Planning Speech Acts in a Logic of Action and Change*

Martin Magnusson and Patrick Doherty
Department of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden
{marma,patdo}@ida.liu.se

Abstract

Cooperation is a complex task that necessarily involves communication and reasoning about others' intentions and beliefs. Multi-agent communication languages aid designers of cooperating robots through standardized speech acts, sometimes including a formal semantics. But a more direct approach would be to have the robots plan both regular and communicative actions themselves. We show how two robots with heterogeneous capabilities can autonomously decide to cooperate when faced with a task that would otherwise be impossible. Request and inform speech acts are formulated in the same first-order logic of action and change as is used for regular actions. This is made possible by treating the contents of communicative actions as quoted formulas of the same language. The robot agents then use a natural deduction theorem prover to generate cooperative plans for an example scenario by reasoning directly with the axioms of the theory.

1 Introduction

Autonomous agents reason about the world to form plans and affect the world by executing those plans. Thus, agents' plans have an indirect effect on the world, and it becomes important for reasoning agents to take other agents' plans into account. Furthermore, they would do well to plan actions that affect other agent's plans and thereby (doubly indirectly) affect the world. Philosophers of linguistics have realized that we humans do this all the time through communication. In particular, Searle's *speech acts* [24] characterize natural language utterances as actions with conditions upon their execution and effects on the mental states of others.

Perrault, Allen, and Cohen [19] establish a useful connection between speech acts and planning. They formalize speech acts as planning operators in a multi-modal logic of belief and intention. Using these an agent can *inform* other agents about some fact, *request* other agents to perform some action, or ask other agents questions by requesting them to inform about some fact. They encoded simplified versions of these actions as STRIPS-like planning operators and used a backward-chaining algorithm to generate plans involving both regular actions and speech acts.

Research on software agents [8] has also adopted speech acts. This body of work depends fundamentally on agent communication languages, which are standardized sets of speech acts that ensure interoperability in agent to agent communication. The two most well known standards, KQML [5] and FIPA/ACL [6], are both based on speech act theory. FIPA/ACL also has a logical semantics defined using multi-modal BDI logic. But the semantics is meant only as a prescriptive guide when implementing software agents. Some researchers try to obtain, and sometimes even prove, conformance between the implementation and the semantics, while most programmers are probably not overly concerned with such matters. Moreover, the communication language is only a wrapper for a content language, which has to provide its own semantics. There is no integration of speech acts within a more general framework of action and change. Instead, these agent communication langauge technologies remain agnostic as to how to plan speech acts and other actions to achieve goals.

Morgenstern [16] offers an integrated theory of both types of actions using a *syntactic* first-order logic that includes quotation. Davis and Morgenstern [2] provide an alternative integration using regular first-order logic. The two theories' semantics cover both the speech acts and their content. However, while the theories were authored with the aim of applications in multi-agent planning, their use has so far been mainly of a prescriptive nature, in the implementation of a STRIPS-like planner in the case of the former theory, and as a specification for future implementations in the case of the latter theory.

In this paper we formalize inform and request speech acts in first-order logic with quotation. The representation is based on Temporal Action Logic (TAL), a first-order language with a well developed methodology for representing time, action, and change. TAL is complemented by syntactic operators that express the modalities of belief and commitment. They take quoted formulas as arguments and allow for the encoding of the effects of speech acts on other agents' beliefs and commitments. The resulting formalism

*This work is supported in part by the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII, the Swedish Research Council Linnaeus Center CADICS, and CENIIT, the Center for Industrial Information Technology.

can be used to represent and reason about both speech acts and their message content, may it be facts, actions, or other speech acts. We automate such reasoning through a natural deduction theorem prover that incorporates a form of abductive planning. The system is applied to a multi-agent planning problem involving the cooperation between two robots through planned goal delegation and knowledge acquisition, which is introduced below.

