation when appearing as `concat`'s arguments and wrapped in `name-of` (which produces a string that can be concatenated). Morgenstern acknowledged the complications of this syntax for larger formulas, and gave three shorthand abbreviations for different patterns of quantifying-in: 1) `^p^` is an antiquote that substitutes into a string the (string) expression the variable `p` represents; 2) !`p`! substitutes into a string the expression that `p` represents after applying the `g` function to this expression to add a layer of quotes; 3) @`p`@ substitutes into a string the standard name of its argument (i.e., `name-of(p)`).

Any other general syntactic theory developed must likewise provide some mechanism(s) for quantifying-in, or else expressiveness will be severely restricted.

2.3.2.6 Evaluation for agency

We now consider the reviewed logics with respect to our goals for agency, particularly their approaches to metareasoning, constraints of interest, and their relation to the notion of an internal logic. As a result of our focus on agency and aspects related to “self,” it is not enough for formulas to simply nest, and it is not enough for formulas to represent the beliefs of an arbitrary agent. Our goal is to develop computational agents that reason logically with what can be characterized as their “own” internal beliefs. Hence, the difference between the structure of the overall language of reasoning and the structure of formulas that it can nest is important. To make this more precise, the issue is whether, for a language L, the formulas that may nest inside a formula of L are also (arbitrary) sentences coming from the same language L, or if they are limited to some narrower or different language L'.

The focus on introspection, such as identified in section 2.3.1, makes this clearer. For an agent that is not described externally by a logic, but employs a logic internally for its reasoning, any formula in this language, L, could potentially represent a belief. If this agent can introspect on any of its beliefs, it can then reason both with and about formulas in L. Introspection requires nested formulas, and so the nesting in the logic must be able to work on any formula from L as well. Higher-order logics do not have this characteristic: these logics nest lower-order formulas and predicates via quantifying over them, and formulas in the full higher-order language are not nested. Of the syntactic theories, Konolige’s metalanguage and object language pair also does not meet this criterion, due to introspection proceeding on the object language only, while metalanguage formulas are still part of an agent's processing.

We now evaluate the theories by Perlis, Morgenstern, and Haas. Each uses a single first-order language, with nested quotation of formulas from the same language. Likewise, each of these approaches is designed to preserve consistency and not lead to paradox from the liar problem or others like it. The trouble with these is their inability to handle more general sources of inconsistency, such as if commonsense world knowledge includes a pair of formulas that contradict each other. As we have indicated, this is a common occurrence in a changing world. Perlis, Morgenstern, and Haas take pains to address paradoxes that arise from specific predicates such as truth, but do not present a means to recover from more mundane contradictions; their systems do not have a more general ability to handle arbitrary contradictions. We thus turn to using active logic as the first-order language, to avoid this problem. Like the theories incorporating quotation that have been surveyed, active logic is resistant to inconsistency and paradox coming from problems like the liar problem. But further, active logic is capable of continuing reasoning unhindered in the presence of other direct inconsistencies as well, due to its means of detecting and neutralizing direct contradictions, and the potential for responses that reinstate formulas.

2.3.2.7 The role for active logic

Active logic, as has been discussed in chapter 1 and section 2.2, is considered an internal logic and gives a useful basis for some aspects of developing agency. However, previously active logic has been deficient in that it has not incorporated a quotation mechanism, and presently cannot refer to formulas within the language. Active logic in combination with quotation would have the capacity to jointly develop the goals of sense of agency, self-reference, and improved metacognition. This leads to the need for enhancing any reasoner implementing active-logic reasoning as well, to incorporate an implementation of quotation, address the issues arising in adapting unification, introspection, and the practicalities of variable handling for quotation, and handle other algorithmic details which in previous work regarding quotation were adequately not advanced. Chapter 3 develops nesting via quotation specific to active logic. As an intermediate step, the remainder of this section introduces motivational examples of quotation that are independent of a particular formalism.

2.3.3 Motivating examples of quotation

In this section, we start with developing some more detailed motivational examples using first-order syntax extended with quotation. A time-sensitive reasoning capability of some form is assumed, where beliefs might be altered or revised as time passes, such as active logic provides. Without having yet introduced a particular approach to quotation for use with active logic, the generalized syntax is intended to motivate an extended form of quotation and to showcase reasoning it can support. We also purposefully avoid requiring syntax or structures specific to an implementation here; hence this chapter has independence of the material in chapter 4. The primary inference rule assumed of the reasoner is a generalized modus ponens, inferring from a conjunction of literals as the antecedent.

2.3.3.1 Referring to and updating an agent's beliefs

An agent reasoner may be integrated as one of several components within a physical system (for example, situated as the means of controlling a robot and issuing executable action commands). One way to enact actions from reasoning is by designating atomic "impetus" formulas (or "iwffs") corresponding to primitive actions. Once an iwff is detected by other system components, the system attempts its corresponding action. Iwffs are not persistent beliefs, but should remain only until prompting the rest of the system; accordingly, other components should update iwff beliefs when the desired action is attempted.

Consider a reasoner that has concluded it ought to move left. This can be represented via the iwff, where `move` is a function:

$$\texttt{do}(\texttt{move}(\texttt{left}))$$

This sentence, when found in the reasoning component's KB, may be invoked by the overall system as a command to the wheels to move a certain amount to its left. Once the action is initiated the reasoner should be instructed to update the transient iwff formula, so that the agent knows the action is now being done. A command to update a formula into another would nest two quoted formulas inside:

$$\texttt{update}(\text{“}\texttt{do(move(left))}\text{”}, \text{“}\texttt{doing(move(left))}\text{”})$$

Similarly, an update from `doing` to `done` should occur when the action completes.

2.3.3.2 Reasoning about the presence or absence of beliefs

Quotation can enable more expressive formulas, which specify patterns of inference that would have otherwise required different instances of schemata.

Positive introspection — consider the following formula:

$$\texttt{goal(G)} \wedge \texttt{fulfills(C,G)} \wedge \texttt{pos\_int(C)} \rightarrow \texttt{completed(G)}$$

If the agent adopts a goal `G`, and a condition `C` fulfills the goal, the agent notes its goal as met when that condition is believed. In this case, the `fulfills` predicate's first argument is a condition that must become true. To keep the theory first-order and not present a variable where a predicate would occur, `C` is used in positive introspection instead of directly as a premise. The truth value of `pos_int` is based on searching the knowledge base for a unifying formula, once its argument variable has been bound to a constrained quoted formula. As an example, consider a case where the agent is outside, it is windy, and the agent doesn't want a box to blow away; it further knows that things filled with sand are heavy and so will not blow away. Then, when based on this scenario, if our `C` above is “`sand_filled(box_256)`” and `G` is “`heavy(box_256)`”, during inference if the formula that the box is filled with sand is in the knowledge base, the completion of the goal to make it heavy would follow. In the absence of quotation devices, a schema instantiating a separate formula for each predicate to appear in place of `C` would be required.

More general positive introspection — introspection should not be restricted to checking only ground formulas; the truth value of `pos_int` should be determined by whether any formula is known that can unify with its argument. Consider if the agent in addition to the goal satisfaction formula above above has the following:

$$\texttt{goal(``heavy(Obj)'')}$$

$$\texttt{fulfills(``sand\_filled(Obj)'', ``heavy(Obj)'')}$$

In this example, if the object is filled with sand, this is a success condition for making that object heavy. The introspective lookup will proceed with the expression "`sand_filled(Obj)`", and a ground term such as `sand_filled(box_256)` may be retrieved as a known matching fact.

Negative introspection — in a logic without a closed world assumption inference rule, reasoning about the absence of certain formulas may still be desired. For instance, an agent reasonably ought to have complete knowledge of its actions, but should not assert numerous formulas about the actions it is not performing at each point in time. Or, returning to the scenario of protecting a box from blowing away, an agent might defeasibly expect that boxes aren't filled, without an explicit negative formula indicating so. Negative introspection applied to formulas that should not appear in the knowledge base gives a flexible means of inferring about their absence:

$$\texttt{box(X)} \land \texttt{neg\_int(``sand\_filled(X)'')} \rightarrow \texttt{can\_blow\_away(X)}$$

2.3.3.3 Distinguishing experience and quoted expressions

Information may come into the knowledge base from many ways, including inference but also from observation, such as by robotic sensors, other agents, the internet, etc. Not all such information may be true (in fact, formulas could be contradictory). A system that has acquired information must be able to evaluate its acceptability. Treating these statements to be evaluated as quoted expression terms achieves this; the formulas can be tracked and reasoned about without committing to assignment of a truth value.

So, an agent whose microphone records the sound of a fire alarm might well believe the formula `went_off(alarm)`. But if the agent instead does not directly experience the sound of the alarm, but rather hears *about* it via a message from an unfamiliar source, the agent can instead record `heard("went_off(alarm)")`. What is heard here certainly will not lead to the same inferences as `went_off(alarm)`. After reasoning such as running an evaluation procedure, the contents may then be withdrawn from quotation and asserted as true (or false). An agent believing the following formula might then decide to evacuate the building in the case of directly sensing an alarm:

$$\texttt{went\_off(alarm)} \rightarrow \texttt{do(initiate\_evacuation\_procedure)}$$

Behavior when the quoted version was heard will depend on the results of any evaluation procedures triggered.

Chapter 3: Quotation in active logic

3.1 Introduction

The need for nesting of propositions and formulas in a logical reasoning system was established in chapter 2.3. That chapter also discussed constraints that affect all of the main forms of nesting, particularly the need for internal reasoning and an internal logic to support agency. Theories of quotation have promise in being able to support goals for agency, but this has been hindered by their development in external rather than internal logics. Now, we introduce how approaches to nesting formulas might be realized in active logic and ultimately incorporated into a reasoner. Active logic has the capacity to address the biggest shortcoming of syntactic theories, as an internal logic system which also is capable of handling direct inconsistencies as they arise.

3.2 Need for quotation in active logic

A mechanism of quotation also offers the means to address shortcomings in active logic's reasoning. As has previously been described, active logic has the ability to introspect on the presence or absence of formulas in the current belief set using the special predicates `pos_int` and `neg_int` for this purpose. This usage grew out of prior development in Purang's active logic work [74, 75] and older ALMA 1.0 reasoner implementation. Introspection was also previously represented in older step-logic work by Elgot-Drapkin and Perlis as `Know`, or alternatively formalized in shorthand via the `K` predicate symbol, for knowledge present in the belief set of the active-logic agent [19, 21]. For example, the binary predicate `K` would be used in a schema like $\mathtt{K}(\mathtt{t}, \mathtt{X})$: the agent has knowledge of a formula, `X`, which is believed at the timestep `t`. Both the means of introspection present in ALMA 1.0, due to Purang, and the older formalism of Elgot-Drapkin with `Know` describe a means of reference to formulas in the active logic language, where the referred formula itself ought to nest within another formula — in this case, within one of the predicates indicating introspection. Elgot-Drapkin and Perlis identified that quotation marks ought to be used around the formulas that are introspection arguments to `Know`. Yet, their work did not go any further in characterizing quotation, and the convention of quotation marks around formulas was not consistently followed within the step-logic papers and examples. The work of Purang did not account for quotation explicitly at all, making no mention in it. As a result, no mechanism accounted for quotation in ALMA 1.0 code for implementing `pos_int` and `neg_int`, and this deficit persisted in later active logic work which built upon ALMA 1.0. The present work seeks to corrects this deficit, first in the active logic theory and subsequently in the implemented reasoner.

In the absence of quotation, any argument to positive or negative introspection is not directly a nested formula, but a composition of first-order functions, which together stand for a nested formula in that specific context. Consider an example from Purang's system, supposing the reasoner has `flies` as a predicate symbol. Ordinary negation can express that a bird doesn't fly, e.g. $\neg\texttt{flies}(\texttt{tweety})$. If an agent must reflect on its beliefs as to whether a particular bird is considered to fly or not, a symbol `flies` is also used inside of one of the introspection predicates, such as `neg_int`:

$$\texttt{bird}(\texttt{X}) \wedge \texttt{neg\_int}(\texttt{not}(\texttt{flies}(\texttt{X}))) \rightarrow \texttt{flies}(\texttt{X}) \tag{3.1}$$

But according to the first-order grammar of active logic, and echoing the usual interpretation of first-order languages, `not` and `flies` in the above formula are interpreted as functors, rather than the negation operator or `flies` predicate symbol, respectively (and likewise would be parsed as functors by a reasoner). A designer of active logic axioms would likewise need to imitate any other logical operators with unary or binary functions in such situations, and need to imitate predicate symbols with functors. Then, to make a connection from this combination of functions to an active logic well-formed formula it is to represent, there is a particular interpretation applied: that the outermost functors with operator names are interpreted as stand-ins for the operators, and that functors directly inside these "operators" are stand-ins for predicate symbols.

At the time, this approach in past work for referring to formulas may have been sufficient. ALMA 1.0 was able to successfully handle many commonsense problems across domains, and the simple reference mechanism using functions was adequate to implement some degree of working introspection. ALMA 1.0's commitments from being built on top of Prolog may have reduced the number of issues related to interpreting functions as operators, since all operators occurring in ALMA 1.0 formulas would have required such an approach, not just nested operators.

Regardless of prior treatments, the present focus on agency and explicit representation of an agent's internal processing motivates a need for nesting and reference beyond introspection contexts of `pos_int` and `neg_int` (e.g. in formulas such as `bel(john, phone_number(mary, number)))`. If nested formulas are to be able to appear anywhere in the language where other terms can, then this old method of interpreting certain function arguments as standing for quoted formulas is quite problematic.

This old approach fails to make a clear distinction between when an embedded function is to be interpreted as standing for an object in the domain or for a formula which could be true or false. For example, consider the formula `p(j, q(m, n))`. There are two different ways `q(m, n)` could be interpreted. The first is that it is a boolean-valued formula, where `q` is a functor that stands for a predicate symbol (e.g. `q` might be `greater_than`, and `q(m, n)` expresses the truth of $m > n$). The second is that it is a function that stands for an object in the domain (e.g. `q` might be `sum`, and and `q(m, n)` expresses the value of $m + n$).

Other information would be necessary to make the determination of whether a function in an active logic formula represents a nested formula (we call this the formula interpretation), or represents an arbitrary function instance (the function interpretation). Some new form of context must aid the process, regardless of whether the information for disambiguation comes from additional formulas and beliefs, which would express the contexts or predicates that interpret functions as nested formulas, or if this is achieved by a requirement for every function to track its status with metadata or extra arguments.

The suggestion to in some way associate with a function an indicator for being under the formula interpretation versus the function interpretation seems a promising direction. Yet several likely approaches tracking this distinction, without a new syntactic device, also seem cumbersome: appending a final argument to function(s) to indicate their interpretation would add bloat and impair readability; storing metadata attached to a function might be practical for coding into a reasoner, but would complicate the active logic theory. A better approach is clearly needed.

3.3 Quotation terms

Our solution to the nesting problem is to introduce a quotation primitive, the *quotation term*, into active logic as a fourth kind of term in the language. A quotation term is thus distinct from variables, functions, or constants. Active logic extended to include this kind of novel term results in a language in which a quoted formula (e.g. "$\mathtt{q}(\mathtt{m},\mathtt{n})$") have the formula interpretation simply by its nature as a different sort of term. Quotation terms are distinguished from a function (e.g. $\mathtt{q}(\mathtt{m},\mathtt{n})$)), which would exclusively have the function interpretation. This choice brings complications as well, particularly in adjustments that will be required to data structure representations in a reasoner, and the unification algorithm to properly accommodate another variety of term. It also contrasts with prior syntactic theories of quotation (see section 2.3.2), which have considered each instance of a nested formula to be a unique form of constant, instead of another sort of term,. The distinction between an atomic formula within a quotation term such as "$\mathtt{q}(\mathtt{m},\mathtt{n})$" and a constant name for $\mathtt{q}(\mathtt{m},\mathtt{n})$ is a minor one if the formula contains no variables — however, in other cases there is a clear distinction between constants and quotation terms, such as in cases with variables or additional structure to the formula.

3.3.1 Formalism

Active logic has formerly used the following rule for a term, as reproduced from the grammar in section 2.2.10:

$\langle \textit{Term} \rangle \rightarrow \langle \textit{Funcname} \rangle$ `(` $\langle \textit{Listofterms} \rangle$ `)`

$|$ $\langle \textit{Variable} \rangle$

$|$ $\langle \textit{Constant} \rangle$

The new syntax to incorporate quotation terms will instead change this to:

$\langle \textit{Term} \rangle \rightarrow$ `quote(` $\langle \textit{Formula} \rangle$ `)`

$|$ $\langle \textit{Funcname} \rangle$ `(` $\langle \textit{Listofterms} \rangle$ `)`

$|$ $\langle \textit{Variable} \rangle$

$|$ $\langle \textit{Constant} \rangle$

Thus, the above production formalizes that a quotation term is a novel type of term in the logic. The nonterminal $Formula$ in the above rule is the symbol associated with the production of an active logic well-formed formula, as defined by the new language of the term production rule modified as above, and the remainder of the grammar of section 2.2.10. An entire quotation term therefore consists of: 1) *quote*,

2) the set of parentheses, and 3) the formula that is contained within these parentheses, which is considered to be quoted due to its context as a part of a quotation term.

Several primary use cases for quotation are for purposes of metacognition and self-reference, as well as representing beliefs about the beliefs of other agents. Hence, this initial restriction to well-formed formulas seems sensible: all of these use cases deal with nested wffs, rather than nesting strings of characters that are not full formulas. If a need to accommodate more arbitrary character sequences arises later, an appropriate extension for this may be considered and devised. Active logic following the extended grammar can include quotation terms in syntax trees for its formulas; this also naturally allows arbitrary nesting of these quotation terms, since any well-formed formula nested within a quotation term may itself contain further quotation terms of its own.

Syntactically, *quote* is not a functor. This follows by virtue of its appearance as a string only in a kind of term that is distinct from a function. Indeed, it must be enforced that the string *quote* cannot appear as a functor symbol, to avoid ambiguity in parsing terms. Further, for the unification behavior of quotation, quotation terms and functions are not unifiable given their status as different sorts of terms. However, the notation here still strongly resembles that of a function. *Quote* is better understood as shorthand for quotation marks wrapping the wff, but the use of parentheses is convenient for specifying quotation terms unambiguously for the reasoner. The quotation term production rule syntax, as well as the fact that its argument is a formula, might potentially suggest to readers a modality.

Although there is a resemblance in form, a quotation term can be distinguished from modality by the fact that it remains a term, not a formula itself, and intuitively can be thought of as denoting an object in the domain of interpretation rather than a truth value. For example, "`p(c)`" is a quotation term that denotes the expression `p(c)`. In this respect, where nesting is viewed as formulas directly present within other formulas (i.e., not as constants that spell out a formula), active logic steps interestingly outside the boundary of purely first-order systems. It also departs from other syntactic theories, in that it requires the nested formulas in quotation terms to be wffs that are interpreted in a quoted context, where they do not bear a truth value. This change has been made with considerations for the implementation and efficiency of use, and will be justified in the following section.

Subsequently to defining the production rule, when there is no ambiguity in notation, quotation marks will be used to wrap the nested wff instead of notion with `quote`.

3.3.2 Quotation term reasoning

From having defined the syntax of quotation terms, we next describe what reasoning ought to be enabled by representing quotation terms. Progressively more sophisticated examples are presented, as motivation for the development of informal semantics for quotation terms.

3.3.2.1 A point about quoted formula meaning

As we begin developing quotation reasoning by key examples, we first clarify a point about the meanings of quoted formulas. We turn to a scenario of the sort introduced in the previous chapter, in which an agent uses quotation to mention information that has been heard secondhand from a (fallible) fellow agent.[1]

The following is a very simple formula using a quotation term to characterize one such scenario:

$$\texttt{heard}(\text{``}\texttt{on\_fire}(\texttt{site1})\text{''}) \land \texttt{smoke}(\texttt{site1}) \to \texttt{on\_fire}(\texttt{site1}) \qquad (3.2)$$

Here, we focus just on the first literal, `heard(“on_fire(site1)”)`, and its possible meanings.

There are two fairly natural interpretations of having heard something, as expressed by the predicate `heard` with a quotation term argument. The first of these is that the quoted formula was heard *verbatim*, in a way that the syntax within quotation precisely matches the syntax of what was uttered by the other agent. We can make this more apparent by also considering a `heard` literal variation which has a variable inside of quotation. Under the first interpretation considered here, `heard(“on_fire(X)”)` means that the other agent used precisely the variable `X` in their utterance. This would then not be interpreted as meaning the same as `heard(“on_fire(Y)”)`, which is a formula employing different syntax (i.e., `Y` instead of `X`) to express things being on fire.

The second of the meanings for `heard` does not look so closely at the syntax

[1] We mention fallibility here so that there is no assumption made that the item heard is necessarily true.

of tokens inside of the quotation term argument. This could instead be termed *heard-that* — so if an agent has heard that locations are on fire, the quotation term indicating this fact can use any (implicitly) quantified variable name to do so. Here with the *heard-that* interpretation, `heard`(“`on_fire(X)`”) has the same meaning as `heard`(“`on_fire(Y)`”).

In this way, an aspect of the present work edges up against natural language. An English version of the two senses of having heard can be demonstrated by pointing out how in one sense having heard “Locations are on fire” is not the same as having heard “Places are on fire,” due to the two using a slightly different series of words. Yet, either one of these sentences expresses a message about sites that are on fire, that is the same as what the other sentence expresses. We acknowledge this overlap of the commonsense reasoning issues at hand with natural language, but put aside the case of being concerned with exactly what an agent has said verbatim (including the particular variables used in an utterance).

For the purposes of this work, when the predicate `heard` is used, it is always to be interpreted as *heard-that*, and hence, variable names may be freely replaced with other names inside the quotation term without any change in meaning (as long as variable clashes are avoided). More generally, we will not consider any predicate which inspects the precise syntax of any argument quotation term. For instance, we exclude predicates for which the intended meaning is defined based on a quotation term argument containing a particular variable, or based on the precise length of the quoted formula string. Putting aside such predicates has implications in the context of quoted formulas for variable bindings and substitution. Most of all, it enables variables occurring within quotation to be bound to other variables in quotation, and eventually substituted for other variables when a binding has been successfully made.

3.3.2.2 A ground quotation term

With the intended meaning of the **heard** predicate clarified, we now return again to the simple formula 3.2. Here we focus on the entirely of the formula, not just the **heard** literal.

$$\texttt{heard}(\text{“}\texttt{on\_fire}(\texttt{site1})\text{”}) \wedge \texttt{smoke}(\texttt{site1}) \rightarrow \texttt{on\_fire}(\texttt{site1}) \qquad (3.2 \text{ revisited})$$

In this case, what was uttered by the fellow agent has been captured inside the quotation term, which is the argument to the **heard** predicate held by the first agent. The **heard** predicate symbol represents an agent-centric sense of the agent representing it having heard that the site is on fire, while the predicate symbol **smoke** can indicate finding physical evidence of smoke in the environment. If what was heard was the report of a particular site (site1) on fire in the building, and that has been verified from detecting smoke, it is concluded that there really is a fire at this site (on_fire(site1)).

Making this inference requires the ability to do unification between formulas containing a quotation term, such as between the implication's first premise and the atomic formula **heard**(“on_fire(site1)”). Intuitively, for ground formulas inside quotation terms, the quotation terms must unify only if the formulas contained have identical structure of predicates, functions, and constants — as is the case in this example. No new bindings will be added to the unifier from such unification.

Formula 3.2 is not especially insightful. It is highly-specialized, dealing only with the case of a fire at the specific location given, makes little use of the quotation term, and the unification behavior described for quotation terms is quite obvious.

3.3.2.3 A variable both inside and outside quotation

Consider an attempt to rewrite formula 3.2 to be more general, by replacing `site1` with a variable `X`:

$$\texttt{heard}(\text{“}\texttt{on\_fire(X)}\text{”}) \land \texttt{smoke(X)} \rightarrow \texttt{on\_fire(X)} \tag{3.3}$$

By making a replacement of the specific site constant with `X`, this formula is intended to express that an agent hearing about *any* site reported as on fire, and seeing smoke visible, should conclude that particular site is on fire. However, the variable name `X` is inside a quotation term and also is used elsewhere outside of the quotation term.

We consider whether formula 3.3 would express the intended meaning, given this usage of `X` both inside and outside of quotation. The consideration that all active logic variables are implicitly universally quantified is essential in addressing this.[2] As was discussed in section 2.3.2.5, universal quantifiers do not reach into quotation without the use of a mechanism for quantifying-in, which we have not yet provided. It is necessary to extend the quotation terms of active logic to include such a mechanism as well; this mechanism is introduced in section 3.4.

The occurrence of `X` within the quotation term therefore cannot indicate the

[2] As indicated in section 2.2.10, existential quantifiers are neither currently a part of the grammar, nor considered to be implicit in formulas, as universal quantifiers are.

same variable that is the argument to both `smoke` and the conclusion `on_fire`, despite these variables being spelled out identically. Hence, formula 3.3 fails to give the intended meaning in which all occurrences of `X` are in fact the same variable. As written, the meaning of that formula is more clearly expressed as the following interchangeable formula, which contains distinct names for what are actually distinct variables, to not confuse the quoted variable with the variable outside of quotation:

$$\texttt{heard}(\text{``}\texttt{on\_fire}(\texttt{X})\text{''}) \land \texttt{smoke}(\texttt{Y}) \rightarrow \texttt{on\_fire}(\texttt{Y}) \tag{3.4}$$

This reveals some notable limitations arising from a lack of quantifying-in. The ability to express a formula with the original intended meaning of formula 3.3 becomes available after we introduce quantifying-in mechanisms in section 3.4. However, despite having a meaning identical to formula 3.4, formula 3.3 remains a well-formed formula in the system of active logic with quotation, and is distinct from the formula that results from quantifying into `heard` on `X`.

We now make explicit the implicit universal quantifiers for variables `X` and `Y`. `Y` only appears outside of quotation marks, and it is clear the scope of the quantifier applies over the entire formula (which we also will refer to as *formula-level scope*), as occurs for quantifiers with other variables that do not involve quotation. Without a means of quantifying-in, by definition a variable that is quantified to have a scope over an entire outer formula cannot also occur inside of a quotation term. This leaves the only possible location for `X`'s quantifier as within the quotation term. For a formula nested in a quotation term as simple as `on_fire(X)`, there is only one possible scope for the quantifier. In the more general case of arbitrarily complex formulas embedded in quotation terms, it is reasonable to require that all universal quantifiers that occur within quotation marks have a scope over their entire quoted formula — the largest possible scope permissible by the intuitive semantics. In subsequent examples, this interpretation is followed for all quoted variables where quantifiers are not explicit. Formula 3.4 is then rendered more traditionally with its quantifiers as:

$$\forall \mathtt{y}[\mathtt{heard}(``\forall \mathtt{x}[\mathtt{on\_fire}(\mathtt{x})]") \land \mathtt{smoke}(\mathtt{y}) \rightarrow \mathtt{on\_fire}(\mathtt{y})] \tag{3.5}$$

On the basis of this quantifier placement, the first premise in English would indicate having heard a statement that all locations are on fire. This principle of quantifier placement also holds in the case of several nested quotation term formulas; for example, consider the following formula in which having heard that all locations are on fire is a belief attributed to the agent Alice:

$$\mathtt{bel}(\mathtt{alice}, ``\mathtt{heard}(``\mathtt{on\_fire}(\mathtt{X})")") \tag{3.6}$$

This formula with its quantifiers made explicit is the following:

$$\mathtt{bel}(\mathtt{alice}, ``\mathtt{heard}(``\forall \mathtt{x}[\mathtt{on\_fire}(\mathtt{x})]")") \tag{3.7}$$

3.3.2.4 Unifying a quotation term containing a variable

Consider a formula that expresses the knowledge that, if the active-logic agent heard an utterance that all locations in the building are on fire (and thus the fire is widespread), and this has been confirmed (perhaps consulting cameras instead of touring the burning building), then an emergency evacuation action should result:

$$\mathtt{heard}(``\mathtt{on\_fire}(\mathtt{X})") \land \mathtt{confirmed}(\mathtt{widespread\_fire}) \rightarrow \mathtt{do}(\mathtt{evacuate}) \tag{3.8}$$

Here the argument to `on_fire` is a variable instead of a constant and so it is somewhat less trivial to determine which formulas ought to unify with the first premise. We now consider the desired behavior for this formula's intended meaning, and which formulas should be unifiable with the `heard` premise.

Intuitively, an atomic formula such as `heard`("`on_fire(site1)`") must not unify with `heard`("`on_fire(X)`"). Knowledge of the former quotation term, with a constant that indicates one site's status, would not be sufficient to meet the condition of finding all locations on fire, as expressed by an implicitly quantified quotation term formula (`heard`("`on_fire(X)`")). On the other hand, `heard`("`on_fire(Y)`"), similarly to `heard`("`on_fire(X)`"), is a broad statement about all sites. Although the two are not syntactically identical, with unique variable names across distinct formulas, it is reasonable to give the same meaning to two instances of `heard` that differ only in a variable name. On this basis, the quotation term including `Y` must clearly be unifiable with the implication premise of formula 3.8. Expressing that "all locations in the building are on fire" does not depend on the variable name used.

From here, we consider more generally what terms ought to unify with a variable inside of quotation. In doing so, it helps to again consider quantifiers made explicit in the formulas, which we recall means that universal quantifiers for quoted variables appear just within the nearest enclosing quotation marks. For two quoted formulas that are identical in structure except for a variable appearing in one formula and a ground term appearing in the second formula in the same places where this variable occurs, the second formula has at least one fewer quantifier within the quotation marks. If binding the variable in question to the ground term were allowed to occur, substituting occurrences of the variable for its binding would result in changing the formula structure by removing a quantifier, and hence yielding a more specific quoted formula. Since quoted formulas are in many ways treated as mentioned formulas that capture a greater degree of specificity, it is sensible that a quoted variable cannot unify with a ground term, preventing specialization when substituting. This is essentially a generalized version of the argument above, where a ground quoted formula about a single fire cannot possibly imply that all locations are on fire. Similarly, a quoted variable should not unify with a non-ground function term containing other quoted variables, because these non-ground terms are also more specific quoted content than the universally quantified variable, regardless of whether the formulas they are embedded in may contain more universal quantifiers in total. Thus, a quoted variable may only unify with another quoted variable, and not another type of term.

The idea that quotation terms, as presented thus far, should not unify with more specific quotation terms also carries through to considering the unification behavior of terms with multiple variables. For example, consider two quotation terms with differing numbers of variables, that express beliefs about what Alice believes other agents may see:

$$\texttt{bel}(\texttt{alice}, \text{``}\texttt{sees}(\texttt{Agent}, \texttt{Agent})\text{''}) \tag{3.9}$$

$$\texttt{bel}(\texttt{alice}, \text{``}\texttt{sees}(\texttt{Agent\_a}, \texttt{Agent\_b})\text{''}) \tag{3.10}$$

We can also make quantifiers explicit in these formulas for clarity:

$$\texttt{bel}(\texttt{alice}, \text{``}\forall \texttt{agent}[\texttt{sees}(\texttt{agent}, \texttt{agent})]\text{''}) \tag{3.11}$$

$$\texttt{bel}(\texttt{alice}, \text{``}\forall \texttt{agent}_{\texttt{a}}, \texttt{agent}_{\texttt{b}}[\texttt{sees}(\texttt{agent}_{\texttt{a}}, \texttt{agent}_{\texttt{b}})]\text{''}) \tag{3.12}$$

Formula 3.11 thus indicates that Alice believes that any agent sees itself, while formula 3.12 indicates that Alice believes that any agent sees all other agents, which is more general. Hence, these should not unify with each other. Formula 3.9 should be unifiable only with a quoted `sees` literal with a single variable used for both arguments, formula 3.10 should be unifiable only with a quoted literal containing two distinct variable arguments. This example generalizes to a broader statement, as the second of two principles characterizing quotation term unification. For two quotation terms to unify: 1. Their quoted formulas must have identical structures of operators, functions, literals, constants, and further-nested quotation terms (with the exception of quoted variables); and 2. For each quoted variable in one formula, all occurrences of this particular variable must correspond to exactly one variable in each corresponding location in the other formula.

Here an interesting distinction from other sorts of terms is evident: a variable within the formula of a quotation term may unify only with another variable, but neither a constant nor a more specialized pattern of variables. In contrast, a non-quoted variable, as in `heard(S)`, does not behave as selectively for unification, and for example `S` can unify with arbitrary constants, such as `site1` and `site2`, or other terms that it subsumes.

3.3.2.5 Inferring a quotation term as true

The previous examples have dealt with specific predicates quoted inside a quotation term, such as `on_fire`. There must be a means of extracting the information mentioned in a quotation term, so that a reasoning agent can have the potential to trust or believe this information. We now consider expressing a formula that is more generalized in its ability to address reasoning based on any kind of utterance heard from another agent. Consider the following formula, for expressing that anything heard and successfully investigated (`confirmed`) becomes a belief:

$$\texttt{heard(X)} \wedge \texttt{confirmed(X)} \rightarrow \texttt{true(X)} \tag{3.13}$$

For more generality, variables are now used in place of quotation term arguments to the predicates that were used in earlier examples. Intuitively, a regular variable should be able to bind to a quote during unification, which would permit `heard`(“`on_fire(site1)`”) to satisfy the first premise of the rule above. The predicate in the consequent, `true`, should achieve the effect that its argument will be trusted as true by an active-logic agent in the following timestep. We introduce an additional specialized inference rule for this purpose:

$$\frac{t : \texttt{true}(\text{“}X\text{”})}{t+1 : X}$$

This process is a method of unquoting. Concluding the example, the implication's conclusion would be `true`(“`on_fire(site1)`”), and then the formula `on_fire(site1)` is subsequently inferred, so that the agent believes to be true what it had previously heard to be true. As with any other active logic belief, this can ultimately be overturned or distrusted, in cases such as when a contradiction arises as a result of a reason to also believe $\neg X$.

3.4 Quasi-quotation

We have previously described the nesting mechanism of quotation terms added into active logic, and provided both a series of examples for quotation terms, as well as the principles of unification drawn from them. We now discuss examples that require quantifying into quotation, introduce the use of quasi-quotation as a means to do so, develop the desired unification behavior for quasi-quotation, and ultimately generalize this behavior to a full unification algorithm for quotation terms and quasi-quotation. This will be essential for representing an active-logic agent's reasoning about beliefs — both the agent's own as well as those of others.

3.4.1 A need for quantifying-in

Quotation terms as introduced thus far have a significant limitation, exemplified by formula 3.3: the same variable that appears within a quotation term cannot also appear outside of that quotation with the same meaning. Hence, quoted variables in their present form are locked inside of quotation, where these variables are mentioned only. This inaccessibility of variables inside quotation terms prevents their binding and substitution for anything other than other quoted variables, by the definition of unification for quotation terms presented in section 3.3.2. This is useful in certain kinds of formulas, particularly formulas that express universal quantification inside of quotes (e.g. formula 3.6). Yet in other cases, it is desirable for particular variables within quotation terms to have their corresponding quantifiers appear beyond the quotation marks. This is once again the problem of quantifying into quotation, as was introduced in section 2.3.2.5. What is desired is a pattern of how certain variables, if indicated for this purpose, may quantify into quotation terms and interpolate within the quotation marks. Such a pattern would allow: 1) the same variable to appear both inside and outside of quotation, 2) these variables to make bindings with other sorts of terms, and 3) substitution into the context of quotation. These abilities clearly exceed what can be done with quotation terms that have no ability to quantify-in. Next, we illustrate this need more vividly with examples using introspection.[3]

Introspective reasoning without binding or substituting quoted variables is sufficient for certain categories of formulas, which includes many formulas that express general knowledge, definitions, or properties. For example, an active-logic agent might check whether all cephalopods are mollusks with a literal such as `pos_int`(“`cephalopod`(`X`) $\rightarrow$ `mollusk`(`X`)”). The existing form of quotation terms suffices for this sort of formula; beliefs about a particular instance of a cephalopod being a mollusk are not unifiable with the quotation, while the general form of the belief about cephalopods as mollusks will unify and satisfy introspection.

However, other kinds of formulas given as arguments for introspective reasoning should succeed without making exact matches, and introspection should also function in an intuitive way for these more particular instances. For example, ideally

[3] Introspection as implemented in ALMA 1.0 with the formula interpretation of functions discussed in section 3.2 has the ability to perform these sorts of bindings and substitution. However, the theoretical and practical limitations identified in that section remain. This approach also has other limits on the expressiveness of formulas. A formula such as 3.8 cannot be represented in the same way without quotation; a formula interpretation of `heard`(`on_fire`(`X`)) is unifiable with both `heard`(`on_fire`(`site1`)) as well as `heard`(`on_fire`(`Y`)), while quotation enables unifying with only the latter (as formula 3.8 does), or the former (as allowed by quantifying-in).

an active-logic agent could check the current time via a formula in its knowledge base much like `pos_int`(“`now(T)`”), and be able to connect the variable `T` to the present integer time value. Yet, presently this succeeds only if exactly `now(T)` is a standalone formula appearing in the knowledge base — a formula that indicates `now(T)` is believed for all values of `T`, which is very unlikely to be a useful formula believed by a reasonable agent.[4] Thus, for positive introspection, an argument will not always be intended as an almost-verbatim match for a knowledge base formula (that is, a verbatim match to a formula except for a change of variables). Therefore instead of matching as in the cephalopod example, two common non-verbatim patterns of introspection use quotation terms differently:

1) Introspection on an argument containing a variable, which is intended to *acquire a binding* as part of unification with another belief. Hence in this pattern, the introspection argument is a formula expected to unify with a more specific knowledge base formula that it subsumes. For example, consider a knowledge base with the formulas `tired(tweety)`, `afternoon(50)`, and the following formula (expressing that a tired bird can nap only when the current point in time is during the afternoon):

$$\texttt{tired(tweety)} \land \texttt{pos\_int}(\text{“}\texttt{now(T)}\text{”}) \land \texttt{afternoon(T)} \rightarrow \texttt{can\_nap(tweety)} \quad (3.14)$$

In this example, the desired result when positive introspection searches for a formula unifiable with `now(T)` is that introspection succeed by binding `T`, by unifying with `now(50)`. The intended most general unifier with the knowledge base described then consists of `T` bound to the constant `50`, which can be substituted into the final

[4] Or more precisely, the present semantics for quoted variables would have introspection on “`now(T)`” only be unifiable with a `now` literal containing a variable argument. This might be another variable than `T` itself; but regardless this does not change the fact that `now` with a variable argument would not be reasonably attributed to an agent.

premise `afternoon`(`T`), which matches the atomic `afternoon` formula in the KB. Obtaining the desired conclusion therefore hinges on finding a binding for `T` in the quotation term *during* introspection.

2) Introspection with an argument containing a variable which has previously acquired a binding, so that *the substitution of this variable for its binding* must be made before introspection is performed. For example, consider a knowledge base containing both the premise `bird`(`tweety`), and the following formula:

$$\texttt{bird}(\texttt{X}) \wedge \texttt{neg\_int}(\text{``}\neg\texttt{flies}(\texttt{X})\text{''}) \rightarrow \texttt{flies}(\texttt{X}) \tag{3.15}$$

This is the same example from Purang that was introduced in section 3.2, only rewritten with the syntax of quotation terms. In evaluating the formula from left to right, the atomic formula `bird`(`tweety`) first unifies with `bird`(`X`), resulting in the binding of `X` to `tweety`. When the first premise has been satisfied, it's appropriate that `tweety` can be substituted into `X` in `flies`(`X`) in the quotation term. Then, the truth of the introspection predicate would be based not on a formula about flying that contains a variable, but on the present belief of `flies`(`tweety`). To obtain a conclusion specific to `tweety`, it is necessary to have some means to substitute into the quotation term a more specific bird than an unknown flier `X`, *before* checking the knowledge base.

While quotation terms as developed so far are presently insufficient for these cases, such patterns of reasoning will be enabled by extending quotation terms to allow the option of quantifying-in, as we now develop.

3.4.2 Formalism

Quantifying-in provides a theoretical mechanism for variables to bind and substitute into the context of quotation terms. We use the term *quasi-quotation* for referring to a syntactic device allowing quantifying-in, whereby variables may be selectively indicated to "escape" the quotation terms that they are contained in. Quasi-quotation was originally introduced by Quine [77], although Quine used the term quasi-quotation slightly differently, unlike our usage for a mark that essentially unquotes particular variables.

