Computational Intelligence, Volume 16, Number 2, 2000

TACKLING THE QUALIFICATION PROBLEM USING FLUENT DEPENDENCY CONSTRAINTS

JONAS KVARNSTRÖM AND PATRICK DOHERTY
Department of Computer and Information Science, Linköping University, Sweden

In the area of formal reasoning about action and change, one of the fundamental representation problems is providing concise modular and incremental specifications of action types and world models, where instantiations of action types are invoked by agents such as mobile robots. Provided the preconditions to the action are true, their invocation results in changes to the world model concomitant with the goal-directed behavior of the agent. One particularly difficult class of related problems, collectively called the qualification problem, deals with the need to find a concise incremental and modular means of characterizing the plethora of exceptional conditions that might qualify an action, but generally do not, without having to explicitly enumerate them in the preconditions to an action. We show how fluent dependency constraints together with the use of durational fluents can be used to deal with problems associated with action qualification using a temporal logic for action and change called TAL-Q. We demonstrate the approach using action scenarios that combine solutions to the frame, ramification, and qualification problems in the context of actions with duration, concurrent actions, nondeterministic actions, and the use of both Boolean and non-Boolean fluents. The circumscription policy used for the combined problems is reducible to the first-order case.

Key words: action theories, nonmonotonic reasoning, temporal logic, qualification problem, circumscription.

1. INTRODUCTION

The primary focus of research in the area of formal reasoning about action and change considers representation problems associated with an autonomous agent, such as a mobile robot (UGV) or an unmanned aerial vehicle (UAV), interacting with a highly complex and dynamic environment in which the agent behaves in a goal-directed manner. A primary goal of the research is to develop modeling and verification tools that can be used by engineers in the development of such agents and by the agents themselves, who require both representations of the environment and limitations of their behavior in the environment in order to execute tasks to achieve goals. Due to the dynamic and causal nature of an agent's interaction with its environment, temporal logic formalisms are ideal candidates for world modeling, task and planning specification, and causal reasoning. The use of temporal logic formalisms provides a suitable basis for both specifying and verifying the complex activity associated with agent interaction with complex environments.

When one focuses on the type of complex environments associated with UGVs and UAVs, it immediately becomes clear that it is in general computationally, epistemologically, and ontologically infeasible to completely represent the environment an agent is embedded in and the action types it has at its disposal when interacting with its environment. This leads to the use of nonmonotonic extensions to temporal formalisms which contribute to providing succinct and modular representations of incomplete world model specifications and action type specifications. This article focuses on the representation of action type specifications and agent task representations in terms of narratives. In our approach, narratives consist of different classes of statements, which

Address correspondence to the authors at the Department of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden; e-mail: jonkv@ida.liu.se; patdo@ida.liu.se

This is an extended version of an article (Doherty and Kvarnström 1998) published at the Fifth International Workshop on Temporal Representation and Reasoning (TIME-98).

include action type specifications, timed action occurrences, observations, domain and dependency constraints, and additional timing information relating statements to each other. Narratives can be viewed as agent programs to be executed by an agent, or as hypothetical courses of action an agent can reason about when generating its own plans, or simply trying to understand how its future actions will affect its external environment, or to what degree its past actions have achieved its previous goals.

Three difficult modeling problems associated with the formal specification of action types in the context of complex, dynamic environments are the *frame*, *ramification*, and *qualification* problems. These problems have been a topic of continual research in the action and change community. Briefly, the frame problem concerns the need to find a concise and efficient means of representing and reasoning about what does not change when an action or actions are executed by an agent. The ramification problem concerns the need to separate the representation of the direct effects of an action type description from the plethora of indirect effects that may ensue when the action is executed successfully. An important aspect of the problem is to deal with the context dependent and causal nature of the chains of indirect effects that may ensue. The qualification problem, which is the problem we focus on, concerns the need to find a concise, incremental, and modular means of characterizing the plethora of exceptional conditions that might qualify an action, but generally do not, without having to explicitly enumerate them in the precondition to the action type. A solution to one of these problems generally implies a solution to the other two due to the interactions between preconditions, postconditions, and indirect effects of action occurrences.

Ascertaining whether one has solutions to each of these problems is as difficult as finding the solutions themselves. The reason for this is that solutions may work well when based on a particular set of assumptions regarding the ontological nature of the environment an agent is embedded in and the particular epistemological constraints placed on the agent itself, but may not work well when these assumptions and constraints are relaxed. Rather than there being one frame, ramification and qualification problem, we would claim that there are different solutions for different combinations of epistemological and ontological assumptions. This working hypothesis is well in line with the approach used by Sandewall (1994) in his study of the frame problem using the Features and Fluents framework.

For example, some ontological assumptions concerning action types are whether actions with duration, nondeterministic actions, or concurrent actions are possible. An epistemological assumption would be whether an agent has complete knowledge about all the effects of an action, or whether one can assume complete and accurate sensory data about the environment. An additional factor when evaluating a solution pertains to what types of reasoning tasks one has in mind for the agent. If one is concerned with a predictive mechanism for the agent used when generating a plan, a solution to the qualification problem which works here might not work if one is concerned with a postdictive mechanism for the agent used after executing a number of actions in a plan and gathering sensor data about the results.

We will first informally discuss some of the different ontological and epistemological choices that may affect the nature of solutions to the qualification problem. We will then present a complex narrative description, the Russian Airplane Hijack (RAH) Scenario, which in order to be adequately represented in any logical formalism would require robust solutions to the frame, ramification, and qualification problems. We say robust because a description of the RAH world requires the representation of concurrent actions, incomplete specification of states, ramification with chaining, the use of non-Boolean fluents, fine-grained dependencies among objects in different

fluent value domains, actions with duration, two types of qualification (*weak* and *strong*), and the use of explicit time, in addition to other features. To our knowledge, this provides one of the more challenging benchmark examples in the literature. It is challenging in the sense that it involves solutions to all three representation problems and the ontological assumptions pertaining to allowable action types are relatively complex.

The RAH narrative description will be used as a vehicle for considering different facets of the qualification problem and demonstrating our solutions to the problem. To do this, we will first introduce TAL-Q (Temporal Action Logic with Qualification), an extension to the already existing TAL family of logics (see Doherty et al. 1998) which has sufficient expressivity to model the RAH scenario. TAL-Q is an incremental extension of an earlier logic called TAL-C (Karlsson and Gustafsson 1999), just as TAL-C is an incremental extension of TAL-RC (Gustafsson and Doherty 1996). In fact, the logical language and minimization policy is roughly the same for TAL-RC, TAL-C, and TAL-Q. The advantages of leaving the logic and minimization policy intact are that the new class of narrative descriptions that can be represented in TAL-Q subsumes previous classes and that any circumscribed scenario in TAL-Q is provably and automatically reducible to a first-order theory implemented in an on-line research tool developed by our group called VITAL (Kvarnström and Doherty 1997), which permits the visualization and querying of narrative descriptions.

After introducing TAL-Q, we will use it to represent the RAH narrative description. This will be done in stages. Initially, we will represent the narrative under the assumption that actions always succeed. We will then modify the representation with qualification conditions for action types and a mechanism for reasoning about qualified action types based on the use of durational fluents and dependency constraints. The use of durational fluents in combination with a simple form of circumscription provides a flexible means for incorporating a default mechanism into TAL-Q.

We will then use TAL-Q to consider a number of additional aspects pertaining to qualification in the context of different ontological choices such as the use of concurrent actions. We will also briefly consider two alternative approaches to qualification that can be represented using TAL-Q. Finally, we will direct our attention toward a number of benchmark examples in the literature, representing them using TAL-Q, and then compare our approach to qualification with a number of other approaches in the literature.

2. THE QUALIFICATION PROBLEM

Before it is possible to design or assess any approach to solving the qualification problem, we must define in more detail what the qualification problem is.

Let us assume that there is an environment, a "real world," in which actions can be executed by one or more agents. Let us also assume that each action has a well-defined *intended effect*. For example, in the well-known blocks world, the intended effect of the action $\mathsf{putdown}(x)$ is that in the resulting state, the block x the agent is currently holding should be on the table and all other blocks should be unaffected by the action.

When reasoning about this world using a temporal action logic, we need a correct description of the preconditions and effects of each action type that can be used by an agent. If it is possible to find a model of the world that is both simple and correct, at least at some level of abstraction, it should be straightforward to find such a description of preconditions and effects of actions. However, in more complex worlds,

describing an action may be far more difficult, and the resulting preconditions may be extremely complicated. This complexity is often due to the large number of conditions that are almost always false, but when satisfied, can cause an action to fail to achieve its intended effects. We will call such exceptional conditions *qualifications*, and if one or more of an action's qualifications hold, the action will be said to be *qualified*. (In some cases, even "nonexceptional" conditions will also be considered as qualifications.)

The potentially large number of qualifications to an action leads to a number of representational and implementational difficulties that are collectively called the *qualification problem*. Some of these difficulties are discussed below.

2.1. Restricting the Problem

The qualification problem is a complex problem with many different aspects, and it would be very optimistic to assume that we can design a single solution that covers all these aspects. Instead, it is necessary to determine in advance which aspects of the problem should be addressed by the solution we are designing or assessing, which reasoning tasks (such as prediction or planning) should be supported by the solution, and which ontological and epistemological assumptions will be made regarding the worlds for which the solution should be applicable and the agents that will apply it. Below, we will consider these questions in some more detail.

Aspects of the Qualification Problem. Although the difficulties associated with the qualification problem are closely related, it is possible to isolate several aspects of the problem that may be tackled separately. Some of these aspects pertain to the following:

- Due to incomplete knowledge about the world one is reasoning about, it may be impossible, or at least very difficult, to find and enumerate all qualifications to an action. A classical example of this aspect of the qualification problem is the "potato in tailpipe" problem (Ginsberg and Smith 1988): In order to start a car, there must be nothing wrong with the battery, there must be gas in the tank, there must not be a potato in the tailpipe, and so on. No matter how many conditions we manage to think of, there will surely always be more.
- Even when it is possible to know all qualifications to an action, the complexity of these conditions may require a highly expressive logic, unless we are willing to abstract away from some aspects of the world and be satisfied with incomplete specifications and a mechanism to deal with this incompleteness.
- The information we do have about the conditions under which an action is qualified needs to be represented in a modular manner, so that conditions may be added or removed incrementally.
- Assuming that actions are normally not qualified, the need to explicitly prove that each qualification condition does not hold may be computationally inefficient.

In this article, we will mainly concentrate on the representational problems associated with qualification—that is, modular and incremental representations of qualified action types.

In order to assess or design a solution to the qualification problem, we also need to specify the reasoning tasks that will be used by an agent in achieving goals via execution of actions. For example, an agent interested in determining why an action failed using postdiction may need a different solution than an agent that is solely interested in predicting the results of invoking a sequence of actions.

We will mainly consider off-line reasoning tasks such as prediction, postdiction, and planning.

Ontological and Epistemological Assumptions. It is also necessary to determine which ontological assumptions will be made regarding the world in which the solution will be applied, as well as which epistemological assumptions will be made about the agent's knowledge of the world and of the effects of its actions. Perhaps the most important such assumption is that of what will happen in the world if the agent invokes a qualified action. The following are some of the assumptions that may be reasonable, depending on the world that is being modeled:

- Invoking a qualified action has no effect at all on the world.
- Invoking a qualified action affects the world, but we always know what effects it will have even when it is qualified.
- Invoking a qualified action affects the world in an unknown way, but only during the time interval when the action is being executed.
- Invoking a qualified action affects the world in an unknown way, and may trigger unknown chains of events that continue affecting the world after the action has finished executing.

However, there are also many other assumptions that may affect the applicability of a solution. The following are some examples of additional questions that need to be answered:

- Is there complete information about the initial state in a narrative? Is there any information about any other state in terms of observations made by the agent?
- Can actions be context-dependent? Can they be nondeterministic? Can they have duration, and if they can, do they have internal state (that is, may fluents change, discretely or continuously, within the duration of an action)? Can they have delayed effects? Can there be concurrent actions? If so, can actions overlap partially?
- Can there be dynamic processes continuously taking place independently of the actions invoked by the agent?
- In the presence of incomplete information and nondeterministic actions, are there domain constraints that exclude certain "impossible" states? Are there domain constraints that exclude certain "impossible" *sequences* of states? Can domain constraints vary over time?
- Are actions allowed to have indirect side effects? Can side effects be delayed (take place after the action has finished executing)? Can they trigger other side effects?

Clearly, the more complex the ontological and epistemological assumptions are, the more restricted our choices will be when attempting to solve the qualification problem for that particular class of worlds. Consequently, we need to determine these assumptions in advance.

Ideally, we would like to formally assess the correctness of different solutions to the qualification problem relative to a given class of narrative descriptions, specified via epistemological and ontological assumptions as Sandewall (1994) has done for the frame problem. However, extending Sandewall's framework for qualification—as well as ramification, concurrent actions, and other extensions we may want to use in combination with qualification—is outside the scope of this article. Instead, we will discuss in a more informal manner some of the questions that need to be considered

when a solution to the qualification problem is designed and some of the effects the choice of reasoning task and our assumptions about the class of worlds we are reasoning about may have on the answers to these questions. We will then provide formal, but formally unassessed, solutions using TAL-Q. Some of the existing solutions in the literature will also be considered from this point of view in Section 10.

2.2. Designing a Solution

We have now considered four questions: Which aspects of the qualification problem a solution should address, which reasoning tasks it should support, which ontological assumptions should be made regarding the worlds to which it is applicable, and which epistemological assumptions should be made regarding the agents that should apply it. For each of these questions, the answer will depend mainly on the class of problems we are trying to solve. For example, for anyone developing an agent controlling a UAV (unmanned aerial vehicle), the computational aspects of the qualification problem are very important; prediction, postdiction and planning may all be useful; and one must probably be able to model context-dependent concurrent actions with duration.

However, there are also certain design choices that may be made more or less independently of the problem or class of problems that should be solved. Some of these choices are discussed in this section.

How Should Qualification Conditions Be Expressed? By definition, an action is qualified if it is somehow prevented from having its intended effects. There are basically two aspects to the problem. In an offline mode, for example, when an agent is generating a plan, a predictive mechanism might simulate the possible future state of the world given that the agent executes a sequence or partially ordered sequence of actions and find that the sequence violates certain domain or dependency constraints. In this case, either the domain or dependency constraints have been incorrectly specified, or the action type descriptions are not precise enough and a qualification condition for one or more actions has to be added. In an online mode, the agent actually executes sequences of actions and finds that one or several have not achieved their intended effects. This information is derived from actual sensory data. Since the world is its own model, either one has inaccurately specified the ontological assumptions that pertain to the world or a rare qualifications has arisen and that qualification condition should be added to the agent's action type specification in an incremental manner so the next time the condition arises, the action will not be executed due to the explicit qualification. So, the qualification problem does not rule out adding a number of qualifications to an action, but any solution tries to minimize the number of explicit qualifications per action, and those that are added are added in a modular and incremental manner. Note that very little research has been done regarding the online execution and modification of action types. Most of the research has focused on generating the proper conclusions in offline or simulation mode, assuming one already has explicit information about at least some of the qualifications per action, and on specifying a mechanism for adding new qualifications in an incremental and modular manner. In this article, we will also focus on the offline mode.

Most formalisms for reasoning about action and change are based on the two-state assumption. There is an initial state in which an action is invoked and a result state in which the effects of the action become true provided the preconditions to the action are true in the initial state. TAL-Q is an exception due to its use of actions with

duration and its use of explicit time. There are basically two classes of solutions in the literature, one focusing on the initial state of an action and the other focusing on the effect state.

When focusing on the initial state, one very straightforward solution would be to strictly treat qualification constraints as preconditions to actions—conditions that must or must not hold in the state in which an action is invoked. For example, the start action can be considered qualified if potato_in_tailpipe is true in the state where an action will be executed. Most solutions in this class encode an assumption that if one cannot explicitly prove that a known qualification to an action is true then that action can be executed. If actions may have duration and internal state, this approach can be extended by also allowing conditions at any time within the interval when the action is executed—for example, even if there was no potato in the tailpipe when the start action was invoked, one may be inserted during its execution.