2 Cooperation and Communication

Consider a motivating scenario involving an autonomous unmanned aerial vehicle named uav1. The robot is equipped with a winch system capable of lifting and dropping supply crates. Suppose it is assigned the task of delivering crate15 to the storage building store23. It would be unwise (although perhaps spectacular) to have the robot *fly into* the building. Instead, UAVs are restricted to operate in designated fly-zones, and storage buildings are not among them.

A class of autonomous unmanned ground vehicles provide services complementary to flying robots. They too can attach crates, using fork lifts, but stick to driving short distances in and between buildings designated as drive-zones. One of the UGVs, named ugv3, happens to sit idle in the building store14.

To succeed at its task the UAV will have to request help from the ground robot to get crate15 into the building, where it can not fly itself. It knows that ground robots have the capability of delivering crates between locations in drive-zones, and it might consider delegating its task to ugv3. But crate15's current location prevents simply delegating the goal since the crate is far outside any drive-zone areas where a ground vehicle could fetch it. Instead, uav1 will have to deliver crate15 to a rendezvous point, accessible to both UAVs and UGVs. Only then is it possible to request ugv3 to see to it that the crate gets to its final destination.

Such a plan is only possible if the two robots manage to coordinate their actions through communication. We would like them to figure out the above plan, including both physical actions and communicative speech acts, completely autonomously. This will require a sufficiently expressive representation and reasoning formalism. We present our proposal next.

3 Temporal Action Logic

First-order logic might serve as a solid foundation. But it is by itself too noncommittal regarding choices of how to represent actions and their effects on time-varying properties of the world. Several alternative logics of action and change are available to aid a logicist researcher. We present work with one such logic, the Temporal Action Logic (TAL).

The origins of TAL are found in Sandewall's model-theoretic Features and Fluents framework [23]. Doherty [3] selected important concepts, such as an explicit time line and the use of occlusion (discussed below), to form TAL and gave it a proof-theoretic first-order characterization. Many extensions since have turned TAL into a very expressive language for commonsense reasoning. Doherty and Kvarnström [4] provide a detailed account of the logic, but the version presented below includes further extensions that make TAL suitable for applications in multi-agent planning and reasoning.

In TAL, properties and relations that may change over time are modeled by *fluents*. A fluent f is a function of time, and its value at a time point t is denoted (value t f). When we talk about a time interval i between two time points t_1 and t_2 we mean the interval (t_1,t_2] that is open on the left and closed on the right. The functions (start i) and (finish i) picks out t_1 and t_2 respectively. An agent carrying out an action a during time interval i is specified by the predicate (Occurs *agent* i a). But the most important feature of TAL is probably its *occlusion* concept. A persistent fluent's value is permitted to change when occluded, but must persist during time intervals when not occluded. The following formula (with free variables implicitly universally quantified and in prefix form to make the representation of quoted formulas more convenient) relates a fluent f's value at the start and end time points of a time interval i:

$$(\rightarrow (\neg \text{(Occlude } i\ f)) \\ (= \text{(value (start } i) f) \text{ (value (finish } i) f))) \quad (1)$$

By assuming that fluents are not occluded unless otherwise specified one is in effect making the frame assumption that things usually do not change. Exceptions are specified by action specifications that explicitly occlude fluents that the action affects. E.g., if uav1 flies between two locations, its location fluent (location uav1) would be occluded during any interval with a non-empty intersection with the movement interval. This prevents any use of Formula 1 for relying on the default persistence of the robot's location that conflicts with the robot's moving about. By exercising fine-grained control over occlusion one gains a flexible tool for dealing with important aspects and generalizations of the frame problem.

3.1 A Syntactic Belief Operator

Previous accounts of TAL lack a representation of agents' mental states and beliefs. Introducing a *syntactic* belief operator provides a simple and intuitive notion of beliefs. To explain this let us first assume that uav1 believes it is at loc1 at noon. The following formula[1] would represent this belief in its knowledge base:

$$(= \text{(value 12:00 (location uav1))} \text{ loc1}) \quad (2)$$

[1] Clock times such as 12:00 are not really part of the logic. We assume a translation scheme between clock times and integers.