We emphasize that this feature may be selectively applied to only some variables. Not all variables should escape from inside of a quotation term; instead it is often useful for universal quantification to remain inside of quotation marks, as is the case in formula 3.8:

$$\texttt{heard}(\text{``}\texttt{on\_fire(X)}\text{''}) \land \texttt{confirmed}(\texttt{widespread\_fire}) \rightarrow \texttt{do}(\texttt{evacuate}) \tag{3.8 revisited}$$

But frequently, quantifying-in via quasi-quotation should enable targeted variables in a formula to escape their containing quotation term during inference and unification, so that these variables might be substituted for or bound. Consider again the example of formula 3.3, which in the absence of quantifying-in was incapable of expressing its originally intended meaning of having the same X throughout the formula:

$$\texttt{heard}(\text{``}\texttt{on\_fire(X)}\text{''}) \land \texttt{smoke}(\texttt{X}) \rightarrow \texttt{on\_fire}(\texttt{X}) \tag{3.3 revisited}$$

Quasi-quoting the first use of X, within the quotation term, now gives a way to achieve the desired meaning: that all appearances of `X` are occurrences of the same variable. We now introduce a formalism for quasi-quotation, as built by extending the existing quotation terms.

Then, quantifying-in is delineated with a special quasi-quotation character (the quasi-quotation mark) appearing before a variable to modify it. The backtick (`` ` ``) is used for this mark, inspired by Lisp.[5] *A variable* specifically has been mentioned as the target of a quasi-quotation mark; it is unclear what other varieties of terms, such as functional expressions, would mean if quasi-quoted when inside of quotation terms, and presently we put these other cases aside.[6] Extending the grammar to include a variable preceded by a quasi-quotation mark can be achieved by adding another nonterminal symbol to the production rule that derives a variable, as introduced in section 2.2.10. With the additional nonterminal, that rule becomes:

$\langle \mathit{Variable} \rangle \rightarrow$ `` ` ``$\langle \mathit{Variable} \rangle$

| [A-Z][a-zA-Z0-9_]*

Hence, the presence of a quasi-quotation mark only before a variable is enforced.

[5] Quasi-quotation in Lisp (and related dialects such as Scheme) is an alternative form of quotation in the language, which serves as a template whereby parameters may be inserted into the template by selectively unquoting expressions via another operator [1, 88]. Some other languages, such as Haskell, also support the feature due to inspiration from the Lisp family [48]. In the programming language domain, this feature is used for program-generating programs, to more easily embed domain-specific languages by specifying syntax and constructing fragments of programs [7].

[6] Further afield from terms, the notion of quasi-quoting other grammatical active logic constructs such as predicates, operators, or whole well-formed formulas presently does not seem justifiable. There do not appear to be reasonable intuitive semantics to describe these and how they would behave in inference, nor an example use case, in contrast to the many examples readily found for variables. One could imagine a case where it may be useful to quantify variable names over quoted predicates (e.g. looking for binding a variable to a predicate name in representing whether John believes a predicate `P` where `P(tweety)` is true). But this would suggest a second-order logic, that is beyond the scope of first-order active logic in this work.

As a first example of quasi-quotation, we finally express formula 3.3 to properly reflect the meaning that had been originally intended:

$$\texttt{heard}(\text{“}\texttt{on\_fire(`X)}\text{”}) \wedge \texttt{smoke(X)} \rightarrow \texttt{on\_fire(X)} \tag{3.16}$$

This formula now successfully captures the meaning that, for any site reported as on fire, and where smoke is seen there, then it is concluded that this particular site is on fire. If the quantifier for X is made explicit, this example, in contrast to formula 3.5, is rendered as:

$$\forall \texttt{x}[\texttt{heard}(\text{“}\texttt{on\_fire(`x)}\text{”}) \wedge \texttt{smoke(x)} \rightarrow \texttt{on\_fire(x)}] \tag{3.17}$$

The production for a variable now has two alternatives, where application of the first alternative yields a quasi-quotation mark and the $Variable$ nonterminal once again. If this first alternative is selected repeatedly, then arbitrarily many quasi-quotation marks may appear in succession. The purpose of allowing multiple quasi-quotation marks is to quantify into nested quotation terms in different ways. For an instance of a variable located within N nested pairs of quotation marks, we refer to this variable instance as appearing in N *levels of quotation*. With multiple quasi-quotation marks, the means of quantifying-in is generalized so that each additional level of quotation added before a particular variable allows the variable to escape one further pair of quotation marks, from the innermost out. We also define the *effective level of quotation*: for a variable in N levels of quotation and M quasi-quotation marks prepended, effective level of quotation is $N-M$. An instance of a variable in N levels of quotation with N quasi-quotation marks is referred to as *fully-escaping*; any fully-escaping variable has a quantifier with scope over its entire formula. A variable that has no quasi-quotation marks and appears outside of any quotation terms also trivially fits this definition.[7] This scope is intuitive given that the effective level of quotation for a fully-escaping variable is zero. Hence, such a variable behaves quite similarly to a variable outside of any quotation. If instead of being able to repeatedly quasi-quote as described above, a single quasi-quotation mark were to immediately allow a variable to fully escape from all nested levels of quotation, this would not have sufficient expressiveness for desired uses of quasi-quotation. It is often convenient for a variable within several nested quotation terms to be only partially quasi-quoted (i.e., not fully-escaping, and having an effective level of quotation greater than zero). A variable in a partially quasi-quoted case, with effective level of quotation greater than zero, is also referred to as a *non-escaping variable*; non-escaping variables may be partly quasi-quoted, or not quasi-quoted at all.

For a variable in several levels of quotation, we return to example 3.6 to demonstrate how multiple quasi-quotation marks affect the quantifiers of a formula:

$$\texttt{bel(alice, “heard(“on\_fire(X)”)”)} \tag{3.6 revisited}$$

We note again that here Alice is another agent, for which the entire quoted wff is a belief of the active-logic agent, regarding a belief that this agent attributes to Alice. The variable `X` appears within two levels of quotation, and thus two new formulas result from quasi-quoting it up to two times:

$$\texttt{bel(alice, “heard(“on\_fire(`X)”)”)} \tag{3.18}$$

$$\texttt{bel(alice, “heard(“on\_fire(``X)”)”)} \tag{3.19}$$

[7] In some cases when we wish to discuss only fully-escaping variables with a nonzero number of quasi-quotation marks, these variables are referred to as fully-escaping quasi-quoted variables.

By default for a quoted variable, the universal quantifier when made explicit occurs just within the nearest quotation marks, as in formula 3.7:

$$\texttt{bel}(\texttt{alice}, \text{“}\texttt{heard}(\text{“}\forall\texttt{x}[\texttt{on\_fire}(\texttt{x})]\text{”})\text{”}) \tag{3.7 revisited}$$

Formula 3.6 expresses that the belief is attributed to Alice that she heard one utterance express that all locations are on fire. Using one quasi-quotation mark (where $\texttt{X}$ is at an effective quotation level of one), the outer quotation term formula is able to quantify into the inner quotation term formula, and the universal quantifier when made explicit is within the outermost quotation marks:

$$\texttt{bel}(\texttt{alice}, \text{“}\forall\texttt{x}[\texttt{heard}(\text{“}\texttt{on\_fire}(\text{`}\texttt{x})\text{”}])\text{”}) \tag{3.20}$$

Formula 3.18 expresses that one belief is attributed to Alice, which is a universally quantified formula expressing that all locations have been heard as on fire. Using two quasi-quotation marks (where $\texttt{X}$ is fully-escaping, with effective quotation level of zero), $\texttt{X}$ is a fully-escaping variable, the outermost formula quantifies into both quotation terms' usage of $\texttt{X}$, and hence its universal quantifier has scope over the entire formula:

$$\forall\texttt{x}[\texttt{bel}(\texttt{alice}, \text{“}\texttt{heard}(\text{“}\texttt{on\_fire}(\text{``}\texttt{x})\text{”})\text{”}]) \tag{3.21}$$

Formula 3.19 expresses that, for each possible value of $\texttt{x}$, a belief is attributed to Alice, which is that she heard an utterance indicating that particular value of $\texttt{x}$ is on fire. In contrast to one belief of Alice's being modeled in example 3.18, in the case of this formula, Alice is modeled as holding an infinite set of such beliefs. This latest example moves beyond matters exclusively about the formalism of the new quasi-quotation structure; Alice is modeled as having heard a set of formulas, of which each ought to contain a particular value appearing in place of the quasi-quoted variable. Considering the contents of such formulas involves issues of binding variables and making substitutions. These issues must be resolved, and most crucially the questions of which bindings would be permissible for quasi-quoted variables, and how substitution should occur in these contexts, must be answered to make use of quasi-quotation.

One final important point about formalism is that, for active logic with quotation and quasi-quotation, the number of quasi-quotation marks that can apply to a variable is up to and including the number of quotation marks enclosing that variable (i.e., a variable may be quasi-quoted only up to being fully-escaping). A formula with excessive quasi-quotation marks before any variable on this basis is not a well-formed formula. Thus the variable `X` in formula 3.6 within two levels of quotation may be quasi-quoted up to two times. This limit based on the number of enclosing quotes provides one source of context-sensitivity for the new language. Since context-sensitivity is not pervasive in the language with quotation and quasi-quotation, presentation of grammars will continue to use context-free rules, although the quasi-quotation limit still applies.

3.4.3 Quasi-quotation reasoning

When previously defining the formalism of quasi-quotation, we distinguished two main categories of variables: *fully-escaping* variables that are not affected by any enclosing quotation marks (which also trivially includes variables outside of quotation altogether), and non-escaping variables which remain partly affected by quotation.

We now move beyond the formalism of quasi-quotation, and proceed to develop reasoning for the language with quasi-quotation, by extending the behavior of unification. Due to the richer structure of a quotation term with embedded quasi-quotation marks inside, there are more possibilities for how this might behave than in the more straightforward case of unifying a quotation term without quasi-quotation. Distinctions between types of variables lead to significantly different outcomes for how they might bind and unify. We will next guide the development of intuitive semantics for quasi-quotation using the same example-driven approach as detailed in section 3.3.2 for the basic kind of quotation terms. Within context of a proof step, we consider all the different cases of unifying two terms, in which a variable within a term can be non-escaping or fully-escaping. Our approach is achieved by illustrating each case, and extracting principles of how to handle them. We then ultimately develop a full unification algorithm, and show it behaves according to our developed principles.

3.4.3.1 Binding a non-escaping quasi-quoted variable

Quasi-quoted variables that are not fully-escaping from quotation in many respects still resemble quoted variables that have no quasi-quotation. Their respective implicit universal quantifiers remain within at least one pair of quotation marks, as exemplified by formula 3.20:

$$\texttt{bel(alice, “}\forall\texttt{x[heard(“on\_fire(\`{}x)”])”)} \qquad \text{(3.20 revisited)}$$

Intuitions much the same as those determining bindings for quoted variables (described in section 3.3.2.4) thus apply here as well. That is, these non-escaping variables are treated much like mentioned variables, and so should not bind to any term that is more specific. Rejecting bindings in these cases once again excludes such a variable binding to a constant, function, or quotation term. Just as a quoted variable without quasi-quotes may bind to another quoted variable without quasi-quotes as described in section 3.3.2, a quasi-quoted variable that isn't fully-escaping should unify with another variable at the *same effective level* of quotation. One consequence of this is that these variables cannot unify with any fully-escaping variable, which necessarily have an effective level, zero, different from that of any non-escaping variable. Contrasting formulas 3.6 and 3.18 provides an example as to why quoted variables with different effective quotation levels should not unify:

$$\texttt{bel}(\texttt{alice}, \text{“}\texttt{heard}(\text{“}\texttt{on\_fire}(\texttt{X})\text{”})\text{”}) \tag{3.6 revisited}$$

$$\texttt{bel}(\texttt{alice}, \text{“}\texttt{heard}(\text{“}\texttt{on\_fire}(\grave{}\texttt{X})\text{”})\text{”}) \tag{3.18 revisited}$$

As was made clear in rendering their quantifiers explicitly with formulas 3.7 and 3.20, respectively, formula 3.6 indicates an agent modeling that Alice believes she heard (the same agent-centric **heard** as before) a single utterance that indicated all locations as on fire, while formula 3.18 indicates an agent modeling that Alice believes she heard all possible utterances that each indicated a location as on fire:

$$\texttt{bel}(\texttt{alice}, \text{“}\texttt{heard}(\text{“}\forall\texttt{x}[\texttt{on\_fire}(\texttt{x})]\text{”})\text{”}) \tag{3.7 revisited}$$

$$\texttt{bel}(\texttt{alice}, \text{“}\forall\texttt{x}[\texttt{heard}(\text{“}\texttt{on\_fire}(\grave{}\texttt{x})\text{”}])\text{”}) \tag{3.20 revisited}$$

The two quotation terms, which an active-logic agent attributes to Alice, are clearly not equivalent. We also can describe this fact in syntactic terms, by noting that there is a difference in the structure of the two quoted `heard` formulas: they have their implicit universal quantifiers in different positions, due to their different effective quotation levels. Hence, we generalize this point into the principle that: *when attempting to make an initial binding, two variables with a difference in effective quotation level cannot bind.*[8]

A key restriction placed on unifying quoted variables in the earlier section was that each occurrence of a quoted variable in one quotation terms must correspond to exactly one variable in the other term. For quasi-quoted variables that are still non-escaping, the same motivation for this requirement exists. The resulting behavior for non-escaping quasi-quoted variables is that a binding is permitted between variables at the same effective level of quotation, as long as this condition on variable matching consistency is followed. This directly generalizes the unification behavior of non-escaping variables without quasi-quotes. However, that result should not be surprising, since the intuitions come from the fact that these variables are non-escaping, whether or not there are quasi-quotation marks involved.

We subsequently develop the unification behavior of fully-escaping variables, first from more trivial examples. In doing so, we also address in much greater detail other reasons why a non-escaping variable should not unify with a fully-escaping variable.

[8] We later will return to differences of effective quotation level when going beyond basic cases of bindings and discussing the recursive unification of bound variables, where there is reason for slightly different behavior.

3.4.3.2 Binding a fully-escaping quasi-quoted variable

We next consider situations when a fully-escaping quasi-quoted variable should create a binding with other expressions. We return to a running motivational example for quasi-quotation to do so:

$$\texttt{heard}(\text{“}\texttt{on\_fire}(\texttt{`X})\text{”}) \wedge \texttt{smoke}(\texttt{X}) \rightarrow \texttt{on\_fire}(\texttt{X}) \qquad \text{(3.16 revisited)}$$

Once again, this formula now successfully expresses that hearing about *any particular* site reported as on fire, and seeing smoke visible, warrants the conclusion that the particular site is on fire. Intuitively, for purposes of unification the fully-escaping `X` should be treated as a blank space in the quoted formula containing it, which can be filled with a wider range of substitutions than a non-escaping variable might.[9]

Prior to discussing how to make bindings, we make a point of how we conceive of bindings. We emphasize that each successful binding is structured as a binding specifically of `X` to a term; and is not a binding of `` `X `` to a term. The purpose of a quasi-quotation mark is to single out this variable for protection from quotation, and thus to change what kinds of terms it can unify with and what kinds of bindings it can make; the associated binding itself exists between simply the variable `X` and another term. In fact, this principle about a binding's structure holds not just for fully-escaping variables, but for all bindings made in cases of quasi-quotation, including whether or not the bound variable in question is fully-escaping. Examples below will illustrate this.

Now, we describe examples that drive the kinds of bindings that formula 3.16

[9] And, of course, other occurrences of the variable `X` throughout the formula will also be filled with the same substitution as made within the quotation term.

should support for `X`.

3.4.3.2.1 Binding to a ground term

Bindings for `X` in formula 3.16 should be permissible with a variety of terms that could indicate locations heard as on fire, so that bound variables can ultimately enable conclusions about the locations to which they were bound. For example, terms that could indicate locations, such as `site1`, `hallway(interior)`, `stairs(level2, level3)`, and `sign("high_voltage")` are all options which should be valid choices for what could become bound to `X`. More generally, it is clear that fully-escaping `X` may be bound to any ground term comprised of constants, function, and ground quotation terms.

An additional pair of examples further illustrates the importance of noting a binding as between merely a variable and a term, rather than between a quasi-quoted variable and a term. We first consider an active-logic agent's knowledge base containing the formula `electrical_hazard(fence)` and the following:

$$\texttt{electrical\_hazard(X)} \rightarrow \texttt{sign(``high\_voltage(`X)")} \tag{3.22}$$

Now, compare this with a separate KB containing `sign("high_voltage(fence)")` and the converse implication:

$$\texttt{sign(``high\_voltage(`X)")} \rightarrow \texttt{electrical\_hazard(X)} \tag{3.23}$$

In both cases, the desired binding is `X / fence`. It is an intuitive result that the same binding be obtained, regardless of if it originated first from a fully-escaping quasi-quoted variable, or from a variable that had no quasi-quotation marks.

3.4.3.2.2 Binding to a variable

Returning to accounting for the kinds of terms that might satisfy the first premise of formula 3.16 (`heard`(“`on_fire`(`` ` ``X)”)), the remaining terms to consider are variables, both quasi-quoted and non-quasi-quoted. However, at this point we have distinguished partly-escaping quasi-quoted variables from fully-escaping variables, and further described the reasons according to which fully-escaping variables unify in the same manner as regular variables inside quotation. In the running example, a formula with another fully-escaping variable in the same position in the formula as `X`, such as `heard`(“`on_fire`(`` ` ``Y)”), should be unifiable with the first premise of formula 3.16, yielding the binding `X / Y`. These two formulas have identical syntax (except for variable names), when including their respective quasi-quotation marks, which also means their implicit quantifiers each have a scope over the whole formula. As with the typical form of unification without quotation, any variable with a scope over the entire formula ought to be able to freely bind with and substitute for another variable of the same scope. Furthermore, in terms of formula meaning, the standalone literal above expresses that, for any location-value of `Y`, the agent currently reasoning has separately heard the formula that this location is on fire. Since every location was separately heard as being on fire, this gives license to believe that any location that is smoking is sufficient for it being on fire (i.e., concluding that `smoke`(Y) $\rightarrow$ `on_fire`(Y)). Both of these reasons thus support making bindings between two fully-escaping quasi-quoted variables.

In contrast, if there is a quoted formula with a non-escaping variable in the same position as `X`, such as `heard`(“`on_fire(Z)`”), there are reasons to reject this formula unifying with the premise `heard`(“`on_fire(`X)`”) of formula 3.16. The clearest argument against this binding is based on the syntax of the two `heard` literals. `X` escapes from quotation due to quasi-quotation, while `Z` is a non-escaping variable and so its quantifier lies within the nearest quotation marks. To allow a binding such as `X / Z` would lead to a substitution of `Z` in place of `X` in formula 3.16, and in doing so would place a new quantifier inside of the quotation term when `X` is replaced by non-escaping `Z`. As was established in section 3.3.2.4, a binding should not change a formula's structure by removing quantifiers, and a natural corollary from this example is that quantifiers should not be introduced within a quotation term outside of positions where a substitution is made. Rather, *a substitution in place of a quasi-quoted variable should only introduce new structures and quantifiers into the formula at positions where the variable had appeared.* Another problem is that permitting a fully-escaping variable binding to a non-escaping variable would also suggest a form of equivalence between variables with and without quasi-quotation, which is directly counter to the initial goal of quasi-quotation. Thus, a fully-escaping variable should not be bound to a non-escaping variable.

The motivations behind accepting bindings of a fully-escaping variable to another fully-escaping variable, and also rejecting bindings of a fully-escaping variable to a non-escaping variable, each additionally help guide further cases.

3.4.3.2.3 Binding to a non-ground term

We next consider whether to unify fully-escaping quasi-quoted variables with non-ground terms. A simpler question to resolve is whether a fully-escaping variable ought to unify with a term containing exclusively other fully-escaping variables. Similarly to the motivation for creating a binding directly between two fully-escaping variables, fully-escaping quasi-quoted variables should also further resemble regular variables outside of quotation by possessing the ability to bind with expressions that contain other variables of formula-level scope — just as variables outside of quotes are able to do. For variables appearing within quotation marks, we recall that only fully-escaping variables have such a scope. In these cases, fully-escaping variables may occur at the same level of quotation as the variable it would bind to; or may occur at a deeper level of quotation, if the term contains additional nested quotation terms. For our running example focusing on the premise `heard`(“`on_fire`(\``X`)”) of formula 3.16, a sample formula containing a variable at the same quotation level as `X` is `heard`(“`on_fire`(`hallway`(\``H`))”) (each hallway was separately heard as on fire), which yields a binding of `X` / `hallway`(\``H`). An example with a variable at a deeper level is `heard`(“`on_fire`(`sign`(“`high_voltage`(\`\``V`)”))”) (each sign indicating a location having high voltage was separately heard as on fire), which yields a binding of `X` / `sign`(“`high_voltage`(\`\``V`)”).

Whether a fully-escaping variable ought to unify with a term that *contains* non-escaping variables is more complicated. The most basic syntax that could be considered consists of a formula with a function of variable arguments, as in `heard`(“`on_fire`(`stairs`(`Lower`, `Upper`))”). Here, the problem persists that new quantifiers would be introduced (for `Lower` and `Upper`) inside of the quotation term if the fully-escaping `X` is bound to `stairs`(`Lower`, `Upper`). Consequently, this binding must be rejected. However, implicit quantifiers are positioned differently in terms containing variables in a deeper level of quotation without quasi-quotation. Consider `heard(“on_fire(sign(“high_voltage(V)”))”)`, for which `V`’s implicit quantifier lies within the innermost quotation term. Since `V` is quantified *within* the quotation term argument of `sign`, this overall formula has a different meaning: that the sign itself, that happens to indicate all locations as high voltage, was heard to be ablaze. A binding such as `X / sign(“high_voltage(V)”)` does not introduce a problematic new quantifier, since the sole quantifier for a variable in the `high_voltage` quotation term is contained in that term. Hence, this binding for `X` is permissible. We summarize this point as follows: *a fully-escaping variable binds to a term T containing non-escaping variables if all implicit quantifiers of these variables remain within T.*

Two further examples illustrate how this principle applies to terms containing partly-escaping variables, in addition to the scenarios involving non-escaping variables without quasi-quotes that were discussed above. The first of these is the literal `heard(“on_fire(sign(“high_voltage(`V)”))”)`. Here, the implicit quantifier for `V` appears just prior to the `on_fire` predicate symbol, so a prospective binding of `X / sign(“high_voltage(`V)”)` would violate the requirement that non-fully-escaping quantifiers remain in the bound term. Hence, this cannot unify with `heard(“on_fire(`X)”)`.

Conversely, consider this example containing a non-escaping variable `Agent`:

`heard(“on_fire(sign(“bel(manager, “review(`Agent, excellent)”)”))”)` (3.24)

Since `Agent` is quasi-quoted once, its implicit quantifier positioned just before the `bel` predicate symbol, inside the second quotation term. The example thus expresses that what was heard as on fire was a sign indicating that, for each agent, the manager believes that particular agent has received an excellent review. With three nested quotation terms, the formula expresses an uncommon thought, especially for its specificity of describing a burning sign. Despite this, the implicit quantifier for `Agent` crucially does not appear beyond the argument to `on_fire`. This allows a binding of `X / sign(“bel(manager, “review(`Agent, excellent)”)”)`. Although this superficially resembles the rejected binding of `X / sign(“high_voltage(`V)”)` due to containing a variable with one quasi-quotation mark, the presence of an additional enclosing pair of quotation marks makes this an allowable binding.

3.4.3.2.4 Binding to fully-escaping variables

Finally, we point out how a pair of fully-escaping quasi-quoted variables handle quite differently a pair of formulas which could not have unified with only non-escaping variables, due to the patterns of variables in each formula. Consider the following:

$$\texttt{bel(alice, “sees(`Agent, `Agent)”)} \tag{3.25}$$

$$\texttt{bel(alice, “sees(`Agent\_a, `Agent\_b)”)} \tag{3.26}$$

This pair has almost identical syntax to formulas 3.9 and 3.10, respectively, although a quasi-quotation mark is added before each variable, resulting in significantly different meanings due to the implicit quantifiers escaping beyond the quotation terms. In formulas 3.9 and 3.10, each of the formulas expresses that Alma models Alice as holding one belief that holds quantifiers about what others see. But, in formulas 3.25 and 3.26, since the quantifiers for variables are fully-escaping, each formula indicates that Alma models Alice as having all possible `sees` beliefs, each with particular agents as arguments. By the quasi-quotation unification principles developed so far, this pair can successfully unify and produce bindings. One possible set of bindings that could result is `Agent / Agent_a` and `Agent_b / Agent_a`. In contrast to the reason for rejecting an analogous binding between non-escaping variables, bindings that replace *fully-escaping variables* with more specialized patterns are permissible, as long as the constraints outlined above are followed. When unpacking the process of unification, the above pair is also a trivial recursive case: once the binding of `Agent / Agent_a` has been made, when comparing `Agent` to `Agent_b`, the term to which `Agent` is bound, `Agent_a`, must be recursively unified with `Agent_b`. Although the recursive case in this example is simple, more complicated examples of recursive unification with quotation terms require additional changes; we describe these in section 3.4.3.5.

3.4.3.3 Exhaustiveness of bindings

Section 3.4.3.1 addressed when a non-escaping variable might create a binding: this is permissible exclusively when matching with select cases of other non-escaping variables (i.e., with the same effective quotation level, and as long as there is consistency in variable matching), and a binding is rejected when the term it is compared to is of any other type — a function, a constant, a quotation term, a fully-escaping variable (explained in more detail in section 3.4.3.2.2), or a non-escaping variable violating the conditions above.

Likewise, section 3.4.3.2 addressed when a fully-escaping variable might create a binding: when matching with another fully-escaping variable, a constant, a function or a quotation term (as long as either of these types does not contain a variable with improperly placed implicit quantifiers). Conversely, a binding is rejected when comparing to a non-escaping variable, for a function or quotation term with improper quantifier placement (according to the principle identified above).

Taken together, these cases give an exhaustive accounting for the behavior during unification involving a variable compared against any kind of term, whether success or failure results. We can also provide an exhaustive accounting for unification behavior when comparing a pair of terms that are both non-variables: functions, constants, and quotation terms. With variables removed from consideration, unification is straightforward in that these terms simply successfully unify when their structures match, and otherwise fail to unify when there isn't a structural match. Hence without regard to variables, a constant unifies exclusively with an equal constant, a function unifies exclusively with another function of matching functor where each term recursively unifies, and a quotation term unifies exclusively with another quotation term containing matching predicates and where each predicate argument recursively unifies. This behavior is made most explicit in the full algorithm presented in section 3.4.4. Yet, in concluding the section that developed principles for binding, we here emphasize the importance of how the above principles achieve exhaustive coverage for terms, and so help make our ultimate unification algorithm precise.

Continuing on, we return to the trivial case of recursive unification that was invoked at the end of section 3.4.3.2.4. As a requisite to developing recursive unification, which is presented in section 3.4.3.5, we first develop the means of substitution based on bindings, which also require some extensions specific to quotation terms.

3.4.3.4 Substitution with quotation and quasi-quotation

Several of the bindings indicated above in section 3.4.3.2.3 stand out, by virtue of containing a term that has more quasi-quotation marks than enclosing quotes. For example, `X / hallway(`H)` has singly-quasi-quoted `H` without quotation marks appearing in the bound term itself, and `X / sign(“high_voltage(``V)”)` has a twice-quasi-quoted `V` with just one pair of enclosing quotes presented in the binding itself. These bindings are justified on the basis that `H` and `W` are fully-escaping variables in their particular formulas — yet, presently the bindings that contain these two variables do not reflect that fact. We therefore extend the information recorded in a binding accordingly, so that a binding on its own makes clear which variables are fully-escaping. A way of providing this context is to indicate as a third piece of information the level of quotation (which we recall is counted from the highest formula level down) at which a binding was made. Since multiple occurrences of a variable might lead to recursive unification of a bound variable, for additional clarity we attach to a binding *the level of quotation in which the bound variable occurs.*

For active logic with quotation, we define a binding as a tuple $\{V, L, T\}$, which consists of a bound variable V, which is within L levels of quotation, bound to a term T. This will also be presented as V `(`L`)` / T. Thus, the two bindings above are clarified by the new notation to be the expanded bindings of `X (1) / hallway(`H)` and `X (1) / sign("high_voltage(``V)")`. `H` in fact is within one level of quotation, confirming that one quasi-quotation mark makes it a fully-escaping variable. Likewise, the quotation term containing `W` is in fact within another, second, level, confirming that two quasi-quotation marks signify it as fully-escaping while remaining well-formed.

We use this new definition of a binding to develop the method of making substitutions for variables when quotation terms are involved. Methods of substitution as presented here extend the typical means of substituting variables for terms in a first-order system without quotation. As a result of the behavior being a direct extension, if a variable V became bound to a term T_0 outside of quotation terms (in the new notation, V `(0) /` T_0), any occurrence of V outside of quotation can trivially be replaced by T_0 in the usual way.

3.4.3.4.1 Substituting for a non-escaping variable

Substituting for variables within quotation marks that have no quasi-quotes, or are quasi-quoted but only still non-escaping, is also straightforward. In the first of these cases, if a variable within a quotation term N_0 is bound, then by the principles developed above, the term to which N_0 is bound is necessarily another non-escaping quoted variable N_1. It is thus a trivial substitution of N_0 for N_1, and moreover since neither variable escapes quotation all substitutions are made within the quotation term where they appear.

Likewise, if a non-escaping variable P_0 is bound, the bound term P_1 is necessarily a variable at the same effective quotation level.[10] The substitution of P_0 for P_1 remains trivial regardless of the number of quasi-quotation marks affecting each variable of this pair; the number of quasi-quotation marks prepended to P_0 will not change. After making a substitution, then P_1 will have the same number of quasi-quotation marks prepended. Once again, no substitution is made outside of the quotation terms containing the implicit quantifiers of P_0 and P_1, but rather all substitutions are localized within these quotation terms.

Bindings made for fully-escaping variables present some more challenging cases that require different treatment, in contrast to non-escaping variables. Some of the complications follow from the fact that fully-escaping variables might be substituted into contexts without any quotation. We now develop the desired substitution behavior in these cases, from initially basic examples.

3.4.3.4.2 Substituting for a fully-escaping variable

Prior to analyzing cases for the behavior of fully-escaping variables and their bindings, we first clarify a notational matter for quasi-quotation in a binding. As was previously described in section 3.4.3.2 and subsequent sub-sections, notationally when a variable V has already been bound, any quasi-quotation marks attached to V are not represented in the binding itself, even if V is fully-escaping and quasi-quoted. The primary reason, which we will elaborate on and provide examples for shortly, is that appearances of V with different amounts of quasi-quotation (or even none at all) should all be substituted appropriately with the given binding. Yet

[10] Per the principles in section 3.4.3.1, that a non-escaping variable may bind exclusively with another non-escaping variable, and only when the pair has equal effective level.

information about the original context where V was bound is not lost either, and can be retrieved from the values in the binding. For a binding V (*L*) / T, if L is greater than zero, the amount of quasi-quotation for V can always be determined. If T is a function, quotation term, or constant, then V must be fully-escaping to have bound it to this term, and so the variable is quasi-quoted L times. If T is another variable, then the number of quasi-quotes prepended to T is the same number as for V, since only variables of equal numbers of quasi-quotation marks are unifiable (whether fully-escaping or not).

Although there is justification for our binding notation omitting the bound variable's quasi-quotes, we also recognize that substituting a fully-escaping variable for its binding should replace any quasi-quotation marks the variable has. Consider two literals already established as unifiable: `heard`(“`on_fire(`X)`”) and `heard`(“`on_fire(site1)`”). According to the binding constraints developed, the binding `X (1) / site1` is obtained. Substitution should be able to transform the first literal into the second. Yet this is only possible if, instead of replacing (all occurrences of) `X` with `site1`, all occurrences of `X` *and any number of preceding quasi-quotation marks* are replaced.

From this insight, we develop the general pattern of appropriately substituting a fully-escaping variable for an arbitrary term. A fully-escaping variable X may appear throughout an arbitrary formula in varying levels of quotation, where each appearance has an amount of quasi-quotation marks corresponding to the quotation level (to keep it at an effective level of zero). We first address substitutions *into the same quotation level* — where all occurrences of X would have the same quotation level — before addressing substitution across quotation levels.

3.4.3.4.3 Substituting into the same quotation level

We consider the problem of how to substitute based on an arbitrary binding V `(`L`)` `/` T, in which V is a fully-escaping quasi-quoted variable and all occurrences of V appear within L levels of quotation. The easiest case occurs when T is a ground term. Here, as described above, to make the formula containing V fully match the formula containing occurrences of T, both V *and* its preceding quasi-quotation marks must be replaced by T. After substituting out both the variable and quasi-quotation marks, what had formerly been the "blank spaces" in the form of fully-escaping variable instances will have disappeared — which is sensible, since a suitable replacement has been made for the content of the term T. For example, consider substitution with the following formula based on a binding `Agent (1) / bob`:

`bel(alice, "sees(``Agent, ``Agent)")` (3.25 revisited)

The result of substituting `Agent` for `bob` is `bel(alice, "sees(bob, bob)")`, based on the binding, which leaves a formula containing no quasi-quotation marks. Replacement of a fully-escaping variable with any ground term also follows the same pattern of substitution.

A different category of examples consists of bindings that are between two fully-escaping variables instead of ground terms. We can formalize these as being of the form V_0 `(`L`)` `/` T: a fully-escaping V_0 is at L levels of quotation, bound to a term T that contains a fully-escaping variable V_1. Hence, V_1 necessarily must have L quasi-quotation marks prepended, to match the quotation level. With a binding between two fully-escaping variables, when substituting one for the other, we again apply the method of replacing all quasi-quotation marks prepended to a fully-escaping variable. After substituting, there will then remain L quasi-quotation marks: the number preceding V_1, as part of T. More concretely, when `heard`(“`on_fire(`X)`”) has unified with `heard`(“`on_fire(`Y)`”), the associated binding is `X (1) / `Y`. Then, `` `X `` is substituted for `` `Y ``, maintaining the same number of quasi-quotation marks.[11] The number of quasi-quotes ultimately remains constant after substituting, which is the desired result.

We finally consider a more generalized binding of V `(`L`) /` T in which V is fully-escaping and T is an arbitrary term. However, we first add the constraint that additionally, all occurrences of V in the overall formula are within L levels of quotation. This section has so far not addressed cases in which T is a function or a quotation term and contains variables which are further inside (whether these are fully-escaping or non-escaping). In these cases where T is either a quotation term or a function term, any fully-escaping variables that are interior to the structure of T clearly have all of their quasi-quotation marks interior to its structure as well, since these immediately precede their respective variables. Likewise, by the constraint detailed in 3.4.3.2.3, any implicit quantifiers for non-escaping variables must be contained inside of T for a binding to succeed. Taken together, these facts lead to a simple process of substituting V and its quasi-quotation marks for T, with uniform behavior no matter which forms of variables are within T.

[11] Or, alternatively there is a binding `Y (1) / `X`, for which a similar substitution is easily made.

3.4.3.4.4 Substituting into a different quotation level

We now broaden the forms of substitution under consideration, to also address substituting variables into a different level of enclosing quotation than the level in which a binding was originally created. This is only possible for fully-escaping variables, as these are the only kind of variables for which different instances of the same variable can appear throughout a formula across different levels of quotation. We formalize this kind of substitution as follows: there must exist a binding V (L_0) / T, and in the formula where substitutions are being made there must also exist at least one other occurrence of V at another quotation level L_1, where L_1 is not equal to L_0. Thus, the problem to address is how to make an appropriate substitution of V at level L_1.

A simple situation in which this problem arises is when the binding in question has been created by first unifying an occurrence of V which is outside of quotation ($L_0 = 0$), and where V additionally reappears elsewhere in the same formula inside at least one quotation term (i.e., $L_1 > 0$). The following formula demonstrates this by reversing the running examples of smoke and fire: here, Alma tells an agent near a smoking location about the location being on fire:

$$\texttt{smoke(X)} \land \texttt{nearby(Agent, X)} \rightarrow \texttt{tell(Agent, "on\_fire(`X)")} \tag{3.27}$$

We suppose that other formulas in Alma's knowledge base have unified to satisfy the premises, in which `smoke(X)` unified with `smoke(sign("high_voltage(W)")` (i.e., that the sign indicating high voltage in all applicable locations is itself smoking), and the agent nearby was satisfied by `nearby(alice, sign("high_voltage(W)")` (i.e., Alice is near this sign). A binding of `X (0) / sign(“high_voltage(W)”)` will thus have been created (and hence, `X` fills the role of V, and `sign(“high_voltage(W)”)` the role of T, in this example). In this case, this binding was made outside of quotation, but is used to substitute for `` `X`` within quotation. Applying the substitution rules developed above, the conclusion is `tell(Agent, “on_fire(sign(“high_voltage(W)”))”)`. This conclusion now contains two quotation terms, inside which `W` is at a deeper level of quotation than the level that this variable first appeared at in the `smoke` literal. We see that once again, any non-escaping variables in the term T will not be a problem when substituting, for the reason that their quantifiers are further inside of T.

However, in contrast with the previous paragraph, the case of *a fully-escaping variable* appearing inside of T, when substituting across different quotation levels, acts differently from the examples (discussed in section 3.4.3.4.3) that substitute into the same level. Consider when formula 3.27 has its `smoke` premise satisfied by unification with the literal `smoke(hallway(H))` (i.e., expressing that smoke is found in all hallways), creating a binding of `X (0) / sign(hallway(H))`. If the `smoke` literal and formula 3.27 were used as arguments to resolution, a conclusion that naively substitutes the term in the binding in place of both `X` and the quasi-quoted `X` would yield the following formula:

$$\texttt{nearby(Agent,hallway(H))} \rightarrow \texttt{tell(Agent, “on\_fire(hallway(H))”)} \qquad (3.28)$$

In the `smoke` literal, `H` began as a variable with an implicit quantifier of formula-level scope, due to its appearance outside of any quotation. Its first occurrence in formula 3.28 retains such a scope, with the implicit quantifier placed before `nearby`. Yet the second occurrence of H in formula 3.28 is a non-escaping variable due to how substitution occurred and its placement inside one level of quotation, which means it has a distinct meaning from the first occurrence. That is, the second H has an implicit quantifier with a scope over the on_fire formula. Alma is then unable to use formula 3.28 to make an appropriate conclusion about what to tell an agent near a particular hallway, due to a lack of quasi-quotation of H in the conclusion. This problem reveals that hallway(H) cannot be substituted into both locations exactly as it is. Rather, when the binding term T contains a variable E with formula-level scope (i.e., E being fully-escaping, which includes E appearing outside of quotes), when substituting into one level of quotation *the substituted term T must have any fully-escaping variable such as E be altered.* In the specific case of formula 3.28, the second occurrence of H that ends up within one level of quotation should be altered to have a single quasi-quotation mark prepended. Thus, the substitution should be done as if the binding were the following: X (1) / sign(hallway(`H)) — and thus, instead of formula 3.28, the following corrected formula is concluded:

$$\texttt{nearby}(\texttt{Agent}, \texttt{hallway}(\texttt{H})) \rightarrow \texttt{tell}(\texttt{Agent}, \text{“}\texttt{on\_fire}(\texttt{hallway}(\texttt{`H}))\text{”}) \tag{3.29}$$

This preserves the intended meaning of the formula: that the agent will be told about the fire for any particular instance of a hallway nearby.

Beyond this example of inserting one quasi-quotation mark to keep the variable H fully-escaping when the term where it appears inside substitutes into one level of quotation, the broader scenario occurs when making a substitution based on a binding V (L_0) / T, in which T contains other fully-escaping variables. In this general case, the context of T must be changed, by modifying the number of quasi-quotation marks preceding its variables that are fully-escaping from at least L_0 levels of quotation. We refer to this change, which alters fully-escaping variables within a term, as *adjusting the term's context.* Whenever an occurrence of V that is instead at L_1 levels of quotation is replaced by T, if $L_0 \neq L_1$, each fully-escaping variable in the substituted copy of T must have its number of quasi-quotation marks changed to remain fully-escaping in T's new context (after substitution) of L_1 levels of quotation. The procedure **Adjust-Context**, utilized as a helper by the complete algorithm **Full-Quasi-Quotation-Unification** detailed below in section 3.4.4, modifies a copy of term T in this manner:[12]

Algorithm 1 Adjust-Context

```
 1: function ADJUST-CONTEXT($T, L_0, L_1$)
 2:     if VARIABLE?($T$) and $T$.quasiquotes = $L_0$ then
 3:         $T$.quasiquotes = $L_1$
 4:         return $T$
 5:     else if FUNCTION?($T$) then
 6:         for $t_i$ in $T$.terms do
 7:             $t_i$ = ADJUST-CONTEXT($t_i, L_0, L_1$)
 8:         end for
 9:         return $T$
10:     else if QUOTE?($T$) then
11:         return ADJUST-CLAUSE-CONTEXT($T,\ L_0 + 1,\ L_1 + 1$)
12:     end if
13: end function
14:
15: function ADJUST-CLAUSE-CONTEXT($C, L_0, L_1$)
16:     for lit in $C$.pos_literals do
17:         for $t$ in lit.terms do
18:             $t$ = ADJUST-CONTEXT($t, L_0, L_1$)
19:         end for
20:     end for
21:     for lit in $C$.neg_literals do
```

[12] Note that this algorithm, along with the algorithms that follow in the remainder of this chapter, does not distinguish a separate case for terms that are constants; for purposes of simplifying the algorithms, a constant is considered to be a function with an empty list of terms and a functor comprised of the constant string. This leads to correct treatment of constants in all cases.

```
22:         for $t$ in lit.terms do
23:             $t$ = Adjust-Context($t, L_0, L_1$)
24:         end for
25:     end for
26:     return $C$
27: end function
```

Thus, if a fully-escaping variable V appears at L_0 levels of quotation within a particular copy of T, **Adjust-Context**(T, L_0, L_1) will modify V to have L_1 preceding quasi-quotation marks, represented as the value of the field `quasiquotes`. If V appears at the deeper level of quotation $L_0 + x$ within T, then it will be given $L_1 + x$ quasi-quotation marks instead, since the procedure recursively descends into further levels of quotation before altering the amount of quasi-quotation marks. Notably, the changes from the procedure also include cases where L_1 is zero, in which case quasi-quotation marks are dropped entirely from a given instance of a variable. We illustrate this principle in action with further examples, both for adjusting the context to increase and decrease quasi-quotation.