In approaches focusing on the effect state (such as Ginsberg and Smith 1988; Lin and Reiter 1994), an action is considered qualified whenever its intended effect would contradict a domain constraint in the effect state. In a sense, this implies a form of hypothetical reasoning that could only be used in off-line mode, where one checks whether the execution of an action would lead to a contradiction in the result state. In Ginsberg and Smith, if this is the case, the action has no effect and the execution and effect states for the action are the same. In the case of Lin and Reiter, a form of precompilation is used to modify the specified preconditions of each action to include the negation of every condition that would cause a contradiction. This assumes that one already has explicit qualification conditions in the theory. As before, this approach can of course be extended to actions with duration and internal state by considering an action qualified whenever it would contradict a domain constraint at any time during its execution. Also, if domain constraints can span multiple states (e.g., relating fluents in one state to fluents in its successor), an action could be considered qualified whenever executing it must eventually result in such a domain constraint being contradicted.

We will pursue the precondition-based approach with TAL-Q, but with a much richer ontology of actions. This richer ontology would lead to problems in the latter approach. For example, if actions can be executed concurrently, it could be the case that the combined effect of two concurrent actions contradicts a domain constraint, but either action alone does not. Do we predict that one action succeeds, which may sometimes be the case, or that neither one does? Additional problems would arise if we allowed delayed side effects, nondeterministic actions, or any of a number of other features that have generally not been considered together with qualification. These problems make the latter approach much less intuitive for these extensions to the logics than it is for a situation calculus–type logic or belief-update approach described in Lin and Reiter (1994) and Ginsberg and Smith (1988). Due to the added expressivity of TAL-Q, these issues must be dealt with in our solution.

What Should Be Entailed about the Effects of Invoking a Qualified Action? In a number of formalisms, one can reason not only about the effects of actions that are unqualified, but also those that are qualified. In other words, an action is invoked even though not all conditions that would *guarantee* its intended effects are satisfied. Reasoning about this type of situation is perhaps more appropriate when considering the online mode, but still has to be defined even for offline mode reasoning if the formalism allows invocation of qualified actions. Ideally, we would like our approach

to the qualification problem to be able to represent whatever knowledge—or *lack* of knowledge—we have regarding the effects of invoking actions, whether qualified or not.

However, there are cases where this might not make sense, or is simply unnecessary. Suppose, for instance, we have an interest in the planning task in offline mode. In this case, reasoning about the effects of qualified actions does not make much sense, since the point to generating a plan is to generate a sequence of actions we assume are all executable and have their intended effects. What would be important is being able to reason about under what conditions an action might be qualified so as to avoid using it under those conditions in the plan generation phase. On the other hand, if one is using the formalism in the online mode, reasoning about the effects of invoking qualified actions may be very important because an agent might on occasion invoke an action without being aware it is qualified—due to faulty sensors, for example. In this case, being able to reason about at least some of the effects of the action would be quite useful in a postdictive or diagnostic phase of reasoning.

Whatever choice is made, it should be made very clear why the choice is being made and what the ontological justification is. Quite often, the choice is simply a side effect of the solution chosen for solving the qualification problem. As we shall see in Section 10, many formalisms behave differently in this respect.

Should It Be Possible to Reason about Qualification within the Logic? One final design issue is whether qualifications to actions should be first-class objects which can be explicitly reasoned about in the formalism itself. This is particularly important in the online reasoning mode, where execution monitoring is a central part of an agent's execution mechanism and determines future courses of action and modification of existing courses of action.

2.3. Reasoning about Undesirable Actions

A problem that often appears during planning is that of determining which actions would have effects that are undesirable. Although this may at a first seem unrelated to the qualification problem, it turns out that both problems can often be specified in terms of conditions that hold when an action is invoked or constraints that should not be violated by the effects of an action. Recall that this is the basis for the two classes of solutions to the qualification problem discussed previously. The difference between reasoning about qualified actions and reasoning about undesirable effects of actions may better be determined in terms of ontological assumptions placed on the worlds we are interested in. For example, should one make explicit distinctions between types of qualifications to actions such as those that if satisfied would make it physically impossible to execute the action satisfactorily, or those that simply involve contingent restrictions associated with the domain in question?

This appears to be the reason why some qualification examples in the literature are in fact examples of actions which *would* have their defined, intended, well-known effects, but which are invoked in a context in which those effects are not desirable. For example, in the lenient emperor scenario (Lin and Reiter 1994), there is a robot that can paint blocks, but an emperor allows at most one block to be yellow at any given time. This is ensured by considering the action paint(*block*, yellow) to be qualified whenever there is already a yellow block (and, of course, by preventing the robot from executing any qualified action).

This approach works well when attempting to define a plan while avoiding undesirable actions. On the other hand, suppose that there is already a yellow block and that we want to predict what would happen if the robot tried to paint another block yellow. Certainly, there is nothing inherently problematic about this course of events even in the context of the emperor's strange rule. Intuitively, the action should succeed, with the conclusion that the robot invoked an action that violates correct social behavior. In the example above, the action is considered to be qualified and the conclusion will be that the action has in fact failed.

Therefore, undesirable actions should probably not be handled in exactly the same way as qualified actions, but they can probably be handled in a technically very *similar* manner, and any solution to the qualification problem may also be interesting in this respect. Several examples in the literature which relate to this issue will be considered.

2.4. Summary

In summary, a solution to the qualification problem that works well for one reasoning task, under one ontological assumption, might not work well given another reasoning task or another ontological assumption, and when the set of problems one considers is extended, one may have to use a different approach previously considered less than optimal.

Due to these considerations, there is probably no single "best" solution to the qualification problem. Instead, there is likely to exist a *set* of good solutions, each of which is useful for a given expressivity and for a given task. Unfortunately, the solutions found in the literature often do not state explicitly what task and expressivity they are intended to handle. This makes it difficult to compare solutions, or build on one another's work. This section's intent was to point these issues out and create a context for the rest of the article. We will now consider the RAH scenario and its formalization in TAL-Q.

3. THE RUSSIAN AIRPLANE HIJACK SCENARIO

In the remainder of this article, we will use the methodology of representative examples as a means of considering and proposing a solution to the qualification problem for a certain class of worlds. The scenario we will use is the Russian Airplane Hijack Scenario (RAH),[1] previously published in Doherty and Kvarnström (1998).

> *A Russian businessman, Boris, travels a lot and is concerned about both his hair and safety. Consequently, when traveling, he places both a comb and a gun in his pocket. A Bulgarian businessman, Dimiter, is less concerned about his hair, but when traveling by air, has a tendency to drink large amounts of vodka before boarding a flight to subdue his fear of flying. A Swedish businessman, Erik, travels a lot, likes combing his hair, but is generally law abiding.*
>
> *Now, one ramification of moving between locations is that objects in your pocket will follow you from location to location. Similarly, a person on board a plane will follow the plane as it flies between cities.*
>
> *Generally, when boarding a plane, the only preconditions are that you are at the gate and you have a ticket. However, if you try to board a plane carrying a gun in your pocket, which will be the case for Boris, this should qualify the action. Also, a condition that could sometimes qualify the boarding action is if you arrive at the gate in a sufficiently inebriated condition, as will be the case for Dimiter. When the boarding action is qualified, attempting to board should have no effect.*

[1] This scenario is an elaboration and concretization of a sketch for a scenario proposed by Vladimir Lifschitz in online discussions in the Electronic Transactions on Artificial Intelligence (ETAI/ENAI).

Boris, Erik, and Dimiter already have their tickets. They start (concurrently) from their respective homes, stop by the office, go to the airport, and try to board flight SAS609 to Stockholm. Both Erik and Boris put combs in their pockets at home, and Boris picks up a gun at the office, while Dimiter is already drunk at home and may or may not already have a comb in his pocket. Who will successfully board the plane? What are their final locations? What will be in their pockets after attempting to board the plane and after the plane has arrived at its destination?

If the scenario is encoded properly and our intuitions about the frame, ramification, and qualification problems are correct then we should be able to entail the following from the RAH scenario, assuming that Boris, Erik, and Dimiter own the combs comb1, comb2, and comb3, respectively:

1. Erik will board the plane successfully, eventually ending up at his destination.
2. An indirect effect of flying is that the person ends up at the same location as the airplane. Additionally, because items in pockets follow the person, a transitive effect results where the items in the pocket are at the same location as the plane. Consequently, Erik's comb (comb2) will also end up at his destination.
3. Boris will get as far as the airport with a gun and comb1 in his pocket. He will be unable to board the plane.
4. Dimiter will get as far as the airport, and may or may not be able to board the plane. If he is able to board the plane, he will eventually end up at his destination. Otherwise, he will remain at the airport. In any case, if he initially carried a comb, it will end up in the same location.

For this scenario, we assume that we know all possible reasons why an action may be qualified, and we are mainly interested in representing qualifications in a modular and intuitive manner. We are also mainly interested in prediction.

This is a rather complex scenario, and modeling it requires a relatively expressive logic. Unfortunately, many approaches to the qualification problem in the literature have been defined with very strong constraints placed on action types and also use the two-state assumption. Modeling this scenario in such logics, or scaling up the expressivity of such a logic to be able to model this scenario, would be difficult. In the next section, we will take advantage of the expressivity already part of TAL-Q in defining a solution.

4. TAL-Q: TEMPORAL ACTION LOGIC WITH QUALIFICATION

Our approach to handling the qualification problem is based on the use of TAL-Q (Temporal Action Logic with Qualification), a member of the TAL (Temporal Action Logics) family of logics for reasoning about action and change.

As it turns out, the approach we will present does not require any new predicates or other changes to the high-level concepts used in previous TAL logics. Instead, it uses well-known concepts from older logics such as TAL-C (Karlsson and Gustafsson 1999) in a new and different way. Therefore, we will begin by describing the logic TAL-Q without considering the qualification problem. In Section 5, we show how the RAH scenario can be modeled in TAL-Q under the assumption that actions never fail, and in Section 6, we define our approach to solving the qualification problem within TAL-Q and demonstrate it by applying it to the RAH scenario.

4.1. The TAL Family of Logics

Temporal Action Logics have their origin in the Features and Fluents framework proposed by Sandewall (1994), where both a variety of logics of preferential entailment for reasoning about action and change and a framework for assessing the correctness of these and future logics were proposed. One of the definitions of preferential entailment, PMON, was proposed by Sandewall and assessed correct for the $\mathscr{K}$*-IA* class of action scenarios, a broad class of scenarios which deals with nondeterministic actions, incomplete specification of state and the timing of actions, and observations at arbitrary states in a scenario. PMON solved the frame problem for the $\mathscr{K}$*-IA* class. Later, Doherty translated and generalized PMON into an order-sorted first-order logic with a circumscription axiom capturing the PMON definition of preferential entailment (Doherty 1994; Doherty and Łukaszewicz 1994).

Recently, a number of additional extensions and generalizations have been added to the original PMON logic. Although the new logics generated belong to what we call the TAL family, each is essentially an incremental addition to the base logic PMON.

TAL-RC, proposed by Gustafsson and Doherty (1996), provides a solution to the ramification problem for a broad, but as yet unassessed class of action scenarios. The main idea is the addition of a specialization of fluent dependency constraints which we called *causal constraints*. The solution was based on the insight that the *Occlude* predicate used to solve the frame problem for PMON was all that was needed to define causal rules which turn out to be very similar to action effect axioms. The solution is also extremely fine-grained in the sense that one can easily encode dependencies between individual objects in the domain, work with both Boolean and non-Boolean fluents and represent both Markovian and non-Markovian dependencies (Giunchiglia and Lifschitz 1995).

TAL-C, proposed by Karlsson and Gustafsson (1999), uses fluent dependency constraints as a basis for representing concurrent actions. A number of phenomena related to action concurrency such as interference between one action's effects and another's execution, bounds on concurrency, and conflicting, synergistic, and cumulative effects of concurrent actions are studied.

TAL-Q, initially proposed by Doherty and Kvarnström (1998) and described in more detail in this article, is intended to provide one approach to solving the qualification problem. We note that TAL-Q is an incremental extension of TAL-C, just as TAL-C is an incremental extension of TAL-RC. In fact, the logical language and minimization policy is roughly the same for TAL-RC, TAL-C, and TAL-Q. The advantages of leaving the logic and minimization policy intact are that the new class of action scenarios which can be represented in TAL-Q subsumes previous classes and that any circumscribed scenario in TAL-Q is provably and automatically reducible to a first-order theory.

The TAL methodology uses two languages for representing and reasoning about narratives. The surface language $\mathscr{L}$(ND) (Narrative Description Language) provides a convenient high-level notation for describing narratives, and can be seen as a set of macros easily translated into the base language $\mathscr{L}$(FL), which is an order-sorted first-order language. We will now provide a brief description of these two languages, the underlying logic and the translation from $\mathscr{L}$(ND) to $\mathscr{L}$(FL), referring the reader to Doherty et al. (1998), Sandewall (1994), Doherty (1994), Doherty and Łukaszewicz (1994), Gustafsson and Doherty (1996), Karlsson and Gustafsson (1999), and Kvarnström and Doherty (1997) for more detailed descriptions of the logics in the TAL family.

The Surface Language $\mathscr{L}$(ND)

A narrative in $\mathscr{L}$(ND) consists of two parts, the narrative background specification and the narrative specification. Each part consists of a collection of labeled statements, the sum of which describes describing both the static and dynamic aspects of a narrative.

The *narrative background specification* (NBS) consists of a *narrative type specification* (NTS) providing fluent value domains and type descriptions for fluents and actions, an *action type specification* (ATS) providing generic definitions of actions, a *domain constraint specification* (DCS) providing statements representing static knowledge about logical dependencies between fluents generally true in every state or across states, and a *dependency constraint specification* (DeCS) providing statements representing knowledge about directional dependencies between fluents true in states or across states.

The *narrative specification* (NS) consists of *observation statements* intended to represent observations of fluent values at specific timepoints, *action occurrence statements* specifying which actions occur and during which time intervals, and *schedule statements*, a class of intermediary statements automatically generated from action type specifications during the translation from $\mathscr{L}$(ND) to $\mathscr{L}$(FL). Note that in action occurrence statements, both the timing between actions and the durations of actions can be incompletely specified.

An Example. In order to make it easier to understand the $\mathscr{L}$(ND) language, we will explain it using a variation of the well-known hiding turkey scenario. In this variation of the scenario, there is a turkey that may or may not be deaf, and there is also a gun. First, we load the gun, which makes some noise—if the turkey is not deaf, it will hide whenever there is noise. However, if the turkey has been hiding for a while and there has not been any noise, the turkey will decide to come out in the open again. After a while, we shoot, and if the turkey is not hiding at that time, it will die.

The Narrative Type Specification; Persistent and Durational Fluents. For the narrative type specification we will require one value domain (the Boolean domain), five Boolean fluents (alive, deaf, hiding, loaded, and noise), and two actions (Load and Fire).

Intuitively, the first four fluents do not change unless something changes them, that the fifth, noise, is different—there is no noise unless someone is currently making noise. This distinction between persistent and durational fluents is important. A persistent fluent can only change when an action or dependency constraint allows it to change (the *persistence assumption* or *inertia assumption*). Otherwise, it retains the same value it had at the previous timepoint. On the other hand, a durational fluent is associated with a default value, and can only take on another value when an action or dependency constraint allows it to (the *default value assumption*). At timepoints when no action or dependency constraint explicitly allows it to take on another value, it will immediately revert to its default value.

Whether a fluent is persistent or durational—or neither—is defined in a set of *persistence statements*, using the $\mathscr{L}$(ND) macros *Per* and *Dur*. For a persistent fluent f, $Per(t, f)$ should be true, and for a durational fluent f with default value v, $Dur(t, f, v)$ should be true. (Although *Per* and *Dur* both take a temporal argument, they are usually specified with universal quantification over t. Note also that some earlier TAL logics used a fixed *nochange axiom* instead of persistence statements.

Using persistence statements provides a more flexible and fine-grained approach to controlling the default behavior of fluents.)