Similarly, if it was ugv3 that believed that uav1 is at loc1, Formula 2 would be in *its* knowledge base. Beliefs about others' beliefs are then really beliefs about what formulas are present in others' knowledge bases. If uav1 believes that ugv3 believes what Formula 2 expresses, then uav1 believes that ugv3 has Formula 2 in its knowledge base. This would be represented in the knowledge base of uav1 by the following formula:

(Believes ugv3 12:00
  '(= (value 12:00 (location uav1)) loc1))

The first argument of the Believes predicate is then the agent holding the belief. The second argument is the time point at which the agent holds the belief. Finally, the third argument is a quoted version of the formula expressing the belief, in this case Formula 2. This is what makes Believes a syntactic operator.

We use the quotation notation from KIF [7], which is a formal variant of Lisp's. An expression preceded by a quote is a regular first-order term that serves as a *name* of that expression. Alternatively one may use a backquote, in which case sub-expressions can be *unquoted* by preceding them with a comma. This facilitates *quantifying-in* by exposing chosen variables inside a backquoted expression for binding by quantifiers. E.g., we use quantifying-in to represent uav1's belief that ugv3 knows its own location, without uav1 having to know the name for that location:

($\exists$ x (Believes ugv3 12:00
    '(= (value 12:00 (location ugv3)) ',x)))

Note that while x ranges over locations[2], it is the *name* of a location that should occur as part of the third argument of Believes. The quote preceding the comma ensures that whatever value x is bound to is quoted to produce the name of that value.

While a quoted formula still looks like a formula, it is in fact a term. This means that standard inference rules such as modus ponens are not applicable to the quoted formulas that appear as arguments in the Believes operator. There are two possible solutions to this limitation. Either we could add axioms that express inference rules for beliefs, or we could employ a theorem prover with special purpose inference rules for beliefs. We pursue the latter alternative in the theorem prover described in Section 5, for efficiency reasons. While it should still be possible to characterize these inference rules in terms of axioms, this is subject to future work.

3.2 Action Occurrences

An action occurs when it is *possible* for an agent to execute the action, during some time interval i, and the agent is *committed* to the action occurring, at the start of the time interval. The predicate (Possible *agent* i *action*) represents physical and knowledge preconditions for an agent carrying out an action during time interval i, while (Committed *agent* t p) represents an agent's commitment at time point t to satisfy the formula p. Both predicates require a quoted expression in their third argument position, which precludes the free use of substitution of equals without regards to the agent's knowledge. Using these predicates we can formalize the above intuition about action occurrences:

($\rightarrow$ ($\wedge$ (Possible *agent* i ``,*action*)
      (Committed *agent* (start i)
        `(Occurs ',*agent* ',i ',*action*)))
  (Occurs *agent* i *action*))

Note the interaction between backquote and quote in ``,*action* to make sure that the argument of Possible is the *name* of the action. The initial backquote turns the following quote into the name of a quote, leaving the variable *action* free for binding. The resulting expression denotes the quoted version of whatever the variable is bound to rather than a quoted variable that can not be bound at all.

3.3 Action Specifications

Each one of an agent's available actions has an action specification that consists of three parts. The first part determines under what conditions an action is possible. It may include physical preconditions, but also involves knowledge preconditions on behalf of the agent executing the action.

Consider e.g. a stock market agent that plans to get rich by buying "the stock that will increase in value". While theoretically correct, the plan is of no practical value unless the agent knows a name that identifies some particular stock that it reasonably expects will increase in value. To make this intuition formal, Moore [14] suggests that an action is only executable if the agent knows *rigid designators* for all of the action's arguments. Morgenstern [15] modifies this suggestion slightly in her requirement that *standard identifiers* are known for the action arguments.

Our action specifications follow Morgenstern and use the syntactic predicate (Identifier x) to single out a name x as a standard identifier. In the stock market example, the action of buying stock would not be executable unless the agent in fact knew the name under which the stock was listed.