As the first example, we show how a fully-escaping variable within a binding term T might also be adjusted to have additional quasi-quotation marks when T is substituted into further levels of quotation. We consider once again formula 3.27:

$$\texttt{smoke(X)} \land \texttt{nearby(Agent, X)} \rightarrow \texttt{tell(Agent, “on\_fire(`X)”)} \qquad \text{(3.27 revisited)}$$

Consider a case in which the first premise of the formula is satisfied by the literal `smoke(sign(“high_voltage(`W)”))` (i.e., that every sign expressing a high-voltage location is smoking). This produces the binding `X (0) / sign(“high_voltage(`W)”)`, which is to be substituted in place of the occurrence of `` `X `` in one level of quotation (using the variables defined above, $L_0 = 0$ and $L_i = 1$, respectively). Thus, the context is adjusted so that `W`, which is fully-escaping from its enclosing $L_0 + 1$ levels of quotation in the binding term, is ultimately preceded by $L_i + 1 = 2$ quasi-quotation marks after substitution in the implication conclusion. This yields the following as the conclusion following from resolution (in which the **smoke** premise has resolved out):

$$\texttt{nearby}(\texttt{Agent}, \texttt{sign}(\text{“}\texttt{high\_voltage}(\texttt{`W})\text{”})) \rightarrow$$
$$\texttt{tell}(\texttt{Agent}, \text{“}\texttt{on\_fire}(\texttt{sign}(\text{“}\texttt{high\_voltage}(\texttt{``W})\text{”}))\text{”}) \qquad (3.30)$$

In this formula, `W` now correctly is fully-escaping from the quotation terms of the conclusion literal, as intended.

Roughly inverting the example and corresponding context change, consider inference making use of formula 3.16:

$$\texttt{heard}(\text{“}\texttt{on\_fire}(\texttt{`X})\text{”}) \wedge \texttt{smoke}(\texttt{X}) \rightarrow \texttt{on\_fire}(\texttt{X}) \qquad (3.16 \text{ revisited})$$

Let its first premise be satisfied by $\texttt{heard}(\text{“}\texttt{on_fire}(\texttt{sign}(\text{“}\texttt{high_voltage}(\texttt{``W})\text{”}))\text{”})$ (i.e., that there has separately been heard, for each sign indicating high-voltage, that this sign is on fire). This produces the binding `X (1) / sign(“high_voltage(``W)”)` (i.e., $L_0 = 1$), to be substituted in place of occurrences of `X` outside of quotation (i.e., $L_1 = 0$). Ultimately, the intended level for `W` is within just the one quotation term argument of **sign** (hence, $L_0 + 1$), and so the context is adjusted so that `W` is preceded by only $L_1 + 1 = 1$ quasi-quotation mark. Then, the following is inferred after the **heard** literal resolves out of formula 3.16:

$$\texttt{smoke}(\texttt{sign}(\text{“}\texttt{high\_voltage}(\texttt{`W})\text{”})) \rightarrow \texttt{on\_fire}(\texttt{sign}(\text{“}\texttt{high\_voltage}(\texttt{`W})\text{”})) \qquad (3.31)$$

This formula expresses the desired result, in which a particular location `W` for a sign indicating high-voltage is fully-escaping in each occurrence. If its context had not been modified when substituting, there would have been one too many quasi-quotes before `W`, and the result would not have been a wff due to this excess.

As a final example, again for inferences using formula 3.16, we also consider a new literal `heard(“on_fire(stairs(`Lower, `Upper))”)` (i.e., there has separately been heard that, for each particular set of stairs, from a lower floor to an upper floor, these stairs are on fire). This leads to a binding of `X (1) / stairs(`Lower, `Upper)`, once again substituted in place of the second and third occurrences of `X` in formula 3.16, which are outside of quotation terms. Yet in this case, there is no quotation term in the binding either. Thus, `Lower` and `Upper` are adjusted to have their quasi-quotation marks removed, and the overall formula inferred after resolution is as follows:

$$\texttt{smoke(stairs(Lower,Upper))} \rightarrow \texttt{on\_fire(stairs(Lower,Upper))} \tag{3.32}$$

Just as quasi-quotation marks were introduced earlier into the conclusion of formula 3.29 to preserve the intended meaning, here the meaning is preserved by dropping quasi-quotation marks.

Hence, **Adjust-Context** for fully-escaping variables during certain substitutions enables the correct handling of these variables when substituting into different levels of quotation. The context would be altered regardless of whether the variables in question begin inside or outside of quotation, and whether amount of quasi-quotation is increased or decreased. This ability also establishes the treatment of the remaining category of cases for substitutions that involve quotation terms.

Given a term T at quotation level L, and binding list θ, the following substitution procedure results in the appropriate replacement of *any* bound variable within T for its bindings, including adjusting context when different levels of quotation require doing so:

Algorithm 2 Quasi-Quotation-Substitute

```
function SUBST-TERM($\theta, T, L$)
    if VARIABLE?($T$) and $\{T\ (L_t)\ /\ Term\} \in \theta$ then
        if $T$.quasiquotes $= L$ and $L_t \neq L$ then
            $T$ = ADJUST-CONTEXT($Term, L_t, L$)
        else
            $T = Term$
        end if
    else if FUNCTION?($T$) then
        SUBST-FUNC($\theta, T, L$)
    else if QUOTE?($T$) then
        SUBST-CLAUSE($\theta, T, L+1$)
    end if
end function

function SUBST-FUNC($\theta, F, L$)
    for $t$ in $F$.terms do
        SUBST-TERM($\theta, t, L$)
    end for
end function

function SUBST-CLAUSE($\theta, C, L$)
    for $lit$ in $C$.pos_literals do
        SUBST-FUNC($\theta, lit, L$)
    end for
    for $lit$ in $C$.neg_literals do
        SUBST-FUNC($\theta, lit, L$)
    end for
end function
```

Next, we define the means of recursive unification with quotation terms, which also utilizes the feature of adjusting a variable's quasi-quotation context. This recursive unification in turn leads to the general unification algorithm.

3.4.3.5 Recursive quasi-quotation unification

As occurs during the process of standard unification of first-order terms without quotation, when a variable V is to be compared with a term T_0 for unification, the growing list of associated bindings must first be checked to determine if there already exists a binding V (L_1) / T_1. When this is the case, instead of unification proceeding with the bound variable V, then T_1 should instead be (recursively) unified with T_0. However, there is now the added dimension of quotation level to consider: the level of quotation, L_0, in which T_0 appears can impact the recursive case. If $L_0 \neq L_1$, proper recursive comparison of T_0 to T_1 requires altering one of the terms via **Adjust-Context**. We illustrate this need developing the following example.

Consider the following formula, expressing (in Alma's KB) that if information has been heard, and yet another agent is considered to lack beliefs of having heard that information, then that knowledge can be told to the agent (allowing Alma to be helpful in providing knowledge):

$$\texttt{heard(X)} \land \neg\texttt{bel(Agent, “heard(`X)”)} \rightarrow \texttt{tell(Agent, X)} \tag{3.33}$$

Additionally, suppose that there are two further beliefs in the knowledge base: `heard(“instructions(`Number, `Content)”)` (i.e., that each numbered instruction step was separately heard) and ¬`bel(bob, “heard(“instructions(``N, ``Step)”)”)` (i.e., that Alma models Bob as not believing having heard these steps). The variables `Number`, `Content`, `N`, and `Step` are all quasi-quoted to be fully-escaping, indicating that each instruction has been communicated separately, and thus justifying the representation with these fully-escaping quasi-quoted variables. Suppose Alma is making inferences with these three wffs, using the inference rule of extended modus ponens, in which the two premises of formula 3.33 are unified in left-to-right order. Then, the first two bindings made are `X (0) /` "`instructions(`Number, `Content)`" and `Agent (0) / bob`.

Subsequently, inference with extended modus ponens must unify the occurrence of `` `X `` in the second premise of formula 3.33 with the inner quotation term "``instructions(``N, ``Step)``" (for convenience, referred to as T_0) within the negated `bel` literal ¬`bel(bob,` "`heard(`"``instructions(``N, ``Step)``"`)`"`)`. Naive recursive unification would attempt to unify T_0 with the term to which `X` is already bound: "`instructions(`Number, `Content)`" (or for convenience, T_1). The intended result is the formation of additional bindings (such as binding `Number` to `N`, and `Content` to `Step`) due to all four variables in these quotation terms being fully-escaping. However, the pursuit of adding these additional bindings leads to a problem: T_0 and T_1 appear in different levels of quotation, as T_0 appears within one level, and T_1 does not appear within any quotation term. The general problem of recursive unification of an arbitrary term T_0 with an arbitrary T_1 raises issues involving terms with two different quotation contexts, much like the problem that was previously handled for substitution in section 3.4.3.4.4. Indeed, in the same manner as the substitution issue was solved, we can once again utilize the procedure **Adjust-Context** to address the problem with recursive unification.

Unification should not be directly attempted between T_0 and T_1, due to the different levels of quotation that each term originally appeared within. Instead, one of the two terms must be selected to be modified by **Adjust-Context**, before proceeding to attempt to unify. When in the process of executing the unification algorithm, since T_1 is the term to which `X` is bound, there exists a copy of T_1 which has been instantiated in the binding list, and that can easily be replaced by the new term that the procedure returns. This is in contrast to terms like T_0 that appear as a term that is a fragment of a knowledge base formula rather than a copy that can be modified more freely. Hence, T_1 is selected as the term that will be modified; the existing binding `X (0) /` "`instructions(`Number, `Content)`" must be altered so that the binding term, T_1, ultimately has a context of one level of quotation (just like T_0). Computing **Adjust-Context**$(T_1, 0, 1)$ (i.e., **Adjust-Context**(T_1, L_1, L_0)) yields this desired term: "`instructions(``Number, ``Content)`". Then, the existing binding in the list is replaced by `X (1) /` "`instructions(``Number, ``Content)`". `X` is now bound to a term within one level of quotation, and this term if unified with $T_0$ ("`instructions(``N, ``Step)`") makes the desired new bindings of `Number (1) / `N` and `Content (1) / `Step` (or, `N (1) / `Number` and `Step (1) / `Content`) when unifying these two quotation terms. This concludes the extended example utilizing formula 3.33.

We generalize from this example to a principle for changing bindings to enable recursive quasi-quotation unification in arbitrary cases. Consider a formula where the fully-escaping variable V and the term T_0 appear in corresponding positions within quotation level L_0. Additionally, consider where V is already bound as V `(`L_1`) /` T_1, and $L_0 \neq L_1$. This binding must be replaced in the binding list by V `(`L_0`) /` **Adjust-Context**`(`T_1, L_1, L_0`)`, which is followed by recursive unification of

V and **Adjust-Context**(T_1, L_1, L_0). Recursive unification then proceeds without encountering issues of different contexts, due to the constraint that adjusting the binding's context precedes unification for different quotation levels.

3.4.4 A quotation unification algorithm

We now pull together the principles and results of our example-driven approach for bindings and substitution into a general unification algorithm, **Full-Quasi-Quotation-Unification**. This algorithm takes as arguments terms X and Y, each at quotation level L, returns a binding list θ when successful.

Algorithm 3 Full-Quasi-Quotation-Unification

```
1: function UNIFY($X, Y, L, \theta$)
2:     if VARIABLE?($X$) and VARIABLE?($Y$) and $X = Y$ then
3:         return $\theta$
4:     else if VARIABLE?($X$) then
5:         return UNIFY-VAR($X, Y, L, \theta$)
6:     else if VARIABLE?($Y$) then
7:         return UNIFY-VAR($Y, X, L, \theta$)
8:     else if FUNCTION?($X$) and FUNCTION?($Y$) then
9:         return UNIFY-FUNCTION($X, Y, L, \theta$)
10:    else if QUOTE?($X$) and QUOTE?($Y$) then
11:        return UNIFY-CLAUSE($X$.clause, $Y$.clause, $L + 1, \theta$)
12:    else
13:        return failure
14:    end if
15: end function
16:
17: function UNIFY-VAR($Var, X, L, \theta$)
18:    if VARIABLE?($X$) then
19:        if $Var$.quasiquotes $\neq$ $X$.quasiquotes or ($Var$.quasiquotes $< L$
             and $\neg$VARS-MATCH?($\theta$.matches, $L - Var$.quasiquotes, $Var, X$))
             then
20:            return failure
21:        end if
22:    else if $Var$.quasiquotes $\neq L$ or VARS-INSUFF-QUOTED?($X, L$) then
23:        return failure
```

```
    end if

    if $\{Var\ (L_{var})\ /\ Term\} \in \theta$ then
        if $Var$.quasiquotes $= L$ and $L_{var} \neq L$ then
            $Term$ = ADJUST-CONTEXT($Term, L_{var}, L$)
            $L_{var} = L$
        end if
        return UNIFY($Term, X, L_{var}, \theta$)
    else if VARIABLE?($X$) and $\{X\ (L_{var})\ /\ Term\} \in \theta$ then
        if $X$.quasiquotes $= L$ and $L_{var} \neq L$ then
            $Term$ = ADJUST-CONTEXT($Term, L_{var}, L$)
            $L_{var} = L$
        end if
        return UNIFY($Var, Term, L_{var}, \theta$)
    else if OCCURS-CHECK?($\theta, Var, X$) then
        return failure
    else
        return $\theta \cup \{Var\ (L)\ /\ X\}$
    end if
end function

function UNIFY-FUNCTION($X, Y, L, \theta$)
    if $X$.term_count $\neq$ $Y$.term_count or $X$.functor $\neq$ $Y$.functor then
        return failure
    end if
    for $x_t, y_t$ in $X$.terms, $Y$.terms do
        $\theta$ = UNIFY($x_t, y_t, L, \theta$)
        if $\theta$ = failure then
            return failure
        end if
    end for
    return $\theta$
end function

function UNIFY-CLAUSE($Var, X, L, \theta$)
    if $X$.pos_count $\neq$ $Y$.pos_count or $X$.neg_count $\neq$ $Y$.neg_count then
        return failure
    end if
    for $x_l, y_l$ in $X$.pos_literals, $Y$.pos_literals do
        $\theta$ = UNIFY-FUNCTION($x_l, y_l, L, \theta$)
        if $\theta$ = failure then
            return failure
        end if
    end for
    for $x_l, y_l$ in $X$.neg_literals, $Y$.neg_literals do
```

```
            $\theta$ = Unify-Function($x_l, y_l, L, \theta$)
            if $\theta$ = failure then
                return failure
            end if
        end for
        return $\theta$
    end function
```

Full-Quasi-Quotation-Unification begins by the top-level procedure **Unify** comparing the types of the argument terms to invoke the corresponding procedures; this is much like the typical form of unification except for the additional inclusion of a quotation level argument, and an added case for X and Y as quotation terms. **Unify-Var**, in which the variable Var is compared against an arbitrary term X, filters out several cases which should not unify based on the principles developed in earlier sections.

The first set of these failure conditions are checked when X is another variable. If Var has a number of quasi-quotation marks different from X, then the two variables are at different quotation levels and there is reason for failure: as described in both section 3.4.3.1 where one variable is non-escaping and section 3.4.3.2.2 where one variable is fully-escaping, the same level of quotation is required for a successful binding. If Var and X have the same number of quasi-quotation marks, which is less than their mutual quotation level L, then these non-escaping variables must the condition for matches between non-escaping variables specified in the end of section 3.3.2.4: each variable in one formula must correspond to exactly one variable in each corresponding location in the other formula. This is verified by the procedure **Vars-Match?**, which computes whether a matching of Var with X (at their effective quotation level) is consistent with the matching of non-escaping variable pairs encountered so far when unifying. That is, Var must have matched with X, or there must not exist a prior matching of either Var or X with any other variable.

When X is not a variable, unification will fail if Var is not a fully-escaping variable, as indicated in section 3.4.3.1. Moreover, if Var is fully-escaping, there is also a failure case for when the term X contains a non-escaping variable with an implicit quantifier that appears beyond the confines of the term X, per the discussion of section 3.4.3.2.3. The procedure call **Vars-Insuff-Quoted?**(X, L) computes whether this is the case by recursively descending into X, returning true when there exists a variable inside at effective quotation level E, such that $0 < E \leq L$. Hence, non-escaping variables within X that do not violate the above condition have either effective level of 0, or of a value greater than L.

Following checking the above set of conditions, which may lead to the variable Var failing to unify with X, the remainder of **Unify-Var** resembles the standard unification algorithm. If Var is bound, or if X is actually a variable and is bound, then the binding is retrieved, and recursive unification will use the binding term instead of the bound variable (after the new addition of modifying the term via **Adjust-Context**, as described in section 3.4.3.4.4). The **Occurs-Check?** procedure, with arguments of variable Var and term X, causes a unification failure when Var appears inside of X, which would otherwise have the potential to cause an infinite regress during substitution. As **Occurs-Check?** recursively checks the entire structure of X, in addition to functions this also includes descending into quotation terms and their formulas, and comparing against Var both non-escaping and fully-escaping variables that are found. Due to standardization of formula objects in ALMA, the occurs-check problem remains the same as for the regular first-order case, and does not become more complicated with quotation. An additional detail to note is that, both in the algorithm's occurs-check and in the first line of **Unify**, a comparison of two variables such as $X = Y$ is not made exclusively on the basis of the variable name string. Rather, variables are assumed to be standardized apart across different formulas and also across different levels of quotation within one formula, making use of some form of metadata annotation when necessary.[13] Hence, the notion of equality between two algorithm variables also reflects the idea that $X = Y$ if and only if X and Y are at the same effective level of quotation.

If there has not been a failure due to the occurs-check, a new binding of Var to X, at their L levels of quotation, is added into the binding list. As a convenience which simplifies the eventual substitution process, upon adding a new binding, the binding list is processed so that each bound variable occurring in the term of a binding will be substituted for their own respective binding terms.

In the remaining functions, **Unify-Function** and **Unify-Clause** continue comparing the structures of their argument pair, processing each term for a function, and each function for the positive and negative literals of a quoted clause. Ultimately, the binding list is returned following successful unification.

As we have seen, in each case where the algorithm goes beyond the standard unification algorithm, the rationale for doing so is traced back to a principle previ-

[13] In the reasoner implementation presented in chapter 4, this is achieved in the form of unique IDs attached to each distinct variable, across formulas and quotation levels.

ously developed. Hence, the algorithm's behavior is governed by these principles.

Chapter 4: The ALMA 2.0 reasoner

4.1 Introduction

The ALMA 2.0 reasoner has been developed as part of the contributions of the present work to implement automated inference for active logic. This reasoner constitutes a research contribution in providing an extensible platform for the extensions of the logic developed here. It additionally provides a basis for being able to be incorporated into agents which may base their reasoning upon active logic.

ALMA 2.0 is the successor to the prior active logic reasoner developed by Purang [74], ALMA 1.0. Deficiencies in ALMA 1.0 are detailed in section 4.8.1; as a result of these issues, the decision was made to use a more practical programming language than Prolog in the new reasoner. ALMA 2.0 (hereinafter, referred to as ALMA when it is unambiguous which version is referenced) is entirely implemented in C, and relies upon the external libraries of Micro Parser Combinators [34] for support of parsing the ALMA language, and the TommyDS [50] for high-performance hash tables and linked data structures in C. Aside from these libraries, the entirety of the present reasoner's code was developed for the current work.

The discussion of the reasoner in the remainder of this chapter focuses on mostly high-level implementation details of the reasoner. However, the full source code for ALMA, including instructions for compilation and execution, and a large body of runnable examples — including the formula files for all of the examples developed in chapter 5 — are all publicly available on GitHub: `https://github.com/mclumd/alma-2.0`. In particular, we note that the implementation of the algorithm **Full-Quasi-Quotation-Unification** and its helper procedures **Adjust-Context** and **Quasi-Quotation-Substitute** are available at `https://github.com/mclumd/alma-2.0/blob/master/alma_unify.c`.

We begin by introducing the basic properties of the reasoner, described below.

4.2 Grammar and parsing

The grammar of the ALMA language was designed to be expressive and not restrict the kinds of active logic formulas that could be accepted and parsed, whether these formulas are provided by users in axiom files or asserted via the add or observation commands (further described in section 4.3). The system internally takes steps to rewrite formulas after parsing, and transform them into a restructured normal form, as we subsequently describe. But, these steps do not restrict the grammar that an axiom writer may use to specify formulas; at this point the user is free to describe much more general formulas, rather than being restricted to narrower categories such as Horn Clauses or any other kind of other common logic program specification.

This grammar for ALMA formulas is based on the active logic grammar defined in section 2.2.10, and continues to follow conventions of the base active logic grammar's syntax. However, it has also been extended with the grammar rules for quotation terms as introduced in section 3.3.1, and for quasi-quotation as introduced in section 3.4.2. As such, it also features the usual first-order operators connecting atomic predicates that can contain arbitrary nesting of terms, and also does not feature explicit quantifiers but rather implicit universal quantifiers for each variable. The standard types of terms of active logic are included in the grammar (i.e., functional expressions, term variables, and constants), as well as terms we have introduced: quotation terms and the variable type which was expanded by the option to be preceded by a series of quasi-quotation marks. ALMA formulas are thus characterized by the following set of production rules:

$\langle \mathit{Alma} \rangle \rightarrow (\langle \mathit{Almaformula} \rangle$

$\quad | \quad \langle \mathit{Almacomment} \rangle)^*$

$\langle \mathit{Almacomment} \rangle \rightarrow$ %[^\n]* \n

$\langle \mathit{Almaformula} \rangle \rightarrow \langle \mathit{Sentence} \rangle$ `.`

$\langle \mathit{Sentence} \rangle \rightarrow \langle \mathit{Fformula} \rangle$

$\quad | \quad \langle \mathit{Bformula} \rangle$

$\quad | \quad \langle \mathit{Formula} \rangle$

$\langle \mathit{Formula} \rangle \rightarrow$ `and(` $\langle \mathit{Formula} \rangle$ `,` $\langle \mathit{Formula} \rangle$ `)`

$\quad | \quad$ `or(` $\langle \mathit{Formula} \rangle$ `,` $\langle \mathit{Formula} \rangle$ `)`

$\quad | \quad$ `if(` $\langle \mathit{Formula} \rangle$ `,` $\langle \mathit{Formula} \rangle$ `)`

$\quad | \quad$ `not(` $\langle \mathit{Formula} \rangle$ `)`

$\quad | \quad \langle \mathit{Literal} \rangle$

$\langle \mathit{Fformula} \rangle \rightarrow \texttt{fif(}\ \langle \mathit{Conjform} \rangle\ \texttt{,}\ \texttt{conclusion(}\ \langle \mathit{Fformconc} \rangle\ \texttt{))}$

$\langle \mathit{Fformconc} \rangle \rightarrow \texttt{and(}\ \langle \mathit{Fformconc} \rangle\ \texttt{,}\ \langle \mathit{Fformconc} \rangle\ \texttt{)}$

$\mid\ \langle \mathit{Fformula} \rangle$

$\mid\ \texttt{not(}\ \langle \mathit{Literal} \rangle\ \texttt{)}$

$\mid\ \langle \mathit{Literal} \rangle$

$\langle \mathit{Bformula} \rangle \rightarrow \texttt{bif(}\ \langle \mathit{Formula} \rangle\ \texttt{,}\ \langle \mathit{Formula} \rangle\ \texttt{)}$

$\langle \mathit{Conjform} \rangle \rightarrow \texttt{and(}\ \langle \mathit{Conjform} \rangle\ \texttt{,}\ \langle \mathit{Conjform} \rangle\ \texttt{)}$

$\mid\ \texttt{not(}\ \langle \mathit{Literal} \rangle\ \texttt{)}$

$\mid\ \langle \mathit{Literal} \rangle$

$\langle \mathit{Literal} \rangle \rightarrow \langle \mathit{Predname} \rangle\ \texttt{(}\ \langle \mathit{Listofterms} \rangle\ \texttt{)}$

$\mid\ \langle \mathit{Predname} \rangle$

$\langle \mathit{Listofterms} \rangle \rightarrow \langle \mathit{Term} \rangle\ (\texttt{,}\ \langle \mathit{Term} \rangle)^*$

$\langle \mathit{Term} \rangle \rightarrow \texttt{quote(}\ \langle \mathit{Formula} \rangle\ \texttt{)}$

$\mid\ \langle \mathit{Funcname} \rangle\ \texttt{(}\ \langle \mathit{Listofterms} \rangle\ \texttt{)}$

$\mid\ \langle \mathit{Variable} \rangle$

$\mid\ \langle \mathit{Constant} \rangle$

$\langle \mathit{Predname} \rangle \rightarrow \langle \mathit{Prologconst} \rangle$

$\langle \mathit{Constant} \rangle \rightarrow \langle \mathit{Prologconst} \rangle$

$\langle \mathit{Funcname} \rangle \rightarrow \langle \mathit{Prologconst} \rangle$

$\langle \mathit{Variable} \rangle \rightarrow \texttt{`}\langle \mathit{Variable} \rangle$

| [A-Z][a-zA-Z0-9_]*

$\langle \mathit{Prologconst} \rangle \rightarrow$ [a-z0-9_][a-zA-Z0-9_]*

Beyond the familiar operators, additional grammar rules of *Fformula*, *Fformconc*, and *Bformula* distinguish rules containing forward-if (fif) and backward-if (bif) kinds of implication. The use of forward-if and backward-if designates formulas that are used as part of reasoning with the forward-if inference rule based on extended modus ponens (described in section 4.5.2), and backward chaining inference based on resolution (described in section 4.5.3), respectively. Forward-if and backward-if operators are recognized and lead to a formula being given a metadata tag of fif or bif, which designates their different usage in inference. Except for maintaining the tag, these operators are treated the same as the usual form of implication for rewriting purposes. For the *Variable* production rule, the first alternative may be repeated enough to yield a number of quasi-quotation marks that is greater than the number of enclosing quotation marks. In this case, the parsed formula will be rejected by the system, since every quasi-quoted variable is verified to have an appropriate number of marks.

As was the case for the grammar given in section 2.2.10, all variables are assumed to be implicitly universally quantified. The scope of implicit quantifiers remains as defined for variables in section 3.4.2, in which the scope of a quantifier for a variable is based on its effective quotation level. Presently, a user of ALMA must accommodate the lack of an existential quantifier (e.g., by Skolemization); extending ALMA for genuine existential quantification may be developed as future work beyond the scope of the present research. However, there is a behavior in ALMA that functions very much like a negated existential quantifier in particular cases of interest: see discussion on negative introspection in section 4.5.2.2. The

ALMA variable notation itself is also inspired by Prolog, in which variables are distinguished from constants by a leading uppercase character. Each formula is terminated with a period.

A certain set of predicate symbols, such as `now`, `contra`, `distrusted`, and others discussed in sections 4.5.2.1 and 4.5.4, may be considered to be reserved symbols. In particular contexts (described in their respective later sections), these are either interpreted by ALMA to be the result of a specific inference rule or process, or they may have procedural effects, and hence these predicate symbols should not be reused for semantically unrelated purposes. Situations in which these predicates trigger procedures include when an instance of such a predicate is concluded as an atomic formula or is evaluated as a premise during inference.

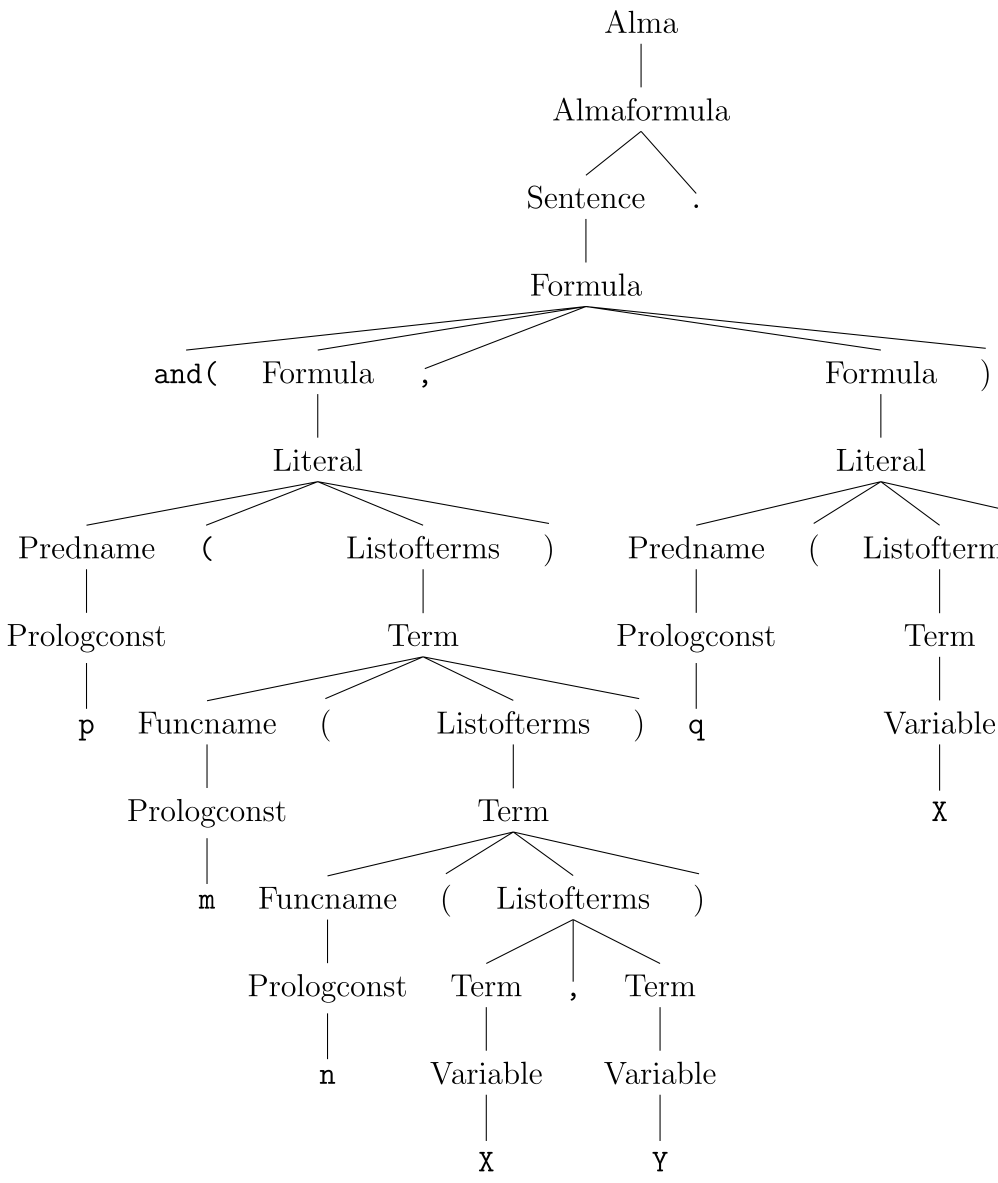
From grammatical ALMA input, the reasoner's parser constructs an abstract syntax tree. The highest-level nonterminal symbol, *Alma*, is used to define an entire knowledge base instantiation, consisting of a collection of individual formulas or comments (which are naturally discarded when parsing formulas). For example, consider the following active logic formula:

$$p(m(n(X, Y))) \wedge q(X) \tag{4.1}$$

This is specified in the ALMA grammar as:

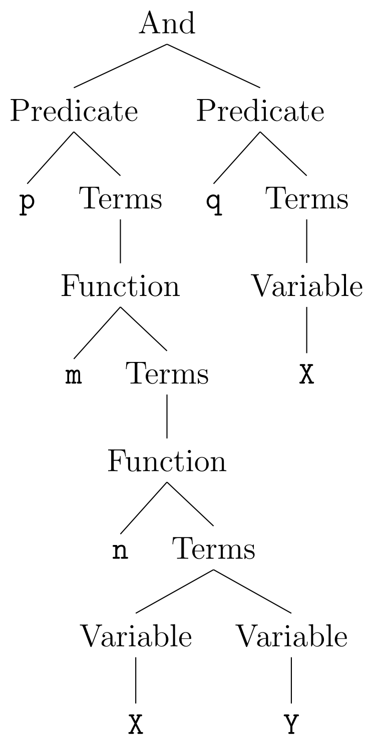
$$\texttt{and(p(m(n(X,Y))),q(X)).} \tag{4.2}$$

The parse tree below reflects the grammar's parse of this formula.



ALMA internally provides data structures for each operator and type of term. Thus, due to the existence of structures to which nonterminal symbols can be mapped, it is straightforward to translate from the abstract syntax tree to a tree composed of these objects. Additionally, any series of *Variable* productions that include one or more quasi-quotation marks will be consolidated into an integer count of the number of quasi-quotation marks that is attached to the variable they modify.

For formula 4.2, the following tree is obtained from this process:



Following translation into a tree that captures the recursive first-order formula structure, ALMA rewrites and standardizes each formula into an equivalent formula in conjunctive normal form (with additional metadata also generated in the process for tagging a forward-if or backward-if formula). This restructuring allows efficient representation in the knowledge base, which can be abstracted as a list of the clauses, where each clause contains two lists for the positive and negative literals, respectively (among other data), thereby avoiding the need to store explicit operators. Storage in clause form also has implications for the process of inference and the inference rules that ALMA supports (described in section 4.5), which rely on clauses. If a backward-if formula is split into several normal form clauses, each retains the bif tag. One exception to standardization into clause form is made for the conclusion of a forward-if formula, which may be a conjunction of arbitrary formulas due to applications of the *Fformconc* production rule; each conclusion conjunct is standardized into a clause, each of which is then stored in a list of conclusions.

Formulas are further rewritten to order literals lexicographically by predicate symbol, facilitating comparisons between formulas (such as for determining duplicate derivations). Forward-if formulas when participating in inference require their premises to be satisfied in left-right order. The structure for this category of formula tracks its premise ordering in metadata, in addition to its lexicographically sorted literals.

More must be said about the motivation for formula rewriting as applied to quotation terms, and their quoted wffs. As defined in the production rule *Term*, a quotation term within `quote` contains a formula as derived using the nonterminal *Formula*:

$\langle \textit{Term} \rangle \rightarrow$ `quote(` $\langle \textit{Formula} \rangle$ `)`

$\mid$ $\langle \textit{Funcname} \rangle$ `(` $\langle \textit{Listofterms} \rangle$ `)`

$\mid$ $\langle \textit{Variable} \rangle$

$\mid$ $\langle \textit{Constant} \rangle$

Detailed discussion in section 3.3.1 established the generality of the nested formula in a quotation term. Given that ALMA standardizes formulas into conjunctive normal form, a natural question arising is what degree of rewriting might be warranted on any formula held inside a quotation term.

One possible sense of a predicate containing quotation term(s) might be that the formula inside the quotation term is being referred to in a verbatim manner (see related discussion in 3.3.2.1). For instance, in a case expressing that the agent Alice believes the formula $\texttt{a} \rightarrow \neg\texttt{b}$, the objection may be raised that transcribing this into $\texttt{believes}(\texttt{alice}, \text{“}\neg\texttt{a} \vee \neg\texttt{b}\text{”})$ is a misrepresentation.[1] That is, Alice may hold a belief of precisely the implication, and not the disjunction — and hence, it could be misleading to indicate that the disjunctive version is Alice's belief. This type of interpretation would appear to preclude some kinds of standardization of formulas within quotation terms. However, if the ALMA reasoner is not utilizing predicates that are dependent on nested formulas as *verbatim* utterances, and indeed if the primary usage of quotation terms would be to refer instead to formulas directly in the knowledge base, then rewriting is on firmer ground. For employing quotation terms for internal reasoning about and with particular formulas that are the ALMA agent's knowledge base beliefs, rewriting a nested formula through use of the same procedures that have been thus far applied to a non-quoted formula poses no risk. The rewritten formula will have a structure matching against formulas in the KB (up to variable names) that have been rewritten in the same way.

Additionally, to further address this concern about rewriting, it is important to recall our emphasis on predicates that do not have their semantics based on exact syntax of quotation term formulas, as was described in section 3.3.2.1. In the later section 4.7, we do deal with applications in which beliefs of another agent are represented. However, all formulas used in applications of ALMA that express beliefs of other agents are not truly beliefs belonging literally to another distinct

[1] Again, note that the underlying *internal reasoning* idea is that an active logic agent Alma has this entire formula in its knowledge base.

agent such as Alice, but rather *ALMA's modeling* of beliefs that it attributes to these agents. Therefore, the ALMA system can rewrite attributed beliefs into standard form. Although, this means that certain equivalences are implicitly attributed to the modeled agents; e.g., Alice is modeled as if implicitly knowing the equivalence between an implication $\texttt{a} \rightarrow \neg\texttt{b}$ and its disjunctive form $\neg\texttt{a} \vee \neg\texttt{b}$.

Hence, rewriting of nested formulas is chosen on the basis that it is advantageous computationally to standardize quoted terms in ALMA. This enables recording, comparing, and unifying formulas much more easily due to obtaining a common structure between quotation terms, even if they originally had distinct parses. ALMA takes the approach of using its existing rewriting procedure with nested formulas that can be converted into a single conjunctive normal form clause, which is recursively applied to the furthest depth of quotation nesting. Presently, reasoning does not support the cases when a nested formula cannot be rewritten into a single clause, due to the complications it would bring to unifying and reasoning; this would be an area for future expansion of ALMA's functionality.

4.3 Control by commands

ALMA is initialized with an axiom set of initial formulas specified in a file, loaded into the reasoner through the steps for parsing and formula construction that are described above. The reasoner then can execute in one of two modes. In the first mode, it is controlled by a series of commands that drive its processing, consisting of: step, print, add, delete, observe, update, halt, and backward search.

Commands allow the running system to be manually controlled via the above set supplied by a human user, or it can be incorporated into a larger program in which another process inputs commands to the ALMA process at desired intervals. In the second mode, ALMA runs automatically in a loop of a step command followed by a print command, as long as it does not reach an idling state. The commands are as follows:

Step — Transitions from the current active logic timestep t to its subsequent successor timestep $t + 1$. As a result, in keeping with the principles of active logic, ALMA makes inferences across the timestep transition. The system's reasoning applies all available inference rules to the set of formulas in the KB at time t and makes the assertions and retractions to update the knowledge base upon the transition from t to $t + 1$. Further description of the mechanisms of high-level control when stepping, the inference algorithms applied, and additional details such as for formula inheritance, are all detailed in sections 4.4 and 4.5.

Print — Dependent on the command-line options, prints verbose output of either the entirety of the knowledge base at the current timestep, or otherwise prints output of the new or modified formulas in the knowledge base since the last print command. When printing, ALMA uses a more human-readable notation than is used internally to store formulas, and even to the prefix style of operators that the grammar requires to specify axioms and formulas. If possible, each clause is printed in sequent form: an implication with an antecedent of a conjunction of positive literals (using infix operator notation without parentheses), and a consequent of a disjunction of positive literals (also in infix notation). In practice, it has been observed that formulas provided by a user of the system very commonly are already in sequent notation or very close to it; hence many formulas which were altered in conversion to conjunctive normal form will be printed in their sequent form identically to, or very close to, their original definition by a user. Metadata such as a formula's integer index, lists by index of parents and children of the formula, formulas connected by equivalence links (described in section 4.7), and the pause status are also printed. Partitions of the knowledge base, such as are created for backward search (section 4.5.3) and agent belief models, are indicated separately from the rest of the knowledge base. Aspects of this design are discussed in the respective sections for these abilities.

Add — Attempts to assert new ALMA formulas. If the argument string is successfully parsed according to the grammar, a set of formulas is obtained. Each formula is added into the knowledge base and will appear in the current belief set after the next step command.