For the extended hiding turkey scenario, we need the following persistence statements:

per1 $\forall t[\mathtt{true} \rightarrow Per(t+1, \mathsf{alive}) \wedge Per(t+1, \mathsf{deaf}) \wedge Per(t+1, \mathsf{hiding}) \wedge Per(t+1, \mathsf{loaded})]$

per2 $\forall t[\mathtt{true} \rightarrow Dur(t, \mathsf{noise}, \mathsf{false})]$.

Some Basic Macros and Formulas. Now that we have defined a set of fluents, we need to be able to talk about their values at any given timepoint. Here, we will show some of the most important macros in $\mathscr{L}$(ND) in order to provide an intuitive understanding of the surface language, again referring the reader to Doherty et al. (1998) for a formal definition.

First, an *isvalue expression* is an expression of the form $f \hat{=} v$, stating that the fluent f has the value v (for Boolean fluents, we will sometimes write f instead of $f \hat{=} \mathsf{true}$). An *atemporal narrative formula* consists of isvalue expressions combined using the standard Boolean connectives and quantification over values. Below, let τ and τ' be timepoints and let α and β be atemporal narrative formulas.

An $\mathscr{L}$(ND) formula of the form $[\tau]\ \alpha$ means that α holds at τ. Closed or semiclosed intervals can also be used. For example, the formula $[\tau, \tau']\ \alpha$ means that α holds between τ and τ', inclusive. We also need to talk about changes in the values of fluents. In order to simplify such formulas, we can use the C_T macro, which stands for *changes to true*. A formula of the form $C_T([\tau]\,\alpha)$ is defined as $(\forall t.\ t+1 = \tau \rightarrow [t] \neg\, \alpha) \wedge [\tau]\alpha$. Note that due to the use of nonnegative time, $C_T([0]\,\alpha)$ is equivalent to $[0]\,\alpha$.

Domain Constraint Specification. Domain constraints represent static knowledge about logical dependencies between fluents generally true in every state or across states. Although no domain constraints are needed for this scenario, we will show one possible constraint as an example: If there is noise at some timepoint, and the turkey is not deaf, then at the next timepoint, the turkey must be hiding.

dom1 $\forall t\ [[t]\ \mathsf{noise} \wedge \neg\, \mathsf{deaf} \rightarrow [t+1]\ \mathsf{hiding}]$.

Action Type Specification and Dependency Constraint Specification. Actions are of course intended to change the world in some way, and dependency constraints describe how the world will change given certain conditions and can be used to specify side effects for actions. But since fluents are normally persistent or have a default value, we cannot change their values without first defining a way to specify exceptions to those general rules. This is done using the R, I, and X macros. They can all be used with a temporal interval, for example $R((\tau, \tau']\,\alpha)$, or a single timepoint, for example $I([\tau']\,\alpha)$. Each operator releases the fluents occurring in α from the persistence and default value assumptions during the given interval or at the given timepoint. However, the operators differ in how they constrain the values of the fluents occurring in α.

The R operator stands for *fluent reassignment*, and ensures that α will hold at τ', the final timepoint in the interval. During the rest of the interval, the fluents occurring in α are allowed to vary.

The I operator stands for *exceptional assignment* and is often but not always used in combination with durational fluents. It ensures that α will hold during the entire interval.

The X operator stands for *occlude assignment*. Its purpose is simply to allow a fluent's value to vary at a timepoint or interval, and therefore it does not constrain the fluents in α.

Returning to the extended hiding turkey scenario, we first need to define the Load and Fire actions: If we load the gun, then at some timepoint in the duration of the action, loaded will become true (the R macro), and during the entire action, noise will be true (the I macro). If we fire the gun, the effects depend on the state in which it is invoked: If it was loaded initially, it will not be loaded after firing, and if additionally the turkey was not hiding, it will no longer be alive.

We also need two dependency constraints: First, if the turkey is not deaf and currently not hiding, and if a noise *begins* (the C_T macro), then at the next timepoint, the turkey will be hiding (this is a delayed dependency constraint). Second, if the turkey has been hiding for ten timepoints, and there has been no noise during that time, it will stop hiding.

acs1 $[t_1, t_2]\,\mathsf{Load} \rightsquigarrow R((t_1, t_2]\,\mathsf{loaded}) \wedge I((t_1, t_2]\,\mathsf{noise})$
acs2 $[t_1, t_2]\,\mathsf{Fire} \rightsquigarrow ([t_1]\,\mathsf{loaded} \wedge \neg\mathsf{hiding} \rightarrow R((t_1, t_2]\neg\mathsf{alive})) \wedge$
$([t_1]\,\mathsf{loaded} \rightarrow R((t_1, t_2]\neg\mathsf{loaded}))$
dep1 $\forall t\ [[t]\neg\mathsf{hiding} \wedge \neg\mathsf{deaf} \wedge C_T([t]\,\mathsf{noise}) \rightarrow R([t+1]\,\mathsf{hiding})]$
dep2 $\forall t\ [[t, t+9]\,\mathsf{hiding} \wedge \neg\mathsf{noise} \rightarrow R([t+10]\neg\mathsf{hiding})]$.

Observation Statements and Action Occurrence Statements. Finally, we also need to provide the actual narrative specification—the observations and action occurrences. To be more exact, we need to observe that the turkey is alive in the initial state, but not hiding, and that the gun is not loaded, and we must also load and fire the gun. We do not observe whether the turkey is deaf or not, and there is no need to state that there is no noise, since noise is false by default.

obs1 $[0]\,\mathsf{alive} \wedge \neg\mathsf{loaded} \wedge \neg\mathsf{hiding}$
occ1 [1, 4] Load
occ2 [5, 6] Fire.

4.3. The Base Language $\mathscr{L}$(FL)

The $\mathscr{L}$(ND) narrative is translated into the base language $\mathscr{L}$(FL), an order-sorted first-order language with equality using the predicates $Holds(t, f, v)$, $Occlude(t, f)$, and $Occurs(t, t', a)$, where t, f, v, and a are variables for timepoints, fluents, values, and actions, respectively.

The *Holds* predicate expresses what value a fluent has at each timepoint, and is used in the translation of isvalue expressions; for example, $[0]\,\mathsf{alive} \hat{=} \mathsf{true} \wedge \mathsf{loaded} \hat{=} \mathsf{false}$ can be translated into $Holds(0, \mathsf{alive}, \mathsf{true}) \wedge Holds(0, \mathsf{loaded}, \mathsf{false})$. The *Occlude* predicate expresses that a persistent or durational fluent is exempt from its persistence or default value assumption, respectively, at a given timepoint. It is used in the translation of the R, I, and X operators, which are intended to change the values of fluents. Finally, the *Occurs* predicate expresses that a certain action occurs during a certain time interval, and is used in the translation of action occurrence statements and schedule statements.

A linear discrete time structure is used in TAL-Q. The minimization policy is based on the use of filtered preferential entailment (Sandewall 1989), where schedule statements and dependency constraints are circumscribed with *Occlude* minimized and all other predicates fixed, and action occurrence statements are circumscribed with

Occurs minimized and all other predicates fixed. The result is then filtered with the observations and domain constraints, some foundational axioms such as unique names and domain closure axioms, the temporal structure axioms, and a set of persistence axioms characterizing the behavior of persistent and durational fluents.

In the following, we use

- Υ to denote the collection of narrative statements contained in a narrative in $\mathscr{L}$(ND), and Υ_{per}, Υ_{obs}, Υ_{occ}, Υ_{scd}, Υ_{domc}, and Υ_{depc} to denote the sets of persistence, observation, action occurrence, schedule, domain constraint, and dependency constraint statements in Υ, respectively.
- Γ to denote the collection of narrative formulas in $\mathscr{L}$(FL) corresponding to the translation of the narrative statements in Υ, and Γ_{obs}, Γ_{occ}, Γ_{scd}, Γ_{domc}, and Γ_{depc} to denote the corresponding sets of observation, action occurrence, schedule, domain constraint, and dependency constraint formulas in Γ, respectively.[2]
- Γ_{fnd} to denote the set of foundational axioms in $\mathscr{L}$(FL), which contains unique names axioms, unique values axioms, etc.
- Γ_{time} to denote the set of axioms representing the temporal base structure. Since the timepoints in TAL-Q use the natural numbers structure, we use the Peano axioms without multiplication.
- Γ_{per} to denote the set of axioms characterizing the behavior of persistent and durational fluents, defined as the translation of the formulas in Υ_{per} into $\mathscr{L}$(FL).

Given a narrative Υ in $\mathscr{L}$(ND), the corresponding theory Δ in $\mathscr{L}$(FL) is $\Gamma \wedge \Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per}$, where $\Gamma = \Gamma_{obs} \wedge \Gamma_{occ} \wedge \Gamma_{domc} \wedge \Gamma_{depc} \wedge \Gamma_{scd}$. Given this theory, we apply a filtered circumscription policy which results in the circumscribed theory $\Delta' = \Gamma' \wedge \Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per}$, where $\Gamma' = \Gamma_{obs} \wedge \Gamma_{domc} \wedge \mathit{Circ}(\Gamma_{occ}; \mathit{Occurs}) \wedge \mathit{Circ}(\Gamma_{depc} \wedge \Gamma_{scd}; \mathit{Occlude})$. It is easily shown that the second-order formulas $\mathit{Circ}(\Gamma_{depc} \wedge \Gamma_{scd}; \mathit{Occlude})$ and $\mathit{Circ}(\Gamma_{occ}; \mathit{Occurs})$ are reducible to logically equivalent first-order formulas (Doherty et al. 1998; Doherty 1996).

The $\mathscr{L}$(FL) formula γ is preferentially entailed by the narrative Υ iff $\Delta' \models \gamma$.

5. REPRESENTING THE RUSSIAN AIRPLANE HIJACK SCENARIO

In this section, we will show how the Russian Airplane Hijack Scenario can be represented in TAL-Q if we do not consider qualifications to actions. This will result in a scenario in which it is assumed that any attempt to board a plane always succeeds, regardless of whether the person carries a gun or is drunk. In Section 6, we will show how the scenario presented here can be modified in order to deal with the qualification problem.

In order to simplify the presentation, being at the airport will be the only normal precondition for boarding a plane.

All formulas in this section are written in the surface language $\mathscr{L}$(ND). Appendix A contains the same formulas, with the exception that the action type definitions have been modified as in Section 6 and some new dependency constraints have been added.

[2] The translation is very straightforward and is defined formally in Doherty (1996), Doherty et al. (1998), and Karlsson and Gustafsson (1999). For some examples, see the $\mathscr{L}$(ND) scenario in Appendix A and its translation into $\mathscr{L}$(FL) in Appendix B.

Appendix B contains the translation of the formulas in Appendix A into the base logic $\mathscr{L}$(FL).

5.1. Narrative Background Specification

First, it is necessary to determine which value domains, fluents and actions are needed. For the RAH Scenario, we will need a Boolean value domain boolean = {true, false}, a domain location = {home1, home2, home3, office, airport, run609, run609b, air} for locations, and a domain thing = {gun, comb1, comb2, comb3, boris, dimiter, erik, sas609} containing everything that has a location. We will also define the subdomains runway = {runway1, runway2} for locations that are runways, plane = {sas609} for things that are airplanes, person = {boris, dimiter, erik} for things that are people, and pthing = {gun, comb1, comb2, comb3} for things that people can pick up.

We also need four fluents: loc(thing): location, inpocket(person, pthing): boolean, onplane(plane, person): boolean, and drunk(person): boolean.

Four actions are necessary: pickup(person, pthing) for picking up things, travel(person, location, location) for traveling between locations in the same city, board(person, plane) for boarding an airplane, and fly(plane, runway, runway) for flying between two runways.

Finally, we need to declare each of the four fluents persistent at all timepoints using a set of persistence statements:

per1 $\forall t, thing\ [\texttt{true} \rightarrow Per(t+1, \mathsf{loc}(thing))]$
per2 $\forall t, person, pthing\ [\texttt{true} \rightarrow Per(t+1, \mathsf{inpocket}(person, pthing))]$
per3 $\forall t, person\ [\texttt{true} \rightarrow Per(t+1, \mathsf{drunk}(person))]$
per4 $\forall t, plane, person\ [\texttt{true} \rightarrow Per(t+1, \mathsf{onplane}(plane, person))]$.

5.2. Initial State

The initial state in a TAL narrative (as well as any other state) can be completely or incompletely specified using observation statements. For this scenario, we must define the initial locations of all things, as well as who is drunk in the initial state. On the other hand, we do not observe which things are in whose pockets.

obs1 $[0]\ \mathsf{loc}(\mathsf{boris}) \hat{=} \mathsf{home1} \wedge \mathsf{loc}(\mathsf{gun}) \hat{=} \mathsf{office} \wedge \mathsf{loc}(\mathsf{comb1}) \hat{=} \mathsf{home1} \wedge \neg \mathsf{drunk}(\mathsf{boris})$
obs2 $[0]\ \mathsf{loc}(\mathsf{erik}) \hat{=} \mathsf{home2} \wedge \mathsf{loc}(\mathsf{comb2}) \hat{=} \mathsf{home2} \wedge \neg \mathsf{drunk}(\mathsf{erik})$
obs3 $[0]\ \mathsf{loc}(\mathsf{dimiter}) \hat{=} \mathsf{home3} \wedge \mathsf{loc}(\mathsf{comb3}) \hat{=} \mathsf{home3} \wedge \mathsf{drunk}(\mathsf{dimiter})$
obs4 $[0]\ \mathsf{loc}(\mathsf{sas609}) \hat{=} \mathsf{run609}$.

5.3. Action Definitions

Four actions were declared in the narrative type specification. The following action type specification defines the meaning of those actions. For example, if fly(*plane*, $runway_1$, $runway_2$) is invoked between t_1 and t_2, then assuming the airplane is initially at $runway_1$, it will be in the air in the interval (t_1, t_2) and finally end up at $runway_2$ at time t_2.

acs1 $[t_1, t_2]\ \mathsf{fly}(plane, runway_1, runway_2) \rightsquigarrow [t_1]\ \mathsf{loc}(plane) \hat{=} runway_1 \rightarrow I((t_1, t_2)\ \mathsf{loc}(plane) \hat{=} \mathsf{air}) \wedge R([t_2]\ \mathsf{loc}(plane) \hat{=} runway_2)$

acs2 $[t_1, t_2]\ \mathsf{pickup}(person, pthing) \rightsquigarrow [t_1]\ \mathsf{loc}(person) \hat{=} value(t_1, \mathsf{loc}(pthing)) \rightarrow R((t_1, t_2]\ \mathsf{inpocket}(person, pthing))$

acs3 $[t_1, t_2]\ \mathsf{travel}(person, loc_1, loc_2) \rightsquigarrow [t_1]\ \mathsf{loc}(person) \hat{=} loc_1 \rightarrow R([t_2]\ \mathsf{loc}(person) \hat{=} loc_2)$

acs4 $[t_1, t_2]\ \mathsf{board}(person, plane) \rightsquigarrow [t_1]\ \mathsf{loc}(person) \hat{=} \mathsf{airport} \rightarrow R([t_2]\ \mathsf{loc}(person) \hat{=} value(t_2, \mathsf{loc}(plane)) \wedge \mathsf{onplane}(plane, person))$.