The second part of an action specification lists additional requirements for any agent that decides to execute the action itself. To execute an action the agent must invoke some piece of computer code implementing it. Since our actions have an explicit time argument we think of the agent as having an execution schedule to which procedure calls can be added at specific time points. Executing an action then involves looking up standard identifiers for its arguments and scheduling the procedure call associated with the action. The effect of deciding to execute an action is that the agent becomes committed to the action occurrence. An alternative way of ensuring commitment to an action is to delegate

[2]TAL is an order sorted logic. In our implementation we indicate variable sorts by prefixes, but ignore these here for readability.

its execution to someone else through the use of the request speech act, as we will see later.

Finally, the third part of an action specification details the effects of the action on the world and on the mental states of agents. This allows agents to reason about actions and form plans to achieve goals.

4 Formalization

We are now able to formalize the agent cooperation scenario presented in Section 2 using TAL. The following unique names assumptions are needed:

$(\not= l_1\ l_2)$ $\quad l_1, l_2 \in \{$base, helipad2, store14, store23$\}$
$(\not= c_1\ c_2)$ $\quad c_1, c_2 \in \{$nil, crate15$\}$

The terms in the first set are locations and the second are crates, where nil denotes a null element of the crate sort. In addition, quoted expressions are considered equal only when they are syntactically identical.

The term names in the following set are standard identifiers that can be used as arguments to procedure calls in the robot's internal action execution mechanism. We might e.g. imagine that the procedure for flying to one of the named locations involves a simple lookup of a GPS coordinate in an internal map data structure.

(Identifier x) $\quad x \in \{$'uav1, 'ugv3, 'crate15, 'base, 'helipad2, 'store14, 'store23$\}$

Operating restrictions on UAVs and UGVs are given by fly- and drive-zones:

(FlyZone base)
(FlyZone helipad2)
(DriveZone store14)
(DriveZone store23)
(DriveZone helipad2)

The above knowledge is common to all agents in our scenario.

4.1 Physical Actions

The bulk of the robots' knowledge base is made up of the action specifications. Each of the three specification parts are given by an implication. Starting with the fly action we note that it is possible for a UAV to fly to a location in a fly-zone if the UAV knows a standard identifier for the location. Secondly, an agent may commit to flying by scheduling a fly procedure call. The constant *self* is a placeholder for the identifier of the agent in whose KB the formula appears, e.g. uav1 or ugv3 in our case. This means that, while an agent can reason about whether it is possible for another agent to fly, it can not schedule a call to the fly procedure in another agent's execution mechanism. Thirdly, at the end of the fly interval the UAV ends up at its destination.

In addition, modified fluents need to be occluded to overrule their default persistence. Flying should occlude the UAV's location fluent in any interval that intersects the flying action since (Occlude i f) means that f is occluded *somewhere* in interval i. This could be expressed as an implication (→ (Intersect i_2 i) (Occlude i_2 (location *uav*))). However, the action specification below uses the contrapositive form of this formula. The reason for this is discussed further in Section 5.

(→ (∧ (Believes *uav* (start i) '(= ',*to* ,x))
(Identifier x)
(FlyZone *to*))
(Possible *uav* i '(fly ',*to*)))
(→ (∧ (= *to* x) (Identifier '',x)
(Schedule self i (fly x)))
(Committed self (start i) '(Occurs self ',i (fly ',*to*))))
(→ (Occurs *uav* i (fly *to*))
(∧ (= (value (finish i) (location *uav*)) *to*)
(→ (¬ (Occlude i_2 (location *uav*))) (Disjoint i_2 i))))

Note that the above might sometimes require the agent to reason about its own beliefs. Suppose, for example, that uav1 is considering the possibility of flying *itself* to ugv3's location. Its knowledge base might contain the formula (= (value t_1 (location ugv3)) helipad2), expressing the belief that ugv3 is at helipad2. Then uav1 would make the belief explicit by asserting (Believes uav1 t_2 '(= (value t_1 (location ugv3)) helipad2)), where t_2 is the current time.