Del — Attempts to delete formulas from the knowledge base. If the argument string is successfully parsed according to the grammar, each resulting formula is compared to formulas in the belief set. Any match (up to variables) of a deletion argument formula against a knowledge base formula leads to the deletion of the latter. Deleting a formula also removes it from the parent set of all of its parents, the children set of all children, the equivalence links of formulas to which it is linked, and deletes any other metadata bookkeeping tracking the formula.

Obs — Attempts to assert its argument as an observation: an atomic formula to be recorded which will be modified to parameterize it with an extra argument appended to the predicate. This argument is a constant that reflects the integer time when the observation will appear in the knowledge base. Due to the timestep delay in a novel formula appearing in the belief set, this means that an observe command given at timestep t will have the temporal argument of $t + 1$. Hence, this temporal argument for a new observation always matches the present `now` when it first appears in the knowledge base. This command can model observation functionality, in which the ALMA system has information arriving from the world outside at particular points in time.

Update — Attempts to replace the formula referred to by its first argument with the second. Successfully updating results in the replacement retaining all parents, children, and other metadata of its predecessor. This is a powerful change that has the potential to be abused and render the derivation graph of ALMA to be nonsensical since an arbitrary formula may be the replacement; therefore it must be used carefully. Use cases for updating a formula are primarily when the new formula put in place is a modification of the original that has been replaced.

Bs — Initiates a backward search to answer the query of whether the argument is entailed by the set of knowledge base beliefs. A new partition of the knowledge base is created for the query-answering backward search, further described in section 4.5.3.

Halt — Terminates the reasoner process.

4.4 High-level control and prospect management

The core of processing in ALMA is from the high-level control of reasoning that initiates from a step command. In implementing automated reasoning following the active logic theory, ALMA exhaustively infers all new formulas that can be derived applying its inference rules from its present belief set as it makes one step from time t to $t + 1$. This expands the set of inferences in a breadth-first manner over timesteps. For efficiency, all progress toward new formula derivations must involve newly acquired formulas. The invariant that all possible inferences from older formulas were already inferred at prior timesteps inductively holds as reasoning advances.

ALMA achieves this through recording and managing *prospects*. A prospect is defined as an abstract structure for storing formulas and other information that is identified as potentially satisfying an inference rule. These are internal data structures for the reasoner tracking progress for inferences, and are not to be considered formulas or aspects of active logic itself. Prospects are further identified and categorized by the method of inference they involve, namely: resolution prospects, forward-if prospects, and backward search prospects. ALMA generates prospects at the end of a step based on the newly-inferred formulas that have just been derived, and these prospects will be evaluated during the beginning of the next step command.

A *resolution prospect* is defined as a tuple of a newly derived clause, an older formula that contains a complementary literal of the same predicate, and the positive and negative complementary literals from these two formulas. These are queued when generated, when evaluated this begins with attempting to unify the complementary literal pair. A *backward search prospect* is defined similarly: a tuple of a new formula in a backward search partition, a formula with a complementary literal of the same predicate from the main region of the knowledge base, and the positive and negative complementary literals from these two formulas. These are queued in the backward search partition, which the former formula is drawn from. The search for resolution and backward search prospects is currently implemented as an exhaustive search over the pairs of clauses with overlapping predicates. Each ALMA formula is indexed by the name and arity of each predicate that it contains, making this search for prospects an efficient one.

In comparison to resolution and backward search prospects, a *forward-if prospect* has a more complex structure. Such a prospect structure consists of: a tuple of a forward-if formula, a set of (partial) bindings of variables to terms, a set of clauses that have been unified with the premises, and a slot for the next candidate clause to check against a premise. Formulas satisfying a forward-if's premises may be acquired over a period of different timesteps, and so a forward-if prospect may be in different states of partial satisfaction at across timesteps. When a new literal formula is found to match the next unsatisfied premise of a partially-satisfied forward-if prospect, this formula is stored in the prospect's slot; hence the prospect is readied to be progressed further during the next step.

Thus, at the beginning of processing for a step command, ALMA progresses the queued prospects. Afterward, resolution and backward search prospects are removed, and any instances of partial forward-if prospects that have advanced their progress to completion are removed. Formulas that result from successful inference are stored in a buffer for new clauses. After the existing prospects have been exhausted, the new clauses are processed from this buffer and inserted into the knowledge base if they are not duplicates of an existing formula. If a formula is found to be a duplicate, the parents that derived it are attached to the version of the formula already in the knowledge base as a novel derivation. The initial batch of new clauses may include meta-knowledge clauses such as are described in section 4.6, which depending on the predicate may derive as children a secondary set of new (non-meta-knowledge) clauses that should enter the knowledge base at the same timestep. Hence, a second pass through the new clause queue is required. Meta-knowledge clauses may additionally derive as children clauses that will enter the KB at a subsequent timestep, which are placed in the queue after a delay of one timestep. After new formulas have been added into the KB, they are used to generate all possible new prospect instances. Collecting of the full set of new clauses in a buffer, and only later inserting them into the KB once the prospects have been exhausted, ensures that the relative input ordering of formulas within a single timestep will not affect which clauses are obtained. That is, the largest possible set of possible new inferences is collected into the buffer, and any new metacognitive formulas (described in subsequent sections) which might prevent or modify new inferences are applied after the point of collecting them.

Following a step command, ALMA determines its state as idling when no new clauses have been derived during the last step and timestep-delayed clauses remain.

Since resolution prospects, backward search prospects, and possible progressions for partial forward-if prospects are exhaustively handled prior to producing new clauses, checking for the idling state does not need to check whether prospects remain.

4.4.1 Inheritance

ALMA formulas by default have temporal persistence and are inherited into future timesteps, unless otherwise modified. This behavior matches with the active logic theory, in which formulas that are not distrusted or otherwise retracted persist — which also gives a practical answer to some frame problem issues. In ALMA, for formulas that are disinherited, the underlying data structure for a formula is not deleted, due to the existence of connections to other formula objects (e.g., its parents and children). Instead, a formula is indicated to be disinherited through metadata flags indicating the reason for not inheriting it: either the formula became distrusted, retired, or handled.

A formula may become *distrusted* as a result of inconsistency and contradiction detection as described in section 4.6.2, or due to the `distrust` procedure described in section 4.5.4. A formula may become *retired* as a result of a fluent formula that once held no longer applying as the world changes. ALMA presently has a weaker sense of fluents and makes little use of this flag. A formula may become *handled* if it is a literal expressing an inconsistent state which has been resolved, as also described in section 4.6.2. Additionally, a fourth flag exists for a *paused* status, which goes into effect for a formula when it has an upward equivalence link that has become distrusted (described in significantly more detail in section 4.7). Deletion of a formula using the command also targets a formula and prevents its inheritance in the strongest sense, when the formula is removed from the knowledge base entirely. However, deletion also removes the record that a formula was present in earlier timesteps, unlike the disinheriting flags.

If a disinheriting flag marks a formula, this formula will neither be used to generate inference prospects nor be accessible to procedures which check the current contents of the knowledge base. Hence, such a formula will participate in any further inference except in the limited cases of procedures that can consult formulas that are not current beliefs (for instance, the `ancestor` procedure can do so). Further information about the interactions of procedures with inheritance is discussed in section 4.5.2.1.

4.5 Methods of inference

ALMA employs several inference algorithms, described below. For each of them, as the inference graph expands from applying them, each formula records its parents and children as metadata attached to the clause data structure. This keeps the overall knowledge base structured as an adjacency-list.

4.5.1 Forward-chaining resolution

ALMA formulas that do not contain a forward-if-tagged or backward-if-tagged implication are used in forward-chaining resolution. Given the constraints on formula structure following the standardization into conjunctive normal form, any formula outside of those two categories is necessarily in clause form. As previously described in section 4.4, resolution prospects are generated in an exhaustive manner so that each complementary literal pair across clauses is identified. The following rule describes resolution in active logic:

$$\frac{t : P_1 \vee ... \vee P_n \vee A \quad Q_1 \vee ... \vee Q_m \vee \neg A}{t+1 : P_1 \vee ... \vee P_n \vee Q_1 \vee ... \vee Q_m}$$

The first step in resolution is attempting the syntactic unification of the complementary pair, A and $\neg A$. Note that this is not to suggest the positive and negative literals A and $\neg A$ must be identical; rather this is an idealization for literals T_i and $\neg T_j$ where T_i and T_j are unifiable terms.

Unification utilizes the algorithm **Full-Quasi-Quotation-Unification**, with the arguments of T_i, T_j, 0 (the level of quotation in which T_i and T_j are found), and an initially empty binding list. ALMA internally attaches distinct metadata values to each variable per formula, which ensures that variables in separate clauses given the same name in the grammar are still considered to be unique across clauses. Likewise, even if two variables with the same name appear in different effective levels of quotation within the same literal, these are standardized apart with different metadata identifiers. This is leveraged during unification — direct comparisons of variables make comparisons with the metadata, so that variables in different contexts as we have identified will not be considered to be identical.

When unification succeeds, the result falls into one of two cases. For resolution in which the two premise clauses are each a single literal (i.e., T_i and $\neg T_j$ directly), the conclusion resolvent is an empty clause. The two literal premises to resolution are thus unifiable formulas in direct contradiction, and it is this empty resolution that is used to detect contradictions. From here, ALMA applies its contradiction-handling faculties, which are described in section 4.6.2 as a specialized rule of inference. When the two resolution premises are not literals, their resolvent is nonempty. A new clause is constructed per the inference rule: as the disjunction of the remaining literals from the premises, for which the set of variables bound in the most general unifier are substituted for their bindings using the quotation substitution algorithm **Quasi-Quotation-Substitute**.

A new formula derived from one step of resolution records in its metadata that its parents are the two premises of resolution, and those parent clauses record the resolvent as a child in their metadata.

4.5.2 Forward-if

ALMA forward-if formulas allow inference of a literal conclusion from a conjunction of literals — a form of generalized modus ponens adapted to active logic. Hence, the following rule describes forward-if inference. As with the inference rule presented for resolution, a literal P_i appearing here as a standalone literal and also as a premise for the forward-if implication does not imply both of these must be precisely the same formula, but rather that a literal P_i must be unifiable with the respective implication premise.

$$\frac{t : P_1 \quad ... \quad P_n \quad P_1 \wedge ... \wedge P_n \xrightarrow{\mathtt{f}} C}{t+1 : C}$$

For example, the forward-if formula $\mathtt{a} \wedge \mathtt{b} \xrightarrow{\mathtt{f}} \mathtt{q}$ reasons in a forward manner to infer `q` when `a` and `b` are simultaneously present. One aspect not captured in the above notation is that of ordering in the antecedent: a particular premise of the forward-if is only checked for whether it is satisfied when its preceding premises have been determined to be satisfied. This is important due to the existence of a class of specialized ALMA predicates which execute procedures, which may take arguments that are dependent upon the satisfaction of earlier premises. These predicates are described further in section 4.5.2.1.

Unlike the incremental progress of resolution, which may derive many descendant formulas in a chain of reasoning toward a particular query, forward-if formulas do not derive intermediate result formulas before the conclusions. To illustrate this, consider again $\mathtt{a} \wedge \mathtt{b} \xrightarrow{\mathtt{f}} \mathtt{q}$ compared to the formula $\mathtt{a} \wedge \mathtt{b} \rightarrow \mathtt{q}$, which contains the typical implication instead of a forward-if. If `a` and `b` are known formulas, after one timestep resolution with $\mathtt{a} \wedge \mathtt{b} \rightarrow \mathtt{q}$ would derive resolvents $\mathtt{b} \rightarrow \mathtt{q}$ and $\mathtt{a} \rightarrow \mathtt{q}$ as intermediate results, and after two timesteps derive `q`. In contrast, as shown in the inference rule, a forward-if implication does not derive any descendant in the knowledge base until one timestep after all premises have been simultaneously satisfied. For the example of $\mathtt{a} \wedge \mathtt{b} \xrightarrow{\mathtt{f}} \mathtt{q}$, the only inference that would result is for `q`. If `a` and `b` enter the knowledge base at the same timestep, the forward-if implication will also be able to derive `q` after a single timestep, unlike the chain of resolution inferences which takes two timesteps.

Because intermediate conclusion formulas do not result, forward-if inference has more possible options as to when and how the space of satisfying premises is searched, as long as the final conclusions are appropriately derived and the order of premises is respected. ALMA progressively builds up prospects that track partially satisfied premises and bindings, and expands these prospects with the cases of successful unification at each new timestep. In this approach, many prospects are partially progressed and may never be satisfied to produce their conclusion, but consume space due to being stored in a state ready for further progression. When the search space branches from having multiple possibilities for unifying the next premise, the relevant partially-progressed forward-if prospects split into new copies, including copying their record of what has been unified. This trades extra space for gain in time, by avoiding repeating the unification calls of the first few shared premises, that have already been examined at earlier timesteps. ALMA thus uses a more space-intensive method, rather than an alternative that uses less space but might repeat for multiple formulas checking the prefixes of a set of satisfying premises.

4.5.2.1 Procedural premise predicates

ALMA reserves a set of predicates that we refer to as *procedural premise predicates*. These procedural predicates are distinguished from other predicates that, when present as premises of a forward-if formula, are satisfied only from unification with other beliefs. In contrast, the predicates described in this section in a forward-if formula associate their satisfaction with the result of executing a procedure, and execute for each instance present as a forward-if premise. These predicates are considered to have particular internal semantics enforced (as viewed by the system): their instances are considered to be true only in the circumstances defined by the success of their procedure, and false otherwise.

For example, `less_than` is one such procedural predicate. Consider the following formula:

$$\texttt{job(J)} \wedge \texttt{now(T)} \wedge \texttt{deadline(D)} \wedge \texttt{less\_than(T,D)} \xrightarrow{\texttt{f}} \texttt{canWorkOn(J)} \tag{4.3}$$

That is, if there is a job to be done, and the current time (indicated with `now`) is prior to the deadline, the agent can work on the job.[2] To infer the conclusion, there is no need for any atomic formula with `less_than` to be directly believed. Rather, whenever a positive `deadline` literal has unified with the premise `deadline(D)`, the procedure for `less_than` is subsequently executed, returning true or false, which becomes the truth value of the `less_than` premise. A procedural premise predicate is given the restriction that it may not begin a forward-if formula, as the first premise of the formula's antecedent.

Each procedure is executed at most once per procedural forward-if premise instance, given that: 1) a new combination of formulas (that is distinct from combinations used on this formula before) satisfies the premises preceding it, and 2) all binding constraints (described below) that are specified for the predicate instance

[2]Note the superficial resemblance of this rule to the deadline-planning active logic work done by Nirkhe, described in section 2.2.4.3.

are met. Execution occurs immediately after the last premise preceding in the ordering has been determined to be satisfied. Hence, the system does not devote an excessive amount of time to procedure execution by not repeating execution when the earlier premises are the same. In select cases, where a particular formula or ALMA application may require the repeated execution of the same procedure, there exists a workaround of the behavior that it executes once per premise set. Namely, an extra premise `now(T)` may be added prior to the procedural predicate. Due to the fact that a derivation of a novel `now` formula is obtained every timestep from the clock rule, every timestep will yield a different set of premises before the procedure, and it will continue to execute each timestep.

For each procedural premise predicate, an optional last argument may be used to specify any binding constraints. A function can be given in this last place, in which case each variable argument of that function must already have a binding (i.e., made when unifying the forward-if premises appearing before the procedural predicate), or else the procedure trivially returns false. If no binding constraints are necessary, the formula can be supplied only the necessary arguments. Binding constraints can be used to allow flexible instances of the procedural premise predicates that use many variable arguments. At the same time, the constraints ensure that the procedures are not underspecified when they are executed.

For example, returning to the `less_than` procedure in formula 4.3, it can be mandated that bindings have been made for the variables `T` and `D` in the first three premises before the procedure executes, by modifying the procedure into the following: `less_than(T, D, bound(T, D))`. The functor `bound` provides the function symbol of the final argument. If bindings were acquired in unifying the premises with `now` and `deadline`, the procedural predicate's truth value becomes the result of evaluating the inequality, for whether the expression to which `T` is bound is less than the expression to which `D` is bound. "Type errors" in which a non-numeric constant is bound to either argument variable will result in this evaluating to false.

Now, we turn to enumerating each procedure of this category. Implemented ALMA procedures generally fall into two categories: metacognitive procedures that check relations between formulas in the knowledge base (such as whether particular beliefs have a certain form, whether a formula was derived by a particular parent), and impure logic procedures (such as those that evaluate mathematical expressions which are inconvenient in pure logic). Each procedure is described below. When the arguments of a procedure are indicated to require a particular structure, such as a quotation term or a function, it also suffices for the argument to be a variable that has become bound to such a type of term, due to the fact that when copies of the arguments are retrieved, any variables will first be substituted for their bindings. We now present the full set of procedural premise predicates.

4.5.2.1.1 Family of procedural premise predicates

`Pos_int`, `pos_int_spec`, `pos_int_gen`, `pos_int_past`, `neg_int`, `neg_int_spec`, `neg_int_gen`, and `neg_int_past` — The family of introspection predicates, described fully in section 4.5.2.2.

`Acquired` — Takes as its first argument a quotation term representing a formula, much like introspection, and as its second argument a variable that cannot have yet been bound. If the match in the knowledge base is found, the variable that is the predicate's second argument is bound to the timestep number when the matching formula was first asserted into the knowledge base.

`Ancestor` — Takes three arguments: the first two represent formulas via quotation terms, and the third represents an integer time value. The second argument must be unifiable with a formula in the knowledge base, where this formula was either acquired at a timestep less than or equal to the time argument, or the formula was disinherited at the time argument. The first argument must then be unifiable with an ancestor of the second, based on any of the derivations in the different parent sets of that second argument. Only when these conditions are met does the procedure return true, retaining any bindings made. Otherwise, the result if the procedure is false, and no new bindings are made. The search through the space of ancestors is breadth-first based on distance from the root descendant.

`Non-ancestor` — The negation of `ancestor`, as a separate predicate due to procedural predicates not supporting the negation operator, as an implementation convenience. Hence, it takes the same three arguments as `ancestor`.

`Parent` — A procedure equivalent to `ancestor`, except that it only attempts to unify the first argument quotation term with the immediate parent of the second argument. Hence, this can determine if a formula is a direct parent rather than a more arbitrary or distance ancestor.

`Parents_defaults` — Takes as its argument a quotation term representing a formula. If a formula in the knowledge base is unifiable with this argument, this procedure inspects the parent sets of the formula to check whether it is considered to be a default conclusion. A formula is considered to be a default conclusion if every parent set that is currently a trusted derivation includes at least one default formula as a parent, where a default is as we introduce in section 5.2. Although more detail is provided in that section, in brief, the parents are checked to see whether they are a forward-if implication which also includes a `neg_int` premise that takes a positive literal of the `abnormal` predicate. The procedure returns true when this condition for the formula being a default conclusion is met, and false otherwise.

`Parent_non_default` — The negation of `parents_defaults`, as a separate predicate due to procedural predicates not supporting the negation operator. Hence, it takes the same argument as `parents_defaults`, and returns true when its argument is not a default conclusion, due to the existence of a trusted non-default parent set.

`Less_than` — Takes two arguments representing ground integer constants. If the arguments meet these criteria on their form, returns the result of evaluating the inequality on its two arguments, and false otherwise.

`Quote_cons` — Takes two arguments, the first of which must be to a bound variable, and the second of which must be to an unbound variable. When the first argument is bound to a function, the second argument will be bound to a newly constructed quotation term consisting of with a singleton clause inside of quotation marks, in which the predicate symbol is the same as the function symbol of the first argument, and each term is copied from the first argument.

`Not_equal` — Takes two arguments representing formulas via quotation terms. Given these two arguments, returns the result of comparing the two formulas for equality. Note that these two formulas are not unified, which tests a more general relation between the formulas. Rather, the two formulas must have exactly the same structure and differ only by a renaming of respective variables.

Much like in Prolog, if all arguments to a procedural predicate are ground (once substituting any bindings), execution of the procedure amounts to checking the truth of this ground relation. If one or more variable appears as an argument to a procedural predicate, this amounts to a query as to whether there exists a solution, which will be bound to give an answer to the query. Certain predicates, such as `less_than`, do not support execution with unbound variable arguments. For an example of a non-ground case, consider the following example:

$$\begin{aligned}&\mathtt{answer}(\mathtt{problem}(\mathtt{X}),\mathtt{Y}) \wedge \mathtt{acquired}(\text{“}\mathtt{answer}(\mathtt{problem}(\grave{}\mathtt{X})\text{”},\mathtt{Y}),\mathtt{T})\\&\xrightarrow{\mathtt{f}} \mathtt{solved\_at}(\mathtt{problem}(\mathtt{X}),\mathtt{T}) \qquad (4.4)\end{aligned}$$

That is, if a particular problem has been found to have an answer, and it's desired to know at what time the problem was solved, the unbound variable `T` that is the second argument of `acquired` will be bound to the answer to the query. In the above case, the formula to check, $\mathtt{answer}(\mathtt{problem}(\mathtt{X}),\mathtt{Y})$, has the same structure as the first premise, and so is guaranteed to unify with the same literal that has already satisfied the first premise. This ensures any conclusion made will have `T` replaced with a constant denoting a particular time value.

4.5.2.2 Introspection

Introspection has previously been introduced in section 2.3 as an important ability for an agent's internal processing, and chapter 3 illustrated how quotation is important for enabling a general ability for introspection in active logic. Now, we discuss specifics of how this is realized in ALMA through a series of procedural predicates.

We return to ALMA's set of procedures which implement both positive and negative introspection, consisting of: `pos_int`, `pos_int_spec`, `pos_int_gen`, `pos_int_past`, `neg_int`, `neg_int_spec`, `neg_int_gen`, and `neg_int_past`. Additionally, the predicate `acquired` operates very similarly to `pos_int`. Each of these predicates has special internal semantics based on a procedural lookup for a formula in the knowledge base. The group of `pos_int` predicates (as well as `acquired`) each evaluates to true when finding a matching unifiable formula in the knowledge base that is presently believed, although with some minor variations in which formulas satisfy each procedure. Similarly, the group of `neg_int` predicates each evaluates to true when *failing* to find such a formula, respectively.

As an example utilizing introspection, consider the following formula using the basic `neg_int` predicate:

$$\texttt{bird(X)} \land \texttt{neg\_int}(\text{“}\neg\texttt{flies(`X)}\text{”}) \xrightarrow{\texttt{f}} \texttt{flies(X)} \tag{4.5}$$

When the first premise has been unified and given `X` a binding such as the constant `tweety`, negative introspection evaluates to true if ¬`flies(tweety)` is not in the KB. If `X` had no binding when evaluating negative introspection, this would have acted very much like a negated existential quantifier, with scope directly around the predicate. That is, `neg_int` would have evaluated to true only if there did not exist any possible binding of `X` that would have caused the introspection argument to match a KB formula. For instance, since active logic tracks the present moment with the special predicate `now`, negative introspection on an atomic `now` formula via `neg_int`(“`now(`T)`”) always evaluates to false. Conversely, positive introspection on this formula, via `pos_int`(“`now(`T)`”), evaluates to true as long as `T` is either not bound when it executes or is already bound to a constant integer matching the current timestep value. If a successful unification makes new bindings when executing `pos_int`, these bindings are added to the partial bindings of the forward-if prospect.

Any introspection procedure's first argument, referred to as Q, must be a quoted formula (or a variable bound to one). However, the semantics of introspection involve comparison of Q against currently believed formulas in the knowledge base, which are clauses outside of quotation and not quotation terms. Introspection predicates use two distinct methods of constructing a quotation term out of a knowledge base formula. The first of these is used in cases for which it is desired that the query unify with a more specialized formula; the latter is used in cases that allow unifying with a formula that might be more generalized.

The first quotation term construction method is more trivial, and can be accomplished simply by placing the formula in an outer set of quotation marks, without any other modification. This is what is done by `pos_int_spec` and `neg_int_spec`. As a consequence of this method of outer quotation placement, a quotation term thus constructed from a KB formula (referred to as X) will have no fully-escaping variables. Given this, if Q successfully unifies with X, by the principles of quotation unification, any function, constant, or quote in Q must have unified with a matching term type in X. Additionally, any fully-escaping variable in Q must have unified with a function, constant, or quote in X.

From this, we can see that when the query Q contains fully-escaping variables, a successful unification is only possible when X is a formula that has a more constrained kind of term in positions where Q has fully-escaping variables. On the other hand, when Q contains no fully-escaping variables, then X has precisely the same level of generality. This method of introspection does not in any case permit Q to unify with a more generalized formula, since that would necessarily require a fully-escaping variable in the other formula. Hence, `pos_int_spec` and `neg_int_spec` implement a form of introspection in which the query formula Q is desired to unify with more specialized content from the knowledge base.

For the second way that a knowledge base formula can be converted into a quotation term to use with introspection, it is helpful to first consider a motivational example. Suppose that Alma believes (has in its KB) that if something has been heard, and what was heard it also currently believes, then it is in agreement with what was heard. Verifying the current belief of a formula would require using positive introspection, which we can see serves essentially as an instance of $\texttt{bel}(\texttt{alma}, \ldots)$, in looking up the agent's own beliefs:

$$\texttt{heard(X)} \land \texttt{pos\_int(X)} \xrightarrow{\texttt{f}} \texttt{agreement(X)} \tag{4.6}$$

Further, suppose that once again $\texttt{heard}(\text{``}\texttt{on\_fire(site1)}\text{''})$ is present in the knowledge base, as well as `on_fire`(`E`). After unifying the `heard` literals and binding `X`, Alma will attempt to determine whether the knowledge base currently includes `on_fire`(`site1`), via the procedure `pos_int`(“`on_fire`(`site1`)”). While precisely `on_fire`(`site1`) is not a current belief, the generalized formula `on_fire`(`E`) indicates that everything is on fire, and trivially implies that `site1` must be as well. However, universal instantiation is not an inference rule that the ALMA system is presently equipped with, for the reason that such a rule would rapidly swamp the knowledge base with an excess of ground formulas implied by its universally quantified beliefs. This means that `on_fire`(`site1`) cannot be derived as a consequence of `on_fire`(`E`). Yet a lack of universal instantiation is no trouble outside of introspective inference and quotation; `on_fire`(`site1`) and `on_fire`(`E`) are easily unified between two basic formulas. Introspection should also be able to succeed at unification in the case under consideration here. What is needed is another form of introspection that allows the query to unify with a more *generalized* formula, in addition to allowing the query to unify with a more specialized KB formula as can be done by `pos_int_spec` and `neg_int_spec`. The procedures `pos_int` and `neg_int` implement this behavior, allowing both specialized and also generalized formulas to answer introspective queries.

As stated above in contrast to `pos_int_spec` and `neg_int_spec`, an introspective query Q unifies with a quotation term Y that is more generalized when Y contains a fully-escaping variable in a position where Q contains a function, constant, or quote. In the running example, what is necessary is constructing a quotation term of “`on_fire(`E)`” out of the KB formula, which thus is unifiable with “on_fire(site1)” as desired. We next consider how to characterize the method of construction this sort of quotation term more generally. An important part of the functionality is taking a variable that was fully-escaping in the KB formula (e.g., the variable E in the example) and adding a quasi-quotation mark so that it remains fully-escaping once a quotation term has been created. This sort of change sounds similar to what is done by the **Adjust-Context** algorithm for adjusting the level of quotation from 0 to 1, and it is initially tempting to consider using this algorithm to prepare a KB formula for unification. However, this doesn't work in all cases; consider the behavior when executing pos_int(“on_fire(X)”). Modifying the KB formula on_fire(E) into “on_fire(`E)” would prevent unification in this case, thus here the desired method of creating a quotation is to simply add enclosing quotes to create “on_fire(E)”. Together these two cases suggest that converting into a quotation term Y ought to add quasi-quotation marks differently depending upon the introspection argument Q that will be unified with Y.

The general principle is as follows: consider an introspection case where the query, Q, ought to unify with a KB formula that is more generalized. Let there be a generalized KB formula F, that is to be converted into a quotation term Y. Each fully-escaping variable from F will be given an additional quasi-quotation mark in Y only if the variable's corresponding position in Q is not a non-escaping variable. This approach achieves the intended outcome for both of the example queries above, through making comparisons to the query to determine when additional quasi-quotation marks ought to be added to variables. Both the pair pos_int and neg_int and the pair pos_int_gen and neg_int_gen selectively add quasi-quotation marks prior to unification in this way. As suggested by their names, `pos_int_gen` and `neg_int_gen` will fail to unify when the formula Y is a specialization of the query Q, as determined by a traversal of the two quotation terms finding a fully-escaping variable in Q at the same position as a non-variable term in Y.

We note that the second method of creating a quotation term out of a KB formula, which allows for unifying with a formula more generalized than the query, is used exclusively by `pos_int`, `neg_int`, `pos_int_gen`, and `neg_int_gen` out of the entire set of procedural predicates. Other procedures which include a lookup of a KB formula for varied purposes (such as a lookup done by `ancestor`, `acquired`, `update`, and others) are viewed as requiring a user to specify the matching argument relatively precisely, for which finding a formula of greater generality is not necessary or especially useful.

Thus, in writing ALMA axioms for introspection, the three forms of specific, regular, and generalized introspection allow fine-grained control over what sorts of formulas will satisfy introspection procedures. The older ALMA 1.0 system could not offer this kind of control without quotation. Consequently, it would always allow an introspection query $\mathtt{p}(\mathtt{X})$ to unify with the ground formula $\mathtt{p}(\mathtt{a})$ even if the intention of the query was determining if a universally-quantified formula was in the KB. In the current system, `pos_int_gen` can allow a restricted search for `p` with a variable argument. Further, ALMA 1.0 could prevent the query $\mathtt{p}(\mathtt{a}, \mathtt{X})$ from unifying with the generalized $\mathtt{p}(\mathtt{Y}, \mathtt{X})$ or $\mathtt{p}(\mathtt{Y}, \mathtt{b})$, while at the same time allowing the query to unify with $\mathtt{p}(\mathtt{a}, \mathtt{b})$. The current system can achieve such behavior by using *pos_int_spec* for the query. Introspection in ALMA 2.0 thus allows more expressive queries than could be performed in the previous reasoner.

Lastly, the six introspection procedures detailed thus far each may also be supplied a second argument corresponding to an integer time s. If this argument is given, introspection will only attempt to unify the query with formulas that were first acquired at timestep s. The final two introspection procedures, `pos_int_past` and `neg_int_past`, go further in the direction of checking time values for a formula, by taking second and third arguments corresponding to integer times s and e. These procedures unify exclusively with disinherited formulas that were believed over an interval of time contained in the interval $[s, e]$, and provide a means for the reasoner to look back on past beliefs. Therefore, there is not a single procedure for introspection that can check both past and present beliefs; both `pos_int_past` as well as one of the positive introspection predicates for present beliefs must be used to query both past and present beliefs. The functionality of inspecting past beliefs is used in applications that are discussed in section 5.3.

4.5.3 Backward-chaining resolution

ALMA can answer queries via backward-chaining refutation resolution, as initiated by the *bs* command with a formula argument, when the reasoner is not run automatically. This query-answering is done in a separately instantiated partition of the knowledge base, due to the need to hold hypothetical beliefs of the query and derived formulas. The first of these hypothetical beliefs added into the partitioned region when a backward search is initialized is the negation of the query formula. After initializing, resolution broadly works as it does in the forward direction. For example, if a backward search is initiated to determine whether the knowledge base can answer a query for the formula $\mathtt{q}$, the backward-chaining segment is initialized, first containing exclusively $\neg\mathtt{q}$. Consider the following formula, which contains an implication marked as a backward-if instance:

$$\mathtt{p} \xrightarrow{\mathtt{f}} \mathtt{q} \tag{4.7}$$

If this formula is present in the knowledge base, it will resolve with an ongoing backward search query for $\neg\mathtt{q}$, to infer a hypothetical $\neg\mathtt{p}$ and add it to the partition. If $\mathtt{p}$ additionally is in the knowledge base, it will resolve with its negation and yield an empty resolvent.

In backward-chaining, an empty resolvent does not indicate a contradiction but rather it provides an affirmative answer to the query. The above example would thus have a positive answer to the query for $\mathtt{q}$. After an empty resolvent is obtained, the answer with proper variable bindings must be extracted. The backward search associates with every clause in the partition a set of bindings for the variables appearing in the original query. These are updated after each resolution that makes a binding to one of the variables, or to another variable in a current binding of a query variable. As a result, after a query answer has been verified to exist, the tracked bindings allow easy substitution into the query answer. When the initial query has obtained an answer, this then inserted into the main region of the knowledge base.

4.5.4 Procedural atomic predicates

ALMA also features a second category of predicates that execute procedures. In contrast to procedures that execute when appearing as a forward-if premise to determine truth, predicates in this other category execute their respective procedures following the derivation or assertion of a formula containing them. Hence, we refer to these as *procedural atomic predicates.* The procedures associated with each predicate in this category have side effects on the knowledge base. The set of procedural atomic predicates consists of:

`Distrust` — If the unary argument to this predicate unifies with a formula in the knowledge base, its status changes to distrusted. Additionally, all descendants of this formula will also be searched; if any descendant does not have any derivation where all parents are trusted, then this descendant formula will also be distrusted. Any formula X that has become distrusted is not inherited by ALMA into future timesteps, and a meta-formula of the form `distrusted(`"X"$, T$`)` will also be added to the knowledge base at timestep T, as a child of the original `distrust` literal. Although the `distrust` procedure derives instances of `distrusted` as children, unlike almost all other ALMA child formulas these appear immediately, without a timestep delay. This is due to the distrust and disinheritance of targeted formulas happening immediately — since a formula that is distrusted should not have an opportunity of one more timestep to be used in inferences before it is withdrawn.

Recall that a formula with distrusted status is not deleted from the knowledge base, as it is important the clause object can be reached in the reasoner's logic, in cases such as of possible reinstatement or use in handling a contradiction. Any distrusted formula will not participate in inference or most procedures due to the disinheriting flag indicating that it is distrusted, thus preserving the abstraction of being disinherited. If multiple formulas are unifiable with the distrust argument, each will become distrusted accordingly.

`Reinstate` — If the first argument to this predicate is a quotation term unifiable with a distrusted formula that is part of a currently ongoing state of contradiction, and the second argument is an integer matching the timestep when this formula was distrusted, the formula is reinstated into full, non-distrusted belief. The idea of an ongoing state of contradiction is based on the contradiction-handling ability discussed in section 4.6.2, in which a present state of contradiction is indicated by a `contradicting` literal. Hence, the reinstatement argument must also unify with one of the two quoted contradictands of a `contradicting` formula that has not yet been disinherited as *handled* (as introduced in section 4.4.1).

A copy of the prior distrusted formula is constructed for this purpose, which includes metadata such as ancestors and descendants copied from the original, and is added as a new belief at the following timestep (and, due to the use of a copy and its appearance as a new belief after another timestep, there is no difference in behavior with regard to when a reinstatement occurs relative to other inferences in the same timestep). In a sense, the reinstated formula is derived from the `reinstate` literal, but it is more intuitive to simply restore the formula with the parents and children of the original without modification — especially since checking the ancestors of this formula should yield formulas from its original derivation(s). A reinstatement also necessarily resolves at least one ongoing contradiction, as described above. For each ongoing `contradicting` literal X that has become handled at timestep t, a meta-formula `handled(`"X"$, t$`)` will also be inferred as a child of the `reinstate` formula. Like `distrusted` formulas, these children also appear at the same timestep as their parent, since this is when the KB side effect of handling occurs. If multiple formulas are unifiable with the reinstatement argument, a series of formulas may become successfully reinstated at once.

`True` — Implements the specialized inference rule for truth of a quoted formula, as introduced in section 3.3.2.5. If an atomic formula of the form `true`("X") is a belief at a given timestep, where its unary argument is a quotation term, the formula X that is nested in the quotation term will be derived a new belief of the agent one timestep later. When X is withdrawn from quotation, its context is adjusted appropriately with the algorithm for doing so that is detailed in section 3.4.3.4.4. The unquoted formula X is obtained as the child of `true`("X").

`Update` — Implements a variation of the top-level reasoner update command, which is gentler by not having destructive effects on KB derivation history. If the first argument is a quotation term expressing the target, and the second is a quotation term expressing its replacement, if the former successfully unifies with a KB formula, this formula will be distrusted as described above, and its replacement will be asserted as new knowledge (with the `update` literal recorded as the parent that caused this effect). When successful, the newly-added updated formula will appear in the knowledge base after one timestep delay, as occurs for typical ALMA inferences.

Due to the updated formula appearing in the KB after this delay, which internally relies upon the buffer for new clauses that also stores other inferences made at the current timestep, the result is the same regardless of the ordering of an update relative to other inferences at the same timestep.

4.6 Specialized inference rules

In addition to the above more general-purpose inference mechanisms, ALMA also realizes other specialized inference rules which are specific to active logic, including the clock rule and contradiction-handling inference rules.

4.6.1 Clock rule

ALMA records the value of the current timestep, as the evolving moment counting up over each transition, via a special atomic formula of the `now` predicate. From the starting state of the reasoner, with any initial axioms first loaded in, the formula `now(0)` also appears among them. As each timestep transitions to the next, the preceding `now` is disinherited, and from the clock inference rule a `now` with an incremented integer argument is derived:

$$\frac{t : \texttt{now(t)}}{t+1 : \texttt{now(t + 1)}}$$

4.6.2 Contradiction handling

ALMA implements the contradiction-handling aspect of active logic. As discussed previously, inferring an empty resolvent when applying forward-chaining resolution between a pair of formulas, P and $\neg P$, signifies a direct contradiction between the two. In such a case, two atomic formulas are derived instead of a conclusion formula being obtained from the resolution inference rule.

$$\frac{t : P \quad \neg P}{t+1 : \texttt{contra\_event}(\text{``}P\text{''}, \text{``}\neg P\text{''}, t) \wedge \texttt{contradicting}(\text{``}P\text{''}, \text{``}\neg P\text{''}, t)}$$

The first conclusion of `contra_event` indicates that an event of a contradiction occurred at the timestep t. The second conclusion indicates that the ongoing state of P and $\neg P$ being in direct contradiction began at time t. This expresses an open interval that remains ongoing at any present timestep greater than t, until the contradictory state is resolved by a `reinstate` procedure (as described above) that targets either contradictand and disinherits this `contradicting` instance with handled status.

The two contradictands will be marked as distrusted with a corresponding inference, in the same manner as the `distrust` literal operates. Hence, the recursive traversal of contradictand descendants is also performed, to also distrust descendants without a remaining trusted derivation.

There also is a slight variation of the contradiction inference rule that we introduce. This is intended to identify when a pair of `reinstate` literals targets both contradictands (P and $\neg P$) out of an ongoing contradiction. Reinstatement exists exclusively to allow correction of unresolved contradictions. If an ALMA knowledge base contains a set of contradiction-response formulas such that a pair of reinstatements for both contradictands would be inferred without being caught, this would result in an infinite loop: formulas P and $\neg P$ are caught as contradictory, then both reinstated, then once more caught as contradictory, etc. The following inference rule helps to address this problem, by detecting a contradiction in the two quoted formulas that a pair of reinstatements are targetting (rather than a direct contradiction between the full formulas, as in the basic contradiction-detection rule):

$$\frac{t : \texttt{reinstate}(\text{“}P\text{”}, x) \quad \texttt{reinstate}(\text{“}\neg P\text{”}, x)}{t+1 : \texttt{contra\_event}(\text{“}\texttt{reinstate}(\text{“}P\text{”}, x)\text{”}, \text{“}\texttt{reinstate}(\text{“}\neg P\text{”}, x)\text{”}, t)}$$

Additionally, a corresponding instance of `contradicting` with the same three arguments is inferred: $\texttt{contradicting}(\text{“}\texttt{reinstate}(\text{“}P\text{”}, x)\text{”}, \text{“}\texttt{reinstate}(\text{“}\neg P\text{”}, x)\text{”}, t)$.

As a matter of reasoning about a contradiction between pieces of reinstatement meta-knowledge, this is higher-level metareasoning (which also is hinted at through the inference rule conclusion using two levels of quotation). However, since the contradiction between `reinstate` formulas uses exactly the same predicates as any other contradiction, ALMA can respond to this contradiction in a similar way, and guide a response using contradiction-handling formulas appropriate for inconsistency in reinstating. The machinery of ALMA's contradiction-response mechanisms (which crucially are employing quotation terms) are able to solve this kind of contradiction like they can solve a lower-level contradiction.

However, it is also important to emphasize that this inference rule can only stop contradictory reinstatements which are added into the knowledge base at the same timestep (i.e., t in the inference rule above). If P were to be reinstated at timestep t_1 and then $\neg P$ were instead reinstated at a later timestep t_2, then it becomes more difficult to contain this type of contradictory response. For one reason, P will have already been added back into the knowledge base at a timestep after t_1. Furthermore, when there is a differece of timesteps between contradictory reinstatements, it is not obvious how an axiom or inference rule might be devised to distrust only the contradictory reinstatements, while leaving other forms of reinstatements as legitimate.