The following action occurrences are also needed. The exact timepoints used below were not specified in the RAH scenario, but have been chosen arbitrarily. Alternatively, exact timepoints could have been avoided by using nonnumerical temporal constants. Note, however, that many of the actions are concurrent, sometimes with partially overlapping intervals.

occ1 $[1, 2]\ \mathsf{pickup}(\mathsf{boris}, \mathsf{comb1})$
occ2 $[1, 2]\ \mathsf{pickup}(\mathsf{erik}, \mathsf{comb2})$
occ3 $[2, 4]\ \mathsf{travel}(\mathsf{dimiter}, \mathsf{home3}, \mathsf{office})$
occ4 $[3, 5]\ \mathsf{travel}(\mathsf{boris}, \mathsf{home1}, \mathsf{office})$
occ5 $[4, 6]\ \mathsf{travel}(\mathsf{erik}, \mathsf{home2}, \mathsf{office})$
occ6 $[6, 7]\ \mathsf{pickup}(\mathsf{boris}, \mathsf{gun})$
occ7 $[5, 7]\ \mathsf{travel}(\mathsf{dimiter}, \mathsf{office}, \mathsf{airport})$
occ8 $[7, 9]\ \mathsf{travel}(\mathsf{erik}, \mathsf{office}, \mathsf{airport})$
occ9 $[8, 10]\ \mathsf{travel}(\mathsf{boris}, \mathsf{office}, \mathsf{airport})$
occ10 $[9, 10]\ \mathsf{board}(\mathsf{dimiter}, \mathsf{sas609})$
occ11 $[10, 11]\ \mathsf{board}(\mathsf{boris}, \mathsf{sas609})$
occ12 $[11, 12]\ \mathsf{board}(\mathsf{erik}, \mathsf{sas609})$
occ13 $[13, 16]\ \mathsf{fly}(\mathsf{sas609}, \mathsf{run609}, \mathsf{run609b})$

5.4. Domain Constraints

We will define three domain constraints: No thing can be be carried by two persons at the same time, no person can be on board two planes at the same time, and any thing in a person's pocket must be at the same location as that person.

dom1 $\forall t, pthing, person_1, person_2$
$[person_1 \neq person_2 \wedge [t]\ \mathsf{inpocket}(person_1, pthing) \rightarrow [t] \neg \mathsf{inpocket}(person_2, pthing)]$

dom2 $\forall t, person, plane_1, plane_2$
$[plane_1 \neq plane_2 \wedge [t]\ \mathsf{onplane}(plane_1, person) \rightarrow [t] \neg \mathsf{onplane}(plane_2, person)]$

dom3 $\forall t, person, pthing$
$[[t]\ \mathsf{inpocket}(person, pthing) \rightarrow [t]\ \mathsf{loc}(pthing) \hat{=} value(t, \mathsf{loc}(person))]$.

5.5. Dependency Constraints

Now, apart from qualifications, only the side effects of actions remain to be modeled: Anything on board an airplane should follow the airplane, and anything a person carries should follow the person. The following two dependency constraints are sufficient for achieving this. For example, if someone is on board a plane and the location of the plane changes to *loc*, the location of the person also changes to *loc*.

dep1 $\forall t, plane, person, loc[$
$[t]\ \mathsf{onplane}(plane, person) \wedge C_T([t]\ \mathsf{loc}(plane) \hat{=} loc) \rightarrow R([t]\ \mathsf{loc}(person) \hat{=} loc)]$

dep2 $\forall t, person, pthing, loc[$
$[t]\ \mathsf{inpocket}(person, pthing) \wedge C_T([t]\ \mathsf{loc}(person) \hat{=} loc) \rightarrow R([t]\ \mathsf{loc}(pthing) \hat{=} loc)]$.

6. REPRESENTING THE QUALIFICATION PROBLEM IN TAL-Q

We have now modeled most of the Russian Airplane Hijack Scenario in TAL-Q, but we have not yet taken care of the qualifications defined by the scenario: Someone who carries a gun cannot board a plane, and someone who is drunk may or may not be able to board.

There are already a number of solutions to various aspects of the qualification problem in the literature, some of which would be applicable to the TAL logics. However, many of these solutions are dependent on the two state assumption with highly constrained action types. We would like to provide a solution that retains the following features of TAL:

- Any state, including the initial state, can be completely or incompletely specified using observations and domain constraints.
- Actions can be context-dependent and nondeterministic. They can have duration and internal state, and the duration may be different for different executions of the action. There may be concurrent actions with partially overlapping execution intervals.
- There can be dynamic processes continuously taking place independently of any actions that may occur.
- Domain constraints can be used for specifying logical dependencies between fluents generally true in every state or across states. They may vary over time.
- Actions can have side effects, which may be delayed and may affect the world at multiple points in time. They may in turn trigger other delayed or nondelayed side effects.

We would also like to retain the first-order reducibility of the circumscription axiom. The following restrictions and assumptions will apply. As discussed in Section 2.2, we will be satisfied with a solution where invoking a qualified action either has no effect or has some well-defined effect. We will also restrict the solution to the offline planning and prediction problems, and not claim a complete solution for the postdiction problem, which would require being able to conclude that an action was qualified because its successful execution would have contradicted an observation of some fluent value after that action was invoked.

6.1. Enabling Fluents

To handle the qualification problem, we will propose a default-based solution where each action type in a narrative is associated with an *enabling fluent*, a Boolean durational fluent with default value true and with the same number and type of arguments as the action type. This fluent will be used in the precondition of the action, and will usually be named by prefixing "poss_" to the name of the action. For example, the boarding action in the RAH scenario will be associated with an enabling fluent poss_board(*person*, *airplane*). We add a persistence statement for this fluent and modify **acs4** as follows:

per5 $\forall t, person, plane\ [\mathsf{true} \rightarrow Dur(t, \mathsf{poss\_board}(person, plane), \mathsf{true})]$

acs4′ $[t_1, t_2]\ \mathsf{board}(person, plane) \rightsquigarrow$
$[t_1]\ \mathsf{poss\_board}(person, plane) \wedge \mathsf{loc}(person) \hat{=} \mathsf{airport} \rightarrow$
$R([t_2]\ \mathsf{loc}(person) \hat{=} value(t_2, \mathsf{loc}(plane)) \wedge \mathsf{onplane}(plane, person))$.

The other action types are modified in a similar way (see Appendix A for more details).

Now, suppose that board(*person*, *plane*) is executed between timepoints t_1 and t_2. If poss_board(*person*, *plane*) is false at t_1 for some reason, the action is qualified, or *disabled*. On the other hand, if the fluent is true at t_1, the action is *enabled*. Of course, it can still be the case that the action has no effects, if other parts of its precondition are false.

To generalize this, a context-independent action that should have no effect at all when qualified can be defined using a simple action definition of the form

acs*m* $[t_1, t_2]\ \mathsf{action} \rightsquigarrow [t_1]\ \mathsf{poss\_action} \wedge \alpha \rightarrow R([t_2]\ \beta)$

where α is the precondition and β specifies the direct effects of the action (context-dependent actions are defined analogously). However, we also wanted to be able to define actions that do have some effects when they are qualified. This can be done by defining a context-dependent action that defines what happens when the enabling fluent is false:

acs*n* $[t_1, t_2]\ \mathsf{action} \rightsquigarrow ([t_1]\ \mathsf{poss\_action} \wedge \alpha_1 \rightarrow R([t_2]\ \beta_1)) \wedge$
$([t_1] \neg \mathsf{poss\_action} \wedge \alpha_2 \rightarrow R([t_2]\ \beta_2))$.

For example, suppose that whenever anyone tries to board a plane but the action is qualified, they should be thrown in jail. In order to model this, we would add a new persistent fluent in_jail(person) : boolean and modify the boarding action from Section 5.3 as follows:

acs4″ $[t_1, t_2]\ \mathsf{board}(\mathit{person}, \mathit{plane}) \rightsquigarrow$
$([t_1]\ \mathsf{poss\_board}(\mathit{person}, \mathit{plane}) \wedge \mathsf{loc}(\mathit{person}) \hat{=} \mathsf{airport} \rightarrow$
$R([t_2]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathit{value}(t_2, \mathsf{loc}(\mathit{plane})) \wedge \mathsf{onplane}(\mathit{plane}, \mathit{person}))) \wedge$
$([t_1] \neg \mathsf{poss\_board}(\mathit{person}, \mathit{plane}) \wedge \mathsf{loc}(\mathit{person}) \hat{=} \mathsf{airport} \rightarrow$
$R([t_2]\ \mathsf{in\_jail}(\mathit{person})))$.

In this alternative scenario, if anyone is at the airport and tries to board a plane, and the action is qualified, they will be thrown in jail. If they are at the airport but the action is not qualified, they will board the plane. If they are not at the airport, none of the preconditions will be true, and invoking the action will have no effect.

Regardless of whether a qualified action has an effect or not, its enabling fluent is a durational fluent with default value true. Therefore, the fluent will normally be true, and the action will normally be enabled. In the remainder of this section, we will examine some of the ways in which we can disable an action using strong and weak qualification.

6.2. Strong Qualification

Let us start with *strong* qualification. When an action is *strongly qualified*, it should definitely not succeed. This can be accomplished by forcing its enabling fluent to be false at the timepoint at which the action is invoked.

For example, suppose that when a person has a gun in his pocket, it should be impossible for that person to board a plane. Then, whenever inpocket(person, gun) holds, we must make poss_board false. This can be achieved using a dependency constraint:

dep3 $\forall t, \mathit{person}, \mathit{plane}\ [[t]\ \mathsf{inpocket}(\mathit{person}, \mathsf{gun}) \rightarrow$
$I([t] \neg \mathsf{poss\_board}(\mathit{person}, \mathit{plane}))]$.

At any timepoint t when a person has a gun in his pocket, we use the I macro both to occlude poss_board(*person*, *plane*) for all airplanes, thereby releasing it from the default value axiom, and to make it false. This implies that as long as a person has a gun in his pocket, poss_board will be false for that person on all airplanes. If the gun is later removed from the pocket, this dependency constraint will no longer be triggered. At that time, assuming no other qualifications affect the enabling fluent, it will automatically revert to its default value, true.

6.3. Weak Qualification

Although strong qualification can often be useful, we may sometimes want to express the fact that an action may succeed, or it may fail, depending on circumstances we may or may not be aware of. We call this *weak* qualification.

For example, we may want to model the fact that when a person is drunk, he *may or may not* be able to board an airplane, depending on whether airport security discovers this or not. We may not be able to determine within our model of the RAH scenario whether airport security does discover that any given person is drunk, and even if we could, it may be of no interest. In this case, whenever drunk(*person*) holds, we must release poss_board from the default value assumption:

dep4 $\forall t, person\ [[t]\ \mathsf{drunk}(person) \rightarrow X([t]\ \forall plane\ [\neg\, \mathsf{poss\_board}(person, plane)])]$.

At any timepoint t when a person is drunk, we occlude poss_board(*person*, *plane*) for all airplanes, but since we do not state anything about the *value* of the enabling fluent, it is allowed to be either true or false.

Although being able to state that an action *may* fail is useful in its own right, it is naturally also possible to restrict the set of models further by adding more statements to the scenario, which could make it possible to infer whether poss_board(dimiter, sas609) is true or false at some or all timepoints. For example, we may know that people boarding sas609 are always checked more carefully, so that it is impossible for anyone who is drunk to be on board that airplane, which could be expressed using a domain constraint. In the context of postdiction, observation statements could be used in a similar manner. For example, adding the observation statement **obs5** [13] onplane(sas609, boris) to the narrative would allow us to infer that Boris did in fact board the plane and that poss_board(boris, sas609) was in fact true. He would then end up at his intended destination. If instead we added the observation statement **obs6** [13] ¬ onplane(sas609, boris), we could infer that he was unable to board the plane and he did not end up at his destination.

The TAL-Q representation of the Russian Airplane Hijack Scenario from Section 3 is now complete. The full $\mathscr{L}$(ND) narrative is listed in Appendix A, and the translation into $\mathscr{L}$(FL) is shown in Appendix B.

The translation into $\mathscr{L}$(FL) was done using VITAL (Kvarnström and Doherty 1997), a research tool that can be used to study problems involving action and change within TAL and generate visualizations of action scenarios and preferred entailments. VITAL was also used for generating Figure 1, which is a color-coded summary of facts true in all preferred models of the RAH scenario. Light gray and dark gray stand for true and false values for Boolean fluents. Medium gray stands for an unknown value, and black stands for a value which is unknown but will be the same as that of the previous timepoint due to inertia. For non-Boolean fluents, "$*\,n\,*$" means that there are n possible values; the values are not shown in the diagram due to lack of space.

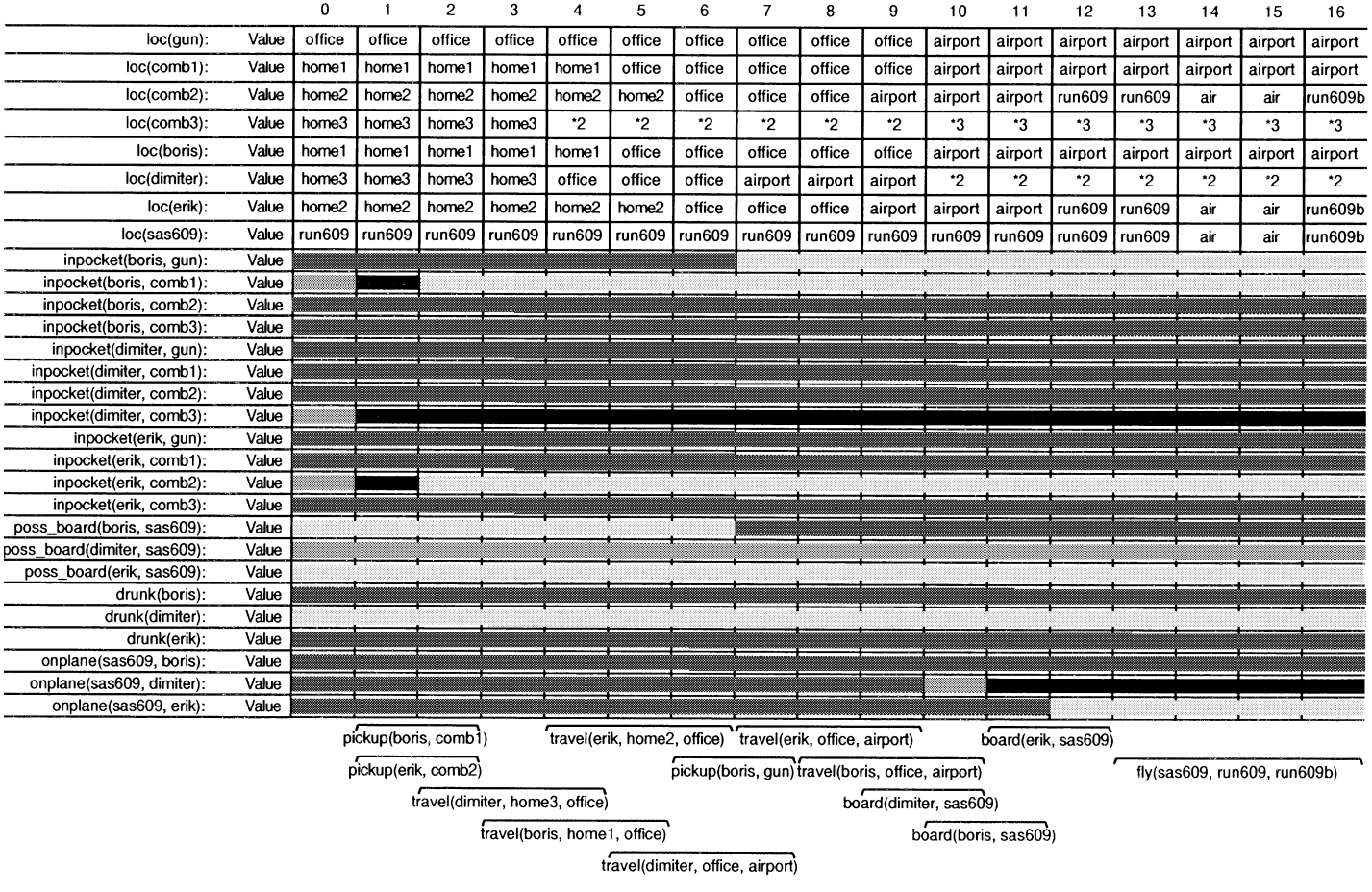

FIGURE 1. Timelines for the Russian Airplane Hijack Scenario.

In this scenario, Dimiter is drunk at all timepoints, and he attempts to board a plane at time 9. There will be two classes of preferred models: In one class, Dimiter will successfully board the plane, and in the other, he will not. As shown in Figure 1, we can not infer poss_board(dimiter, sas609) or its negation at any timepoint. In other words, we will not assume that the action succeeds merely because it is *possible* that it will succeed.

It should be noted that this approach has similarities to a standard default solution to the qualification problem, but with some subtle differences. For example, it permits more control of the enabling precondition, even allowing it to change during the execution of an action. More importantly, it involves no changes to the minimization policy already used in TAL to deal with the frame and ramification problems, and the circumscription policy inherits first-order reducibility.