Ground vehicles have a very similar action that allows them to drive to locations in drive-zones:

(→ (∧ (Believes *ugv* (start i) '(= ',*to* ,x))
(Identifier x)
(DriveZone *to*))
(Possible *ugv* i '(drive ',*to*)))
(→ (∧ (= *to* x) (Identifier '',x)
(Schedule self i (drive x)))
(Committed self (start i) '(Occurs self ',i (drive ',*to*))))

(→ (Occurs *ugv* i (drive *to*))
(∧ (= (value (finish i) (location *ugv*)) *to*)
(→ (¬ (Occlude i_2 (location *ugv*))) (Disjoint i_2 i))))

Both types of agents can carry one crate at a time, and the fluent (carrying *agent*) indicates which one it is at the moment. To attach a crate the agent must not already be carrying anything, indicated by the value nil, and the agent and crate must be at the same place. The action effects occlude the crate's location (as well as the carrying fluent) since we can no longer depend on the frame assumption that it will remain in the same place.

(→ (∧ (Believes *agent* (start i) '(= ',*crate* ,x))
(Identifier x)
(= (value (start i) (carrying *agent*)) nil)
(= (value (start i) (location *agent*))
(value (start i) (location *crate*))))
(Possible *agent* i '(attach ',*crate*)))

(→ (∧ (= *crate* x) (Identifier ‘’,x)
(Schedule self i (attach x)))
(Committed self (start i)
‘(Occurs self ’,i (attach ’,*crate*))))
(→ (Occurs *agent* i (attach *crate*))
(∧ (= (value (finish i) (carrying *agent*)) *crate*)
(→ (¬ (Occlude i_2 (location *crate*)))
(Disjoint i_2 i))
(→ (¬ (Occlude i_3 (carrying *agent*)))
(Disjoint i_3 i))))

Detaching a crate has the effect that the crate ends up at the same location that the agent is currently at:

(→ (∧ (Believes *agent* (start i) ‘(= ’,*crate* ,x))
(Identifier x)
(= (value (start i) (carrying *agent*)) *crate*))
(Possible *agent* i ‘(detach ’,*crate*)))
(→ (∧ (= *crate* x) (Identifier ‘’,x)
(Schedule self i (detach x)))
(Committed self (start i)
‘(Occurs self ’,i (detach ’,*crate*))))
(→ (Occurs *agent* i (detach *crate*))
(∧ (= (value (finish i) (carrying *agent*)) nil)
(= (value (finish i) (location *crate*))
(value (finish i) (location *agent*)))
(→ (¬ (Occlude i_2 (location *crate*)))
(Disjoint i_2 i))
(→ (¬ (Occlude i_3 (carrying *agent*)))
(Disjoint i_3 i))))

4.2 Speech Acts

Speech acts can be used to communicate knowledge to, and to incur commitment in, other agents. We reformulate Allen's speech acts [1] in TAL using the syntactic belief and commitment predicates. More complex formulations have been suggested in the literature, e.g. to allow indirect speech acts [18]. But our robots will stick to straight answers and direct requests, without regard for politeness (although see Section 7 for a discussion of this).

The type of information we will be interested in is *knowing what* a particular value is. This is straight forwardly communicated by standard identifiers. E.g., if ugv3 wishes to inform uav1 that its location is store14 at noon, it may schedule an action of the following form:

(inform uav1
’(= (value 12:00 (location ugv3)) store14)) (3)

However, this is complicated when uav1 wishes to *ask* ugv3 what its location is. In accordance with much research in speech acts, we view questions as requests for information. The UAV should thus request that the UGV perform the inform action in Formula 3. Though since uav1 does not know where ugv3 is, which is presumably the reason why it is asking about it in the first place, it can not know what action to request.

Again we follow Allen's directions and introduce an informRef action designed to facilitate questions of this type. The informRef action does not mention the value that is unknown to the UAV agent, which instead performs the following request:

(request ugv3
’(Occurs ugv3 i2 (informRef uav1
(value 12:00 (location ugv3)))))

The above request still contains the unknown time interval i2, which ugv3 may instantiate in any way it chooses. The explicit time representation used by TAL opens up the possibility of a general account of the knowledge preconditions and knowledge effects of action's start and end time points, but formulating it is part of future work.