More generally, we can consider whether a set of ALMA axioms might be expected to converge to a fixed point belief set over time, or would fail to converge to due a pattern such as an infinite loop of contradiction and reinstatement. The case that we have just identified, in which both contradictands might be reinstated are different points in time, demonstrates that in the most general case, beliefs may fail to converge due to recurring contradictions. In particular, an ALMA-based agent with a large collection of different contradiction-handling beliefs may become more prone to such difficulties, as in this case it may become more likely that several principles for reinstatement would produce conflicting results. Thus, when developing examples in ALMA which require reinstatement, as has been done in the present work and is detailed in chapter 5, reinstatement rules must be carefully devised, and tested to ensure that a fixed point is reached in each example. We further identify future work on this limitation in section 6.2.2.

4.6.3 Inferring as true

This inference rule is further described in previous sections 3.3.2.5 and 4.5.4.

4.7 Agent belief modeling

With the ability of nesting formulas by means of quotation terms, we have seen that active logic can also represent formulas attributed as beliefs of other agents, most commonly utilizing a binary predicate `bel` with the first argument for an agent name and the second for a quotation term of the formula considered to be believed by that agent. Once again, any such formula `bel`$(\ldots,\ldots)$ is an internal belief of the Alma agent, which attributes a belief to (a model of) another agent, rather than necessarily representing a formula such an agent truly believes. A number of such `bel` formulas were used in developing of the motivating examples for quotation, namely formulas 3.6, 3.9, 3.24, and 3.33).

The formalism of quotation is essential to ALMA attributing beliefs to other agents. Any atomic `bel` literal can be used in inference within ALMA itself, by unifying with existing forward-if premises and resolving with other KB wffs. But, we additionally have an interest in the ability of ALMA to model the inferences that *other* agents might be able to make (that is, model the inferences that are made by other agents, *not* seeking to model their inference mechanisms). We consider whether we can achieve this.

There are different levels of generality at which this modeling may occur. As an initial example, suppose that Alma represents the agent Alice having the inference rule of modus ponens (for which we later describe the issues of representing this), and also represents Alice believing the following formula:

$$\texttt{tell}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Confidant}) \rightarrow \texttt{heard}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Speaker}) \tag{4.8}$$

Formula 4.8 indicates that a speaker who communicates an utterance to a confidant will hear their own utterance. ALMA would express Alice's belief in 4.8 as follows, using a quotation term:

$$\begin{aligned}&\texttt{bel}(\texttt{alice}, \text{“}\texttt{tell}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Confidant}) \rightarrow \\ &\texttt{heard}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Speaker})\text{”})\end{aligned} \tag{4.9}$$

It is also possible to express the result of Alice applying modus ponens with respect to the particular implication inside the quotation term of formula 4.9. In this case, when Alma considers Alice to hold the belief indicated in the implication 4.9, as well as a belief of a particular `tell` instance, then Alice will also believe an appropriate `heard` instance (ignoring time here):

$$\begin{aligned}&(\texttt{bel}(\texttt{alice}, \text{“}\texttt{tell}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Confidant}) \rightarrow \\ &\quad \texttt{heard}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Speaker})\text{”}) \wedge \\ &\texttt{bel}(\texttt{alice}, \text{“}\texttt{tell}(\grave{}\texttt{S}, \grave{}\texttt{U}, \grave{}\texttt{C})\text{”})) \\ &\xrightarrow{\texttt{f}} \texttt{bel}(\texttt{alice}, \text{“}\texttt{heard}(\grave{}\texttt{S}, \grave{}\texttt{U}, \grave{}\texttt{S})\text{”})\end{aligned} \tag{4.10}$$

This longer formula uses quasi-quotation marks on each of the arguments to Alice's modeled belief of `tell` (and hence, they are fully-escaping), which leads to the correct arguments in Alice's modeled conclusion that repeats the same fully-escaping quasi-quoted variables. However, this example is quite specialized to the particular example and its use of the two predicates `tell` and `heard`. It would be useful for Alma to be able to represent more generalized schemata of another agent applying a form of modus ponens.

An idealized version of modus ponens represented using quotation would be the following:

$$\mathtt{bel}(\mathtt{Agent}, \text{“}\mathtt{A} \rightarrow \mathtt{B}\text{”}) \wedge \mathtt{bel}(\mathtt{Agent}, \text{“}\mathtt{A}\text{”}) \rightarrow \mathtt{bel}(\mathtt{Agent}, \text{“}\mathtt{B}\text{”}) \tag{4.11}$$

However, there are some difficulties that arise when attempting to more generally model formulas along the lines of formula 4.11. First, this is not a grammatical wff (with respect to our running grammar for active logic with quotation, as defined in section 4.2). Throughout the example formula, variables `A` and `B` occur inside of quotation marks and are meant to stand for entire wffs (e.g., as an arbitrary antecedent and arbitrary conclusion of the implication in the first premise's second argument). Yet this is not consistent with the rules and grammar for quotation terms that have been developed, since a quotation term contains exclusively a formula nested inside quotation marks, which must be derived by applying the production rules to some form of predicate (or combination of predicates connected with operators). We are thus not able to support a variable appearing in lieu of a predicate to represent an arbitrary wff.

Crucially the system of active logic with quotation remains first-order, and does not have a sort of variable that can range over wffs. All variables remain a kind of *term*, and cannot unify with constructions that are not also terms. Even a formula as simple as $\mathtt{bel}(\mathtt{Agent}, \text{“}\mathtt{A}\text{”})$ in formula 4.11 makes this mistake. Likewise, there is not a way in which a set of first-order predicates of different arities may all be quantified over within ALMA, which is a simpler problem.[3] We are able to quantify over *quotation terms* by a formula such as `bel(Agent, A)` that follows the grammar we have defined and keeps `A` as a quotation term variable. Although `bel(Agent, A)` can quantify over any quotation term that may appear as the second argument, this is distinct from quantifying over general wffs. A term variable has limitations compared to a hypothetical variable over wffs, such as also remaining unifiable with other kinds of terms alongside quotation terms, and a variable cannot add constraints on the structure of unifiable quotation terms (i.e., any quotation term that satisfies the constraints of the unification algorithm might bind to `A`).

Therefore, the above attempt at modus ponens for another agent will not work even in the simple case where the implication premise `A` is a single (arbitrary) literal. Similar limitations prevent constructing general axioms for modeling another agent using resolution, once again even limiting expressing simple cases. In sum, the goal of generalized modeling of an agent's expected reasoning must be worked toward by other means.

Although modus ponens cannot be represented in a general form as in 4.11, ALMA is able to use modus ponens for formulas outside of quotation, as is implemented as a core part of the reasoner and which forms the basis for forward-if reasoning. Likewise the generalized form of resolution is also directly built into the system as a core inference rule. Hence, we exploit the existing forms of reasoning that are already a core part of the ALMA reasoner, for purposes of modeling agents.

[3] Although, we use a technique in section 5.2 of an interpretation that shifts a predicate symbol into an argument of the predicate, allowing quantifying over a limited category of predicates of a particular specified arity.

That is, we have further designed ALMA to achieve a form of agent inference modeling, subject to certain constraints, by a novel feature: the use of *partitioned regions* of the knowledge base for modeling other agents. When a formula in the ALMA KB is of the form `bel`(*Agent*, "*A*"), the formula *A* is placed into a KB partition specific to the particular *Agent*, which directly uses the same existing inference mechanisms of modus ponens and resolution as the core KB. Although this approach uses a specialized treatment for the belief predicate in particular, it thus still uses quotation overall rather than a modality for belief; hence the system is kept first-order while the benefits of using quotation terms still apply. We describe this functionality in detail in the following sections. Applications using this feature of ALMA are explored in section 5.4, while limitations of this approach are discussed in section 6.2.2.2.

4.7.1 Agent belief model partitions

As we introduce partitioned regions of the ALMA knowledge base to model the beliefs of other agents, we also use terminology to refer to these areas. The modeling of an agent belief occurs in a new region of ALMA's knowledge base we call an *agent KB* (with the proviso that these remain wffs internal to ALMA), and the rest of the knowledge base we call the *core KB* (which has been the focus of essentially all KB discussion up to this point). When the first belief attributed to another specific agent is added into the core KB as a new formula, an agent KB will be instantiated for the agent in question.

Consider an example scenario in which the following formula begins in the ALMA core KB at timestep 0:

$$\texttt{bel}(\texttt{bob}, \text{“}\texttt{p}(\texttt{X})\text{”}) \tag{4.12}$$

At that same timestep, when this is the first positive belief of Bob to appear, the KB for modeling positive beliefs of Bob will be initialized. Then, the corresponding belief `p(X)` will be added into this agent's KB by directly withdrawing the nested formula from the quotation term that is the formula's second argument. If necessary, **Adjust-Context** is applied to bring the formula from one to zero levels of quotation. ALMA performs this initialization as a result of detecting a formula that is solely a `bel` literal with two arguments, for which the first is a constant and the second is a quotation term. Hence, a negated belief literal will also satisfy those conditions; ALMA would construct a second agent KB for the *negative* belief formulas of Bob if $\neg$`bel(bob, “p(X)”)` were to also appear in its core KB. This is used to also enable modeling of what Alma expects other agents to lack beliefs or knowledge regarding. Alma having a belief that $\neg$`bel(bob, “p(X)”)` (i.e., Bob lacks a belief that `p(X)` is true) is naturally distinct from a positive `bel` literal with a negated nested formula such as `bel(bob, “¬p(X)”)` (i.e., Bob does believe that $\neg$`P(X)` is true).

An agent KB likewise has the ability to recursively initialize its own, further nested, agent KBs. Suppose that ALMA has the following formula:

$$\texttt{bel}(\texttt{bob}, \text{“}\texttt{bel}(\texttt{carol}, \text{“}\texttt{p}(\texttt{X})\text{”})\text{”}) \tag{4.13}$$

Then, first “`bel(carol, “p(X)”` will be inserted into the positive agent KB for Bob. But this formula also models Bob's belief that Carol holds a certain belief. If this is the first belief modeling Carol within the agent KB for Bob, a new agent KB for

Bob modeling Carol will be recursively constructed inside of the positive agent KB for Bob. Then, `p(X)` will be inserted into this innermost agent KB for Carol. At this point, ALMA's entire KB will be partitioned into ALMA's core KB and the Bob agent KB, which is itself partitioned into the core KB *for the model of Bob's beliefs*, and the agent KB for *the model of Bob's modeling of Carol's beliefs*. ALMA has been designed to recursively instantiate nested agent KBs to arbitrary depth. This level of generality does ultimately raise some issues in practical examples, and problems that arise are discussed further in section 5.4.

These pairs of example formulas seen so far — `bel(bob`, "`p(X)`") and its related `p(X)`; `bel(bob`, "`bel(carol`, "`p(X)`")") and "`bel(carol`, "`p(X)`"; "`bel(carol`, "`p(X)`" and `p(X)` — are all intuitively strongly related. In fact, there appears to be some form of equivalence between them. For instance, if Alma were to distrust `bel(bob`, "`p(X)`"), then as a result, additional inferences based upon the formula `p(X)` in the agent KB for Bob should not be possible. We introduce the notion of an *equivalence link* to conceptually be an edge connecting a pair of formulas `bel`(*Agent*, "*A*") and *A*. In the implementation, the equivalence links are achieved by storing pointers as metadata for these clauses.

An instance of a formula F belonging to the simulation of an agent $Agent_a$ may have at most two equivalence links connecting it to other formulas: it may have a link that connects it to a formula E (a belief of the agent, $Agent_b$, that models $Agent_a$) of the form `bel`($Agent_a$, "F"), and it might be that F is a `bel` literal containing a quoted formula G as the belief of an $Agent_c$, in which case F would have a link connecting it to G. We call the first of these an *upward equivalence link* (or upward link) and the second a *downward equivalence link* (or downward link). Hence in this current example, formula F has an upward link to E, and also F has a downward link to G. Meanwhile E has a downward link connecting to F, and no upward link if there only exists the three formulas E, F, and G; while G has an upward link connecting to F and no downward link.

Thus, we have characterized the basic mechanisms of agent belief partitions, when they are created, and the means by which connected formulas are joined across partitions using equivalence links. Next, we describe how these partitions are used to make new inferences, and how beliefs synchronize between the KB regions as this occurs.

4.7.2 Belief inference and synchronization

With agent belief modeling, the process of high-level control of ALMA as described in section 4.4 is slightly expanded. At the same point when prospects are generated based on formulas present in the core ALMA KB, as previously described, if there exist any instantiated agent KBs (i.e., in the partitions of the ALMA KB), then the same procedures for resolution and forward-if prospect generation will recursively generate prospects for each agent KB region. Any KB region thus tracks its own queue of prospects, which are tuples of formulas that are all contained within that same partitioned region.

Then, at the beginning of processing a *step* command, ALMA first progresses the queued prospects for the core KB, but now additionally also progresses the queued prospects for any agent KB recursively and stores any resulting formulas into each agent KB's respective buffer. We emphasize that progressing agent KB prospects executes exactly the same code for making new inferences as the core KB uses to progress its own prospects during inference, with both resolution and forward-if formulas.[4] Therefore, this has the effect of the Alma agent modeling other agents as having the abilities of inference via resolution and a form of extended modus ponens. This includes inference rules regarding formula equivalences (e.g., that an implication is equivalent to its disjunctive form, that conjunctions with either order of arguments are equivalent) as part of the specific forms of these inference rules that ALMA uses beginning by formula standardization. We further discuss in section 6.2.2.2 some limitations relating to this commitment that other agents are modeled as reasoning exactly as Alma does. ALMA additionally models agents as having the ability to further recursively model other agent reasoning, in the same way as the core KB can model others.

Once the prospects have been exhausted and any resulting formulas added into the buffers of new inferences for the core KB and all agent KBs, there is a new step for synchronization of new formulas between KB partitions. Just as when the formulas for timestep 0 were first loaded into ALMA the appearance of a formula `bel`$(Agent, \text{“}F\text{”})$ led to the initialization of F in an agent KB, there is also a process of synchronization necessary during any step command. First, new beliefs are synchronized downward into nested agent KBs. Beginning from the core KB,

[4] However, we note that some minor changes were necessitated to selected procedural premise predicates, to allow the correct behavior when interacting with equivalence links; we describe these in the following section.

any new positive or negative `bel` literals from the new clause buffer will insert into their appropriate agent KB clause buffer the formula F within quotes, and construct equivalence links. This continues in a depth-first fashion with preorder traversal of the KB tree, so that any formula with multiple nested quotation terms of `bel` will sync down as far as possible. Subsequently, a postorder traversal of the KB tree will recursively synchronize belief formulas from the leaf agent KBs upward, so that for any new inference F made in an arbitrary agent KB, `bel`($Agent$, "F") will be instantiated in the larger KB that is modeling $Agent$. One traversal down to the leaves followed by a traversal back up to the root is sufficient to sync all agent beliefs across KB regions. After synchronization, the contents of each new clause buffer are inserted into their respective KB.

In addition to the synchronization which occurs across timesteps as new inferences are made, there is also a secondary form of synchronization that arises when a formula with a downward link become distrusted. Returning to the example discussed above, if `bel(bob, "p(X)")` has become distrusted, then its downward link formula of `p(X)` should not be used to infer new formulas in Bob's agent KB. However, `p(X)` is not considered distrusted itself. In the example, it is the agent modeling Bob (either Alma, or another agent, itself modeled by either Alma itself, or another agent model) that distrusts their belief that Bob believes `p(X)`, which is distinct from this agent believing that Bob himself distrusts `p(X)`. Hence, we use the status of *paused* as introduced in section 4.4.1, which stops the creation of new inferences from `p(X)`. Marking a formula with the paused status also recursively pauses its children and the formula connected by a downward link. The paused status is revoked by a reinstatement of one of the distrusted formulas that had prompted recursive pausing (in this case, by a reinstatement of `bel(bob, "p(X)")`), which recursively unpauses the affected formulas.

4.7.3 Effects on ALMA procedures

A subset of the ALMA procedural premise predicates detailed in section 4.5.2.1 have behavior which accounts for the presence of equivalence links. If a formula X is joined to a formula Y by a downward link, the parents of Y also should be treated as alternative parents for X (albeit, parents which used a distinct derivation method involving the agent KB containing Y). The closure of this property also applies, so that if Y has a downward link to a formula Z, the parents of Z also should serve as alternative parents for both Y and X. However, this property does not apply for traversals that begin by following upward links. A procedure executing in the context of an agent KB means that it is the model of the agent performing this procedure, and hence the inner model of an agent should not have any access to the formulas of whoever is modeling it.

Usage of downward links for alternative derivations has an impact on the procedures that check a formula's derivation. This consists of the pair `parents_default` and `parent_non_default`, and also the trio of `ancestor`, `non_ancestor`, and `parent`. With regard to procedures `parents_default` and `parent_non_default`, a formula is determined to have a non-default derivation if it has a downward link to a formula with unpaused parents. For the three procedures checking parents or ancestors, an arbitrary X with a downward link to Y will be treated as having the upward-linked formulas for Y's parents as its own parents (and the closure of this property, recursively).

Finally, distrust and reinstate have effects as described above for pausing or unpausing formulas connected by downward links. Further, the `distrust` procedure used on an arbitrary X with a downward link to Y will also distrust the upward-linked formulas for Y's children, which are to be treated as children of X (as well as the closure of this property, recursively).

4.8 Comparison with the prior reasoner

Lastly, we compare ALMA 2.0 to the prior reasoner, ALMA 1.0. Notable deficiencies in ALMA 1.0 rendered it unsuitable for further research development. However, it nevertheless gave useful insights and behavior specification to learn from.

4.8.1 Deficiencies in ALMA 1.0

ALMA 1.0 represents active logic formulas by mapping into the predicates and functions of Prolog, and uses Prolog's built-in unification algorithm. Because ALMA 1.0 has this reliance on Prolog's logical primitives and unification, deeper changes to the terms, structure, and inference in active logic are prevented, since none of these are managed in data structures or code of the reasoner itself. The present work crucially required introducing new terms into the language and altering unification that inference can make use of these new terms, among other substantial differences in the reasoner. ALMA 1.0 was therefore fundamentally unable to support this development because of its Prolog foundation.

Significant limitations were also found in the methods of introspection of ALMA 1.0, which are restricted in their use due to requiring ground arguments rather than more arbitrary arguments, and that the system further did not track formulas from prior timesteps of the belief history.

4.8.2 Behavioral overlap

ALMA 2.0 was first created with the goal of obtaining a new active logic reasoner that was to be more maintainable than ALMA 1.0, and to use a language other than Prolog. Despite the flaws of ALMA 1.0, there remains ample literature describing its behavior (or desired behavior), which was used as a blueprint for developing its successor. The formal active logic grammar was based on the grammar used by ALMA 1.0, although parsing infrastructure and formula rewriting was entirely new by virtue of implementing outside of Prolog. Similarly, the intended high-level behavior of control via commands and the specification of inference methods (particularly the partitioned approach to backward search) also were derived from the approach of ALMA 1.0. However, the opacity of the ALMA 1.0 code left determining how to realize these behaviors as problems to be solved throughout the development of ALMA 2.0.

Chapter 5: Applications to commonsense reasoning

5.1 Introduction

ALMA is a platform that is aimed toward supporting general commonsense reasoning. From the description of ALMA's architecture, structures, and mechanisms of inference, as detailed in chapter 4, we now move to application of the reasoner to some commonsense problems, which also can incorporate quotation terms. In particular, ALMA is applied to a series of existing commonsense reasoning problems from the literature. Importantly, the novel combination of active logic with quotation terms, as implemented in ALMA, allows for solving these problems in novel settings and versions. The development will get complicated, but this seems a necessary aspect of practical commonsense reasoning.[1]

[1] While a source of some of the complexity is due to time-based reasoning, to ignore using a time-based approach would lose a huge amount of essential reasoning capabilities, as we had discussed in chapter 1. We argue that solving commonsense problems such as this chapter addresses, without active logic, via an alternative formal mechanism that is nevertheless time-sensitive, would bring up similar issues — particularly if the alternative has similar purposes to active logic such as avoiding omniscience and other issues of external logic.

5.2 Default reasoning with nested beliefs

Reasoning on the basis of default assumptions is widely regarded as essential for an agent with wide-ranging reasoning abilities. In some sense, nearly any formula in ALMA may be considered a default belief, since the formula is not an ironclad belief, but rather might become distrusted or otherwise be overturned at a later timestep.[2] However, there is also a form of default reasoning in which the conclusions of certain implications are specifically represented as default conclusions. That is, a particular formula might be known to be a default conclusion, by virtue of having been inferred from an implication that is represented as a default rule. The ability to determine whether a formula might be considered a default belief can be quite useful. Depending on the belief set, it may be the case that a default formula is more likely to be overturned, or it might alternatively be the case that when an inconsistency arises between a default and a non-default formula, a reinstatement of the latter is preferred due to it not allowing exceptions as the default does.

Traditional approaches commonly either center the entire theory around default rules, or, if this is not the case, introduce a new default operator which is distinct from the standard implication operator, to indicate making a default conclusion [52, 78]. However, while ALMA largely follows a default-based approach, it does so in an evolving time setting. As the means of representing default implication formulas in ALMA, we use quotation terms within additional premises, which take

[2] However, some counterexamples are formulas such as `now`, or examples of introspection. Although it is possible that an agent may be mistaken regarding these in the most general case — such as by not knowing the correct time, or having a faulty clock — we don't deal with these cases in the present work, and assume these formulas to be correct.

the role of ensuring that any justification for a default conclusion is met.

To introduce this, we return to an earlier motivating scenario: that of birds flying. The following is a non-default formula expressing that all birds fly:

$$\texttt{bird}(\texttt{B}) \rightarrow \texttt{flies}(\texttt{B}) \tag{5.1}$$

Yet not all birds fly — although typically an arbitrary bird does, unless it happens to belong to a category that is an exception to the rule. However, this intuition about the flight of typical birds is not reflected in formula 5.1. What is desired is inferring, on the basis of a lack of contrary information, that a bird flies. This is a classic example of a default rule.

There exists a span of approaches of how formula 5.1 might be repaired and fleshed out into such a default, thereby fixing its mistake of expressing that *all* birds fly. Reiter's approach to default logic relies on an added condition that it is consistent to assume the conclusion (i.e., that birds fly) [78]. Levesque uses a logic of "only-knowing" based upon modal operators for both belief and indicating that *only* particular formulas are believed [44]. McCarthy's circumscription is in between the above two, and doesn't prove consistency but captures something close, in attempting to minimize abnormality with respect to flying [52]. Finally, the last of methods we consider is basing a bird flying on a lack of present knowledge that it doesn't fly. This was employed in ALMA 1.0 examples from Purang, using negative introspection. Specific to the present incarnation of ALMA, the following is a formulation of how the system might conclude that a bird flies, which was presented in chapter 3 in developing the syntax of quotation:

$$\texttt{bird}(\texttt{X}) \wedge \texttt{neg\_int}(\text{“}\neg\texttt{flies}(\grave{}\texttt{X})\text{”}) \xrightarrow{\texttt{f}} \texttt{flies}(\texttt{X}) \tag{5.2}$$

Here, negative introspection checks that the ALMA KB does not include the belief that a particular bird doesn't fly, rather than checking a condition reflecting an exception. Due to our focus on internal methods for an agent, the consultation by an agent of its belief set for negative introspection is attractive, in contrast to more external approaches based on provability or consistency. However, formula 5.2 ideally should better capture our intuitions that a bird can fly when it isn't abnormal in a capacity that affects the ability to fly.

Thus, we argue for a representation that still devises default rules as determined on the basis of present knowledge, yet bases its extra premise upon abnormality. Although we do not argue for using circumscription itself, the following is an example of a formalism is inspired by McCarthy's representation of abnormality:

$$\texttt{bird(X)} \land \neg\texttt{abnormal(aspect2(X))} \rightarrow \texttt{flies(X)} \tag{5.3}$$

Here the `abnormal` predicate of the second premise indicates that the bird in question is abnormal with respect to a particular aspect denoted by a unique function, `aspect2`. A problem with this kind of approach is that the relation of `aspect2` to birds and to flying is not made clear outside of the context of this implication that connects all three. Additionally, it would become unwieldy if introducing a specialized function symbol were necessary for each aspect that might be abnormal for some object with respect to a property.

Active logic with quotation allows us to express such a relation more efficiently, without a large class of functions for abnormality, and also computationally efficiently by simply scanning its KB rather than executing some larger procedure. Let us consider for this example that Tweety is a specific bird, and cannot fly due to being a penguin. What is desired is expressing a formula much like the following:

$$\texttt{abnormal}(\texttt{tweety}, \texttt{bird}, \text{“}\neg\texttt{flies}(\texttt{tweety})\text{”}) \tag{5.4}$$

That is, this ternary predicate for abnormality expresses that, for a particular object in question (e.g., tweety), this object is abnormal with respect to a property (e.g., being a bird), due to a specific failure given in a quotation term (e.g., the nested formula expressing that tweety does not fly, by virtue of being a penguin). This concept of using quotation to represent a schema for abnormality of the form $\texttt{abnormal}(\texttt{Object}, \texttt{Property}, \texttt{Failure})$ is quite powerful, and allows moving past the usage of aspects as employed by McCarthy, to instead make use of fairly general predicate expressions for abnormality that additionally are in line with time-sensitive commonsense reasoning. We use negative introspection with an argument of a quoted `abnormal` literal, which does not require a litany of explicit literals. In these cases, the final argument to `Failure` will be a quoted formula that typically expresses the negation of the property concluded as the consequent of the default formula.

There is one additional detail to be addressed as part of the means of abnormality representation with quotation. The second argument for an abnormality literal indicates that the object (e.g., `tweety`) is abnormal with regard to a *property* (e.g., `bird`), due to a failure (e.g., $\neg\texttt{flies}(\texttt{tweety})$). For each constant used to name such a property, there also must exist a corresponding predicate expression, denoting the fact of an object having this property (e.g., $\texttt{bird}(\texttt{X})$ in the running example). As we utilize abnormality formulas, it is desired that the symbols for relating an object to a property be made available to quantify over — yet the system of active logic with quotation cannot quantify over predicate symbols. In several of the commonsense example domains ahead, we therefore rewrite a group of relations via changing the predicate symbol to the newly introduced symbol `rel`, and shifting the original symbol for a property to now occur as the first constant argument to `rel`. We formalize this as follows: to rewrite a N-ary predicate expression $\mathtt{prop}(A_0, A_1, \ldots, A_{n-1})$, we change this to the (N+1)-ary predicate expression $\mathtt{rel}(Prop, A_0, A_1, \ldots, A_{n-1})$.

Applying this to a specific example, we consider a formula that rewrites a non-default version of the rule for birds flying:

$$\mathtt{rel}(\mathtt{bird}, \mathtt{B}) \xrightarrow{\mathtt{f}} \mathtt{rel}(\mathtt{flies}, \mathtt{B}) \tag{5.5}$$

The intended formulation of this as a default rule (5.8 below) thus adds the additional `neg_int` premise with a quotation term containing a nested `abnormal` literal, in the way described above, in which the reason for abnormality is the bird not flying. For generality of negative introspection, inside the quotation term, quasi-quotation marks precede the variables representing birds, so that they are fully-escaping.

$$\begin{aligned}&\mathtt{rel}(\mathtt{bird}, \mathtt{B}) \wedge \mathtt{neg\_int}(\text{``}\mathtt{abnormal}(\grave{}\mathtt{B}, \mathtt{bird}, \text{``}\neg\mathtt{rel}(\mathtt{flies}, \grave{}\grave{}\mathtt{B})\text{''})\text{''})\\ &\xrightarrow{\mathtt{f}} \mathtt{rel}(\mathtt{flies}, \mathtt{B})\end{aligned} \tag{5.6}$$

Here, 5.4 would be one such example of how a bird might be abnormal.

Thus, the following formula attributes a property to an object if it is of a particular kind — a far more generalizated formulation of the kind of formula 5.5. This implication is not a default rule.

$$\mathtt{rel}(\mathtt{Kind}, \mathtt{Obj}) \xrightarrow{\mathtt{f}} \mathtt{rel}(\mathtt{Prop}, \mathtt{Obj}) \tag{5.7}$$

And finally, the corresponding default formulation is created by adding in a generalized negative introspection argument, yielding the following:

$$\texttt{rel}(\texttt{Kind},\texttt{Obj}) \wedge \texttt{neg\_int}(\text{“}\texttt{abnormal}(\texttt{`Obj}, \texttt{`Kind}, \text{“}\neg\texttt{rel}(\texttt{``Prop}, \texttt{``Obj})\text{”})\text{”})$$
$$\xrightarrow{\texttt{f}} \texttt{rel}(\texttt{Prop},\texttt{Obj}) \tag{5.8}$$

This generalized formula makes clearer that, as we use the `abnormal` predicate in the present work, it can be determined whether an object that is abnormal with respect to a property has or lacks the property, based on whether the last argument to `abnormal` contains a positive or negative formula quoted formula. And likewise, any object which is normal with respect to the property necessarily must then have the negation apply.

Having developed this form of representing default reasoning in ALMA, which was aided by the use of quotation terms, we next turn to a series of particular case studies where we apply such active logic default formulas.

5.2.1 Nested defaults case study

Fahlman [23] identified the need for a knowledge-based system to handle a series of nested default rules, where each more-specialized case contradicts more general knowledge. Fahlman gave the specific example:

1. A mollusk typically is a shell-bearer.
2. A cephalopod is a mollusk, but typically is not a shell-bearer.
3. A nautilus is a cephalopod, but typically is a shell-bearer.
4. A naked nautilus is a nautilus but typically is not a shell-bearer.

Traditional nonmonotonic reasoning does not deal with actually undoing conclusions over an episode of reasoning. For example, if a particular animal is simply known to be a nautilus, then a traditional approach would conclude that it is a shell-bearer. But if the reasoning occurs over time, if a creature is first learned to be a cephalopod, and later the belief is acquired that this creature is a nautilus as well, then the above rules lead to a contradiction regarding whether the creature is a shell-bearer or not. Ideally, this should be repaired by giving up the belief the creature is not a shell-bearer. Active logic's handling of direct contradictions allows reasoning to proceed even if direct inconsistencies arise. Then, by using nesting to refer to the contradictory beliefs, reinstatements can be made to serve as intended contradiction repairs.

We present solutions to a series of problems centered around reasoning with formulas for interacting defaults, here and in later actions, based initially on Fahlman's problem of reasoning with mollusk categories in an ontology. Our solutions extend the scope of this problem in several ways. First, we consider how ontology formulas that originally are *not* defaults might be revised to be updated into default formulas in real time. For instance, the KB will begin with a non-default formula expressing that "a mollusk *is* a shell-bearer," without allowing for any exception. In this section, through a pair of examples, we develop axioms that can *make these revisions into defaults*. Second, we devise axioms to appropriately reinstate contradictand formulas, when contradictions have arisen due to the lack of defaults. Third, additional axioms for a separate more complex ontology are also developed in section 5.2.2, which remain compatible with examples utilizing the mollusk ontology. Fourth, in section 5.2.4, we apply the combined set of default reasoning axioms to develop a range of examples of varying complexity, involving a variety of the temporal orderings for acquiring knowledge about individuals in both ontologies. Our set of axioms also pervasively uses quotation terms and quasi-quotation terms, further demonstrating the usefulness of these features.

5.2.1.1 Initial mollusk ontology

We begin with the following ALMA axiom set, which encodes the non-default form of knowledge about the categories of mollusks:

$$\texttt{rel}(\texttt{is\_a}, \texttt{cephalopod}, \texttt{mollusk}) \tag{5.9}$$

$$\texttt{rel}(\texttt{is\_a}, \texttt{nautilus}, \texttt{cephalopod}) \tag{5.10}$$

$$\texttt{rel}(\texttt{is\_a}, \texttt{naked\_nautilus}, \texttt{nautilus}) \tag{5.11}$$

$$\texttt{rel}(\texttt{mollusk}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{shell\_bearer}, \texttt{X}) \tag{5.12}$$

$$\texttt{rel}(\texttt{cephalopod}, \texttt{X}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{X}) \tag{5.13}$$

$$\texttt{rel}(\texttt{nautilus}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{shell\_bearer}, \texttt{X}) \tag{5.14}$$

$$\texttt{rel}(\texttt{naked\_nautilus}, \texttt{X}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{X}) \tag{5.15}$$

These formulas express that there exists an ontology of a series of progressively narrower categories of mollusks: a cephalopod as a specialized kind of mollusk, a nautilus as a specialized kind of cephalopod, and a naked nautilus as a specialized kind of nautilus. Additionally, formulas 5.12–5.15 express the status of these groups as being a shell-bearer or not, as in the numbered items above from Fahlman.

We also include the following four formulas. The first pair are necessary to make use of the ontology, while the latter are more general information for the ALMA agent to use.

$$\texttt{rel}(\texttt{is\_a}, \texttt{X}, \texttt{Y}) \wedge \texttt{rel}(\texttt{is\_a}, \texttt{Y}, \texttt{Z}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{is\_a}, \texttt{X}, \texttt{Z}) \tag{5.16}$$

$$\texttt{rel}(\texttt{is\_a}, \texttt{A}, \texttt{B}) \wedge \texttt{rel}(\texttt{A}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{B}, \texttt{X}) \tag{5.17}$$

$$\texttt{obs}(\texttt{X}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{X}) \tag{5.18}$$

$$\texttt{abnormal}(\texttt{Obj}, \texttt{Kind}, \texttt{Failure}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{Failure}) \tag{5.19}$$

Formula 5.16 establishes the transitivity of the `is_a` relation, and formula 5.17 establishes that relations applying to a smaller ontology category will transfer to a larger one that contains it. Finally, formulas 5.18 and 5.19 express that any formula indicated as an observation is believed to be true, and a formula provided in a quotation term to **`abnormal`**, as a reason for why the object fails to be normal, should be believed to be true as well.

From this starting belief set, we developed additional axioms as became necessary for the correct handling varying sets of mollusks as inputs, to obtain the intended final beliefs. Next, we describe a series of examples as motivating examples for the axiom expansions. The discussion of examples in the subsequent section is at a high level, without exhaustively providing each detail of ALMA inferences per example. However, we also provide full traces for ALMA executing these examples, which can be found in the GitHub repository for the mollusk, vehicle, and combined domains, respectively.

5.2.1.2 First example: cephalopod as a counterexample

The very first example we consider begins with information about a cephalopod arriving, in a form such as the following:

$$\texttt{rel}(\texttt{cephalopod}, \texttt{steve}) \tag{5.20}$$

For simplicity, we assume that this information arrived as an observation that the agent has made, and so the belief in Steve being a cephalopod is considered beyond reproach. After one timestep, $\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ is inferred together with formula 5.13. Additionally, the belief that $\texttt{rel}(\texttt{mollusk}, \texttt{steve})$ is also inferred from formulas 5.9 (that a cephalopod is a mollusk), 5.17 (is-a relations transferring), and 5.20. After one more timestep, Steve also being a mollusk in combination with formula 5.12 produces that $\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$. Thus the knowledge base now attributes both having and not having a shell to Steve, which is a direct inconsistency.

At this point, it is evident that one problem that led to the contradiction is that formula 5.12 is an absolute formula, rather than a default formula, and observing Steve has provided a counterexample. Thus, formula 5.12 should also be updated to amend it into a default formula. We achieve this effect through the following new axiom, which is lengthy due to being designed for generality:

$$\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge$$

$$\texttt{rel}(\texttt{Kind}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{is\_a}, \texttt{Kind\_spec}, \texttt{Kind})\wedge$$

$$\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”}, \text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge$$

$$\begin{aligned}
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind\_spec}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\\
&\xrightarrow{\texttt{f}}\\
&\texttt{update}(\text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Ob}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{Ob})\text{”},\\
&\quad \text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Ob}) \wedge \texttt{neg\_int}(\text{“}\texttt{abnormal}(\grave{}\texttt{Ob}, \grave{}\grave{}\texttt{Kind}, \text{“}\neg\texttt{rel}(\grave{}\grave{}\grave{}\texttt{Pred}, \grave{}\grave{}\texttt{Ob})\text{”})\text{”})\\
&\quad \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{Ob})\text{”})\\
&\wedge\\
&\texttt{rel}(\texttt{Kind}, \texttt{Ab\_Obj}) \wedge \neg\texttt{rel}(\texttt{Pred}, \texttt{Ab\_Obj})\\
&\quad \xrightarrow{\texttt{f}} \texttt{abnormal}(\texttt{Ab\_Obj}, \texttt{Kind}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Ab\_Obj})\text{”}) \qquad (5.21)
\end{aligned}$$

The first premise of formula 5.21 requires two formulas presently contradicting, where the pair of contradictands has the structure of unary predicates (as expressed by the `rel` predicate, i.e., $\texttt{rel}(\texttt{Pred}, \texttt{Obj})$). To ensure that these unary relations are specific to ontology-related reasoning, the formula has additional premises requiring that the `Obj` arguments of the contradicting formulas be inferred from parent formulas $\texttt{rel}(\texttt{Kind}, \texttt{Obj})$ and $\texttt{rel}(\texttt{Kind\_spec}, \texttt{Obj})$, for which $\texttt{rel}(\texttt{is\_a}, \texttt{Kind\_spec}, \texttt{Kind})$ expresses the hierarchical relationship of `Kind_spec` as a more specialized category of `Kind`. In the running example, these premises are satisfied as `Obj` is bound to `steve`, `Kind` to `mollusk`, and `Kind_spec` to `cephalopod` (as hierarchically related by formula 5.9).

Furthermore, formula 5.21 requires that the positive contradictand has a non-default implication as a parent, of the form $\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Obj}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{Obj})$ (e.g., 5.12 in the example). Thus, an English summation of the premises is that the negative contradictand provides evidence that the narrower ontology category is a counterexample to this non-default parent formula, which states that for an object `Obj` in the wider category `Kind`, a positive `Pred` relation follows.

Two conclusion formulas follow as a result of meeting the premises of formula 5.21. The first conclusion is an `update` literal (identified as a procedural atomic predicate in section 4.5.4), which updates the non-default implication into a default formula. In this default formula, negative introspection must indicate the lack of `Ob` being abnormal for type `Kind` with the reason that $\neg\mathtt{rel}(\mathtt{Pred}, \mathtt{Ob})$ holds for particular `Pred` and `Ob` in question. Thus in the case of Steve, formula 5.12 is updated into the following:

$$\begin{aligned}&\mathtt{rel}(\mathtt{mollusk}, \mathtt{Ob}) \wedge \\ &\quad \mathtt{neg\_int}(\text{“}\mathtt{abnormal}(\grave{}\mathtt{Ob}, \mathtt{mollusk}, \text{“}\neg\mathtt{rel}(\mathtt{shell\_bearer}, \grave{}\grave{}\mathtt{Ob})\text{”})\text{”}) \\ &\xrightarrow{f} \mathtt{rel}(\mathtt{shell\_bearer}, \mathtt{Ob})\end{aligned} \tag{5.22}$$

Note that instances of `Ob` in formula 5.21 are not fully-escaping variables, but at one effective level of quotation, thereby ensuring that 5.22 quantifies over any possible constant that binds to `Ob` that represent be a mollusk. The second conclusion from formula 5.21 is an implication that gives the means by which other objects can be inferred to be abnormal, with respect to the predicates that have been used. For instance, in the example of Steve, this second conclusion is the following:

$$\begin{aligned}&\mathtt{rel}(\mathtt{mollusk}, \mathtt{Ab\_Obj}) \wedge \neg\mathtt{rel}(\mathtt{shell\_bearer}, \mathtt{Ab\_Obj}) \\ &\xrightarrow{f} \mathtt{abnormal}(\mathtt{Ab\_Obj}, \mathtt{mollusk}, \text{“}\neg\mathtt{rel}(\mathtt{shell\_bearer}, \grave{}\mathtt{Ab\_Obj})\text{”})\end{aligned} \tag{5.23}$$

An object that is known to be a mollusk and which does not have a shell is thus concluded to be abnormal as a mollusk, with respect to being a shell-bearer, for this very reason of lacking one. As is the case with formula 5.22, this formula is specialized to particular predicates (i.e., `mollusk` and `shell_bearer`) in places where formula 5.21 had fully-escaping variables. Thus, formula 5.21 enables the desired revision of a piece of ALMA ontology knowledge into a default rule (e.g., formula 5.22), and additionally instantiates another belief that allows the conclusion of objects being abnormal (e.g., formula 5.23).