7. ADDITIONAL ASPECTS CONCERNING THE QUALIFICATION PROBLEM

7.1. Qualification and Concurrency

One of the requirements we stated previously was that our solution should be able to handle concurrent actions. Here, there are two different cases, depending on whether the effects of the actions are independent or can interact in various ways. As we have seen when modeling the RAH scenario, the former case does not present a problem: Any number of people could attempt to board the plane at the same time, and the correct, intuitive conclusions would be obtained.

However, the latter case is far more interesting and presents a problem for approaches where actions are qualified when their successful execution would contradict a domain constraint, due to the difficulties associated with determining exactly which of all concurrent actions was the cause of the contradiction. It is more easily handled with an approach where qualifications are conditions evaluated in the state where the action is invoked, such as our TAL-Q approach.

Assume, for example, that it is impossible for two people to board the same airplane at the same time (a resource limitation problem). Similar situations have already been considered in the context of TAL-C in Karlsson and Gustafsson (1999), where bounds on concurrency and limited resources are handled using fluent dependency constraints. In this approach, actions are decoupled from their effects using *influences*, Boolean durational fluents which indicate that the world is inclined to change in some specific way, and a similar approach can be used for qualification. Below, we will show how the specific problem mentioned above can be modeled in TAL-Q using the same approach.

First, we add a new durational influence fluent want_to_board(person, plane) with default value false. We change the definition of board so that instead of altering the onplane fluent directly, the action simply makes want_to_board(person, plane) true at a single timepoint. Then, we add a new dependency constraint **dep5** that is triggered whenever want_to_board(person, plane) is true. This dependency constraint contains what were previously the direct effects of the action.

acs4‴ $[t_1, t_2]\ \mathsf{board}(person, plane) \rightsquigarrow I([t_2]\ \mathsf{want\_to\_board}(person, plane))$

dep5 $\forall t, person, plane\ [[t]\ \mathsf{want\_to\_board}(person, plane) \wedge \mathsf{poss\_board}(person, plane) \rightarrow I([t]\ \mathsf{onplane}(plane, person))]$.

The scenario above is essentially a reformulation of the original RAH scenario, and will entail exactly the same facts. However, it is modeled using the TAL-C influence framework, which provides some additional flexibility in reasoning about actions and their effects. Specifically, there is now a simple way to define what should happen when two people try to board the same plane at the same time. Clearly, for that airplane, want_to_board(*person*, *plane*) will be true for more than one person, and we must make poss_board(*person*, *plane*) false for all except one of them. We add a new fluent can_board(plane) : person whose value at any given time is the unique person that can board the plane at that time. We then add two dependency constraints: One stating that if there is at least one person trying to board a certain plane *plane*, then can_board(*plane*) will be one of those people, and one stating that can_board(*plane*) is the only person who can board *plane*.

dep6 $\forall t, plane\ [\exists person\ [[t]\ \mathsf{want\_to\_board}(person, plane)] \rightarrow \exists person_2$
$[[t]\ \mathsf{want\_to\_board}(person_2, plane) \wedge I([t]\ \mathsf{can\_board}(plane) \hat{=} person_2)]]$
dep7 $\forall t, plane, person\ [\neg([t]\ \mathsf{can\_board}(plane) \hat{=} person) \rightarrow$
$I([t] \neg \mathsf{poss\_board}(person, plane))]$.

It is easy to imagine several variations on this problem. For example, if two or more people try to board a plane simultaneously, it could be the case that none of them should succeed, or that there should be priorities (the "strongest" one should succeed). This can easily be modeled by adapting other techniques presented in Karlsson and Gustafsson (1999).

7.2. Qualification: Not Only For Actions

As we have shown, this approach to qualification is based on general concepts already present in earlier TAL logics, such as durational fluents and fluent dependency constraints, instead of introducing new predicates, entailment relations, or circumscription policies specifically designed for dealing with the qualification problem. This is appealing not only because we avoid introducing new complexity into the logic, but also because reusing these more general concepts adds to the flexibility of the approach. In this section, we will show how we can use exactly the same approach to specify qualifications not only for actions but for any generic rule or constraint.

Qualifying Qualification Constraints. When we initially considered the boarding action, the "natural" preconditions were that one had to be at the airport; this is the precondition encoded in the definition of board (**acs4**). Later, we found another condition that should qualify the action: No one should be able to board a plane carrying a gun. Now, however, we may discover that this qualification does not always hold: Airport security *should* be able to board a plane carrying a gun.

Assuming that there is a fluent is_security(person) : boolean, this exception to the general qualification rule could of course be modeled by changing the dependency constraint **dep3** in the following way:

dep3′ $\forall t, person, plane$
$[[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \wedge \neg \mathsf{is\_security}(person) \rightarrow$
$I([t] \neg \mathsf{poss\_board}(person, plane))]$.

However, we may later discover additional conditions under which a person should be able to board a plane with a gun, and we do not want to modify **dep3** each time.

Instead, the qualification itself should be qualified. Although we have so far only applied qualifications to actions, the same approach can easily be applied to the qualifications themselves. We add a new enabling fluent guns_forbidden(person, plane) : boolean for the qualification constraint, and we modify **dep3** as follows:

dep3″ $\forall t, person, plane\ [[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \wedge \mathsf{guns\_forbidden}(\mathsf{person}, \mathsf{plane}) \rightarrow I([t] \neg \mathsf{poss\_board}(person, plane))]$.

Now, we can qualify the qualification **dep3** simply by making guns_forbidden false for some person and airplane. In order to do this, we add a new dependency constraint:

dep8 $\forall t, person, plane\ [[t]\ \mathsf{is\_security}(person) \rightarrow I([t] \neg \mathsf{guns\_forbidden}(person, plane))]$.

Weakening Qualifications. It may also be the case that we want to qualify a strong qualification in order to "replace" it with a *weak* qualification. For example, suppose that a gun is made of a special kind of plastic that may or may not be detected by airport security. Assuming that we have already added dependency constraints **dep3″** and **dep8** as defined above, and that there is a fluent gun_is_plastic : boolean, we can achieve this in two different ways. First, we can use strong qualification for the guns_forbidden fluent, so that having a gun is definitely not a qualification to board, and then add a new weak qualification for the boarding action:

dep9 $\forall t, person, plane$
$[[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \wedge \mathsf{gun\_is\_plastic} \rightarrow I([t] \neg \mathsf{guns\_forbidden}(person, plane))]$

dep10 $\forall t, person, plane$
$[[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \wedge \mathsf{gun\_is\_plastic} \rightarrow X([t] \neg \mathsf{poss\_board}(person, plane))]$.

Second, we can use weak qualification for the guns_forbidden fluent, so that having a gun may or may not qualify the boarding action:

dep11 $\forall t, person, plane$
$[[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \wedge \mathsf{gun\_is\_plastic} \rightarrow X([t] \neg \mathsf{guns\_forbidden}(person, plane))]$.

Qualifying Dependency Constraints. As we have just shown, the same technique we used for qualifying actions could also be used for qualifying qualifications. Obviously, we could also apply the same technique to other parts of a narrative, such as ordinary dependency constraints. This allows us to express qualified side effects in TAL-Q, which we will demonstrate in Section 9.2.

7.3. Defining Enabling Fluents

In some approaches, qualification conditions are directly tied to specific actions, which can have certain advantages. For example, in our approach, it would have been possible to avoid the need to declare each enabling fluent and to explicitly include them in the corresponding action preconditions. This could be done by introducing a fixed *qualified*(t, a) predicate expressing the fact that a specific action a is qualified at a timepoint t, and then modifying the translation of action type specifications from

$\mathscr{L}$(ND) into $\mathscr{L}$(FL) in the appropriate manner. However, the fact that enabling fluents are ordinary fluents turns out to give us some additional flexibility in the way they are defined and used.

First, there is no strict requirement that a fluent must be *enabling*; we can also reverse its meaning and define a *disabling fluent*, if that is better suited for a particular scenario.

Second, there is of course also no formal requirement that the name of an enabling fluent is named by prefixing "poss_" to the name of the action—this is only a useful convention, which may be relaxed, especially when an enabling fluent is used for qualifying something other than an action.

Third, although our examples always associate a single unique enabling fluent with each action, it is possible to let multiple actions share the same enabling fluent, and one can also use multiple enabling fluents for the same action in order to model the fact that an action can be qualified for any of a set of possible reasons. This may be very useful when modeling larger scenarios. For example, if there is a robot that can move in four directions (actions move_north, move_south, move_east and move_west), and anything that makes the robot unable to move affects either all or none of these actions, we may want to use a single enabling fluent poss_move.

7.4. Interacting Qualifications

Since we are using two kinds of qualification—weak and strong—we must consider what will happen when an action is weakly and strongly qualified at the same time. By definition, this means that both $X([t]\neg \mathsf{poss\_action})$ and $I([t]\neg \mathsf{poss\_action})$ hold at the same timepoint t. But the X operator only releases the enabling fluent from the default value assumption, while the I operator both releases it and constrains its value; in this case, it forces poss_action to be false. In other words, the strong qualification takes precedence, and the action is strongly qualified.

7.5. Ramifications as Qualifications

Another problem related to the qualification problem occurs in formalisms where ramification constraints and qualification constraints are expressed as domain constraints (Ginsberg and Smith 1988; Lin and Reiter 1994). Assume, for example, that we are reasoning about the blocks world, and that have the following domain constraint (expressed using TAL syntax), stating that no two blocks can be on top of the same block:

dom $\forall t, x, y, z\ [[t]\ \mathsf{on}(x, z) \wedge \mathsf{on}(y, z) \rightarrow x = y]$.

Now, suppose that the direct effect of the action put(A, C) is on(A, C), and the action is executed in a state where on(B, C) is true. Then, we cannot determine syntactically whether the domain constraint should be interpreted as a ramification constraint (since no two blocks can be on top of C, B must be removed) or as a qualification constraint (since no two blocks can be on top of C, the action should fail).

In TAL-Q, however, all indirect effects of an action must be expressed as *directed* dependency constraints. Therefore, this problem simply does not arise. For example, if we want a ramification constraint, we can use the following dependency constraint:

dep $\forall t, x, y, z\ [[t]\ \mathsf{on}(x, z) \wedge C_T([t+1]\ \mathsf{on}(y, z)) \wedge x \neq y \rightarrow R([t+1]\neg \mathsf{on}(x, z))]$.

If x is on z, and we then place y on z, then an indirect effect is that x is removed from z.

On the other hand, if we want a qualification constraint, we can introduce an enabling fluent $\mathsf{poss\_put(A,C)}$ and add the following qualification condition:

dep $\forall t, x, y, z\ [[t]\ \mathsf{on}(x, z) \wedge x \neq y \rightarrow I([t] \neg \mathsf{poss\_put}(y, z))]$.

Clearly, the problem of determining whether a constraint should be interpreted as a qualification or a ramification does not arise in this approach.

8. ALTERNATIVE APPROACHES TO THE QUALIFICATION PROBLEM

We have now presented one approach to solving the qualification problem within the TAL framework, but this approach is certainly not the only one. Below, we will examine in somewhat less detail some alternative approaches.

8.1. Using Domain Constraints

Although our main approach to the qualification problem is based on qualifying an action whenever a condition holds in the state in which it is invoked, it is also interesting to investigate approaches based on qualifying an action whenever its execution would contradict a domain constraint.

One variation of this approach would involve simply adding the proper domain constraints to the scenario, and concluding that an action is qualified whenever the resulting narrative is inconsistent. For example, the constraint that no guns are allowed on board airplanes can be stated as follows:

dom4 $\forall t, \mathit{plane}\ [[t] \neg (\mathsf{loc(gun)} \triangleq \mathit{value}(t, \mathsf{loc}(\mathit{plane})))]$.

Now, assume that we add this constraint to the initial version of the RAH scenario (from Section 5), where qualification was not considered. Since Boris tries to board the plane carrying a gun, we can infer that the gun will be on board the plane, but from **dom4** we can infer that no gun will ever be on board a plane, so the scenario is inconsistent, which means that some action must be qualified.

Obviously, this approach does not provide the correct conclusions about the results of invoking a qualified action, and due to the inconsistency it may not even seem like a solution at all. However, as discussed in Section 2.2, there are some cases where such approaches may still be useful, such as when we are doing planning. But even if this is the case, a more serious problem still occurs when this approach is used together with nondeterministic actions or incomplete information about the initial state. For example, suppose that Dimiter may (or may not) have a gun in the initial state. If he tries to board a plane, **dom4** will allow us to infer that he did *not* have a gun, when the intuitive conclusion would have been that the action may or may not be qualified.

8.2. Fault Fluents

By modifying the previous approach slightly, we can define another approach that may also have its uses. Instead of stating that a certain domain constraint must hold, we state that whenever it does *not* hold, a *fault fluent* should become true. For

example, **dom4** from the previous section can be modified as follows:

dom4′ $\forall t, plane\ [[t]\ \mathsf{loc(gun)} \triangleq value(t, \mathsf{loc}(plane)) \rightarrow I([t]\ \mathsf{fault\_gun\_on\_airplane})]$.

Now, whenever someone is on board a plane and is carrying a gun, fault_gun_on_airplane will be true. From this, the agent can infer that an action must have been qualified.

This approach has several advantages. First, if some action is qualified, it is easier to find out which one and why it was qualified, since only the fault fluents need to be considered. Second, invoking a qualified action does not make the entire narrative inconsistent. And third, incomplete information and nondeterministic actions are not a problem. As before, suppose that Dimiter may (or may not) have a gun in the initial state. If he tries to board a plane, there will be two classes of models: One in which he did not have a gun and boards the plane without triggering the fault_gun_on_airplane fault fluent, and one where he did have a gun, boards the plane, and does make fault_gun_on_airplane true.

Unfortunately, the fault fluent approach still does not provide the correct conclusions if there is some qualified action, since it assumes that all actions succeed. However, there is another use for this approach, for which it appears to be perfectly suited. As mentioned in Section 2.3, qualification has sometimes been used for predicting whether the result of invoking a certain action would be *undesirable*. This has usually resulted in predicting that invoking an undesirable action is impossible or has no effect, when, in reality, invoking the action would be possible and the action would have its undesirable effects.

Probably, what is needed for such scenarios is not the use of qualification but the use of a similar *mechanism* for providing "undesirability" conditions in an intuitive and modular way. That can be provided by the fault fluent approach, which always predicts that an action succeeds but can "flag" undesirable results by making a fault fluent true. For this task, the fault fluent approach would provide the correct results.

An interesting feature of this approach is that it can easily be combined with our main approach: True qualifications may be expressed as conditions holding in the invocation state, and undesirable results are expressed in terms of conditions that should hold in the resulting state.

9. ADDITIONAL EXAMPLES

In this section, we will show how some qualification examples from the literature can be represented in TAL-Q, and we will also show an extension to one of those examples. Since the narrative type specifications are obvious from the examples, they will be omitted.

9.1. Dead Birds Don't Walk

We will begin with a relatively straightforward qualification example. There is a turkey, Fred, who can take walks. One constraint on this world is that it is not possible to walk when you are dead. Therefore, if Fred dies, one should conclude that he is no longer walking. On the other hand, if the walk action for Fred is invoked, we would intuitively want the action to be qualified—Fred should not suddenly become alive in order to satisfy the domain constraint (McCain and Turner 1995).

In the TAL-Q representation of this scenario, we use the two Boolean fluents alive and walking mentioned above, but we also need one enabling fluent per action type (**acs1**, **acs2**). If Fred is not alive, he cannot be walking (**dom1**). A domain constraint is not adequate for inferring directed side effects for actions. In this case, we use a dependency constraint (**dep1**) stating that when Fred dies (not at every timepoint where he is dead—note the use of C_T, "changes to true"), he stops walking. We also need a qualification (**dep2**) stating that when Fred is dead (C_T is not used), he cannot start walking. Together with the observation statement **obs1** and the action occurrences **occ1** and **occ2**, this allows one to infer that Fred is initially walking, then dies (and stops walking), and then cannot resume walking.

acs1 $[t_1, t_2]$ Die $\rightsquigarrow [t_1]$ poss_die $\rightarrow R([t_2] \neg$ alive)
acs2 $[t_1, t_2]$ Walk $\rightsquigarrow [t_1]$ poss_walk $\rightarrow R([t_2]$ walking)
dom1 $\forall t\ [[t] \neg$ alive $\rightarrow \neg$ walking]
dep1 $\forall t[C_T([t] \neg$ alive$) \rightarrow R([t] \neg$ walking$)]$
dep2 $\forall t\ [[t] \neg$ alive $\rightarrow I([t] \neg$ poss_walk$)]$
obs1 $[0]$ alive $\wedge$ walking
occ1 $[0, 1]$ Die
occ2 $[1, 2]$ Walk.