However, in the example, the newly inferred formulas 5.22 and 5.23 are not sufficient to obtain the desired KB of beliefs about Steve. The existing contradiction between the $\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ and $\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ remains, and hence it cannot yet be inferred from formula 5.23 that Steve is an abnormal mollusk. Furthermore, in the absence of the abnormality formula, the negative introspection premise of formula 5.22 will evaluate to true, and after another timestep the system will once again infer that $\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$. There is thus a need for an axiom that produces a reinstatement, when a direct contradiction occurs due to conflicting ontology categories. We introduce the following axiom for this purpose:

$$\begin{aligned}
&\texttt{contradicting}(\text{``}\texttt{rel(`Pred,`Obj)}\text{''}, \text{``}\neg\texttt{rel(`Pred,`Obj)}\text{''}, \texttt{T})\wedge\\
&\texttt{neg\_int}(\text{``}\texttt{obs(}\text{``}\texttt{rel(``Pred,``Obj)}\text{''}\texttt{)}\text{''})\wedge\\
&\texttt{rel}(\texttt{Kind}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{is\_a}, \texttt{Kind}, \texttt{Kind\_gen})\wedge\\
&\texttt{parent}(\text{``}\texttt{rel(`Kind,`Obj)}\text{''}, \text{``}\neg\texttt{rel(`Pred,`Obj)}\text{''}, \texttt{T})\wedge\\
&\texttt{parent}(\text{``}\texttt{rel(`Kind\_gen,`Obj)}\text{''}, \text{``}\texttt{rel(`Pred,`Obj)}\text{''}, \texttt{T})\wedge\\
&\texttt{non\_ancestor}(\text{``}\texttt{rel(is\_a,`Kind\_spec,`Kind)}\text{''}, \text{``}\texttt{rel(`Kind,`Obj)}\text{''}, \texttt{T})
\end{aligned}$$

$$\xrightarrow{\texttt{f}} \texttt{reinstate}(\text{``}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{''}, \texttt{T}) \tag{5.24}$$

This formula reinstates the negative contradictand under the following conditions: the contradictand must have a parent formula that the object is of type `Kind`, where the positive contradictand has a parent of more general type `Kind_gen` (related to `Kind` by an `is_a` relation), and where there does not exist a more specialized ontology category as a parent for the belief that the object is of type `Kind`. Therefore, this formula can be summarized as reinstating the negative contradictand when it has been inferred from the narrowest ontology category out of the two contradictands. This encodes one of the major intuitions behind why Steve lacking a shell is given credence: a narrower ontology category might provide an exception to a rule that applies differently to a broader category in the hierarchy.

Formula 5.24 allows for restoring the negated formula which had been a contradictand. However, due to the timing of ALMA's inferences across timesteps, there are some additional steps in the restoration. As we have described, in the running example, the default formula will infer `rel(shell_bearer, steve)` from 5.22. That inference occurs at the same timestep as either reinstatement formula places ¬`rel(shell_bearer, steve)` back into the KB, immediately creating a second direct contradiction that must be handled. Yet derivation from the narrowest ontology category, the condition upon which formula 5.24 makes a reinstatement, still applies to the negative contradictand, and thus reinstates it once more after an additional timestep.[3] After handling the second contradiction, one timestep later a

[3] The positive contradictand will *not* be inferred once again from 5.22, due to ALMA's behavior for new inferences, for which the same formulas that have already expanded a forward-if prospect will not duplicate an inference with reuse at any later timestep.

reinstated $\neg$`rel`(`shell_bearer`, `steve`) will be used as a premise for formula 5.23, to infer `abnormal`(`steve`, `mollusk`, "$\neg$`rel`(`shell_bearer`, `steve`)"). Hence, the correct conclusions have been made about Steve lacking a shell, and that he is an abnormal mollusk, in addition to the formula 5.23 and the default formula 5.22. At this point, the system idles without further contradictions appearing regarding Steve.

5.2.1.3 Second example: counterexample from observation

We next consider a variation of the first example, in which information about the shell-bearer status of a mollusk arrives directly as observations, as indicated by the `obs` predicate:

$$\texttt{obs}(\text{``}\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{bob})\text{''}) \tag{5.25}$$

$$\texttt{obs}(\text{``}\texttt{rel}(\texttt{mollusk}, \texttt{bob})\text{''}) \tag{5.26}$$

By formula 5.18, the quoted formulas in this pair will inferred as `true` by Alma, and then inferred to be direct beliefs by the inference rule for `true`. Subsequently, a contradiction arises between `rel`(`shell_bearer`, `bob`) (derived from formula 5.12) and $\neg$`rel`(`shell_bearer`, `bob`), much like the contradiction arises in the first example.

It was observed that Bob lacks a shell, once again providing a counterexample to the belief expressed in formula 5.12 that all mollusks have shells. However, the negative contradictand is here obtained not from an ontology, but rather from an observation, and in fact there is not a belief in the KB to indicate whether Bob is a subtype of mollusk or not. Formula 5.21 hence will not apply, and an additional axiom is necessary to weaken formula 5.12 when the evidence against the non-default implication has come from observation. We introduce the following axiom to do so:

$$
\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge \\
&\texttt{obs}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}) \wedge \texttt{rel}(\texttt{Kind}, \texttt{Obj})\wedge \\
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”}, \text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T}) \\
&\xrightarrow{\texttt{f}} \\
&\texttt{update}(\text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Ob}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{Ob})\text{”}, \\
&\quad \text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Ob}) \wedge \texttt{neg\_int}(\text{“}\texttt{abnormal}(\grave{}\texttt{Ob}, \grave{}\grave{}\texttt{Kind}, \text{“}\neg\texttt{rel}(\grave{}\grave{}\grave{}\texttt{Pred}, \grave{}\grave{}\texttt{Ob})\text{”})\text{”}) \\
&\quad \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{Ob})\text{”}) \\
&\wedge \\
&\texttt{rel}(\texttt{Kind}, \texttt{Ab\_Obj}) \wedge \neg\texttt{rel}(\texttt{Pred}, \texttt{Ab\_Obj}) \\
&\quad \xrightarrow{\texttt{f}} \texttt{abnormal}(\texttt{Ab\_Obj}, \texttt{Kind}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Ab\_Obj})\text{”}) \qquad (5.27)
\end{aligned}
$$

This axiom is quite similar to formula 5.21. In fact, it has an identical pair of conclusions: the first which updates a non-default implication with a conclusion matching the positive contradictand into a default, and the second which gives the means to conclude that certain objects (such as the negative contradictand's object) are abnormal. Summarized in English, its premises also check that the positive contradictand has such an implication parent in the same manner that formula 5.21 does, although in the case of this formula, the condition which is checked regarding the negative contradictand is that it was an argument to an `obs` formula. Thus, formula 5.27 will produce exactly the same conclusions of formulas 5.22 and 5.23 as the earlier formula 5.21, albeit after checking different conditions.

Additionally, if the negative contradictand was directly observed by the system, then formula 5.24 will not apply, since Bob lacking a shell was not inferred from Bob belonging to a specialized category of mollusk. A new reinstatement method is needed to restore the belief that $\texttt{obs}(\text{“}\texttt{rel}(\texttt{mollusk}, \texttt{bob})\text{”})$. The following more concise formula can reinstate this contradictand, on the basis of the Alma agent placing a high degree of trust in its observation system:

$$\begin{aligned}&\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\ &\texttt{obs}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”})\\ &\xrightarrow{\texttt{f}} \texttt{reinstate}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\end{aligned} \tag{5.28}$$

Formula 5.28 allows for restoring the negated formula which had been a contradictand. Then in this running example, it restores the inference that Bob is not a shell bearer, and subsequently ALMA will further infer that Bob is abnormal as a mollusk, with respect to not having a shell.

In this section, and the preceding section 5.2.1.2, we introduced four new axioms (i.e., formulas 5.21, 5.24, 5.27, and 5.28) for default reasoning, which can revise beliefs into defaults, enable implications for abnormality, and reinstate contradictions. Next, in section 5.2.2 we turn to a second category of examples, for which we introduce additional default reasoning axioms. These axioms are also compatible with those developed above; we ultimately combine both sets, as well as their matching counterparts presented in the section 5.2.3, for a larger corpus of examples, as presented in section 5.2.4.

5.2.2 Interacting defaults case study

We next consider another category of commonsense reasoning with default formulas, which is distinct from the use of nested ontology categories in the mollusk case study based on Fahlman's example. Here, a natural-language summarization of the case study's high-level knowledge is the following:

1. A car typically is powered.

2. A gas tank car that is fueled is typically powered.

3. A gas tank car that is not fueled is typically not powered.

4. An electric car that is charged is typically powered.

Yet once again, as with the mollusk KB, we consider a starting KB that does not use default knowledge (and rather, expresses non-default formulas knowledge such as "a car *is* powered"). Although gas tank cars and electric are both subcategories of the broader ontology category of car, there is not a subset relationship between the two subtypes, as occurred in the mollusk case study. Hence, novel axioms will be introduced for handling default reasoning in which ontology subcategories are not fully ordered by is-a relations. Following the first few motivating examples below, in section 5.2.4, the axioms are tested on a larger set of examples of varying complexity and temporal ordering.

5.2.2.1 Initial car ontology

We begin with the following ALMA axiom set, which encodes the non-default form of knowledge about the categories of cars, and when they are powered:

$$\texttt{rel}(\texttt{is\_a}, \texttt{gas\_tank\_car}, \texttt{car}) \tag{5.29}$$

$$\texttt{rel}(\texttt{is\_a}, \texttt{electric\_car}, \texttt{car}) \tag{5.30}$$

$$\texttt{rel}(\texttt{car}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{powered}, \texttt{X}) \tag{5.31}$$

$$\texttt{rel}(\texttt{gas\_tank\_car}, \texttt{X}) \wedge \texttt{rel}(\texttt{fueled}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{powered}, \texttt{X}) \tag{5.32}$$

$$\texttt{rel}(\texttt{gas\_tank\_car}, \texttt{X}) \wedge \neg\texttt{rel}(\texttt{fueled}, \texttt{X}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{powered}, \texttt{X}) \tag{5.33}$$

$$\texttt{rel}(\texttt{electric\_car}, \texttt{X}) \wedge \texttt{rel}(\texttt{charged}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{powered}, \texttt{X}) \tag{5.34}$$

These formulas express that gas tank cars and electric cars are narrower categories of the broader class of car, as well as when they are powered. Formula 5.31 appears a bit naive in indicating that all cars are powered; yet this might be a belief formed by an agent that has not yet seen any example of a car that is not powered — and, in the examples developed, the belief will be revised to become more realistic in allowing for exceptions. We also repeat the following four formulas, as introduced in section 5.2.1.1:

$$\texttt{rel}(\texttt{is\_a}, \texttt{X}, \texttt{Y}) \wedge \texttt{rel}(\texttt{is\_a}, \texttt{Y}, \texttt{Z}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{is\_a}, \texttt{X}, \texttt{Z}) \tag{5.16 revisited}$$

$$\texttt{rel}(\texttt{is\_a}, \texttt{A}, \texttt{B}) \wedge \texttt{rel}(\texttt{A}, \texttt{X}) \xrightarrow{\texttt{f}} \texttt{rel}(\texttt{B}, \texttt{X}) \tag{5.17 revisited}$$

$$\texttt{obs}(\texttt{X}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{X}) \tag{5.18 revisited}$$

$$\texttt{abnormal}(\texttt{Obj}, \texttt{Kind}, \texttt{Failure}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{Failure}) \tag{5.19 revisited}$$

5.2.2.2 First example: resolving conflicting categories

The first example in the vehicle domain involves information about a car that is both an electric car and also has a gas tank, as represented with the following:

$$\texttt{rel}(\texttt{electric\_car}, \texttt{prius}) \tag{5.35}$$

$$\texttt{rel}(\texttt{gas\_tank\_car}, \texttt{prius}) \tag{5.36}$$

$$\texttt{rel}(\texttt{charged}, \texttt{prius}) \tag{5.37}$$

$$\neg\texttt{rel}(\texttt{fueled}, \texttt{prius}) \tag{5.38}$$

After one timestep, ALMA will have inferred $\texttt{rel}(\texttt{car}, \texttt{prius})$ (from formulas 5.29, 5.36, and 5.17, as well as a second derivation from 5.30, 5.35, and 5.17), $\neg\texttt{rel}(\texttt{powered}, \texttt{prius})$ (from formulas 5.36, 5.38, and 5.33), and $\texttt{rel}(\texttt{powered}, \texttt{prius})$ (from 5.35, 5.37, and 5.34). At the next timestep, the direct contradiction between beliefs about a Prius being powered or not is detected.

Intuitively, we can see that the desired belief is that of a Prius actually being powered, due to its electric charge. Therefore, the positive contradictand should be reinstated, formula 5.33 should be weakened into a default rule, and the Prius should be able to be concluded as abnormal as a gas tank car. We can see that in this example, the axioms developed so far for updating a non-default formula into a default are insufficient. The Prius was not a subject of direct observation in this example, and so formula 5.27 does not apply; furthermore, usage of a narrowest ontology category cannot justify the desired changes. In fact, 5.24 instead drives the reinstatement of the *negative* contradictand, which must also be addressed. Formula 5.21 also does not apply in this case, since it is dependent upon non-default formulas having a single premise, unlike the formulas seen in this category of example.

We consider further the motivation behind why formula 5.33 should be weakened into a default rule. This formula expresses that a car with a gas tank and a lack of fuel leads to the conclusion that this car is not powered, and in combination with 5.32, expresses that a car is powered *precisely when* it is fueled. However, beyond the related pair of formulas 5.32 and 5.33, there is also the formula 5.34 that presents an additional way that a car might be powered. In this way, formula 5.34 gives an exception to 5.33, since a car that has an empty gas tank but has an electric charge will remain powered. This intuition points the way toward how an update to a non-default fromula may be recognized in problems such as this.

We develop the following general axiom, which also serves the purpose of modifying domain-specific formulas like 5.33 into a default:

$$\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{rel}(\texttt{Kind\_a}, \texttt{Obj}) \wedge \neg\texttt{rel}(\texttt{Prop\_a}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{Kind\_b}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{Prop\_b}, \texttt{Obj})\wedge\\
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind\_a}, \texttt{O}) \wedge \neg\texttt{rel}(\grave{}\texttt{Prop\_a}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”},\\
&\quad \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{pos\_int}(\text{“}\texttt{rel}(\grave{}\texttt{Kind\_a}, \texttt{O}) \wedge \texttt{rel}(\grave{}\texttt{Prop\_a}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”})\wedge\\
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind\_b}, \texttt{O}) \wedge \texttt{rel}(\grave{}\texttt{Prop\_b}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”},\\
&\quad \text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{not\_equal}(\text{“}\texttt{rel}(\grave{}\texttt{Kind\_a}, \texttt{O}) \wedge \texttt{rel}(\grave{}\texttt{Prop\_a}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”},\\
&\quad \text{“}\texttt{rel}(\grave{}\texttt{Kind\_b}, \texttt{O}) \wedge \texttt{rel}(\grave{}\texttt{Prop\_b}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”})
\end{aligned}$$

$$\xrightarrow{\mathtt{f}}$$

$$\mathtt{update}(\text{“}\mathtt{rel}(\grave{}\mathtt{Kind\_a},\mathtt{Ob}) \wedge \neg\mathtt{rel}(\grave{}\mathtt{Prop\_a},\mathtt{Ob}) \xrightarrow{\mathtt{f}} \neg\mathtt{rel}(\grave{}\mathtt{Pred},\mathtt{Ob})\text{”},$$

$$\text{“}\mathtt{rel}(\grave{}\mathtt{Kind\_a},\mathtt{Ob}) \wedge \neg\mathtt{rel}(\grave{}\mathtt{Prop\_a},\mathtt{Ob})\wedge$$

$$\mathtt{neg\_int}(\text{“}\mathtt{abnormal}(\grave{}\mathtt{Ob},\grave{}\grave{}\mathtt{Kind\_a},\text{“}\mathtt{rel}(\grave{}\grave{}\grave{}\mathtt{Pred},\grave{}\grave{}\mathtt{Ob})\text{”})\text{”})$$

$$\xrightarrow{\mathtt{f}} \neg\mathtt{rel}(\grave{}\mathtt{Pred},\mathtt{Ob})\text{”})$$

$$\wedge\, \mathtt{reinstate}(\text{“}\mathtt{rel}(\grave{}\mathtt{Pred},\grave{}\mathtt{Obj})\text{”},\mathtt{T})\wedge$$

$$\mathtt{rel}(\mathtt{Kind\_a},\mathtt{Ab\_Obj}) \wedge \neg\mathtt{rel}(\mathtt{Prop\_a},\mathtt{Ab\_Obj}) \wedge \mathtt{rel}(\mathtt{Pred},\mathtt{Ab\_Obj})$$

$$\xrightarrow{\mathtt{f}} \mathtt{abnormal}(\mathtt{Ab\_Obj},\mathtt{Kind\_a},\text{“}\mathtt{rel}(\grave{}\mathtt{Pred},\grave{}\mathtt{Ab\_Obj})\text{”}) \tag{5.39}$$

First, the axiom must verify the typical beginning of an ongoing contradiction, and after this premise, must check that the object Obj is of types Kind_a and Kind_b. From here, the axiom checks the existence of the pair of formulas which appear in quotes as “$\mathtt{rel}(\grave{}\mathtt{Kind\_a},\mathtt{O}) \wedge \neg\mathtt{rel}(\grave{}\mathtt{Prop\_a},\mathtt{O}) \xrightarrow{\mathtt{f}} \neg\mathtt{rel}(\grave{}\mathtt{Pred},\mathtt{O})$” and “$\mathtt{rel}(\grave{}\mathtt{Kind\_a},\mathtt{O}) \wedge \mathtt{rel}(\grave{}\mathtt{Prop\_a},\mathtt{O}) \xrightarrow{\mathtt{f}} \mathtt{rel}(\grave{}\mathtt{Pred},\mathtt{O})$”, for which the former is a parent of the negative contradictand. In the running example, the bindings are made of Kind_a to gas_tank_car, Prop_a to fueled, and Pred to powered. Next, another premise checks whether there exists a parent formula of the positive contradictand of the form “$\mathtt{rel}(\grave{}\mathtt{Kind\_b},\mathtt{O}) \wedge \mathtt{rel}(\grave{}\mathtt{Prop\_b},\mathtt{O}) \xrightarrow{\mathtt{f}} \mathtt{rel}(\grave{}\mathtt{Pred},\mathtt{O})$”, which provides an alternative way for the positive $\mathtt{rel}(\grave{}\mathtt{Pred},\mathtt{O})$ to be obtained for the object (for which in the example Kind_b is bound to electric_car, and Prop_b is bound to charged — a car that is electric can also be powered if it is charged). The final premise verifies that the pair of formulas with Kind_a and Prop_a versus Kind_b and Prop_b are distinct.

On this basis, the negative contradictand's parent is updated into a default, the positive contradictand is reinstated, and a new implication is inferred so that negative literals may be obtained as abnormal with respect to the relation $\texttt{Pred}$ in question. For the Prius, the domain-specific formula 5.33 is updated into the following:

$$\begin{aligned}&\texttt{rel}(\texttt{gas\_tank\_car}, \texttt{Ob}) \wedge \neg\texttt{rel}(\texttt{fueled}, \texttt{Ob})\wedge\\ &\quad \texttt{neg\_int}(\text{``}\texttt{abnormal}(\text{`}\texttt{Ob}, \texttt{gas\_tank\_car}, \text{``}\neg\texttt{rel}(\texttt{powered}, \text{``}\texttt{Ob})\text{''})\text{''})\\ &\xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{powered}, \texttt{Ob}) \end{aligned} \tag{5.40}$$

This example also makes the reinstatement of the following:

$$\texttt{rel}(\texttt{powered}, \texttt{prius}) \tag{5.41}$$

Lastly, the third conclusion is the following:

$$\begin{aligned}&\texttt{rel}(\texttt{gas\_tank\_car}, \texttt{Ab\_Obj}) \wedge \neg\texttt{rel}(\texttt{fueled}, \texttt{Ab\_Obj}) \wedge \texttt{rel}(\texttt{powered}, \texttt{Ab\_Obj})\\ &\xrightarrow{\texttt{f}} \texttt{abnormal}(\texttt{Ab\_Obj}, \texttt{gas\_tank\_car}, \text{``}\neg\texttt{rel}(\texttt{powered}, \text{`}\texttt{Ab\_Obj})\text{''})\end{aligned} \tag{5.42}$$

This gives the means by which other objects can be inferred to be abnormal as a gas tank car, with respect to being powered despite a lack of fuel.

Yet, as we noted above, simultaneously formula 5.24 drives the reinstatement of the negative contradictand. The pair of conflicting reinstatements will arrive at the same timestep. As a general principle, we do not consider it to be an issue that conflicting reinstatements might arise. We expect that a broadly capable system will have many approaches to handling different contradictions that might appear in its KB, and then realistically a subset of these contradiction-handling formulas will inevitably occasionally conflict. An agent would then be best-served by choosing among a set of inconsistent reinstatement formulas when this happens. As noted in section 4.5.4, ALMA also has an inference rule for detecting a contradiction between `reinstate` formulas. From this, the system is able to catch the conflicting reinstatements in this example for whether the Prius is powered. To handle this higher-level contradiction between the reinstate beliefs, we introduce the following formula:

$$\begin{aligned}&\texttt{contradicting(“reinstate(`X, `Time)”, “reinstate(`Y, `Time)”, T)}\land\\ &\texttt{parent(“rel(is\_a, `Kind, `Kind\_gen)”, “reinstate(`Y, `Time)”, T)}\\ &\xrightarrow{\texttt{f}}\texttt{reinstate(“reinstate(`X, `Time)”, T)}\end{aligned} \tag{5.43}$$

This formula expresses the principle that a reinstate derived from ontology knowledge, such as a reinstate from formula 5.24, can be secondary to another form of reinstatement, such as from formula 5.39. As a result, the contradiction between conflicting reinstatements for whether the Prius is powered or not will be resolved by inferring the following:

$$\texttt{reinstate(“reinstate(“rel(powered, prius)”, 2)”, 3)} \tag{5.44}$$

If the contradictory reinstatements were inferred at timestep 2, this formula reinstates the desired reinstatement, which itself reinstates the appropriate formula about the Prius.

5.2.2.3 Second example: repetition after an update

The previous example, using formula 5.39, made a reinstatement of the literal for a Prius being powered as a conclusion alongside the update to a non-default formula. In a situation where a second example of an electric car arrives after the earlier example, the premises of formula 5.39 will no longer be satisfied, due to being dependent on a non-default formula. We therefore also introduce the following axiom, which serves to enable the appropriate reinstatement in many cases such as this where a new default formula such as formula 5.40 has already been instantiated:

$$\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \text{“}\neg\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T})\wedge \\
&\texttt{parents\_defaults}(\text{“}\neg\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T}) \\
&\texttt{parent\_non\_default}(\text{“}\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T}) \\
&\xrightarrow{\texttt{f}} \texttt{reinstate}(\text{“}\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T})
\end{aligned} \tag{5.45}$$

That is, if a contradiction occurs for which all derivations of one contradictands include a default parent formula, and the other contradictand has a derivation without a default formula, than the latter is reinstated, due to having been inferred from at least one source that is presently considered an absolute formula.

5.2.3 Counterpart axioms

Above, we developed some lengthy axioms for commonsense default reasoning, each of which deals with either updating a non-default formula into a default, or with reinstating a contradictand, or both. Each axiom from this group was specialized to target either a positive or a negative case, due to how we can specify formulas within quotation terms. Thus, each such axiom also has a corresponding counterpart. For completeness, we provide these counterpart axioms below, along with some discussion of each such as an example in which they become relevant.

To motivate in more detail the first counterpart axiom that we present, we note that formula 5.21 is based on the negative contradictand providing evidence for revising a non-default formula into a default. Any pairs of conclusions derived from it also use a negated formula for determining abnormality (as we also see in the specific example conclusions 5.22 and 5.23). The set of axioms developed for default reasoning thus also includes a corresponding axiom that is a counterpart to 5.21, and is instead based around evidence from the *positive* contradictand:

$$
\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \text{“}\neg\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T})\wedge \\
&\texttt{rel}(\texttt{Kind}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{is\_a}, \texttt{Kind\_spec}, \texttt{Kind})\wedge \\
&\texttt{parent}(\text{“}\texttt{rel}(\texttt{`Kind}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{`Pred}, \texttt{O})\text{”}, \text{“}\neg\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T})\wedge \\
&\texttt{parent}(\text{“}\texttt{rel}(\texttt{`Kind\_spec}, \texttt{`Obj})\text{”}, \text{“}\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{”}, \texttt{T}) \\
&\xrightarrow{\texttt{f}} \\
&\texttt{update}(\text{“}\texttt{rel}(\texttt{`Kind}, \texttt{Ob}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{`Pred}, \texttt{Ob})\text{”}, \\
&\quad \text{“}\texttt{rel}(\texttt{`Kind}, \texttt{Ob}) \wedge \texttt{neg\_int}(\text{“}\texttt{abnormal}(\texttt{`Ob}, \texttt{``Kind}, \text{“}\neg\texttt{rel}(\texttt{```Pred}, \texttt{``Ob})\text{”})\text{”}) \\
&\quad \xrightarrow{\texttt{f}} \neg\texttt{rel}(\texttt{`Pred}, \texttt{Ob})\text{”}) \\
&\wedge \\
&\texttt{rel}(\texttt{Kind}, \texttt{Ab\_Obj}) \wedge \texttt{rel}(\texttt{Pred}, \texttt{Ab\_Obj}) \\
&\quad \xrightarrow{\texttt{f}} \texttt{abnormal}(\texttt{Ab\_Obj}, \texttt{Kind}, \text{“}\texttt{rel}(\texttt{`Pred}, \texttt{`Ab\_Obj})\text{”})
\end{aligned}
\tag{5.46}
$$

An example utilizing formula 5.46 is when the following information is added alongside the original cephalopod axiom set of section 5.2.1.2:

$$\texttt{rel}(\texttt{nautilus}, \texttt{nate}) \tag{5.47}$$

A contradiction will be obtained for whether Nate has a shell or not, due to formula 5.14 indicating that nautiluses have shells, while Nate is also a cephalopod which formula 5.13 indicates are not shell-bearers. Formula 5.46 will thus allow weakening 5.13 into a default, and concluding an implication determining when a cephalopod is abnormal as a shell-bearer (that is, an abnormal cephalopod such as Nate actually has a shell). Similarly, the following is a counterpart to 5.24, which is also employed in the example with Nate:

$$\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{neg\_int}(\text{“}\texttt{obs}(\text{“}\neg\texttt{rel}(\grave{}\grave{}\texttt{Pred}, \grave{}\grave{}\texttt{Obj})\text{”})\text{”})\wedge\\
&\texttt{rel}(\texttt{Kind}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{is\_a}, \texttt{Kind}, \texttt{Kind\_gen})\wedge\\
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind}, \grave{}\texttt{Obj})\text{”}, \text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind\_gen}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{non\_ancestor}(\text{“}\texttt{rel}(\texttt{is\_a}, \grave{}\texttt{Kind\_spec}, \grave{}\texttt{Kind})\text{”}, \text{“}\texttt{rel}(\grave{}\texttt{Kind}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\\
&\xrightarrow{\texttt{f}} \texttt{reinstate}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T}) \qquad (5.48)
\end{aligned}$$

In the example, this axiom reinstates the belief that $\texttt{rel}(\texttt{shell\_bearer}, \texttt{nate})$.

The following axiom is the counterpart to 5.27, for the alternative case in which the positive contradictand was a direct observation:

$$\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\wedge\\
&\texttt{obs}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}) \wedge \texttt{rel}(\texttt{Kind}, \texttt{Obj})\wedge\\
&\texttt{parent}(\text{“}\texttt{rel}(\grave{}\texttt{Kind}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})\\
&\xrightarrow{\texttt{f}}
\end{aligned}$$

$$\texttt{update}(``\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Ob}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{Ob})",$$

$$``\texttt{rel}(\grave{}\texttt{Kind}, \texttt{Ob}) \wedge \texttt{neg\_int}(``\texttt{abnormal}(\grave{}\texttt{Ob}, \grave{}\grave{}\texttt{Kind}, ``\texttt{rel}(\grave{}\grave{}\grave{}\texttt{Pred}, \grave{}\grave{}\texttt{Ob})")")$$

$$\xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{Ob})")$$

$$\wedge$$

$$\texttt{rel}(\texttt{Kind}, \texttt{Ab\_Obj}) \wedge \texttt{rel}(\texttt{Pred}, \texttt{Ab\_Obj})$$

$$\xrightarrow{\texttt{f}} \texttt{abnormal}(\texttt{Ab\_Obj}, \texttt{Kind}, ``\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Ab\_Obj})") \tag{5.49}$$

And similarly, the following is an alternative to 5.28, which reinstates a positive contradictand that originated as a direct observation:

$$\texttt{contradicting}(``\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", ``\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", \texttt{T}) \wedge$$

$$\texttt{obs}(``\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})")$$

$$\xrightarrow{\texttt{f}} \texttt{reinstate}(``\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", \texttt{T}) \tag{5.50}$$

The following is the counterpart to formula 5.39:

$$\texttt{contradicting}(``\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", ``\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", \texttt{T}) \wedge$$

$$\texttt{rel}(\texttt{Kind\_a}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{Prop\_a}, \texttt{Obj}) \wedge \texttt{rel}(\texttt{Kind\_b}, \texttt{Obj}) \wedge \neg\texttt{rel}(\texttt{Prop\_b}, \texttt{Obj}) \wedge$$

$$\texttt{parent}(``\texttt{rel}(\grave{}\texttt{Kind\_a}, \texttt{O}) \wedge \texttt{rel}(\grave{}\texttt{Prop\_a}, \texttt{O}) \xrightarrow{\texttt{f}} \texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})",$$

$$``\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", \texttt{T}) \wedge$$

$$\texttt{pos\_int}(``\texttt{rel}(\grave{}\texttt{Kind\_a}, \texttt{O}) \wedge \neg\texttt{rel}(\grave{}\texttt{Prop\_a}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})") \wedge$$

$$\texttt{parent}(``\texttt{rel}(\grave{}\texttt{Kind\_b}, \texttt{O}) \wedge \neg\texttt{rel}(\grave{}\texttt{Prop\_b}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})",$$

$$``\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})", \texttt{T}) \wedge$$

$$\texttt{not\_equal}(``\texttt{rel}(\grave{}\texttt{Kind\_a}, \texttt{O}) \wedge \neg\texttt{rel}(\grave{}\texttt{Prop\_a}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})",$$

$$``\texttt{rel}(\grave{}\texttt{Kind\_b}, \texttt{O}) \wedge \neg\texttt{rel}(\grave{}\texttt{Prop\_b}, \texttt{O}) \xrightarrow{\texttt{f}} \neg\texttt{rel}(\grave{}\texttt{Pred}, \texttt{O})")$$

$$\xrightarrow{f}$$

$$\texttt{update}(\text{``}\texttt{rel}(\texttt{`Kind\_a}, \texttt{Ob}) \wedge \texttt{rel}(\texttt{`Prop\_a}, \texttt{Ob}) \xrightarrow{f} \texttt{rel}(\texttt{`Pred}, \texttt{Ob})\text{''},$$
$$\text{``}\texttt{rel}(\texttt{`Kind\_a}, \texttt{Ob}) \wedge \texttt{rel}(\texttt{`Prop\_a}, \texttt{Ob}) \wedge$$
$$\texttt{neg\_int}(\text{``}\texttt{abnormal}(\texttt{`Ob}, \texttt{``Kind\_a}, \text{``}\neg\texttt{rel}(\texttt{```Pred}, \texttt{``Ob})\text{''})\text{''})$$
$$\xrightarrow{f} \texttt{rel}(\texttt{`Pred}, \texttt{Ob})\text{''})$$
$$\wedge\, \texttt{reinstate}(\text{``}\neg\texttt{rel}(\texttt{`Pred}, \texttt{`Obj})\text{''}, \texttt{T}) \wedge$$
$$\texttt{rel}(\texttt{Kind\_a}, \texttt{Ab\_Obj}) \wedge \texttt{rel}(\texttt{Prop\_a}, \texttt{Ab\_Obj}) \wedge \neg\texttt{rel}(\texttt{Pred}, \texttt{Ab\_Obj})$$
$$\xrightarrow{f} \texttt{abnormal}(\texttt{Ab\_Obj}, \texttt{Kind\_a}, \text{``}\neg\texttt{rel}(\texttt{`Pred}, \texttt{`Ab\_Obj})\text{''}) \qquad (5.51)$$

To see an example which makes use of this axiom, we introduce another axiom for electric cars:

$$\texttt{rel}(\texttt{electric\_car}, \texttt{X}) \wedge \neg\texttt{rel}(\texttt{charged}, \texttt{X}) \xrightarrow{f} \neg\texttt{rel}(\texttt{powered}, \texttt{X}) \qquad (5.52)$$

This new formula, formula 5.34, and formula 5.32 will unify with key premises of axiom 5.51, and will ultimately update formula 5.34 into a default on the basis that there is an alternative to an electric gar being powered solely by charge, since it might also be able to be powered if a gas tank car with fuel. Likewise, this scenario will produce a contradiction between reinstatements, and this axiom will allow a resolution in favor of the latter contradictand, as a counterpart to formula 5.43:

$$\texttt{contradicting}(\text{``}\texttt{reinstate}(\texttt{`X}, \texttt{`Time})\text{''}, \text{``}\texttt{reinstate}(\texttt{`Y}, \texttt{`Time})\text{''}, \texttt{T}) \wedge$$
$$\texttt{parent}(\text{``}\texttt{rel}(\texttt{is\_a}, \texttt{`Kind}, \texttt{`Kind\_gen})\text{''}, \text{``}\texttt{reinstate}(\texttt{`X}, \texttt{`Time})\text{''}, \texttt{T})$$
$$\xrightarrow{f} \texttt{reinstate}(\text{``}\texttt{reinstate}(\texttt{`Y}, \texttt{`Time})\text{''}, \texttt{T}) \qquad (5.53)$$

Finally, the last counterpart axiom provides a counterpart to formula 5.45:

$$\begin{aligned}
&\texttt{contradicting}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T}) \wedge \\
&\texttt{parents\_defaults}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T}) \\
&\texttt{parent\_non\_default}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T}) \\
&\xrightarrow{f} \texttt{reinstate}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Obj})\text{”}, \texttt{T})
\end{aligned} \tag{5.54}$$

An example utilizing formula 5.54 is when the following information is added alongside the original vehicle axiom set of section 5.2.2:

$$\texttt{rel}(\texttt{gas\_tank\_car}, \texttt{ford}) \tag{5.55}$$

$$\neg\texttt{rel}(\texttt{fueled}, \texttt{ford}) \tag{5.56}$$

In this example, once formula 5.31 has been updated into a default, the contradictand $\neg\texttt{rel}(\texttt{powered}, \texttt{ford})$ will be successfully reinstated as the contradictand obtained from a non-default set of parents.

5.2.4 Example corpus and testing

We now combine the axioms developed in prior sections 5.2.1, 5.2.2, and 5.2.3 for a larger range of examples, which test a representative combination of temporal orderings, both for problems with different mollusks, for problems with different vehicles, and for some problems which combine both mollusk and vehicle reasoning. The resulting difficulty of a problem varies depending upon these orderings. However, the set of axioms that we have developed in the above sections for meta-reasoning, and most crucially for updating and reinstatement, are able to successfully handle all characteristic examples of this category, regardless of temporal ordering.

The additional examples for the mollusk domain fall into the following categories:

1. *Additional basic examples* — tests for some simpler cases of examples, which incrementally expand the first few examples presented above. For instance, `cephalopod-repeat` is the same as the example in 5.2.1.2, yet with information of a second cephalopod entering into the KB a few timesteps later, to demonstrate that this can be properly handled even when an update is not performed, since an update was part of the solution in the earlier case. The example `nautilus-repeat` likewise makes this check for two examples of a nautilus over time, where the first nautilus instigates updating, and the second simply uses the updated formulas. Next, `single-specific` begins with presentation of a naked nautilus, and then at successively later timesteps adds in additional knowledge that this same creature is in the wider ontology categories, in order from nautilus to cephalopod to mollusk. Yet, each formula for a wider category was updated into a default when obtaining the creature was naked nautilus, so this example remains effectively quite similar to the cephalopod case in terms of contradiction-handling and updating. The example `single-specific-obs` is very similar to the preceding case, but uses observation formulas and makes use of the alternative for contradiction-handling and updating based upon this. The example `double specific` consists of learning of a naked nautilus, then later that a separate creature is a cephalopod, and again does its updating behavior for each mollusk category when the former arrives, and hence the cephalopod is more easily dealt with here.

2. *Examples of obtaining by observation* — testing the axioms for responding to contradictions and updating based upon one of the formulas being learned by observation. In these cases, the contradiction-handling formulas make reinstatements relatively straightforward. The example scenario files for this group are `learn-by-obs`, which observes a shell-less mollusk Bob as was described in section 5.2.1.3; `learn-by-obs2`, which observes and handles a shell-less cephalopod; and `learn-by-obs3`, which observes and handles a shell-less naked nautilus. We note that for these examples, two contradictions appear in the course of each example. First, there is a contradiction between what is observed and what the ontology categories say (which for these three examples is that the creatures are all mollusks expected to have shells). After the first reinstatement of each creature lacking a shell, a secondary contradiction occurs based on the newly-created default formulas, due to timing where the abnormality of the creature in question cannot be inferred before the default rule incorrectly determines it to have a shell (as was also detailed above, as described in discussion following formula 5.23).

3. *Simultaneous updating* — Examples in which a group of different kinds of mollusks are encountered at once, to demonstrate inference chains of updating and reinstating successfully occurring in parallel, without adverse effects from other ontology categories being altered at the same time. The example `narrower-together` acquires three creatures that are a cephalopod, a nautilus, and a naked nautilus examples at same timestep. In tandem, ALMA obtains contradictions for each, updates the respective formulas about shells or their lack for mollusks, cephalopods, and nautiluses into defaults, reinstates, then again obtains contradictions due to

timing of inferring abnormality, and after this, using formulas 5.21 and 5.46, reinstates the correct formulas, infers the creatures as abnormal, and stops.

In the same way, `narrower-together2` has an equivalent process for a group of mollusk, cephalopod, and nautilus, which makes the cephalopod and mollusk implications into defaults. We note that a belief in the example mollusk having a shell is distrusted when its parent implication is updated, but this is re-derived from the default version shortly thereafter. Similarly, `narrower-together3` has a group of a mollusk, a nautilus, and a naked nautilus, and makes mollusk, cephalopod, and nautilus implications into defaults — demonstrating that even if a mollusk isn't the immediate parent category (as it is not for a nautilus), the formulas still entail weakening it into a default, due to its original status as a non-default rule specifying that all in its category have a shell. Lastly, `narrower-together4` has a group of a mollusk, a cephalopod, and a naked nautilus, and succeeds in updating the formulas for mollusks, cephalopods, and nautiluses, in the same pattern of the other examples of this group.

4. *Narrower categories over time* — In contrast to `single-specific`, learning extra knowledge that creatures are in *narrower* ontology categories at successively later timesteps is a more interesting category of example due to needing to repeatedly update axioms as new information arrives. In `narrower-single`, the example begins with a belief that Steve is a mollusk, and at later timesteps progressively adds that he is also a cephalopod, nautilus, and naked nautilus, in turn. The sequence of updating and reinstating, and then producing another contradiction for a narrower category the timestep after reinstatement, which is resolved in return — which is one of the

most common patterns for these examples, thus continues until all categories have been appropriately updated. The example narrower-later introduces three distinct animals over time, in the order of cephalopod, then naked nautilus, then nautilus; `narrower-later2` introduces a naked nautilus, then a cephalopod, then a nautilus; `narrower-later3` introduces a cephalopod then a naked nautilus — in each of these, the steady-state beliefs obtained are what is desired, and the temporal ordering does not pose an issue for the axioms, even as new animal information arrives while the earlier inference processes continue.

The additional examples for the vehicle domain fall into the following categories:

1. *Additional basic examples* — tests for examples which incrementally expand the first few examples presented above. The example `ford` contains a gas tank car without fuel, and updates formula 5.31 into a default, on the basis of the non-powered Ford. In `non-powered`, the input is a gas tank car with an empty tank, a hybrid with no gas or charge, and an uncharged electric vehicle — which leads to the expected effect of a lone update to formula 5.31, since the other novel updating case for the overlapping defaults of cars occurs when there are reasons to believe a car is driveable. In `crossed`, the vehicles are two hybrids in which one has gas but no charge, and the other has a charge but no gas, which ultimately repairs contradictions by updating the belief that a car is powered and that a gas tank car without gas isn't powered. As occurs in the typical case for these examples, secondary contradictions due to abnormality timing occur and are reinstated appropriately.

2. *Electric car examples* — testing expansions of the scenario such as in section 5.2.2.2 (as is also encoded in example `prius`). The variant `prius2` tests the same example but with information originating from observations; `prius-double` verifies that a second appearance of an electric car after initial updating also succeeds despite a different pattern of being corrected. Lastly, `prius-later` involves one electric car in which there is a temporal delay between obtaining that there is a car and that it is an electric car with charge and no gas; which functions as desired despite temporal delay.

3. *Examples with an electric and gas tank car* — tests varying combinations from each of the overlapping ontology groups appearing together. In `ford-prius`, the belief set is much like `crossed`, although the car with fuel is purely gas-powered, it achieves similar updates and results. A related version is `ford-fueled-prius`, identical to the previous example except that the Prius also is fueled; this leads to an easier solution as a result because this car does not produce a contradiction. In `ford-before-prius`, the belief in the electric Prius occurs later in time, while `ford-prius-later` works similarly and also doubles up the appearance of these examples as an additional test, and `prius-before-ford` intuitively flips the temporal order of the vehicle pair. In all of these examples, the system reasons to the correct final belief set, without additional conflict or failure of the update or reinstatement algorithms to behave as intended in designing them.

And lastly, outside of the two domains tested separately, several additional examples were used to test that the combined mollusk and vehicle domain can produce appropriate results without interference between the beliefs used by either group.

Scripting has also been developed to automate the testing of the example corpus. The formulas specific to each commonsense example are stored in a separate file, and for each example ALMA executes with input of 1) the base axiom set, consisting of formulas 5.16–5.19 and the update and reinstatement axioms developed in previous sections; 2) initial scenario axioms such as formulas 5.9–5.15, or 5.29–5.34, or both; and 3) a set of formulas for the specific example. When ALMA runs using these files, it writes a text log of the print output for each timestep, which provides the history of the knowledge base over time. The test script reconstructs the evolution of the KB's derivation forest from each log file, and compares against a text file of the expected beliefs of ALMA for the respective example. For instance, in the example from section 5.2.1.2, the expected beliefs are:

$$\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve}) \tag{5.57}$$

$$\texttt{abnormal}(\texttt{steve}, \texttt{mollusk}, \text{“}\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{”}) \tag{5.58}$$

$$\texttt{rel}(\texttt{mollusk}, \texttt{steve}) \tag{5.59}$$

$$\texttt{rel}(\texttt{cephalopod}, \texttt{steve}) \tag{5.60}$$

The script verifies that, for each of these formulas, they are inferred by ALMA in the course of execution at some timestep (perhaps repeatedly if they become distrusted and then reinstated), and there does not exist a distrust formula after the latest timestep where the formula is obtained.

In effect, the script for the example corpus provides a unit testing framework of sorts for the process of designing ALMA axioms, in which the expected belief files form test cases. This system proved quite useful in developing the set of update and reinstatement axioms, especially verifying during development that the formulas were suitable for all examples of the corpus. Even with a modest number of axioms, as in the examples of this dissertation, the process of axiom design in effect is quite similar to creation of a logic program — so it is intuitive that an analog to unit testing is a helpful feature to support development with the reasoner.

For a summary of some of the properties of example corpus for the mollusk and vehicle default reasoning domains, we first note that the combined default reasoning axiom set (which includes the updating and reinstatement axioms described above), contains 18 axioms. There are additionally 7 topic-specific axioms for the mollusk domain (i.e., formulas 5.9–5.15), and 6 topic-specific axioms for the vehicle domain (i.e., 5.29–5.34). The lines of output for examples of the mollusk domain ranged from 86 to 191, with an average length of 135 lines. For the vehicle domain, lines of output ranged from 81 to 215, with an average length of 121 lines.

Importantly, each of the corpus examples executed with a brief runtime of at most approximately a tenth of a second of clock time on a Macbook with a 2.7 GHz dual-core processor. Although ALMA was not explicitly optimized to minimize execution speed, nevertheless favorable execution times such as these are obtained for all examples that do not involve extremely large KBs (i.e., the large output of the surprise birthday present examples, which is discussed in section 5.4.3.5).

5.3 Question-answering

Prior work on active logic has invoked a claimed ability of the logic to support inferences making use of past beliefs no longer held, from earlier timesteps before the present moment of reasoning. Similarly, past discussion of Purang's ALMA 1.0 has characterized the prior reasoner as able to reflect on a history of any beliefs that have been retracted or disinherited. In fact, neither of these is strictly true. Any belief that has become disinherited is no longer considered to be trusted or believed from that timestep onward, and must be represented using quotation (i.e., mentioned) to avoid assigning it a truth value.[4] With this limitation, it is less clear what behavior regarding past beliefs ALMA was surmised to have implemented; the documentation of ALMA 1.0 suggests that simply logging formula history may have been misinterpreted as this reasoning capacity.

The importance of a metacognitive agent being able to make use of past beliefs, however, was indicated in section 2.3.1. Formalizing the use of past beliefs requires a means of using nested formulas such as quotation provides. Thus only now can this be adequately supported by the extensions to active logic in the present research, as the expressiveness of active logic is now sufficient for nesting the syntax of past beliefs. Reasoning about past beliefs thus provides further applications of quotation terms and quasi-quotation mechanisms, and also demonstrates how the system has been augmented in a self-referential way.

[4] It may be tempting to assume these formulas are false, but since active logic does not apply the closed world assumption, it is generally not reasonable to assume the negation of retracted beliefs. In particular, formulas distrusted due to inconsistency give a counterexample, where there is a particular reason not to assume that their negation holds.

As described in section 4.5.2.2, ALMA specifically implements the procedures `pos_int_past` and `neg_int_past` that query the system's past beliefs via quotation term arguments. We have developed a set of axioms that endow ALMA with abilities for basic question-answering regarding whether a query formula X is believed by the Alma agent (expressed with `query_belief`), or considered to be true by the Alma agent (expressed with `query_truth`) — which are subtly different in certain cases where the formula is not a present belief of the KB (for example, a formula X that is presently in a state of contradiction is not a current belief, yet the agent has uncertainty as to whether it might be true). Question-answering may be best thought of not as answering a question posed by another agent (human or otherwise), but as self-questioning in which the Alma agent is (agentively) assessing what it does or doesn't know — although precisely when and why an agent asks itself about particular beliefs is beyond the scope of this work. ALMA thus uses the `pos_int_past` and `neg_int_past` procedures as part of question-answering if past beliefs are relevant to the query — alongside other procedures including `acquired`, `pos_int`, and `neg_int`. We note that some of the question-answering axioms rely on the convention of `rel` as a predicate symbol, with its first argument as a constant meant to stand for a more particular predicate symbol.

The two query predicates, `query_belief` and `query_truth`, are binary predicates in which the first argument is the query formula, Q, and the second is the timestep for which the query is asked (which the query will only execute during, so that a query is tied to a particular time). Each query is ultimately answered with a formula of the schema $\texttt{answer}(\text{“}Q\text{”}, A, \texttt{reason}(\text{“}R\text{”}))$, in which A will be one of the constants `yes`, `no`, or `unsure`, and R provides a formula that justifies the answer. Whenever possible, a current belief is provided for this reason R. We now present the full set of question-answering axioms.

5.3.1 Axioms

The first group of axioms concerns when the query argument `X` is presently part of an ongoing state of contradiction. When this is the case, the agent necessarily does not believe `X` at this point in time, because it is distrusted. This is captured by the following formula, which ensures that an unhandled contradicting state applies to `X`, and provides the answer with a reason mentioning this fact:

$$\begin{aligned}&\texttt{query\_belief}(\texttt{X},\texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge \texttt{contradicting}(\texttt{X},\texttt{Y},\texttt{T})\\ &\xrightarrow{\texttt{f}} \texttt{answer}(\text{“}\texttt{query\_belief(`X,`Asktime)}\text{”},\texttt{no},\\ &\quad \texttt{reason}(\text{“}\texttt{contradicting(`X,`Y,`T)}\text{”}))\end{aligned} \tag{5.61}$$

Similarly, a matching formula checks whether `X` was the second argument to an unhandled state of `contradicting`. Likewise, the agent would be unsure whether `X` is true at such a moment when the formula is part of a contradiction. The following formula for a query of `query_truth` is quite similar to how `query_belief` is handled, except for this difference in answer:

$$\begin{aligned}&\texttt{query\_truth}(\texttt{X},\texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge \texttt{contradicting}(\texttt{X},\texttt{Y},\texttt{T})\\ &\xrightarrow{\texttt{f}} \texttt{answer}(\text{“}\texttt{query\_truth(`X,`Asktime)}\text{”},\texttt{unsure},\\ &\quad \texttt{reason}(\text{“}\texttt{contradicting(`X,`Y,`T)}\text{”}))\end{aligned} \tag{5.62}$$

Once again, a paired formula checks whether `X` was the second argument to an ongoing `contradicting` state.

The next axiom group concerns when the query argument is a current belief, and the group's conclusions provide a positive answer for both whether the agent believes this formula and whether it considers it to be true, respectively. Each conclusion expresses in a quoted literal that the queried formula was acquired at a timestep `T`.

$$\begin{gathered}\texttt{query\_belief}(\texttt{X}, \texttt{Asktime}) \land \texttt{now}(\texttt{Asktime}) \land \texttt{pos\_int}(\texttt{X}) \land \texttt{acquired}(\texttt{X}, \texttt{T}) \\ \xrightarrow{\texttt{f}} \texttt{answer}(\text{``}\texttt{query\_belief}(\grave{}\texttt{X}, \grave{}\texttt{Asktime})\text{''}, \texttt{yes}, \texttt{reason}(\text{``}\texttt{acquired}(\grave{}\texttt{X}, \grave{}\texttt{T})\text{''}))\end{gathered} \tag{5.63}$$

In this case, the formulas for `query_belief` and `query_truth` are identical except for the latter using `query_truth` in place of the other query name:

$$\begin{gathered}\texttt{query\_truth}(\texttt{X}, \texttt{Asktime}) \land \texttt{now}(\texttt{Asktime}) \land \texttt{pos\_int}(\texttt{X}) \land \texttt{acquired}(\texttt{X}, \texttt{T}) \\ \xrightarrow{\texttt{f}} \texttt{answer}(\text{``}\texttt{query\_truth}(\grave{}\texttt{X}, \grave{}\texttt{Asktime})\text{''}, \texttt{yes}, \texttt{reason}(\text{``}\texttt{acquired}(\grave{}\texttt{X}, \grave{}\texttt{T})\text{''}))\end{gathered} \tag{5.64}$$

Next, if a query argument is not a current belief, but rather its negation is, then queries can also be answered, although with a negative answer. Because active logic variables do not quantify over wffs, there is not a fully general way that a variable `X` can be used to express the pair of both `X` and $\neg$`X`. Hence, the examples of this kind specify the structure of the query formula nested in quotation, and also do the same for its negation. Intuitively, this check for the KB containing the negation of a query would make the most sense to be done when the query is a literal, and not a more generalized formula (and in fact the negation of a clause may not even remain a clause itself if it has a form other than just a literal, due to the formula standardization in ALMA). Accordingly, the question-answering examples of this group focus on query formulas which consist of a single literal. Specifying the formula structure also means that a separate question-answering rule is necessary to represent each particular arity for a literal. The axiom set has been developed to represent unary, binary, and ternary examples.[5] Below, we provide the unary case for `query_belief`:

$$\begin{aligned}&\texttt{query\_belief}(\text{“}\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Arg})\text{”}, \texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge \\ &\neg\texttt{rel}(\texttt{Pred}, \texttt{Arg}) \wedge \texttt{acquired}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Arg})\text{”}, \texttt{T}) \\ &\xrightarrow{\texttt{f}} \texttt{answer}(\text{“}\texttt{query\_belief}(\text{“}\texttt{rel}(\grave{}\grave{}\texttt{Pred}, \grave{}\grave{}\texttt{Arg})\text{”}, \grave{}\texttt{Asktime})\text{”}, \texttt{no}, \\ &\quad \texttt{reason}(\text{“}\texttt{acquired}(\text{“}\neg\texttt{rel}(\grave{}\grave{}\texttt{Pred}, \grave{}\grave{}\texttt{Arg})\text{”}, \grave{}\texttt{T})\text{”})) \qquad (5.65)\end{aligned}$$

The binary and ternary cases provide instead the literals $\texttt{rel}(\texttt{Pred}, \texttt{Arg1}, \texttt{Arg2})$ and $\texttt{rel}(\texttt{Pred}, \texttt{Arg1}, \texttt{Arg2}, \texttt{Arg3})$, respectively, in each place where formula 5.65 provides its unary `rel` literal. In each occurrence of the binary or ternary literals, their arguments are once again quasi-quoted to be fully-escaping.

Due to formula 5.65 specifying only a positive query literal in its first premise, we also have three distinct axioms used to answer a query regarding a negated literal. The first of these three is the following, which provides a counterpart for formula 5.65:

$$\texttt{query\_belief}(\text{“}\neg\texttt{rel}(\grave{}\texttt{Pred}, \grave{}\texttt{Arg})\text{”}, \texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge$$

[5] We note that these arities refer in the examples to the number of arguments to instances of `rel` which are not the predicate symbol, and hence `rel` itself will have an arity that is one greater. Additionally, our choice to provide axioms of this category only up to queries of ternary arity is based on convenience and not a particular principle; as needed for an application the number of axioms may be expanded for higher arities.

$$\mathtt{rel}(\mathtt{Pred},\mathtt{Arg}) \wedge \mathtt{acquired}(\text{``}\mathtt{rel}(\grave{}\mathtt{Pred},\grave{}\mathtt{Arg})\text{''},\mathtt{T}) \xrightarrow{\mathtt{f}} \mathtt{answer}(\text{``}\mathtt{query\_belief}(\text{``}\neg\mathtt{rel}(\grave{}\grave{}\mathtt{Pred},\grave{}\grave{}\mathtt{Arg})\text{''},\grave{}\mathtt{Asktime})\text{''},\mathtt{no}, \mathtt{reason}(\text{``}\mathtt{acquired}(\text{``}\mathtt{rel}(\grave{}\grave{}\mathtt{Pred},\grave{}\grave{}\mathtt{Arg})\text{''},\grave{}\mathtt{T})\text{''})) \quad (5.66)$$

The binary and ternary versions have a very similar form. Also, there once again exist corresponding axioms for the `query_truth` predicate. Here, these six formulas (for unary, binary, and ternary positive query arguments, and unary, binary, and ternary negative arguments) are once more identical to the six for `query_belief` beyond the predicate symbol.

Yet another category for question-answering consists of when the query argument `X` is not a current belief, but was a belief in the past. In such a case, ALMA records evidence of this through a metareasoning literal expressing both whether `X` was distrusted, retired, or handled disinherited and when it was disinherited. The following formula addresses the query with a negative answer, when `X` was distrusted:

$$\mathtt{query\_belief}(\mathtt{X},\mathtt{Asktime}) \wedge \mathtt{now}(\mathtt{Asktime}) \wedge \mathtt{neg\_int}(\mathtt{X}) \wedge \mathtt{pos\_int\_past}(\mathtt{X},\mathtt{S},\mathtt{E}) \wedge \mathtt{distrusted}(\mathtt{X},\mathtt{E}) \xrightarrow{\mathtt{f}} \mathtt{answer}(\text{``}\mathtt{query\_belief}(\grave{}\mathtt{X},\grave{}\mathtt{Asktime})\text{''},\mathtt{no},\mathtt{reason}(\text{``}\mathtt{distrusted}(\grave{}\mathtt{X},\grave{}\mathtt{E})\text{''})) \quad (5.67)$$

Here, once negative introspection determines the lack of `X`, `pos_int_past` attempts to retrieve the start and end times for its belief interval, so that the ending time will be used in looking up a `distrusted` formula for `X`. Similarly, the variations for when the reason to disinherit `X` was retiring or handling the formula instead use `retired(X, E)` and `handled(X, E)` as the final premise, respectively. When answering with query_truth whether the system considers to be true a belief that is not a current belief but a past belief, each answer instead contains the unsure constant — but otherwise is the same as the corresponding query_belief formula:

$$\texttt{query\_truth}(\texttt{X}, \texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge \texttt{neg\_int}(\texttt{X}) \wedge$$
$$\texttt{pos\_int\_past}(\texttt{X}, \texttt{S}, \texttt{E}) \wedge \texttt{distrusted}(\texttt{X}, \texttt{E})$$
$$\xrightarrow{\texttt{f}} \texttt{answer}(\text{“}\texttt{query\_truth}(\grave{}\texttt{X}, \grave{}\texttt{Asktime})\text{”}, \texttt{unsure}, \texttt{reason}(\text{“}\texttt{distrusted}(\grave{}\texttt{X}, \grave{}\texttt{E})\text{”})) \tag{5.68}$$

Lastly, the query X may fail to be found as both a current belief and a belief in the past (i.e., both neg_int and neg_int_past succeed). This category cannot use a quotation term nesting an existing KB formula to the absence of X. Hence, the novel literal never_believed expresses this information. For query_belief, the answer is negative:

$$\texttt{query\_belief}(\texttt{X}, \texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge \texttt{neg\_int}(\texttt{X}) \wedge \texttt{neg\_int\_past}(\texttt{X}, \texttt{S}, \texttt{E})$$
$$\xrightarrow{\texttt{f}} \texttt{answer}(\text{“}\texttt{query\_belief}(\grave{}\texttt{X}, \grave{}\texttt{Asktime})\text{”}, \texttt{no}, \texttt{reason}(\text{“}\texttt{never\_believed}(\grave{}\texttt{X})\text{”})) \tag{5.69}$$

For query_truth, this information is insufficient for a clear answer, and hence the answer is uncertain:

$$\texttt{query\_truth}(\texttt{X}, \texttt{Asktime}) \wedge \texttt{now}(\texttt{Asktime}) \wedge \texttt{neg\_int}(\texttt{X}) \wedge \texttt{neg\_int\_past}(\texttt{X}, \texttt{S}, \texttt{E})$$
$$\xrightarrow{\texttt{f}}$$
$$\texttt{answer}(\text{“}\texttt{query\_truth}(\grave{}\texttt{X}, \grave{}\texttt{Asktime})\text{”}, \texttt{unsure}, \texttt{reason}(\text{“}\texttt{never\_believed}(\grave{}\texttt{X})\text{”})) \tag{5.70}$$

We can see that in practice, several of different answers for the above axioms may be true at once. For example, if a query F has never been believed, query_truth will return unsure due to formula 5.70. Yet it is also possible that $\neg F$ is a present belief, which will return no due to the query_truth counterpart to formula 5.65. When an answer of unsure exists in combination with an answer of no, the latter negative answer should take precedence. Hence, there is one final axiom which updates the answer containing unsure in a case such as this:

$$\texttt{answer(Query, no, Reason\_a)} \wedge \texttt{acquired(“answer(`Query, no, `Reason\_a)”, T)} \wedge$$
$$\texttt{acquired(“answer(`Query, unsure, `Reason\_b)”, T)} \xrightarrow{\texttt{f}}$$
$$\texttt{update(“answer(`Query, unsure, `Reason\_b)”, “answer(`Query, no, `Reason\_b)”)} \tag{5.71}$$

5.3.2 Examples

As a basic example, we first consider a KB containing the following pair of formulas at timestep 0, which therefore has no prior formulas with foo or its negation:

$$\neg\texttt{foo, x)} \tag{5.72}$$

$$\texttt{query\_truth(“rel(foo, x)”, 0)} \tag{5.73}$$

After one timestep, the following formulas are inferred:

$$\texttt{answer(“query\_truth(“rel(foo, x)”, 0)”, no,}$$
$$\texttt{reason(“acquired(“}\neg\texttt{rel(foo, x)”, 0)”))} \tag{5.74}$$

$$\texttt{answer(“query\_truth(“rel(foo, x)”, 0)”, unsure,}$$
$$\texttt{reason(“never\_believed(“rel(foo, x)”)”))} \tag{5.75}$$

One timestep later, formula 5.71 is applied, since the system has just simultaneously inferred an answer of `no` and `unsure` for the same query. The following inferences result:

$$\begin{aligned}&\texttt{update(“answer(“query\_truth(”rel(foo,x)”,0)”,unsure,}\\&\texttt{reason(“never\_believed(“rel(foo,x)”)”))”,}\\&\texttt{“answer(“query\_truth(“rel(foo,x)”,0)”,no,}\\&\texttt{reason(“never\_believed(“rel(foo,x)”)”))”)}\end{aligned} \tag{5.76}$$

$$\begin{aligned}&\texttt{answer(“query\_truth(“rel(foo,x)”,0)”,no,}\\&\texttt{reason(“never\_believed(“rel(foo,x)”)”))}\end{aligned} \tag{5.77}$$

$$\begin{aligned}&\texttt{distrusted(“answer(“query\_truth(“rel(foo,x)”,0)”,unsure,}\\&\texttt{reason(“never\_believed(“rel(foo,x)”)”))”,2)}\end{aligned} \tag{5.78}$$

In this group, formula 5.76 updates formula 5.75 into a slight variation, via the answer `no`, which appears as formula 5.77. Per the behavior of the procedural atomic `update` predicate, the original formulation is then distrusted, as expressed by formula 5.78.

A second and more involved example is based on adding question-answering queries to the default reasoning example of section 5.2.1.2, in which only a cephalopod is considered, but this still causes a series of changes to the formulas believed. In this example, a query is raised each timestep, where only the timestep argument changs; for example:

$$\texttt{query\_belief(“rel(shell\_bearer,steve)”,3)} \tag{5.79}$$

As the beliefs regarding the cephalopod Steve change over time, we see the query answers change as well.

The first query in effect occurs at timestep 3, when the first contradiction between $\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ and $\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ has occurred. In this case, due to the contradiction, the query is answered as follows, on the basis of formulas 5.61 and 5.67, respectively:

$$\begin{aligned}&\texttt{answer}(\text{``}\texttt{query\_belief}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 3)\text{''}, \texttt{no},\\ &\texttt{reason}(\text{``}\texttt{contradicting}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''},\\ &\quad \text{``}\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 3)\text{''}))\end{aligned} \tag{5.80}$$

$$\begin{aligned}&\texttt{answer}(\text{``}\texttt{query\_belief}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 3)\text{''}, \texttt{no},\\ &\texttt{reason}(\text{``}\texttt{distrusted}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 3)\text{''}))\end{aligned} \tag{5.81}$$

The answers to the query remain the above pair until timestep 6, at which point both $\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ and $\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})$ have been freshly re-derived at the same timestep. As a result, the queries are answered in a contradictory manner, where each cites the positive and negative formula, respectively, due to formulas 5.63 and 5.65, respectively:

$$\begin{aligned}&\texttt{answer}(\text{``}\texttt{query\_belief}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 5)\text{''}, \texttt{yes},\\ &\texttt{reason}(\text{``}\texttt{acquired}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 5)\text{''}))\end{aligned} \tag{5.82}$$

$$\begin{aligned}&\texttt{answer}(\text{``}\texttt{query\_belief}(\text{``}\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 5)\text{''}, \texttt{no},\\ &\texttt{reason}(\text{``}\texttt{acquired}(\text{``}\neg\texttt{rel}(\texttt{shell\_bearer}, \texttt{steve})\text{''}, 5)\text{''}))\end{aligned} \tag{5.83}$$

However, when the contradiction between the `shell_bearer` pair is also detected at timestep 6, the above pair of query answers are also distrusted.

At timestep 8, the negative contradictand has been reinstated, and thus at timestep 9 the query answer reflects this fact, that the positive contradictand remains distrusted as a belief, while its negation is now believed (and these can again be attributed to 5.65 and 5.67, respectively):

$$\texttt{answer(“query\_belief(“rel(shell\_bearer, steve)”, 8)”, no,}$$
$$\texttt{reason(“acquired(“}\neg\texttt{rel(shell\_bearer, steve)”, 8)”))} \tag{5.84}$$
$$\texttt{answer(“query\_belief(“rel(shell\_bearer, steve)”, 8)”, no,}$$
$$\texttt{reason(“distrusted(“rel(shell\_bearer, steve)”, 6)”))} \tag{5.85}$$

At this point, the cephalopod knowledge has reached a steady state, and ALMA idles.

5.3.3 Toward explainability

We also note that the basic question-answering ability for ALMA that has been presented above also appears to be applicable to pursuing explainable reasoning. Explainable artificial intelligence is an area of much focus and work, and introspection and question-answering are well-suited to supporting this. Between the capacity for answering self-queries regarding the reason for a specific belief, an ALMA-based agent could also combine the usage of `query_belief` or `query_true` with ancestry procedural predicates such as `ancestor` or `parent` to recursively investigate the origin of older beliefs. Although it is not pursued in the present work, it would be a natural extension of this work to combine these features into a more detailed self-explanatory ability which can account for a thread of reasoning.

5.4 Modeling agent beliefs: the surprise birthday present

We now turn to an example scenario in which ALMA is applied to reason about both self and other agents, using the agent belief modeling partitions of the system as were developed in section 4.7.1. Thus, the case study requires modeling the beliefs and inferences of fellow agents — which pushes the use of formula nesting and quotation beyond a focus that until now was mostly turned inward in representing the agent's own internal beliefs. This study demonstrates an additional larger use of quotation, which is pervasive in attributing the modeled beliefs to other agents.

We consider a form of a commonsense reasoning problem originally described by Davis [15], which in its original formulation regarded the axiomatization of two agents, Alice and Bob, attempting to plan giving a surprise gift to a third agent, Carol, on her birthday two weeks in the future.

Morgenstern [59] developed a first-order monotonic axiomatization for a version of the problem, yet a primary objective of that work was on developing a formalized theory of surprise, and the multi-agent planning aspects of the problem. The way that Morgenstern's work represents agent knowledge is through an external method, with a god's eye view of what an agent will know in time. The work also does not include any of the agents modeling the beliefs of the other two. A recent book surveying commonsense psychology also cited the surprise birthday present problem [26], showing it is still recognized as notable, despite comparatively little attention.

For our purposes, the present work instead deals with how the active-logic-based agent Alma (which replaces the role of Alice in the example, for convenience in naming) might model the beliefs of Bob and Carol, which is a novel angle of the problem. Hence, we put aside both formalizing surprise and developing multi-agent planning, the latter of which ALMA 2.0 is presently not designed to support.

The narrower focus of the group of scenarios that we develop based around this problem concerns what follows when Alma tells Bob of having made the decision to give Carol a cake:

$$\texttt{tell}(\texttt{alma}, \text{“}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{”}, \texttt{bob}) \tag{5.86}$$

We consider that Alma and Bob want to surprise Carol, and the issue is how Alma can represent and reason about what Alma believes are Bob's and Carol's evolving beliefs, as Alma and Bob have made this decision. For the part of Alma modeling Carol, if Alma believes Carol is nearby to be able to overhear what is told to Bob, then intuitively Carol ought to be able to determine that the gift is coming — and by expecting the gift, Carol fails to be surprised. Conversely, if Alma does not believe Carol to have had an opportunity to overhear, then Carol will not be modeled as ultimately unsurprised. Additionally, if Alma considers Bob to have much the same knowledge of the situation, then Alma ought to also model Bob reaching the same conclusion about whether Carol fails to be surprised.

We are thus considering a need for Alma to be in possession of certain knowledge relevant to this commonsense situation — such as when an agent might hear information, when an agent is not surprised due to an expectation of a gift, and so on. But additionally, Alma should also be able to attribute to Bob and Carol the same knowledge of the situation. In fact, beyond that, there is a case of further nesting reflected in the English summary: if Alma models Bob's concluding that Carol is unsurprised, then Alma should also be able model Bob has attributing knowledge of the situation to Carol too.

5.4.1 Attributing common knowledge

We would like Alma to treat a set of formulas for the problem scenario as common knowledge (at least among Alma, Bob, and Carol) and to also model other agents as recursively treating axioms as common knowledge themselves (which will allow a model of Bob to successfully further model Carol's beliefs for the scenario). We next detail a series of axioms which enable this process for common knowledge, in which we represent a belief that is common knowledge as a quoted formula within the `common_knowledge` predicate. First, as a base case, Alma itself naturally must be able to believe formulas of common knowledge:

$$\texttt{common\_knowledge(X)} \xrightarrow{\texttt{f}} \texttt{true(X)} \tag{5.87}$$

From this formula, after another timestep the value of `X` will be withdrawn. Then, for another agent (whose existence Alma indicates through the unary `agent` literal, and whose name is distinct from the Alma agent's own name, which is expressed using `agentname`) must also be modeled so that what Alma considers to be common knowledge is believed to be this other agent's own respective common knowledge. This implication itself is also a piece of common knowledge:

$\texttt{common\_knowledge(“}$

$\texttt{agent(Agent)} \land \texttt{neg\_int(“agentname(`Agent)”)} \land \texttt{common\_knowledge(X)}$

$$\xrightarrow{f} \texttt{bel}(\texttt{Agent}, \text{“}\texttt{common\_knowledge(`X)}\text{”})$$

$$\text{”}) \tag{5.88}$$

The preceding formula allows Alma to pass what it believes to be common knowledge into each agent KB that Alma holds, due to the conclusion of the form $\texttt{bel}(\texttt{Agent}, \text{“}\texttt{common_knowledge(`X)}\text{”})$, which per the design for agent KBs will instantiate a belief into the KB for `Agent`. To enable the transfer of common knowledge down into more-deeply nested agent KBs, the following axiom is also needed:

$$\begin{aligned}&\texttt{common\_knowledge}(\text{“}\\ &\quad \texttt{agent(Agent)} \land \texttt{neg\_int}(\text{“}\texttt{agentname(`Agent)}\text{”})\\ &\quad \xrightarrow{f} \texttt{bel}(\texttt{Agent}, \text{“}\texttt{common\_knowledge(X)} \xrightarrow{f} \texttt{true(X)}\text{”})\\ &\text{”})\end{aligned} \tag{5.89}$$

This axiom takes a role for nested agent KBs similar to what 5.87 does for the top-level Alma belief set, by allowing modeled agents to hold their own beliefs of the truth of common knowledge.

The above set of three axioms related to common knowledge thus avoids in the axiom set for Alma a need for repetition of distinct formulas formulas attributing knowledge of the situation to Bob or Carol. Alma simply must be aware of the existence of its fellow agents, and believe that scenario axioms are common knowledge, and the beliefs will be appropriately passed to the models of Bob and Carol.

Similarly, we also model cases in which an agent such as Alma considers knowledge to be common among a group of agents, but not universally believed as common knowledge by all. In this case, the group that is modeled as sharing a set of beliefs can be built up as a collection of pairwise shared beliefs, using the `pair_knowledge` predicate. If agents A and B are believed to have a belief shared between the two of them, then this means that other agents have meta-knowledge indicating that A and B to each believe that formula (and, this fact about paired knowledge is common knowledge):

$$\begin{aligned}
&\texttt{common\_knowledge}(``\\
&\quad \texttt{pair\_knowledge}(\texttt{Agent\_a}, \texttt{Agent\_b}, \texttt{Bel}) \wedge \texttt{agentname}(\texttt{Self})\\
&\quad \texttt{not\_equal}(``\texttt{agent}(\grave{}\texttt{Agent\_b})", ``\texttt{agent}(\grave{}\texttt{Self})")\\
&\quad \xrightarrow{\texttt{f}} \texttt{bel}(\texttt{Agent\_b}, ``\texttt{pair\_knowledge}(\grave{}\texttt{Agent\_a}, \grave{}\texttt{Agent\_b}, \grave{}\texttt{Bel})")\\
&")
\end{aligned} \tag{5.90}$$

$$\begin{aligned}
&\texttt{common\_knowledge}(``\\
&\quad \texttt{pair\_knowledge}(\texttt{Agent\_a}, \texttt{Agent\_b}, \texttt{Bel}) \wedge \texttt{agentname}(\texttt{Self})\\
&\quad \texttt{not\_equal}(``\texttt{agent}(\grave{}\texttt{Agent\_a})", ``\texttt{agent}(\grave{}\texttt{Self})")\\
&\quad \xrightarrow{\texttt{f}} \texttt{bel}(\texttt{Agent\_a}, ``\texttt{pair\_knowledge}(\grave{}\texttt{Agent\_a}, \grave{}\texttt{Agent\_b}, \grave{}\texttt{Bel})")\\
&")
\end{aligned} \tag{5.91}$$

The above pair checks that the pair knowledge is not involving the current agent before inferring the conclusion, so that an agent does not attempt to model its own beliefs within `bel`. To fill this gap, the following formulas deal with the case where the current agent is part of the pair:

$$\begin{aligned}
&\texttt{common\_knowledge}(``\\
&\quad \texttt{agentname}(\texttt{Self}) \wedge \texttt{pair\_knowledge}(\texttt{Self}, \texttt{Other}, \texttt{Bel}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{Bel})
\end{aligned}$$

$$\text{”)} \tag{5.92}$$

$$\texttt{common\_knowledge}(\text{“}$$

$$\texttt{agentname}(\texttt{Self}) \wedge \texttt{pair\_knowledge}(\texttt{Other}, \texttt{Self}, \texttt{Bel}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{Bel})$$

$$\text{”)} \tag{5.93}$$

5.4.2 Birthday surprise ontology

The following axioms constitute the initial beliefs of an agent for the specific problem of giving a surprise birthday present. The first few axioms model how the other agents are considered by Alma to have heard the utterance that begins each example (formula 5.86). First, as an assumption for another agent than the current agent, if this agent in question isn't near the speaker of an utterance (and isn't the speaker), by default the conclusion to be made is that this other agent didn't hear the utterance:

$$\texttt{tell}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Confidant}) \wedge \texttt{agent}(\texttt{Agent}) \wedge$$

$$\texttt{neg\_int}(\text{“}\texttt{agentname}(\grave{}\texttt{Agent})\text{”}) \wedge \texttt{neg\_int}(\text{“}\texttt{near}(\grave{}\texttt{Agent}, \grave{}\texttt{Speaker})\text{”})$$

$$\xrightarrow{\texttt{f}} \neg\texttt{heard}(\texttt{Agent}, \texttt{Utterance}, \texttt{Speaker}) \tag{5.94}$$

Second, any agent near the speaker will have heard the utterance. This includes the confidant, although typically not the speaker unless they are considered near to themself:

$$\texttt{tell}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Confidant}) \wedge \texttt{agent}(\texttt{Agent}) \wedge \texttt{near}(\texttt{Agent}, \texttt{Speaker})$$

$$\xrightarrow{\texttt{f}} \texttt{heard}(\texttt{Agent}, \texttt{Utterance}, \texttt{Speaker}) \tag{5.95}$$

Third, the speaker always hears their own utterance:

$$\texttt{tell}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Confidant}) \wedge \texttt{agent}(\texttt{Speaker})$$

$$\xrightarrow{\texttt{f}} \texttt{heard}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Speaker}) \tag{5.96}$$

The next axioms model how agents that heard an utterance may additionally attribute hearing (or not hearing) to other agents. Additionally, remaining axioms from from this point onward all are expressed as common knowledge as well. First, an agent that has heard an utterance from a speaker considers that other agents near the speaker have also heard the utterance:

$$\texttt{common\_knowledge}(``$$

$$\texttt{agentname}(\texttt{Self}) \wedge \texttt{heard}(\texttt{Self}, \texttt{Utterance}, \texttt{Speaker}) \wedge \texttt{near}(\texttt{Agent}, \texttt{Speaker})$$

$$\xrightarrow{\texttt{f}} \texttt{heard}(\texttt{Agent}, \texttt{Utterance}, \texttt{Speaker}$$

$$")\tag{5.97}$$

Second, an agent that has heard an utterance from a speaker considers the speaker to have also heard the utterance:

$$\texttt{common\_knowledge}(``$$

$$\texttt{agentname}(\texttt{Self}) \wedge \texttt{heard}(\texttt{Self}, \texttt{Utterance}, \texttt{Speaker})$$

$$\xrightarrow{\texttt{f}} \texttt{heard}(\texttt{Speaker}, \texttt{Utterance}, \texttt{Speaker})$$

$$")\tag{5.98}$$

Third, an agent that has heard an utterance from a speaker considers by default that other agents not known to be near the speaker have not heard the utterance:

$$\texttt{common\_knowledge}(``$$

$$\texttt{agentname}(\texttt{Self}) \wedge \texttt{heard}(\texttt{Self}, \texttt{Utterance}, \texttt{Speaker}) \wedge \texttt{agent}(\texttt{Agent})$$

$\texttt{not\_equal(“agent(`Agent), “agent(`Speaker)”)}\wedge$

$\texttt{neg\_int(“near(`Agent, `Speaker)”)}$

$\xrightarrow{\texttt{f}} \neg\texttt{heard(Agent, Utterance, Speaker)}$

$\texttt{”)}$ (5.99)

Alongside the formulas for attribution of hearing (or not hearing), there are formulas expressing the consequences of this, whether positively hearing or its absence. We assume that agents are credulous and believe what they hear in our scenarios, without considering reasoning related to falsehood (and thus, the giving of a surprise birthday gift does not depend on deceiving Carol). Then, the following expresses how an agent believes what is heard:

$\texttt{common\_knowledge(“}$

$\texttt{agentname(Self)} \wedge \texttt{heard(Self, Utterance, Speaker)} \xrightarrow{\texttt{f}} \texttt{true(Utterance)}$

$\texttt{”)}$ (5.100)

An agent might believe another heard an utterance for several reasons, such as in formulas described above (e.g., 5.95). When this is the case, the conclusion is that other agents would believe in what the present agent expects them to have heard:

$\texttt{common\_knowledge(“}$

$\texttt{agentname(Self)} \wedge \texttt{heard(Agent, Utterance, Speaker)}$

$\texttt{not\_equal(“agent(`Agent)”, “agent(`Self)”)}$

$\xrightarrow{\texttt{f}} \texttt{bel(Agent, “heard(`Agent, `Utterance, `Speaker)”)}$

$\texttt{”)}$ (5.101)

Similarly, if another agent is considered by the present agent to not have heard an utterance, this other agent is modeled as lacking a belief in the utterance:

$$\begin{aligned}
&\texttt{common\_knowledge}(``\\
&\quad \texttt{agentname(Self)} \wedge \neg\texttt{heard(Agent,Utterance,Speaker)}\\
&\quad \texttt{not\_equal}(``\texttt{agent(`Agent)}", ``\texttt{agent(`Self)}") \xrightarrow{\texttt{f}} \neg\texttt{bel(Agent,Utterance)}\\
&")
\end{aligned} \tag{5.102}$$

Lastly, given the usage in the axioms of assumptions that an agent has not heard, based on negative introspection and a lack of present knowledge, naturally a contradiction may arise on over the issue of hearing an utterance. Since in the present work we do not focus on an agent being mistaken about what it has directly heard, it is straightforward to give a preference to the positive contradictand as a contradiction response:

$$\begin{aligned}
&\texttt{common\_knowledge}(``\\
&\quad \texttt{contradicting}(``\texttt{heard(`Agent,`Bel,`Speaker)}",\\
&\qquad ``\neg\texttt{heard(`Agent,`Bel,`Speaker)}", \texttt{T})\\
&\quad \xrightarrow{\texttt{f}} \texttt{reinstate}(``\texttt{heard(`Agent,`Bel,`Speaker)}", \texttt{T})\\
&")
\end{aligned} \tag{5.103}$$

Next are the critical two axioms for determining when there will be a failure to surprise Carol with the gift. First, there is the common knowledge that a decision for a gift to be given to an agent will lead to this agent expecting to receive the gift:

$$\begin{aligned}
&\texttt{common\_knowledge}(``\\
&\quad \texttt{decision(Giver,give(Gift,Agent))} \xrightarrow{\texttt{f}} \texttt{expectation(receive(Agent,Gift))}
\end{aligned}$$

$$\text{”)} \tag{5.104}$$

The expectation that a certain agent will receive a gift might be believed by any agent. Yet, when the expectation is held by the ultimate recipient of the gift, who moreover has heard about the decision to give it to them, this recipient will not be surprised when they receive their gift:

$$\begin{aligned}&\texttt{common\_knowledge}(\text{“}\\&\quad \texttt{expectation}(\texttt{receive}(\texttt{Recipient}, \texttt{Gift}))\land\\&\quad \texttt{heard}(\texttt{Recipient}, \text{“}\texttt{decision}(\texttt{`Giver}, \texttt{give}(\texttt{`Gift}, \texttt{`Agent}))\text{”}, \texttt{Speaker})\\&\quad \xrightarrow{\texttt{f}} \neg\texttt{future\_surprise}(\texttt{Recipient}, \texttt{gift}(\texttt{Gift}))\\&\text{”)}\end{aligned} \tag{5.105}$$

Inference of the conclusion of axiom 5.105 by an agent model of Carol thus naturally represents a failure of the desired result by Alice and Bob.