9.2. A Simple Electric Circuit

Thielscher (1997) discusses qualified ramifications and presents a scenario in which there is an electric circuit with two batteries bat1 and bat2, two switches sw1 and sw2, and one light bulb. There is only one action, toggle(switch), whose only direct effect is that the given switch is toggled.

If you close switch sw1, the first battery is connected to the light bulb. Normally, this has the side effect that the light is turned on. But there are three qualifications to this ramification: The light is not turned on if the bulb is broken, if bat1 is malfunctioning, or if the wiring is loose. Similarly, if you close switch sw2, the second battery is connected to the light bulb. Unfortunately, the voltage is too high, so usually this will have the side effect that the bulb breaks. But here, there are also some qualifications: The bulb does not break if bat2 is malfunctioning, or if the wiring is loose. Finally, there is normally no light when the bulb is broken.

Although our approach does not handle qualification for postdiction, we can easily handle the prediction problem for this scenario. One possible formalization is the following, using the persistent fluents closed(switch), light, broken, malfunc(battery) and loose_wiring and the enabling fluents poss_light, poss_break and no_light_when_broken.

obs1 $[0] \neg$ closed(sw1) $\wedge \neg$ closed(sw2)
obs2 $[0] \neg$ broken $\wedge \neg$ loose_wiring $\wedge \forall$ *battery* $[\neg$ malfunc(*battery*)$]$
acs1 $[t_1, t_2]$ toggle(*switch*) $\rightsquigarrow ([t_1]$ closed(*switch*)$) \rightarrow R([t_2] \neg$ closed(*switch*$))) \wedge$
$([t_1] \neg$ closed(*switch*$) \rightarrow R([t_2]$ closed(*switch*$)))$
dom1 $\forall t\ [[t]$ no_light_when_broken $\rightarrow$ (broken $\rightarrow \neg$ light$)]$
dep1 $\forall t\ [[t]$ poss_light $\wedge\ C_T([t]$ closed(sw1)$) \rightarrow R([t]$ light$)]$
dep2 $\forall t\ [[t]$ poss_break $\wedge\ C_T([t]$ closed(sw2)$) \rightarrow R([t]$ broken$)]$
dep3 $\forall t\ [[t]$ broken $\vee$ malfunc(bat1) $\vee$ loose_wiring $\rightarrow I([t] \neg$ poss_light$)]$
dep4 $\forall t\ [[t]$ malfunc(bat2) $\vee$ loose_wiring $\rightarrow I([t] \neg$ poss_break$)]$.

9.3. Yellow Blocks Are Forbidden

Returning once more to scenarios where only actions are qualified, we will now consider a scenario presented in Lin and Reiter (1994): A blocks world scenario where blocks may have different colors. There is a single robot that can paint blocks (paint(*block*, *color*)), but since yellow is traditionally reserved for the emperor, the robot is not allowed to paint any block yellow. In Lin and Reiter, this is handled using qualification, by adding a domain constraint stating that no block may be yellow. Consequently, in that approach, the preconditions of the action paint(*block*, yellow) will always be false.

One possible translation to TAL-Q would use two domains, block and color, a fluent col(block): color, and an enabling fluent poss_paint(block, color) : boolean with default value true, together with the following $\mathscr{L}$(ND) statements. (Note that in Lin and Reiter 1994, all fluents will be undefined in the state resulting from invoking paint(x, yellow), while in our approach, the action will have no effect.)

obs1 $[0]\ \forall b\ [\neg(\mathsf{col}(b) \hat{=} \mathsf{yellow})]$
acs1 $[t_1, t_2]\ \mathsf{paint}(b, c) \rightsquigarrow [t_1]\ \mathsf{poss\_paint}(b, c) \rightarrow R([t_2]\ \mathsf{col}(b) \hat{=} c)$
dep1 $\forall t, b\ [I([t] \neg \mathsf{poss\_paint}(b, \mathsf{yellow}))]$.

However, the fault fluent approach (Section 8.2) may be more appropriate, since it is more likely that the action would actually succeed, even though its effects were "illegal":

acs2 $[t_1, t_2]\ \mathsf{paint}(b, c) \rightsquigarrow R([t_2]\ \mathsf{col}(b) \hat{=} c)$
dep2 $\forall t, b\ [[t]\ \mathsf{col}(b) \hat{=} \mathsf{yellow} \rightarrow I([t]\ \mathsf{fault\_block\_is\_yellow}(b))]$.

Using this approach, we will predict that painting a block yellow will succeed, but also that the fault fluent fault_block_is_yellow will become true for that block: We have performed an action that has undesirable results.

9.4. The Lenient Emperor

There is also a variation of the previous scenario in which the emperor is more lenient and allows at most one yellow block to exist. If we had thought ahead and provided an enabling fluent for **dep1** above, we could have handled this by qualifying the old qualification. Since we did not, we have to modify the existing qualification **dep1**. For example, it can be replaced with the following constraint:

dep1′ $\forall t\ [\exists b\ [[t]\ \mathsf{col}(b) \hat{=} \mathsf{yellow}] \rightarrow \forall b\ [I([t] \neg \mathsf{poss\_paint}(b, \mathsf{yellow}))]]$.

If we want to be able to paint a yellow block yellow again, we can use the following alternative:

dep1″ $\forall t\ [\exists b\ [[t]\ \mathsf{col}(b) \hat{=} \mathsf{yellow}] \rightarrow$
$\forall b\ [[t] \neg(\mathsf{col}(b) \hat{=} \mathsf{yellow}) \rightarrow I([t] \neg \mathsf{poss\_paint}(b, \mathsf{yellow}))]]$.

Again, the fault fluent approach may be more appropriate: If more than one block is yellow, we signal an error for each yellow block.

dep2′ $\forall t\ [\exists b_1, b_2\ [[t]\ \mathsf{col}(b_1) \hat{=} \mathsf{yellow} \wedge \mathsf{col}(b_2) \hat{=} \mathsf{yellow} \wedge b_1 \neq b_2] \rightarrow$
$\forall b\ [[t]\ \mathsf{col}(b) \hat{=} \mathsf{yellow} \rightarrow I([t]\ \mathsf{fault\_block\_is\_yellow}(b))]]$.

9.5. The Lenient Emperor—with Concurrency

An interesting variation of the lenient emperor scenario, which has not previously been considered in the literature, arises when there may be more than one agent in the world. For example, it may be the case that when no block is yellow, but a number of agents concurrently attempt to paint two or more blocks yellow, exactly one of them will succeed.

A similar scenario was discussed in Section 7.1, where at most one person could board a plane at any given timepoint. That, however, would be analogous to allowing at most one *new* yellow block at each timepoint. However, it turns out that the concurrent lenient emperor scenario can be modeled in a similar manner. First, we will reformulate the scenario using the TAL-C approach, using a durational influence fluent want_to_paint(block, color):

obs1 $[0]\ \forall b\ [\neg(\mathsf{col}(b) \hat{=} \mathsf{yellow})]$
acs1 $[t_1, t_2]\ \mathsf{paint}(b, c) \rightsquigarrow I([t_2]\ \mathsf{want\_to\_paint}(b, c))$
dep1 $\forall t, b, c\ [[t]\ \mathsf{want\_to\_paint}(b, c) \wedge \mathsf{poss\_paint}(b, c) \rightarrow R([t]\ \mathsf{col}(b) \hat{=} c)]$.

Although the influence fluent is not strictly necessary for this example, it can still be an advantage to model the scenario in this way due to the added flexibility in case the scenario ever needs to be changed. In this case, however, the important difference in the new scenario is that in **dep1**, poss_paint must be true at the *same* timepoint when the block should change color. This means that we only need to make sure that whenever any block is yellow, no block *except possibly that one* can be painted yellow. Note that this allows us to repaint a yellow block with the same color, and it also allows us to concurrently paint one block yellow and paint another, previously yellow block in another color.

dep2 $\forall t, b_1, b_2\ [[t]\ \mathsf{col}(b_1) \hat{=} \mathsf{yellow} \wedge b_1 \neq b_2 \rightarrow I([t] \neg \mathsf{poss\_paint}(b_2, \mathsf{yellow}))]$.

For this scenario, the fault fluent approach would be identical to that for the non-concurrent lenient emperor scenario.

10. COMPARISONS

Having considered some qualification examples and how they can be represented in TAL-Q, we will now compare our approach to some other approaches in the literature, beginning with McCarthy's introduction of circumscription (1980, 1986) and continuing with Lifschitz (1987), Shanahan (1997), Ginsberg and Smith (1988), Lin and Reiter (1994), McCain and Turner (1995), and finally Thielscher (1996a, 1996b).

Although these approaches have many differences, there are also many important similarities. Perhaps the most important of these similarities is that all of these approaches are based on the assumption that there is a single agent executing a simple sequence of actions without duration, and that all change in the world is caused by that agent. For example, there can be no concurrent actions, no delayed side effects, and no dynamic processes taking place in the background. Sometimes, not even nondeterministic actions are allowed. In other words, these approaches are not expressive enough to model the Russian Airplane Hijack Scenario.

For some of the approaches, it may be possible to extend them for more complex worlds without requiring major changes—in other words, a graceful scaling up. As we will see, however, several approaches are strongly dependent on the fact that actions

and side effects can be represented as a function from the current state and the action to be performed to the successor state, or possibly the set of successor states. This is especially true for the approaches where qualification is based on constraints that must not be violated by an action, rather than on conditions that must or must not hold when the action is invoked (Ginsberg and Smith 1988; Lin and Reiter 1994).

10.1. McCarthy

McCarthy (1977, 1980) introduces circumscription and discusses how it can be used for conjecturing that any action will succeed unless there is anything preventing its success. This is achieved using a *prevents*(*reason*, *action*, *state*) predicate which holds whenever some specific reason prevents an action from having its usual effects in the given state. Each such reason is then defined explicitly. For example, in a blocks world, we may say that heavy blocks cannot be moved: $\forall x, y, s.(tooheavy(x) \rightarrow prevents(weight(x), move(x, y), s))$. The *prevents* predicate is circumscribed relative to the conjunction of all such reasons, which allows us to predict that the action will succeed unless one of its qualifications holds when the action is invoked.

Clearly, this is very similar to the way we defined our main approach in Section 6. Like our approach, it can not be used for inferring qualifications based on observing that an action failed, since *prevents* is only circumscribed relative to the explicit qualification conditions. One important difference, however, is that our approach does not minimize qualifications—we minimize the occlusion predicate, which means that we minimize *potential* qualification. This is what allows us to express weak qualification.

In McCarthy (1986), a slightly different approach is used within the situation calculus. Instead of using a *prevents* predicate for qualification, a single *ab* ("abnormal") predicate is used for both qualification and many other tasks. The argument of *ab* is an *aspect*, an abstract object. For example, in the blocks world, we can move a block to a location unless the *move* action is abnormal in the first aspect: $\forall x, l, s. \neg ab(aspectl(x, move(x, l), s))) \rightarrow loc(x, result(move(x, l), s)) = l$. Then, we may not be able to lift heavy blocks: $\forall x, l, s.(tooheavy(x) \rightarrow ab(aspectl(x, move(x, l), s)))$. An interesting aspect of this approach is that if a qualified action is invoked, each fluent which would normally have been affected is released from the inertia assumption, but is not given a new value and is therefore allowed to vary freely. Fluents which would not have been affected retain their previous values.

10.2. Lifschitz: Formal Theories of Action

Unfortunately, as Lifschitz (1987) notes, global minimization of abnormality is not sufficient, since it sometimes leads to unintended models. He presents an alternative solution for the prediction task (or *temporal projection*), where it is assumed that all changes in the values of fluents are caused by actions.

Two new predicates are added to the situation calculus: $causes(a, p, f)$ expresses that the action a causes the primitive fluent p to have the same value that the fluent f had when the action was invoked, and $precond(f, a)$ expresses that the fluent f is a precondition to the action a. Given these new predicates, it is possible to define any number of preconditions to an action in an incremental manner. The situation-independent predicates *causes* and *precond* are then circumscribed, and an action is assumed to succeed iff all its preconditions hold when the action is invoked. If any of

its preconditions do not hold, the action will be assumed to have no effect on the world.

This approach produced the correct results for the scenarios where McCarthy's earlier approach failed. However, apart from allowing more complex worlds to be modeled, the approach presented in Section 6 is also more flexible in the way qualification conditions can be specified. For example, our qualification conditions may vary over time, and may also depend on states other than the state in which the action is invoked. Due to the fact that enabling fluents are not directly tied to actions, we can also represent qualified qualifications and qualified side effects, while Lifschitz' approach does not allow side effects at all.

10.3. Shanahan: Solving the Frame Problem

Shanahan (1997) uses an approach similar to that of Lifschitz (1987), the main difference being that the *precond* predicate takes three arguments: $precond(f, v, a)$ expresses the fact that the action a is only executable when the fluent f has the value v. Consequently, the two approaches share many of the same advantages and disadvantages. As in Lifschitz' approach, if any precondition does not hold, an action will be assumed to have no effect on the world.

10.4. Ginsberg and Smith: Reasoning about Action II—The Qualification Problem

Ginsberg and Smith (1988) argue that specifying qualifications as preconditions to actions often leads to complicated formulas, due to the need to take all possible ramifications into account. Accordingly, they define a possible worlds approach in which each action is associated with a set of qualification constraints on the form of domain constraints. Given an action, the set of possible successors of the current world is first calculated without considering the qualification constraints. Then, any such world which does not satisfy all qualification constraints is discarded. If no possible successor remains, the action was qualified, and is assumed not to change the world at all.

This approach works very well for the examples examined by Ginsberg and Smith. However, if concurrent actions or delayed side effects were allowed, it would no longer be possible to reason about whether a single action would violate a domain constraint: For concurrent actions, it would be necessary to take into account all actions being performed at the same time, and it would be more difficult to determine exactly which action should be qualified. Similarly, if delayed side effects were allowed, one would have to know exactly which actions are invoked up to the time when the delayed side effect takes place. Qualified side effects would of course be even more problematic, since one would have to determine somehow whether it is the action itself or one of its side effects that should be qualified. In other words, this approach would be quite difficult to extend to handle complex scenarios such as the Russian Airplane Hijack Scenario.

10.5. Lin and Reiter: State Constraints Revisited

Lin and Reiter (1994) present a solution to the qualification problem within the situation calculus. The solution is based on generating an exact definition of the $Poss(a, s)$ predicate, which states that it is possible to execute the action a in the state

s. The definition of *Poss* is generated using both a set $\mathscr{D}_{nec}$ of formulas of the form $Poss(a, s) \supset \phi$ and a set $\mathscr{D}_{qual}$ of domain constraints that must hold in the state resulting from executing any action. The domain constraints in $\mathscr{D}_{qual}$ are regressed, and the results are combined with the formulas in $\mathscr{D}_{nec}$ to form an exact definition of *Poss*.

Since all qualification conditions are compiled into the definition of the *Poss* predicate, it is possible to infer that an action is qualified by evaluating *Poss* in the current situation. The situation $do(a, s)$ resulting from executing a qualified action a is completely undefined, since the successor state axioms only define fluent values in situations resulting from executing actions whose preconditions hold.

Like the approach used by Ginsberg and Smith, this solution also depends on the restricted expressivity of the logic being used—in fact, it does so to an even greater degree, due to the compilation of qualification conditions into a definition of *Poss*.