Lastly, there is a small collection of axioms necessary for the scenarios to function. It is common knowledge that the three agents exist, that being near is a symmetric relation, and also that each agent believes and is identified by their own name:

$$\texttt{common\_knowledge}(\text{“}\texttt{agent}(\texttt{alma})\text{”}) \tag{5.106}$$

$$\texttt{common\_knowledge}(\text{“}\texttt{agent}(\texttt{bob})\text{”}) \tag{5.107}$$

$$\texttt{common\_knowledge}(\text{“}\texttt{agent}(\texttt{carol})\text{”}) \tag{5.108}$$

$$\texttt{common\_knowledge}(\text{“}\texttt{near}(\texttt{A}, \texttt{B}) \xrightarrow{\texttt{f}} \texttt{near}(\texttt{B}, \texttt{A})\text{”}) \tag{5.109}$$

$$\begin{aligned}&\texttt{common\_knowledge}(\text{“}\\&\quad \texttt{agent}(\texttt{Agent}) \land \texttt{neg\_int}(\text{“}\texttt{agentname}(\texttt{`Agent})\text{”})\end{aligned}$$

$$\xrightarrow{f} \texttt{bel}(\texttt{Agent}, \text{``}\texttt{agentname(`Agent)}\text{''})$$

$$\text{''}) \tag{5.110}$$

5.4.3 Solutions and discussion

Now, we return to solutions to the surprise birthday problem, in which the modeling of agent beliefs enables Alma to infer about failures of surprise if Carol overhears crucial information. Once again, all scenarios which we discuss begin with the following formula of Alma telling an utterance to Bob:

$$\texttt{tell}(\texttt{alma}, \text{``}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{''}, \texttt{bob}) \qquad \text{(5.86 revisited)}$$

In practice when these scenarios are executed by ALMA, for each axiom of the form `common_knowledge`("X"), due to the axioms for inferring the truth of common knowledge such as 5.87 for Alma and 5.89 within a modeled agent's KB, there is a several step derivation of `true`("X") following, and then X alone as a belief. For simplicity, we refer to X as a formula extracted from common knowledge, in relation to `common_knowledge`("X").

As with the earlier examples in section 5.2, the subsequent discussion is at a high level, without exhaustive detail of inferences per example. Full traces for ALMA execution can be found in the GitHub repository.

5.4.3.1 Alma's perspective

The first example scenario we consider for the problem is when all three agents are near each other:

$$\texttt{common\_knowledge}(\text{“}\texttt{near}(\texttt{alma}, \texttt{bob})\text{”}) \tag{5.111}$$

$$\texttt{common\_knowledge}(\text{“}\texttt{near}(\texttt{bob}, \texttt{carol})\text{”}) \tag{5.112}$$

$$\texttt{common\_knowledge}(\text{“}\texttt{near}(\texttt{alma}, \texttt{carol})\text{”}) \tag{5.113}$$

This intuitively should ultimately produce a failure to surprise Carol, since she will be able to overhear the utterance and should make inferences based upon what is heard.

First, we break down how reasoning proceeds for Alma expecting that Carol should be unsurprised. From formula 5.96 that a speaker hears their utterances, and formulas 5.86 and 5.106, Alma infers the following:

$$\texttt{heard}(\texttt{alma}, \text{“}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{”}, \texttt{alma}) \tag{5.114}$$

From formula 5.114, the axiom $\texttt{agentname}(\texttt{alma})$ built into the reasoner so that it has access to its own name, and Alma's belief of formula 5.100 extracted from common knowledge, Alma infers $\texttt{true}(\text{“}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{”})$, and subsequently believes its nested formula:

$$\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol})) \tag{5.115}$$

Formula 5.115 satisfies the premise of 5.104, producing this conclusion indicating that Alma expects Carol to receive the gift:

$$\texttt{expectation}(\texttt{receive}(\texttt{carol}, \texttt{cake})) \tag{5.116}$$

In a second derivation thread, from formula 5.95 that an agent near a speaker overhears, and additional formulas 5.86, 5.108, and 5.113, Alma infers that Carol has also heard the utterance:

$$\texttt{heard}(\texttt{carol}, \text{``}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{''}, \texttt{alma}) \tag{5.117}$$

Yet, due to the timing of inference and the need to unpack formula 5.113 from `common_knowledge`, at the timestep when 5.117 is inferred, one timestep earlier its negation has been inferred based on the formula 5.94 and a lack of knowledge at that timestep that Carol was near. A contradiction thus occurs between whether Carol has heard or not heard, which is subsequently resolved by formula 5.103. Following this reinstatement, between the reinstated formula 5.117, as well as formulas 5.116 and 5.105 extracted from common knowledge, Alma infers that Carol won't be surprised by receiving the gift:

$$\neg\texttt{future\_surprise}(\texttt{carol}, \texttt{gift}(\texttt{cake})) \tag{5.118}$$

5.4.3.2 Carol's perspective

We next address how the agent KB modeling Carol (which is itself a model that is part of Alma's KB) is employed for the first scenario with formulas 5.111–5.113, and trace how reasoning proceeds in Alma's KB partition for the modeling of the positive beliefs of Carol. This process begins with an inference from Alma rather than the model of Carol — from `agentname(alma)` and formulas 5.101 and 5.117, leading to Carol modeled as believing that she heard Alma's utterance:

$$\texttt{bel}(\texttt{carol}, \text{``}\texttt{heard}(\texttt{carol}, \text{``}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{''}, \texttt{alma})\text{''}) \tag{5.119}$$

This belief is synchronized into the agent KB modeling the positive beliefs of Carol, as $\texttt{heard}(\texttt{carol}, \text{“}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{”}, \texttt{alma})$, per ALMA mechanisms introduced in section 4.7.2. Additionally, Alma models Carol as having access to her own name, via formulas 5.108 and 5.110, which lead to the following inference:

$$\texttt{bel}(\texttt{carol}, \text{“}\texttt{agentname}(\texttt{carol})\text{”}) \tag{5.120}$$

This is also synchronized into the partitioned region for the model of Carol, as $\texttt{agentname}(\texttt{carol})$. And lastly, for the final Alma inference that affects the model of Carol for the scenario, from formulas 5.108 and 5.89, Alma infers that Carol will be able to extract common knowledge:

$$\texttt{bel}(\texttt{carol}, \text{“}\texttt{common\_knowledge}(\texttt{X}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{X})\text{”}) \tag{5.121}$$

Then, the $\texttt{common\_knowledge}(\texttt{X}) \xrightarrow{\texttt{f}} \texttt{true}(\texttt{X})$ is instantiated for Carol's positive KB — and from here the model of Carol can extract formulas from common knowledge in the same manner as is used by Alma itself to withdraw the formulas and believe them.

The inferences themselves from Carol's perspective continue in the nested KB for her beliefs. From the combination of her belief in having heard the utterance, her belief in her name, and a belief in formula 5.100 as extracted from common knowledge, the modeled Carol infers that $\texttt{true}(\text{“}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{”})$, and subsequently the formula that was nested as its argument:

$$\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol})) \tag{5.122}$$

Where we note that, in contrast to 5.115, formula 5.122 is specific to Carol, and hence Alma would hold the belief of $\texttt{bel}(\texttt{carol}, \text{“}\texttt{decision}(\texttt{alma}, \texttt{give}(\texttt{cake}, \texttt{carol}))\text{”})$, which is instantiated and connected to formula 5.122 by an equivalence edge.

Carol's belief in her copy of the commonsense implication 5.104, alongside formula 5.122, produces an inference that she expects to receive the gift:

$$\texttt{expectation}(\texttt{receive}(\texttt{carol}, \texttt{cake})) \tag{5.123}$$

Unlike the perspective for Alma, there is not a contradiction reached regarding `heard` literals in the model of Carol — and therefore, from formula 5.123, Carol's equivalent of 5.119, and Carol's belief of 5.105 extracted from common knowledge, Carol also infers that she won't be surprised by receiving the gift:

$$\neg\texttt{future\_surprise}(\texttt{carol}, \texttt{gift}(\texttt{cake})) \tag{5.124}$$

Hence, in addition to Alma ultimately reasoning that Carol should not be surprised when she ultimately receives the gift, Alma also successfully models that *Carol herself* can reason out this same fact as a consequence of being nearby and able to overhear Alma's utterance about the gift. Formula 5.124 will instantiate into the core KB the following formula, which we contrast with formula 5.118 to highlight the difference between the result from the model of Carol with the reasoning without her model:

$$\texttt{bel}(\texttt{carol}, \text{“}\neg\texttt{future\_surprise}(\texttt{carol}, \texttt{gift}(\texttt{cake}))\text{”}) \tag{5.125}$$

In fact, in this contrast we see a difference that resembles the distinction we have drawn between internal and external accounts of reasoning; formula 5.124 is an internal representation for the model simulating Carol's reasoning processes about a failure of surprise, while formula 5.118 is an external representation outside of the model of Carol.

5.4.3.3 Bob's perspective of Carol

In the same running scenario, as another agent near Alma when the utterance is made, Bob is also modeled by Alma as hearing the utterance, and then modeled as believing the set of common knowledge formulas following their synchronization into Bob's agent KB. Thus, the inference process also leads to Bob's model expecting Carol to be unsurprised in the future (i.e., Bob's model producing its own final inference of the form of formula 5.124), along essentially the same lines. This is instantiated into the core KB as follows:

$$\texttt{bel}(\texttt{bob}, \text{``}\neg\texttt{future\_surprise}(\texttt{carol}, \texttt{gift}(\texttt{cake}))\text{''}) \tag{5.126}$$

More interesting is the fact that, due to beliefs of Bob's model including a copy of formula 5.88, and the common knowledge of the other agents and their positioning near each other, the model of Bob will also further be able to attribute these common knowledge beliefs to a more deeply-nested agent KB for Carol. Thus, the process of inference for Bob's model having its own inner model of Carol proceeds with the same inference process as was used by Alma modeling Carol and broken down in section 5.4.3.2 — which unfolded identically since Alma and Bob have the same information about agent proximity. After sufficient timesteps, Bob's model of Carol infers formula 5.124, Bob's model obtains the equivalent of 5.125, and the core KB obtains the following:

$$\texttt{bel}(\texttt{bob}, \text{``}\texttt{bel}(\texttt{carol}, \text{``}\neg\texttt{future\_surprise}(\texttt{carol}, \texttt{gift}(\texttt{cake}))\text{''})\text{''}) \tag{5.127}$$

ALMA's pattern of exhaustive reasoning from the available beliefs and prospects that can be obtained from them continues even for agent KB reasoning. Therefore, we emphasize that each agent model's reasoning, as well as the reasoning of more deeply-nested models, can all progress simultaneously at the same timestep. However, since there is a longer temporal chain of inference for inner agent models, the core KB will obtain a formula such as 5.118 at timestep 8, before 5.125 at timestep 12 and formula 5.127 later still at timestep 14. Alma will thus acquire the belief Carol modeled as failing to be surprised before the model of Bob reasons fully to this conclusion. This opens an opportunity in future work in which Alma can anticipate Bob's model shortly making the same conclusions. More broadly, patterns of reasoning of this kind illustrate how the ALMA reasoner has the capability for reasoning to be carried out by one agent over time as its own beliefs — as well as those of other agents — are undergoing inferential changes, and have those changes be reflected upon.

5.4.3.4 Other scenarios

For a second scenario, we consider the following to be the only knowledge agents have of positioning relative to each other:

$$\texttt{common\_knowledge}(\text{``}\texttt{near}(\texttt{alma}, \texttt{bob})\text{''}) \tag{5.128}$$

Without either Alma or Bob possessing a belief about Carol nearby, both Alma and its model of Bob derive respective beliefs that they expect Carol to eventually receieve her gift (i.e., $\texttt{expectation}(\texttt{receive}(\texttt{carol}, \texttt{cake}))$, obtained in the same derivation as described above for Alma and Bob). Yet, they do not infer that Carol will not be surprised, since by formula 5.105 Carol must also be considered to have heard the utterance.

Finally in a third scenario, we consider a case where it is common knowledge that Alma is nearby Bob, but where only Alma and Carol have shared pair knowledge expressing that Carol is also near Alma and Bob. In a physical space, this may be for a reason such as Carol having sneaked up behind Bob, while Alma faces both Bob and Carol. These are the three axioms for the scenario:

$$\texttt{common\_knowledge}(\text{“}\texttt{near}(\texttt{alma}, \texttt{bob})\text{”}) \tag{5.129}$$

$$\texttt{pair\_knowledge}(\texttt{alma}, \texttt{carol}, (\text{“}\texttt{near}(\texttt{bob}, \texttt{carol})\text{”}) \tag{5.130}$$

$$\texttt{pair\_knowledge}(\texttt{alma}, \texttt{carol}, \text{“}\texttt{near}(\texttt{alma}, \texttt{carol})\text{”}) \tag{5.131}$$

Since Alma and Carol know each other to be near, Alma is able to model Carol such that formula 5.118 is inferred. Similarly, Carol's model is able to conclude that formula 5.124 is true. However, once again Bob has no ability to reach such a conclusion.

5.4.3.5 Discussion

Collectively, the example solutions demonstrate the ability of ALMA to successfully model agent reasoning of some complexity, through the use of reasoner partitions for agent belief modeling. Achieving this relied on the mechanism of representing as common knowledge both the axioms for the scenario, as well as general meta-reasoning axioms about common knowledge. Yet importantly, the active logic reasoning for variations of the surprise birthday present problem did not require attributing inference rules to the modeled agents, and quoted beliefs for common knowledge sufficed for the examples. We again note that this can be attributed to the design of the nested agent KBs described in section 4.7.1.

Our approach to aspects of the surprise birthday present problem is novel for both the focus on agents modeling each other's reasoning, and for the manner in which we achieve this, centered on the agent KB partitions and the synchronization of equivalently-linked formulas into the core ALMA KB. The problem thus provides another example of how evolving-time reasoning and quotation both can work together to facilitate commonsense reasoning, particularly practical examples for reasoning about what inferences another agent might draw.

The design of the axioms for common knowledge leads an issue due to the recursive instantiation of agent common knowledge beliefs. We see that the implication 5.88 yields the conclusion that a piece of common knowledge should be believed by another agent that is known (which here we refer to as agent A). If there doesn't exist an agent KB for A, it will be created, and the KB for A will be populated with common knowledge such as formula 5.88 itself. Then, whenever the model of A has a belief that another agent B exists, its own modeled belief in formula 5.88 will be satisfied, and begin to model B. Here there is a possibility of infinite recursion for instantiating ever-deeper agent KBs.

This occurs in our examples, since the axioms above have also taken the existence of the three agents to be common knowledge (as formulas 5.106, 5.107, and 5.108) — which was necessary for the supporting both Alma modeling Carol as well as the model of Bob going further to model Carol. In fact, with our set of axioms for this problem, any model of an agent will eventually model the other two agents (e.g., Carol will eventually model Bob and Alma), leading to an exponential increase in the set of modeled agent beliefs as more timesteps pass and every agent model continues to expand additional, more deeply-nested models. The surprise birthday present instances surveyed are able to attain the final conclusions outlined above, such as formulas 5.118 and 5.125, before the exponential growth is prohibitively large. However, to allow the ALMA reasoner to terminate, we have implemented a somewhat blunt fix of a parameter for the system that limits the maximum recursive depth for nesting of an agent KB. As a result, when ALMA is set to infer automatically, it eventually exhausts all inferences up to its specified limit of N levels of nesting, and terminates. When this max nesting depth parameter is set to 3, ALMA still achieves the intended results for modeling Bob, Carol, and Bob's model of Carol, as discussed in detail above. A more general solution than this is discussed as an avenue of future work in section 6.2.2.2.

While in section 6.2.2.2, we separately identify a limitation that is related to ALMA modeling other agents as quite ALMA-like in terms of their inference rules and even executable procedures. Yet here, we also point out an upside of this similarity that ALMA attributes to the agents it models. Suppose the agent Alma is interacting with another agent, and modeling them as possessing ALMA-like inference abilities. If some disparities are revealed where this other agent acts in a manner that diverges from what ALMA's inference rules suggest for the agent model, this may provide an opening for the Alma agent to point out a new perspective to the other agent. Or, if the other agent's beliefs diverge in a way that doesn't conform to valid rules of inference, Alma might be able to identify an apparent mistake in the other agent's reasoning, as in the motivating vision of Perlis et al. [66].

We now summarize some properties of the surprise birthday present problem examples, both for the axioms and for execution. The initial axiom set includes 7 axioms of general common knowledge (i.e., formulas 5.87–5.93), 19 axioms specific to the surprise birthday problem (17 of which are detailed as formulas 5.94–5.110), and finally additional axioms which present a specific scenario (e.g., such as indicating which agents are near each other). The lines of output for the surprise birthday present examples, in which the depth of maximum agent belief nesting was set to 3 as described above, ranged from 5197 to 6139, with an average length of 5640 lines. Despite the size of the knowledge bases, and each producing several thousand lines of output, each of the agent-modeling examples nevertheless executed with a runtime still under a second of clock time — with the slowest execution taking 0.96 seconds.

Chapter 6: Conclusions and future directions

We conclude by reviewing the main contributions of this dissertation and results obtained, as well as identifying some limitations that offer future directions in which the work might be extended.

6.1 Summary of results

6.1.1 Quotation and quasi-quotation

The present work provided meaningful advancement for active logic beyond the prior state of work in this category of logic outlined in section 2.2, through the development of formula nesting and quotation. This builds on earlier work in active-logic treatments where evolving time is central, but where agency and quotation were not given explicit formalization. In addition, we developed a full unification algorithm for quotation and quasi-quotation, based upon the general principles drawn in an example-driven manner from the categories of terms, which exhaustively accounts for the categories of term pairings.

As a contrast, we note that the literature for syntactic theories of quotation reviewed in section 2.3.2 primarily presented syntactic details on their respective methods of quantifying into quotation, rather than algorithmic details. This surveyed work also differs significantly in the basics of their approaches to nesting formulas in comparison to quotation terms, and consequently in the definition of quantifying-in methods between the approaches. Thus, our approach is novel both for defining quasi-quotation as specializing the quotation term construct, but also for a focus on practical example-driven commonsense reasoning, on providing a full inference algorithm, and on incorporation into a time-sensitive reasoning engine.

6.1.2 ALMA 2.0

This dissertation also presented ALMA 2.0, a reasoning engine for active logic which also implemented the novel extensions to active logic of quotation and quasi-quotation. ALMA 2.0 supports wide-ranging automated reasoning with active logic, as evidenced by the applications discussed in this dissertation, in addition to other research which employed the reasoner, such as by Clausner et al. [14] and Brody et al. [12]. Furthermore, it exhibits novel features enabled by the ability of quotation, such as more fine-grained self-reflection via introspection on the presence or lack of formulas in the ALMA agent's beliefs, as well as metacognitive access to other information within the ALMA agent's internal reasoning, such as formula derivations and ancestry, and past beliefs. Additionally, the belief model partition regions of ALMA provide a novel means of simulating beliefs attributed to other agents, while utilizing quotation to remain just beyond a first-order language. Particularly in contrast to the previously-developed active logic reasoner, the new system is more extensible in design for incorporating new features, and better suited to integration into larger systems (as also is demonstrated by the work of Clausner et al. and Brody et al.).

6.1.3 Commonsense reasoning applications

Finally, this dissertation presented the application of the new developments for active logic, as well as ALMA 2.0, to a set of commonsense reasoning problems:

1. Several traditional types of "nested default" problem, but solved with active logic's temporally evolving constraints requiring knowledge-retractions beyond the reach of standard methods — as well as variations on the above where categories can overlap, resulting in greater need for retractions, as additional tests of the approach.

2. A self-questioning capacity that allows an agent to answer queries regarding present and past beliefs.

3. A treatment of a problem on reasoning about modeled beliefs of others, attending to the actual reasoning carried out by one agent over time as its own beliefs, as well as those of other agents, are undergoing inferential changes.

The work in these areas, particularly for the forms of default reasoning and agent belief modeling, revealed that a surprisingly great deal of knowledge representation, as well as of temporally sensitive bookkeeping devices, are required for human-level agent-based commonsense reasoning. These appear to be largely independent of any scenario-specific details of the reasoning. While this dissertation has by no means captured the full extent of such knowledge representation, it has made a serious start in several areas. Furthermore, we now provide an outline of further aspects needing study.

6.2 Future work

We identify and discuss a series of issues recognized as some limitations of the current work. Extending active logic and ALMA 2.0 to address these is beyond the scope of this dissertation, yet these remain important problems for future work.

6.2.1 Quotation and quasi-quotation

Regarding ALMA's representation for quotation terms, we have noted in chapter 4 that presently, ALMA reasoning does not support the cases when a nested formula cannot be rewritten into a single clause. This limits the expressiveness of certain forms of formulas within quotation terms. However, if this limitation were to be lifted, and a quotation term could contain a truly arbitrary nested formula, it would add complications to unification and reasoning, which presently consider the only possible exceptions to formulas being in clause form to be a forward-if formula's set of conclusions. Additionally, the meaning of a predicate of a form such as `pos_int`("$\texttt{foo(X)} \land \texttt{bar(Y)}$") is not immediately clear, if $\texttt{foo(X)} \land \texttt{bar(Y)}$ cannot be a single KB formula when the ALMA belief set is standardized into conjunctive normal form. Related to the expressiveness of quoted formulas, it would also be a helpful advance to develop quotation terms in future work for an option to express verbatim content in a quotation term's nested formula, without being subject to rewriting rules of the ALMA reasoner.

Throughout the present work, we have by design restricted quotation terms to exclusively contain a nested complete formula, beginning from the creation of the newly developed ALMA grammar. Future work might also move such a construction of a quotation term, in which there could exist a type of nesting construction for quoting a partial formula, such as just a term of functional expression.

6.2.2 Commonsense reasoning applications

Next, we break down several limitations in the applications for commonsense reasoning, which may be addressed in future work.

6.2.2.1 Default reasoning

As we saw in the example scenarios developed in section 5.2, in some cases ALMA may produce contradictory reinstatements when reinstating following the original contradiction that had occurred. While the system's ability to detect direct contradictions has been extended to also distrust pairs of reinstatements that target both contradictands and thereby avoid some cases of this problem, it remains possible that an ALMA agent might still enter into infinite loops of reinstatement in other situations. Ideally, an ALMA-based agent should be able to automatically avoid these problems in a more thorough manner, and perhaps select its contradiction responses in a more sophisticated manner than the axioms developed in the present work.

Furthermore, while the contradiction-response formulas developed in section 5.2 have aimed to capture a few relatively broad intuitions (such as concerning trust in observations, or favoring formulas which have a derivation from a non-default formula), further progress may be made on axiom design. Future work should further investigate the development of contradiction-response rules that have minimal interference between pairs of such formulas, while simultaneously having the ability to handle a large range of contradiction cases.

6.2.2.2 Agent-modeling

Through the development of ALMA's modeling of the beliefs of other agents, the assumption was implicitly made that other agents use the same inference rules as ALMA, due to the application of the system's own inference rules for the agent KB partitions. However, this assumption is not reasonable in all cases; even if there might be a small core of simpler inference rules that an agent could broadly expect other agents to have (although this would be a defeasible assumption), more complicated rules such as resolution, or specific procedures such as for introspection, may fall outside of this category. Progress on attributing inference rules is then an important problem for ALMA being able to better model other agents and their reasoning.

In section 5.4, we saw that the system for attributing beliefs as common knowledge to agents being modeled was prone to swamping with ever-deeper nesting, due to the recursive nature of these formulas. Our means of stopping this is presently a blunt method of specifying a maximum nesting depth for agent modeling, which was sufficient to permit enough nesting for the birthday problem solutions of interest. However, there is a need for a more sophisticated fix than is provided by such a depth limit. Several approaches appear promising avenues for doing so in future work.

The first approach is to augment ALMA to better pursue goal-directed reasoning. In this case, the system would have a particular goal such as regarding Bob or Carol's belief about the birthday present. Then, one option for goal-directed reasoning would be to stop its reasoning process with the respective agent KBs when the system obtains the answer; or alternatively to use the goal of solving the birthday present problem as a premise for unpacking common knowledge into a deeper-nested agent KB, thereby better controlling how more agents are recursively modeled. Yet regardless of the approach, the intention would be that just as a reasonable agent would not indefinitely produce the infinite closure from expanding the axioms for elementary arithmetic, ALMA would use goals to avoid regress with agent modeling. The second approach is to introduce a new means for controlling the number of inferences that the reasoner makes per timestep. Brody et al. [12] developed a neural network-based architecture for ALMA to control inferential glut via learning a neural heuristic for inferences to be considered, which showed promise in preliminary studies. Such an approach could be pursued on a larger scale for additional scenarios, including modeling agent beliefs, where it would give greater flexibility than a limit on nesting depth.

6.2.3 ALMA-based agency

The present work has made progress on a number of aspects related to pursuing a more robust notion of agency for an agent's internal reasoning. However, this treatment is far from fully general. A number of aspects which suggest themselves as extensions of this work, as natural further steps to pursue, are the following:

1. Extending the ability of an ALMA-based agent to reason with its past beliefs in a deeper way, such as by reasoning using Allen's interval relations [2].

2. Augmenting the logic and system's awareness of reasoning about change in the external world, time passing, and fluents, which remains underdeveloped for active logic despite its strengths at the inference process being situated in time.

3. Improved ability of an active logic system to reason about its performance of actions in time as they occur, and to have greater self-knowledge about events the agent is involved in.

Bibliography

[1] N I Adams, IV, D H Bartley, G Brooks, R K Dybvig, D P Friedman, R Halstead, C Hanson, C T Haynes, E Kohlbecker, D Oxley, K M Pitman, G J Rozas, G L Steele, Jr, G J Sussman, M Wand, and H Abelson. Revised report on the algorithmic language Scheme. *SIGPLAN Notices*, 33(9):26–76, September 1998.

[2] James F Allen. Towards a general theory of action and time. *Artif. Intell.*, 23(2):123–154, July 1984.

[3] Michael Anderson and Donald Perlis. Logic, Self-Awareness and Self-Improvement: The metacognitive loop and the problem of brittleness. *J. Logic Comput.*, 15(1):21–40, 2005.

[4] Michael L Anderson, Walid Gomaa, John Grant, and Don Perlis. Active logic semantics for a single agent in a static world. *Artif. Intell.*, 172(8):1045–1063, May 2008.

[5] Michael L Anderson, Tim Oates, Waiyian Chong, and Don Perlis. The metacognitive loop I: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *J. Exp. Theor. Artif. Intell.*, 18(3):387–411, 2006.

[6] Mikael Asker and Jacek Malec. Reasoning with limited resources: Active logics expressed as labelled deductive systems. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, pages 69–78, 2005.

[7] Alan Bawden. Quasiquotation in lisp. In *Partial Evaluation and Semantic-Based Program Manipulation*, pages 4–12. Citeseer, 1999.

[8] Niclas Braun, Stefan Debener, Nadine Spychala, Edith Bongartz, Peter Sörös, Helge H O Müller, and Alexandra Philipsen. The senses of agency and ownership: A review. *Front. Psychol.*, 9:535, 2018.

[9] Justin Brody, Michael T Cox, and Donald Perlis. The processual self as cognitive unifier. In *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy*, 2013.

[10] Justin Brody, Michael T Cox, and Donald Perlis. Incorporating elements of a processual self into active logic. In *2014 AAAI Spring Symposium Series*, 2014.

[11] Justin Brody, Don Perlis, and Jared Shamwell. Who's Talking?–Efference copy and a robot's sense of agency. In *2015 AAAI Fall Symposium Series*, 2015.

[12] Justin D Brody, Bobby Austin, Omar Khater, Christopher Maxey, Matthew D Goldberg, Timothy Clausner, Darsana Josyula, and Donald Perlis. Using neural networks to control glut in the active logic machine. In *NeSy'20/21: Workshop on Neuro-Symbolic Learning and Reasoning*, 2021.

[13] Waiyian Chong. *Reflective Reasoning.* PhD thesis, University of Maryland, College Park, 2006.

[14] Timothy Clausner, Christopher Maxey, Matthew D Goldberg, Paul Zaidins, Justin Brody, Darsana Josyula, and Don Perlis. Overgenerality from inference in Perspective-Taking. In *Proceedings of the AAAI Fall Symposium 2021*, 2021.

[15] Ernest Davis. The surprise birthday present problem. https://commonsensereasoning.org/problem_page.html#surprisebirthday, 2001. Accessed: 2021-8-25.

[16] Ernest Davis. Logical formalizations of commonsense reasoning: a survey. *J. Artif. Intell. Res.*, 59:651–723, 2017.

[17] Jennifer Drapkin, Michael Miller, and Donald Perlis. A memory model for real-time common sense reasoning. Technical Report TR-86-21, Systems Research Center, University of Maryland, 1986.

[18] Jennifer Elgot-Drapkin, Michael Miller, and Donald Perlis. Memory, reason and time: The Step-Logic approach. In Robert C Cummins, editor, *Philosophy and Ai*, pages 79–103. Cambridge: MIT Press, 1991.

[19] Jennifer J Elgot-Drapkin. Step-Logic and the Three-Wise-Men problem. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 412–417, 1991.

[20] Jennifer J Elgot-Drapkin and Donald Perlis. Reasoning situated in time i: basic concepts. *J. Exp. Theor. Artif. Intell.*, 2(1):75–98, January 1990.

[21] Jennifer Jill Elgot-Drapkin. *Step-logic: Reasoning Situated in Time.* PhD thesis, University of Maryland at College Park, College Park, MD, USA, 1988.

[22] Ronald Fagin, Yoram Moses, Joseph Y Halpern, and Moshe Y Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

[23] Scott E Fahlman, David S Touretzky, and Walter Van Roggen. Cancellation in a parallel semantic network. In *IJCAI*, pages 257–263, 1981.

[24] Matthew D Goldberg, Justin Brody, Timothy C Clausner, and Donald Perlis. The overlooked role of self-agency in artificial systems. In *Workshop on Shortcomings in Vision and Language at European Conference on Computer Vision*, 2018.

[25] Matthew D Goldberg, Darsana Josyula, and Don Perlis. Quotation for real-time metacognition. In *Advances in Cognitive Systems*, 8, 2020.

[26] Andrew S Gordon and Jerry R Hobbs. *A Formal Theory of Commonsense Psychology: How People Think People Think*. Cambridge University Press, September 2017.

[27] Andrew R Haas. *Planning mental actions*. PhD thesis, University of Rochester, 1982.

[28] Andrew R Haas. The syntactic theory of belief and knowledge. Technical Report 5368, Bolt Beranek and Newman Inc., 1983.

[29] Andrew R Haas. A syntactic theory of belief and action. *Artif. Intell.*, 28(3):245–292, 1986.

[30] Andrew R Haas. Sentential semantics for propositional attitudes. *Comput. Linguist.*, 16(4):213–233, December 1990.

[31] Emily Hand, Darsana Josyula, Matthew Paisner, Elizabeth McNany, Donald Perlis, and Michael T Cox. Two approaches to implementing metacognition. In *The Sixth International Conference on Advanced Cognitive Technologies and Applications*, 2014.

[32] Thorben Ole Heins. A case study of active logic. *Master's thesis, Department of Computer Science, Lund University*, 2009.

[33] Kaarlo Jaakko Juhani Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Ithaca, NY, USA: Cornell University Press, 1962.

[34] Daniel Holden. Micro parser combinators. https://github.com/orangeduck/mpc, 2018.

[35] Darsana Josyula, Michael Anderson, and Donald Perlis. Towards domain-independent, task-oriented, conversational adequacy. In *Proceedings of the Eighteenth international Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1637–1638, January 2003.

[36] Darsana P Josyula, Scott Fults, Michael L Anderson, Shomir Wilson, and Don Perlis. Application of MCL in a dialog agent. In *Third Language and Technology Conference*, 2007.

[37] Darsana P Josyula, Matthew D Goldberg, Anthony Herron, Christopher Maxey, Paul Zaidins, Timothy Clausner, Justin Brody, and Don Perlis. Knowledge of self and other within a broader commonsense setting. In *Proceedings of the AAAI Fall Symposium 2021*, 2021.

[38] Darsana Purushothaman Josyula. *A unified theory of acting and agency for a universal interfacing agent.* PhD thesis, University of Maryland, College Park, 2005.

[39] Kurt Konolige. A First-Order formalization of knowledge and action for a multiagent planning system. Technical Report 232, SRI International Menlo Park CA Artificial Intelligence Center, 1980.

[40] Robert Kowalski. Database updates in the event calculus. *j. log. program.*, 12(1-2):121–146, January 1992.

[41] Robert Kowalski and Fariba Sadri. The situation calculus and event calculus compared. In *Proceedings of the 1994 International Symposium on Logic programming*, pages 539–553. unknown, January 1994.

[42] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, March 1986.

[43] Gerhard Lakemeyer and Hector J Levesque. Only-Knowing meets nonmonotonic modal logic. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 350–357, May 2012.

[44] Hector J Levesque. All I know: A study in autoepistemic logic. *Artif. Intell.*, 42(2):263–309, March 1990.

[45] Hector J Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases.* MIT Press, February 2001.

[46] Vladimir Lifschitz. Computing circumscription. In *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 1*, IJCAI'85, pages 121–127, San Francisco, CA, USA, August 1985. Morgan Kaufmann Publishers Inc.

[47] Vladimir Lifschitz. Circumscription. In *Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*, pages 297–352. Oxford University Press, Inc., USA, April 1994.

[48] Geoffrey Mainland. Why it's nice to be quoted: quasiquoting for Haskell. In *Proceedings of the ACM SIGPLAN workshop on Haskell workshop*, Haskell '07, pages 73–82, New York, NY, USA, September 2007. Association for Computing Machinery.

[49] Jacek Malec. Active logic and practice. In *Linköping Electronic Conference Proceedings*, pages 49–53. Linköping University Electronic Press, 2009.

[50] Andrea Mazzoleni. TommyDS. https://github.com/amadvance/tommyds/, 2018.

[51] John McCarthy. Circumscription — a form of Non-Monotonic reasoning. *Artif. Intell.*, 13:27–39, 1980.

[52] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.*, 28(1):89–116, 1986.

[53] John McCarthy and Patrick J Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bonnie Lynn Webber and Nils J Nilsson, editors, *Readings in Artificial Intelligence*, pages 431–450. Morgan Kaufmann, January 1981.

[54] J-J Ch Meyer and W van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press, 1995.

[55] Michael Miller and Donald Perlis. Presentations and this and that: logic and action. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, USA, 1993.

[56] Michael J Miller. *A View of One's Past and Other Aspects of Reasoned Change in Belief*. PhD thesis, University of Maryland at College Park, College Park, MD, USA, 1993.

[57] Leora Morgenstern. A first order theory of planning, knowledge, and action. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 99–114, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[58] Leora Morgenstern. *Foundations of a logic of knowledge, action, and communication*. PhD thesis, New York University, 1988.

[59] Leora Morgenstern. A first-order axiomatization of the surprise birthday present problem: Preliminary report. In *Proceedings of the Seventh International Symposium on Logical Formalizations of Commonsense Reasoning*, 2005.

[60] Erik T Mueller. *Commonsense Reasoning: An Event Calculus Based Approach*. Morgan Kaufmann, November 2014.

[61] Madhura Nirkhe. *Time-situated Reasoning Within Tight Deadlines and Realistic Space and Computation Bounds*. PhD thesis, University of Maryland at College Park, College Park, MD, USA, 1995.

[62] Madhura Nirkhe. How to (plan to) meet a deadline between now and then. *J. Logic Comput.*, 7(1):109–156, February 1997.

[63] Madhura Nirkhe, Sarit Kraus, and Don Perlis. Thinking takes time: A modal active-logic for reasoning in time. In *Proceedings of the Bar Han Symposium On Foundations of AI*, page 11, 1995.

[64] Madhura Nirkhe, Sarit Kraus, and Donald Perlis. Situated reasoning within tight deadlines and realistic space and computation bounds. In *Proceeedings of the Second Symposium on Logical Formalizations of Commonsense Reasonin*, 1993.

[65] Madhura Nirkhe, Donald Perlis, and Sarit Kraus. Reasoning about change in a changing world. In *Proceedings of FLAIRS'93*, 1993.

[66] Don Perlis, Clifford Bakalian, Justin Brody, Timothy Clausner, Matthew D Goldberg, Adam Hamlin, Vincent Hsiao, Darsana Josyula, Chris Maxey, David Sekora, Jared Shamwell, and Jesse Silverberg. Live and learn, ask and tell: Agents over tasks. In *International Workshop on Spoken Dialogue Systems, Special Session on Dialog Systems and Lifelong Learning*, Siracusa, Italy, 2019.

[67] Don Perlis, Justin Brody, Sarit Kraus, and Michael Miller. The internal reasoning of robots. In *Thirteenth International Symposium on Commonsense Reasoning*, 2017.

[68] Donald Perlis. *Language, Computation, and Reality.* PhD thesis, University of Rochester, 1981.

[69] Donald Perlis. Languages with self-reference I: Foundations. *Artif. Intell.*, 25(3):301–322, 1985.

[70] Donald Perlis. Languages with self-reference II: Knowledge, belief, and modality. *Artif. Intell.*, 34(2):179–212, March 1988.

[71] Donald Perlis. Meta in logic. In *Meta-Level Architectures and Reflection*, pages 37–49. Elsevier Science Publishers BV, North-Holland, 1988.

[72] Donald Perlis. Consciousness as self-function. *Journal of Consciousness Studies*, 4(5-6):509–525, 1997.

[73] A Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, October 1977.

[74] Khemdut Purang. Alma/Carne: implementation of a time-situated metareasoner. In *Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001*, pages 103–110, November 2001.

[75] Khemdut Purang. *Systems that detect and repair their own mistakes.* PhD thesis, University of Maryland, College Park, 2001.

[76] W V Quine. Quantifiers and propositional attitudes. *J. Philos.*, 53(5):177–187, 1956.

[77] Willard Quine. *Mathematical Logic, Revised Edition.* Harvard University Press, 1981.

[78] Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[79] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, July 2001.

[80] Bryan Renne. *Dynamic Epistemic Logic with justification.* PhD thesis, City University of New York, Ann Arbor, United States, 2008.

[81] Matthew Schmill, Michael Anderson, Scott Fults, Darsana Josyula, Tim Oates, Don Perlis, Hamid Shahri, Shomir Wilson, and Dean Wright. The metacognitive loop and reasoning about anomalies. In Michael T Cox and Anita Raja, editors, *Metareasoning: Thinking about Thinking*, pages 183–198. MIT Press, March 2011.

[82] Matthew Schmill, Darsana Josyula, Michael Anderson, Shomir Wilson, Tim Oates, and Don Perlis. Ontologies for reasoning about failures in AI systems. In *Proceedings from the Workshop on Metareasoning in Agent Based Systems at the Sixth International Joint Conference on Autonomous Agents and Multiagent Sytems*, 2007.

[83] Stuart C Shapiro. The SNePS semantic network processing sytem. In Nicholas V Findler, editor, *Associative Networks*, pages 179–203. Academic Press, January 1979.

[84] Stuart C Shapiro. The CASSIE projects: An approach to natural language competence. In *EPIA 89*, pages 362–380. Springer Berlin Heidelberg, 1989.

[85] Stuart C Shapiro. Embodied cassie. In *Cognitive Robotics: Papers from the 1998 AAAI Fall Symposium, Technical Report FS-98*, volume 2, pages 136–143. aaai.org, 1998.

[86] Stuart C Shapiro and Jonathan P Bona. The GLAIR cognitive architecture. *International Journal of Machine Consciousness*, 02(02):307–332, 2010.

[87] Stuart C Shapiro, William J Rapaport, Michael Kandefer, Frances L Johnson, and Albert Goldfain. Metacognition in SNePS. *1*, 28(1):17–31, March 2007.

[88] Guy Steele. *Common LISP: The Language.* Elsevier, June 1990.

[89] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic.* Springer Netherlands, May 2007.

[90] M M Vinkov and I B Fominykh. Argumentation semantics for active logic step theories with granulation of time. *Scientific and Technical Information Processing*, 43(5):346–350, December 2016.

[91] Mikhail M Vinkov and Igor B Fominykh. Stepping theories of active logic with two kinds of negation. *Advances in Electrical and Electronic Engineering*, 15(1):84–92–92, March 2017.

[92] Franciscus Petrus Johannes Maria Voorbraak. *As far as I know: Epistemic logic and uncertainty*. PhD thesis, Utrecht University, 1993.

[93] Pei Wang. *Non-axiomatic Logic: A Model of Intelligent Reasoning*. World Scientific, 2013.

[94] Pei Wang and Patrick Hammer. Issues in temporal and causal inference. In *Artificial General Intelligence*, pages 208–217. Springer International Publishing, 2015.

[95] Pei Wang, Xiang Li, and Patrick Hammer. Self in NARS, an AGI system. *Front. Robot. AI*, 5, 2018.

[96] Yanjing Wang. Beyond knowing that: A new generation of epistemic logics. In Hans van Ditmarsch and Gabriel Sandu, editors, *Jaakko Hintikka on Knowledge and Game-Theoretical Semantics*, pages 499–533. Springer International Publishing, Cham, 2018.

[97] Mark Whitsey. Logical omniscience: A survey. Technical Report NOTTCS-WP2003-2, School of Computer Science and IT, University of Nottingham, 2003.