For example, if nondeterministic actions were allowed, we may only know that an action *may* contradict a domain constraint, so finding an exact definition of *Poss* would not be possible. Similarly, if actions with duration and internal state were introduced, the compilation procedure would be far more complicated due to the need to ensure that no intermediate state contradicts the domain constraints in $\mathscr{D}_{qual}$. If concurrent actions, delayed side effects or domain constraints referring to multiple states or domain constraints depending on time were allowed, this approach could not be used at all, since it would not be sufficient to consider the single action and situation used as arguments to the *Poss* predicate.

On the other hand, if the world one is reasoning about is simple enough, this solution does provide a way of specifying qualification constraints that is often more intuitive than using enabling fluents.

10.6. McCain and Turner: A Causal Theory of Ramifications and Qualifications

McCain and Turner (1995) provide a combined solution to the ramification and qualification problems in which every change must be *caused*. An action is qualified if it would imply a change that it did not cause. Causal laws are expressed on the form $\phi \Rightarrow \psi$ (if ϕ holds, ψ is caused to hold). Pure ramifications can be expressed on the form $True \Rightarrow \phi$, and pure qualifications on the form $\neg \phi \Rightarrow False$, but other forms of constraints can also be used.

A pure qualification $\neg \phi \Rightarrow False$ essentially defines a condition that must hold in any state resulting from executing an action. It does not cause fluents to change as a side effect of executing the action, but if the condition ϕ does not hold, *False* must hold in any resulting state, so there can be no resulting state, which means that the action was qualified. It is also possible to express "combined" ramification and qualification constraints. For example, the constraint $\neg Alive \Rightarrow \neg Walking$ may act as a ramification when *Alive* is caused to become false, but as a qualification when *Walking* is caused to become true.

Since the result of invoking a qualified action in this approach is an empty set of possible resulting states, it is not possible to reason about which value a fluent would take on after a qualified action was invoked; it is only possible to determine that the action would be qualified. Therefore, this approach is mainly useful for planning, or for prediction in the case where we are not interested in the result of invoking a qualified action.

If we consider an empty set of possible resulting states in this approach to be equivalent to an inconsistent scenario in the TAL formalism, any qualification con-

straint that can be expressed in this approach—pure or not pure—can also be expressed using our alternative approach from Section 8.1. Each qualification constraint becomes an ordinary domain constraint, while each ramification constraint is expressed as a fluent dependency constraint.

On an abstract level, this solution is very similar to the approaches used by Ginsberg and Smith (1988) and Lin and Reiter (1994), in the sense that pure qualifications are domain constraints that must not be violated by an action. The solution also has similar limitations in expressibility. However, the technical solution and the reasoning behind it are different, and so are the sets of possible states resulting from invoking a qualified action: In Ginsberg and Smith there was a single possible resulting state where nothing had changed, in Lin and Reiter the result was undefined, and in McCain and Turner (1995), there is no resulting state.

10.7. Thielscher: Causality and the Qualification Problem

Thielscher's (1996a, 1996b) approach to the qualification problem is quite different from the previous three approaches. In fact, it turns out to be more similar to the approach we have presented in Section 6 in that it uses fluents to represent qualifications. On the other hand, there are also quite a few differences.

Thielscher uses persistent *disqualification fluents*, which are assumed to be false in the *initial state* unless something forces them to be true, while our enabling fluents are durational, and are normally true in every state. While using durational fluents has the advantage of not needing to explicitly make a disqualification fluent false when a qualification no longer holds, Thielscher's approach has the advantage of being able to handle some cases of postdiction. For example, if the start action can only be qualified when potato is true, and if we observe that the action is qualified, then the conclusion would be that potato must have been true in the initial state.

This, of course, does not handle the case where we initially observed that there was no potato, then waited a while and tried to start the car, and starting failed. Thielscher handles this using *miraculous disqualification*, which allows an action to be qualified even though every explanation for its qualification is proven to be false, and which is globally minimized at a higher priority than ordinary qualification. (Unfortunately, this also allows us to prove that there was in fact no potato.) There is also a method for qualifying ramification constraints within the same framework (Thielscher 1997).

The result of executing a sequence $\langle a_1, \ldots, a_n \rangle$ of actions is a state defined by the function $Res(\langle a_1, \ldots, a_n \rangle)$, which is undefined when some action in the sequence is qualified. However, an observation on the form F after $\langle a_1, \ldots, a_n \rangle$ is still defined in this case, although it is always false for any formula F. This is yet another different definition of the state resulting from invoking a qualified action: Not even the tautology T is considered to hold.

But although this approach has certain advantages, it once again assumes a world where there are no concurrent actions, no actions with duration and internal state, no dynamic processes in the background, no delayed side effects, and no qualified side effects, and therefore, it would not be possible to model the Russian Airplane Hijack Scenario with this approach.

10.8. Summary

We have compared our approach to six other approaches in the literature. There turn out to be some similarities between all of these approaches, perhaps most

importantly that they are designed for simple worlds in which a single agent performs actions in a sequential manner, and where any side effects, if allowed at all, take place in the ending state of the action. Therefore, none of these approaches are powerful enough to model the Russian Airplane Hijack Scenario, although for simpler scenarios they sometimes provide more intuitive methods for specifying which actions are qualified.

There also appear to be two main approaches to the way in which qualification conditions are specified: Either as conditions holding in the initial state or as domain constraints that must not be violated by actions. Here, Thielscher's approach is an exception: It is possible to directly observe the qualification of an action, after which one can postdict the reasons for the qualification.

However, there is also one aspect in which the approaches are quite different: If we execute a qualified action, what can be said about the resulting state? Most approaches turn out to have their own answer to this question:

- In McCarthy (1986), the fluents that would ordinarily be affected by the action are released from the inertia assumption in the resulting state, but all other fluents remain inert. This could be emulated in TAL-Q using the X operator.
- In Lifschitz (1987), Shanahan (1997), and Ginsberg and Smith (1988), the action has no effect on the world. This is normally also the case in TAL-Q, unless an alternative effect has been specified.
- In Lin and Reiter (1994), the resulting state is completely undefined.
- In McCain and Turner (1995), there is no resulting state. This could be considered equivalent to an inconsistent scenario in TAL-Q.
- In Thielscher (1996a, 1996b), the result is a "state" in which *nothing* holds—not even a tautology.

11. CONCLUSION

We have presented an approach to the qualification problem based on the use of dependency constraints and durational fluents in the context of a highly expressive temporal logic of action and change called TAL-Q. TAL-Q permits the use of action types that are nondeterministic, context dependent, durational, and concurrent. This degree of expressivity introduces additional issues in solving the qualification problem not present in any of the previously proposed formalisms and solutions in the literature. We have also tried to show that whether any given approach to solving the qualification problem is useful or not often depends on both the reasoning task and the characteristics of the class of worlds we are interested in reasoning about. Although many solutions have been proposed in the literature, they often do not make such assumptions explicit, and often turn out to be useful only for a small class of worlds. The intent of this article was to present a solution to the qualification problem for TAL-Q in this context. Several of the ideas in the article are tentative and will be pursued in future research. One of the more important topics of research is to clarify the distinctions between online and offline reasoning modes and how these modes affect solutions to the qualification problem. In addition, pursuing the formal assessment of correctness for the proposed solutions to the qualification problem using TAL-Q is an important future research issue as are more formal comparative analyses of the alternative formalisms considered in the article.

ACKNOWLEDGMENTS

This research was supported in part by the Wallenberg Foundation, the Swedish Research Council for Engineering Sciences and the ECSEL/ENSYM graduate studies program.

REFERENCES

DOHERTY, P. 1994. Reasoning about action and change using occlusion. *In* Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94). John Wiley & Sons, New York, pp. 401–405.

DOHERTY, P. 1996. PMON$^+$: A fluent logic for action and change, formal specification, version 1.0. Technical Report LITH-IDA-96-33, Department of Computer and Information Science, Linköping University, Linköping, Sweden. http://www.ida.liu.se/publications/techrep/96/tr96.html.

DOHERTY, P., J. GUSTAFSSON, L. KARLSSON, and J. KVARNSTRÖM. 1998. TAL: Temporal Action Logics, language specification and tutorial. Linköping Electronic Articles in Computer and Information Science, **3**(15). http://www.ep.liu.se/ea/cis/1998/015.

DOHERTY, P., and J. KVARNSTRÖM. 1998. Tackling the qualification problem using fluent dependency constraints: Preliminary report. *In* Proceedings of the Fifth International Workshop on Temporal Representation and Reasoning (TIME-98). *Edited by* L. Khatib and R. Morris. IEEE Computer Society, Los Alamitos, CA, pp. 97–104.

DOHERTY, P., and W. ŁUKASZEWICZ. 1994. Circumscribing features and fluents. *In* Proceedings of the First International Conference on Temporal Logic. *Edited by* D. Gabbay and H. J. Ohlbach. Volume 827 of Lecture Notes in Artificial Intelligence. Springer-Verlag, NY, pp. 82–100.

GINSBERG, M. L., and D. E. SMITH. 1988. Reasoning about action {II}: The qualification problem. Artificial Intelligence Journal, **35**:311–342.

GIUNCHIGLIA, E., and V. LIFSCHITZ. 1995. Dependent fluents. *In* Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95). Morgan Kaufmann Publishers, San Francisco, pp. 1964–1969.

GUSTAFSSON, J., and P. DOHERTY. 1996. Embracing occlusion in specifying the indirect effects of actions. *In* Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96). *Edited by* L. C. Aiello, J. Doyle, and S. C. Shapiro. Morgan Kaufmann Publishers, San Francisco, pp. 87–98.

KARLSSON, L., and J. GUSTAFSSON. 1999. Reasoning about concurrent interaction. Journal of Logic and Computation, **9**(5):623–650.

KVARNSTRÖM, J., and P. DOHERTY. 1997. VITAL: A research tool for visualizing and querying action scenarios in TAL. http://www.ida.liu.se/~jonkv/vital.html.

LIFSCHITZ, V. 1987. Formal theories of action. *In* Proceedings of the Workshop on the Frame Problem in Artificial Intelligence. Morgan Kaufmann Publishers, San Francisco.

LIN, F., and R. REITER. 1994. State constraints revisited. Journal of Logic and Computation, **4**:655–678.

MCCAIN, N., and H. TURNER. 1995. A causal theory of ramifications and qualifications. *In* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95). Morgan Kaufmann Publishers, San Francisco.

MCCARTHY, J. 1977. Epistemological problems of artificial intelligence. *In* Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77). Morgan Kaufmann Publishers, San Francisco, pp. 1038–1044. http://www-formal.stanford.edu/jmc/epistemological.html.

MCCARTHY, J. 1980. Circumscription—A form of non-monotonic reasoning. *Artificial Intelligence*, **13**:27–39. http://www-formal.stanford.edu/jmc/circumscription.html. [Reprinted in McCarthy, J. 1990. Formalization of Common Sense, Papers by John McCarthy. *Edited by* V. Lifschitz. Ablex.]

MCCARTHY, J. 1986. Applications of circumscription to formalizing common sense knowledge. Artificial Intelligence, **28**:89–116. http://www-formal.stanford.edu/jmc/applications.html. [Re-

printed in McCarthy, J. 1990. Formalization of Common Sense, Papers by John McCarthy. *Edited by* V. Lifschitz. Ablex, Norwood, NJ.

SANDEWALL, E. 1989. Filter preferential entailment for the logic of action and change. *In* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89). Morgan Kaufmann Publishers, San Francisco.

SANDEWALL, E. 1994. Features and Fluents: A Systematic Approach to the Representation of Knowledge about Dynamical Systems, Vol. 1. Oxford University Press, Oxford.

SHANAHAN, M. 1997. Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia. MIT Press, Cambridge, MA.

THIELSCHER, M. 1996a. Causality and the qualification problem. *In* Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96). *Edited by* L. C. Aiello, J. Doyle, and S. C. Shapiro. Morgan Kaufmann Publishers, San Francisco, pp. 51–62.

THIELSCHER, M. 1996b. Qualification and causality. Technical Report TR-96-026, International Computer Science Institute (ICSI), Berkeley, CA.

THIELSCHER, M. 1997. Qualified ramifications. *In* Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97). *Edited by* B. Kuipers and B. Webber. MIT Press, Cambridge, MA, pp. 466–471.

APPENDIX A: RAH NARRATIVE IN $\mathscr{L}$(ND)

For a narrative background specification for this scenario, see Section 5.1.

Persistence Statements

per1 $\forall t, thing\ [\mathtt{true} \rightarrow Per(t+1, \mathsf{loc}(thing))]$
per2 $\forall t, person, pthing\ [\mathtt{true} \rightarrow Per(t+1, \mathsf{inpocket}(person, pthing))]$
per3 $\forall t, person\ [\mathtt{true} \rightarrow Per(t+1, \mathsf{drunk}(person))]$
per4 $\forall t, plane, person\ [\mathtt{true} \rightarrow Per(t+1, \mathsf{onplane}(plane, person))]$
per5 $\forall t, person, plane\ [\mathtt{true} \rightarrow Dur(t, \mathsf{poss\_board}(person, plane), \mathsf{true})]$
per6 $\forall t, person, pthing\ [\mathtt{true} \rightarrow Dur(t, \mathsf{poss\_pickup}(person, pthing), \mathsf{true})]$
per7 $\forall t, person, loc_1, loc_2\ [\mathtt{true} \rightarrow Dur(t, \mathsf{poss\_travel}(person, loc_1, loc_2), \mathsf{true})]$
per8 $\forall t, plane, runway_1, runway_2\ [\mathtt{true} \rightarrow Dur(t, \mathsf{poss\_fly}(plane, runway_1, runway_2), \mathsf{true})]$

Observations, Action Occurrences, and Timing

obs1 [0] $\mathsf{loc}(\mathsf{boris}) \hat{=} \mathsf{home1} \wedge \mathsf{loc}(\mathsf{gun}) \hat{=} \mathsf{office} \wedge \mathsf{loc}(\mathsf{comb1}) \hat{=} \mathsf{home1} \wedge \neg \mathsf{drunk}(\mathsf{boris})$
obs2 [0] $\mathsf{loc}(\mathsf{erik}) \hat{=} \mathsf{home2} \wedge \mathsf{loc}(\mathsf{comb2}) \hat{=} \mathsf{home2} \wedge \neg \mathsf{drunk}(\mathsf{erik})$
obs3 [0] $\mathsf{loc}(\mathsf{dimiter}) \hat{=} \mathsf{home3} \wedge \mathsf{loc}(\mathsf{comb3}) \hat{=} \mathsf{home3} \wedge \mathsf{drunk}(\mathsf{dimiter})$
obs4 [0] $\mathsf{loc}(\mathsf{sas609}) \hat{=} \mathsf{run609}$
occ1 [1, 2] pickup(boris, comb2)
occ2 [1, 2] pickup(erik, comb2)
occ3 [2, 4] travel(dimiter, home3, office)
occ4 [3, 5] travel(boris, home1, office)
occ5 [4, 6] travel(erik, home2, office)
occ6 [6, 7] pickup(boris, gun)
occ7 [5, 7] travel(dimiter, office, airport)
occ8 [7, 9] travel(erik, office, airport)
occ9 [8, 10] travel(boris, office, airport)
occ10 [9, 10] board(dimiter, sas609)
occ11 [10, 11] board(boris, sas609)
occ12 [11, 12] board(erik, sas609)
occ13 [13, 16] fly(sas609, run609, run609b)

Action Types

acs1 $[t_1, t_2]\ \mathsf{fly}(plane, runway_1, runway_2) \rightsquigarrow$
$[t_1]\ \mathsf{poss\_fly}(plane, runway_1, runway_2) \wedge \mathsf{loc}(plane) \hat{=} runway_1 \rightarrow$
$I((t_1, t_2)\mathsf{loc}(plane) \hat{=} \mathsf{air}) \wedge R([t_2]\ \mathsf{loc}(plane) \hat{=} runway_2)$

acs2 $[t_1, t_2]\ \mathsf{pickup}(person, pthing) \rightsquigarrow$
$[t_1]\ \mathsf{poss\_pickup}(person, pthing) \wedge \mathsf{loc}(person) \hat{=} value(t_1, \mathsf{loc}(pthing)) \rightarrow$
$R((t_1, t_2]\ \mathsf{inpocket}(person, pthing))$

acs3 $[t_1, t_2]\ \ \mathsf{travel}(person, loc_1, loc_2) \rightsquigarrow [t_1]\ \ \mathsf{poss\_travel}(person, loc_1, loc_2) \wedge$
$\mathsf{loc}(person) \hat{=} loc_1 \rightarrow R([t_2]\ \mathsf{loc}(person) \hat{=} loc_2)$

acs4 $[t_1, t_2]\ \mathsf{board}(person, plane) \rightsquigarrow [t_1]\ \mathsf{poss\_board}(person, plane) \wedge$
$\mathsf{loc}(person) \hat{=} \mathsf{airport} \rightarrow R([t_2]\ \mathsf{loc}(person) \hat{=} value(t_2, \mathsf{loc}(plane)) \wedge$
$\mathsf{onplane}(plane, person))$

Domain Constraints

dom1 $\forall t, pthing, person_1, person_2$
$[\, person_1 \neq person_2 \wedge [t]\ \mathsf{inpocket}(person_1, pthing) \rightarrow$
$[t] \neg \mathsf{inpocket}(person_2, pthing)]$

dom2 $\forall t, person, plane_1, plane_2$
$[\, plane_1 \neq plane_2 \wedge [t]\ \mathsf{onplane}(plane_1, person) \rightarrow [t]\ \neg \mathsf{onplane}(plane_1, person)]$

dom3 $\forall t, person, pthing\ [[t]\ \mathsf{inpocket}(person, pthing) \rightarrow$
$[t]\ \mathsf{loc}(pthing) \hat{=} value(t, \mathsf{loc}(person))]$

Dependency Constraints

dep1 $\forall t, plane, person, loc\ [[t]\ \mathsf{onplane}(plane, person) \wedge C_T([t]\ \mathsf{loc}(plane) \hat{=} loc) \rightarrow$
$R([t]\ \mathsf{loc}(person) \hat{=} loc)]$

dep2 $\forall t, person, pthing, loc\ [[t]\ \mathsf{inpocket}(person, pthing) \wedge C_T([t]\ \mathsf{loc}(person) \hat{=} loc)$
$\rightarrow R([t]\ \mathsf{loc}(pthing) \hat{=} loc)]$

dep3 $\forall t, person, plane\ [[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \rightarrow$
$I([t] \neg \mathsf{poss\_board}(person, plane))]$

dep4 $\forall t, person, plane\ [[t]\ \mathsf{drunk}(person) \rightarrow X([t] \neg \mathsf{poss\_board}(person, plane))]$

Intermediate Schedule Statements

The following statements are generated from the action type specifications.

scd1 $\forall t_1, t_2, plane, runway_1, runway_2.\ [t_1, t_2]\ \mathsf{fly}(plane, runway_1, runway_2) \rightarrow$
$([t_1]\ \mathsf{poss\_fly}(plane, runway_1, runway_2) \wedge \mathsf{loc}(plane) \hat{=} runway_1 \rightarrow$
$I((t_1, t_2)\mathsf{loc}(plane) \hat{=} \mathsf{air}) \wedge R([t_2]\ \mathsf{loc}(plane) \hat{=} runway_2))$

scd2 $\forall t_1, t_2, person, pthing.\ [t_1, t_2]\ \mathsf{pickup}(person, pthing) \rightarrow$
$([t_1]\ \mathsf{poss\_pickup}(person, pthing) \wedge \mathsf{loc}(person) \hat{=} value(t_1, \mathsf{loc}(pthing)) \rightarrow$
$R((t_1, t_2]\ \mathsf{inpocket}(person, pthing)))$

scd3 $\forall t_1, t_2, person, loc_1, loc_2.\ [t_1, t_2]\ \mathsf{travel}(person, loc_1, loc_2) \rightarrow$
$([t_1]\ \mathsf{poss\_travel}(person, loc_1, loc_2) \wedge \mathsf{loc}(person) \hat{=} loc_1 \rightarrow$
$R([t_2]\ \mathsf{loc}(person) \hat{=} loc_2))$

scd4 $\forall t_1, t_2, person, plane.\ [t_1, t_2]\ \mathsf{board}(person, plane) \rightarrow$
$([t_1]\ \mathsf{poss\_board}(person, plane) \wedge \mathsf{loc}(person) \hat{=} \mathsf{airport} \rightarrow$
$R([t_2]\ \mathsf{loc}(person) \hat{=} value(t_2, \mathsf{loc}(plane)) \wedge \mathsf{onplane}(plane, person)))$

APPENDIX B: RAH NARRATIVE IN $\mathscr{L}$(FL)

Persistence Statements

per1 $\forall t, thing, v\ [\neg\ Occlude(t+1, \mathsf{loc}(thing)) \rightarrow (Holds(t, \mathsf{loc}(thing), v) \leftrightarrow Holds(t+1, \mathsf{loc}(thing), v))]$

per2 $\forall t, person, thing, v\ [\neg\ Occlude(t+1, \mathsf{inpocket}(person, thing)) \rightarrow (Holds(t, \mathsf{inpocket}(person, thing), v) \leftrightarrow Holds(t+1, \mathsf{inpocket}(person, thing), v))]$

per3 $\forall t, person, v\ [\neg\ Occlude(t+1, \mathsf{drunk}(person)) \rightarrow (Holds(t, \mathsf{drunk}(person), v) \leftrightarrow Holds(t+1, \mathsf{drunk}(person), v))]$

per4 $\forall t, plane, person, v\ [\neg\ Occlude(t+1, \mathsf{onplane}(plane, person)) \rightarrow (Holds(t, \mathsf{onplane}(plane, person), v) \leftrightarrow Holds(t+1, \mathsf{onplane}(plane, person), v))]$

per5 $\forall t, person, plane\ [\neg\ Occlude(t, \mathsf{poss\_board}(person, plane)) \rightarrow Holds(t, \mathsf{poss\_board}(person, plane), \mathsf{true})]$

per6 $\forall t, person, pthing\ [\neg\ Occlude(t, \mathsf{poss\_pickup}(person, pthing)) \rightarrow Holds(t, \mathsf{poss\_pickup}(person, pthing), \mathsf{true})]$

per7 $\forall t, person, loc_1, loc_2\ [\neg\ Occlude(t, \mathsf{poss\_travel}(person, loc_1, loc_2)) \rightarrow Holds(t, \mathsf{poss\_travel}(person, loc_1, loc_2), \mathsf{true})]$

per8 $\forall t, plane, runway_1, runway_2\ [\neg\ Occlude(t, \mathsf{poss\_fly}(plane, runway_1, runway_2)) \rightarrow Holds(t, \mathsf{poss\_fly}(plane, runway_1, runway_2))]$

Observations, Action Occurrences, and Timing

obs1 $Holds(0, \mathsf{loc}(\mathsf{boris}), \mathsf{home1}) \wedge Holds(0, \mathsf{loc}(\mathsf{gun}), \mathsf{office}) \wedge Holds(0, \mathsf{loc}(\mathsf{comb1}), \mathsf{home1}) \wedge \neg\ Holds(0, \mathsf{drunk}(\mathsf{boris}), \mathsf{true})$

obs2 $Holds(0, \mathsf{loc}(\mathsf{erik}), \mathsf{home2}) \wedge Holds(0, \mathsf{loc}(\mathsf{comb2}), \mathsf{home2}) \wedge \neg\ Holds(0, \mathsf{drunk}(\mathsf{erik}), \mathsf{true})$

obs3 $Holds(0, \mathsf{loc}(\mathsf{dimiter}), \mathsf{home3}) \wedge Holds(0, \mathsf{loc}(\mathsf{comb3}), \mathsf{home3}) \wedge Holds(0, \mathsf{drunk}(\mathsf{dimiter}), \mathsf{true})$

obs4 $Holds(0, \mathsf{loc}(\mathsf{sas609}), \mathsf{run609})$

occ1 $Occurs(1, 2, \mathsf{pickup}(\mathsf{boris}, \mathsf{comb1}))$

occ2 $Occurs(1, 2, \mathsf{pickup}(\mathsf{erik}, \mathsf{comb2}))$

occ3 $Occurs(2, 4, \mathsf{travel}(\mathsf{dimiter}, \mathsf{home3}, \mathsf{office}))$

occ4 $Occurs(3, 5, \mathsf{travel}(\mathsf{boris}, \mathsf{home1}, \mathsf{office}))$

occ5 $Occurs(4, 6, \mathsf{travel}(\mathsf{erik}, \mathsf{home2}, \mathsf{office}))$

occ6 $Occurs(6, 7, \mathsf{pickup}(\mathsf{boris}, \mathsf{gun}))$

occ7 $Occurs(5, 7, \mathsf{travel}(\mathsf{dimiter}, \mathsf{office}, \mathsf{airport}))$

occ8 $Occurs(7, 9, \mathsf{travel}(\mathsf{erik}, \mathsf{office}, \mathsf{airport}))$

occ9 $Occurs(8, 10, \mathsf{travel}(\mathsf{boris}, \mathsf{office}, \mathsf{airport}))$

occ10 $Occurs(9, 10, \mathsf{board}(\mathsf{boris}, \mathsf{sas609}))$

occ11 $Occurs(10, 11, \mathsf{board}(\mathsf{boris}, \mathsf{sas609}))$

occ12 $Occurs(11, 12, \mathsf{board}(\mathsf{erik}, \mathsf{sas609}))$

occ13 $Occurs(13, 16, \mathsf{fly}(\mathsf{sas609}, \mathsf{run609}, \mathsf{run609b}))$

Schedule Statements

scd1 $\forall t_1, t_2, plane, runway_1, runway_2\ [Occurs(t_1, t_2, \mathsf{fly}(plane, runway_1, runway_2)) \rightarrow (Holds(t_1, \mathsf{poss\_fly}(plane, runway_1, runway_2)) \wedge Holds(t_1, \mathsf{loc}(plane), runway_1) \rightarrow \forall t\ [t_1 < t \wedge t < t_2 \rightarrow Holds(t, \mathsf{loc}(plane), \mathsf{air}) \wedge Occlude(t, \mathsf{loc}(plane))] \wedge Holds(t_2, \mathsf{loc}(plane), runway_2) \wedge Occlude(t_2, \mathsf{loc}(plane)))]$

scd2 $\forall t_1, t_2, person, pthing\ [Occurs(t_1, t_2, \mathsf{pickup}(person, pthing)) \rightarrow$
$(Holds(t_1, \mathsf{poss\_pickup}(person, pthing)) \wedge$
$Holds(t_1, \mathsf{loc}(person), val(t_1, \mathsf{loc}(pthing))) \rightarrow$
$Holds(t_2, \mathsf{inpocket}(person, pthing), \mathsf{true}) \wedge$
$Occlude(t_2, \mathsf{inpocket}(person, pthing)))]$

scd3 $\forall t_1, t_2, person, loc_1, loc_2\ [Occurs(t_1, t_2, \mathsf{travel}(person, loc_1, loc_2)) \rightarrow$
$(Holds(t_1, \mathsf{poss\_travel}(person, loc_1, loc_2)) \wedge Holds(t_1, \mathsf{loc}(person), loc_1) \rightarrow$
$Holds(t_2, \mathsf{loc}(person), loc_2) \wedge Occlude(t_2, \mathsf{loc}(person)))]$

scd4 $\forall t_1, t_2, person, plane\ [Occurs(t_1, t_2, \mathsf{board}(person, plane)) \rightarrow$
$(Holds(t_1, \mathsf{poss\_board}(person, plane), \mathsf{true}) \wedge Holds(t_1, \mathsf{loc}(person), airport) \rightarrow$
$Holds(t_2, \mathsf{loc}(person), val(t_2, loc(plane))) \wedge$
$Holds(t_2, \mathsf{onplane}(plane, person), \mathsf{true}) \wedge$
$Occlude(t_2, \mathsf{loc}(person)) \wedge Occlude(t_2, \mathsf{onplane}(plane, person)))]$

Domain Constraints

dom1 $\forall t, pthing, person_1, person_2\ [person_1 \neq person_2 \wedge$
$Holds(t, \mathsf{inpocket}(person_1, pthing), \mathsf{true}) \rightarrow$
$\neg\, Holds(t, \mathsf{inpocket}(person_2, pthing), \mathsf{true})]$

dom2 $\forall t, person, plane_1, plane_2\ [plane_1 \neq plane_2 \wedge$
$Holds(t, \mathsf{onplane}(plane_1, person), \mathsf{true}) \rightarrow$
$\neg\, Holds(t, \mathsf{onplane}(plane_2, person))]$

dom3 $\forall t, person, pthing\ [Holds(t, \mathsf{inpocket}(person, pthing), \mathsf{true}) \rightarrow$
$Holds(t, \mathsf{loc}(pthing), val(t, \mathsf{loc}(person)))]$

Dependency Constraints

dep1 $\forall t, plane, person, loc\ [Holds(t, \mathsf{onplane}(plane, person), \mathsf{true}) \wedge$
$Holds(t, \mathsf{loc}(plane), loc) \wedge$
$\forall u\ [t = u + 1 \rightarrow \neg\, Holds(u, \mathsf{loc}(plane), loc)] \rightarrow Holds(t, \mathsf{loc}(person), loc) \wedge$
$Occlude(t, \mathsf{loc}(person))]$

dep2 $\forall t, person, pthing, loc\ [Holds(t, \mathsf{inpocket}(person, pthing), \mathsf{true}) \wedge$
$Holds(t, \mathsf{loc}(person), loc) \wedge$
$\forall u\ [t = u + 1 \rightarrow \neg\, Holds(u, \mathsf{loc}(person), loc)] \rightarrow Holds(t, \mathsf{loc}(pthing), loc) \wedge$
$Occlude(t, \mathsf{loc}(pthing))]$

dep3 $\forall t, person, plane\ [Holds(t, \mathsf{inpocket}(person, \mathsf{gun}), \mathsf{true}) \rightarrow$
$\neg\, Holds(t, \mathsf{poss\_board}(person, plane), \mathsf{true}) \wedge$
$Occlude(t, \mathsf{poss\_board}(person, plane))]$

dep4 $\forall t, person, plane\ [Holds(t, \mathsf{drunk}(person), \mathsf{true}) \rightarrow$
$Occlude(t, \mathsf{poss\_board}(person, plane))]$

Temporal Structure and Foundational Axioms

Apart from the narrative formulas above, we need axioms Γ_{time} for the temporal structure. The Peano axioms without multiplication. We also need the foundational axioms, Γ_{fnd}, which contain unique names axioms for the value sorts, fluent sorts, and actions. The foundational axioms also contain a set of axioms that relate the *Holds* predicate to the *val* function and ensure that each fluent has exactly one value at each

timepoint:

$$\forall t, thing, loc\ [Holds(t, \mathsf{loc}(thing), loc) \leftrightarrow val(t, \mathsf{loc}(thing)) = loc]$$
$$\forall t, person, pthing, v\ [Holds(t, \mathsf{inpocket}(person, pthing), v) \leftrightarrow val(t, \mathsf{inpocket}(person, pthing)) = v]$$
$$\forall t, person, v\ [Holds(t, \mathsf{drunk}(person), v) \leftrightarrow val(t, \mathsf{drunk}(person)) = v]$$
$$\forall t, plane, person, v\ [Holds(t, \mathsf{onplane}(plane, person), v) \leftrightarrow val(t, \mathsf{onplane}(plane, person)) = v]$$
$$\forall t, person, plane, v\ [Holds(t, \mathsf{poss\_board}(person, plane), v) \leftrightarrow val(t, \mathsf{poss\_board}(person, plane)) = v]$$
$$\forall t, person, pthing, v\ [Holds(t, \mathsf{poss\_pickup}(person, pthing), v) \leftrightarrow val(t, \mathsf{poss\_pickup}(person, pthing)) = v]$$
$$\forall t, person, loc_1, loc_2, v\ [Holds(t, \mathsf{poss\_travel}(person, loc_1, loc_2), v) \leftrightarrow val(t, \mathsf{poss\_travel}(person, loc_1, loc_2)) = v]$$
$$\forall t, plane, runway_1, runway_2, v\ [Holds(t, \mathsf{poss\_fly}(plane, runway_1, runway_2), v) \leftrightarrow val(t, \mathsf{poss\_fly}(plane, runway_1, runway_2)) = v]$$