The informRef preconditions require that the informing agent *knows what* the value is, which is being informed about. The effects assert the existence of a value for which the speaker knows a standard name.

Note that an agent that commits to *executing* the action schedules an *inform* procedure call, plugging in the sought value. In contrast, an agent that only *reasons* about the effects of the informRef action, as in the question example above, knows that the value will become known, but need not yet know its name.

(→ (∧ (Believes *speaker* (start i) ‘(= ’,*value* ,x))
(Identifier x)
(Believes *speaker* (start i) ‘(= ’,*hearer* ,y))
(Identifier y))
(Possible *speaker* i ‘(informRef ’,*hearer* ’,*value*)))
(→ (∧ (= *value* x) (Identifier ‘’,x)
(= *hearer* y) (Identifier ‘’,y)
(Schedule self i (inform y ‘(= ’,*value* ’,x))))
(Committed self (start i)
‘(Occurs self ’,i (informRef ’,*hearer* ’,*value*))))
(→ (Occurs *speaker* i (informRef *hearer value*))
(∃ x (∧ (Believes *hearer* (finish i) ‘(= ’,*value* ,x))
(Identifier x))))

Many other formalizations of speech acts restrict requests to action occurrences. Our formulation of requests supports any well formed formulas, whether they are declarative goals or action occurrences. The effect is that the agent that is the target of the request is committed to satisfying the formula.

(→ (∧ (Wff *formula*)
(Believes *speaker* (start i) ‘(= ’,*hearer* ,x))
(Identifier x))
(Possible *speaker* i ‘(request ’,*hearer* ’,*formula*)))
(→ (∧ (Wff *formula*)
(= *hearer* x) (Identifier ‘’,x)
(Schedule self i (request x *formula*)))
(Committed self (start i)
‘(Occurs self ’,i (request ’,*hearer* ’,*formula*))))
(→ (Occurs *speaker* i (request *hearer formula*))
(Committed *hearer* (finish i) *formula*))

The Wff predicate determines whether the quoted expression is a well formed formula. While we could write axioms defining it, since quoted expressions are terms in our language, we find it convenient to view it as defined by semantic attachment.

Finally, to delegate declarative goals an agent must know something about the capabilities of other agents. In our scenario, UAVs know that ground robots are able to transport crates between locations in drive-zones. This allows uav1 to delegate its goal task and trust that it will indeed be satisfied.

(→ (∧ (DriveZone (value (start i) (location *crate*)))
(DriveZone *to*)
(Committed *ugv* (start i)
'(= (value (finish ',i) (location ',*crate*)) ',*to*)))
(= (value (finish i) (location *crate*)) *to*))

This concludes our formalization of the robot cooperation scenario. We turn our attention next towards the question of how to perform automated reasoning with it.

5 Automated Natural Deduction

Earlier work with TAL has made use of a model-theoretic tool for automated reasoning called VITAL [9]. But this tool relies upon the set of actions being pre-specified and consequently does not support planning. Later work made deductive planning possible through a compilation of TAL formulas into Prolog programs [10]. But Prolog's limited expressivity makes it inapplicable to interesting planning problems involving incomplete information and knowledge producing actions, such as speech acts. Instead, our current work concentrates on an implementation of a theorem prover based on *natural deduction*, inspired by similar systems by Rips [22] and Pollock [20].

Natural deduction is an interesting alternative to the widely used resolution theorem proving technique. A natural deduction prover works with the formulas of an agent's knowledge base in their "natural form" directly, rather than compiling them into clause form. The set of proof rules is extensible and easily accommodates special purpose rules that can make reasoning in specific domains or using a specific formalism like TAL more efficient. We are actively experimenting with different rule sets so the description below is of a preliminary nature.

Rules are divided into *forward* and *backward* rules. Forward rules are applied whenever possible and are designed to converge on a stable set of conclusions so as not to continue generating new inferences forever. Backward rules, in contrast, are used to search backwards from the current proof goal and thus exhibits goal direction. Combined, the result is a bi-directional search for proofs.

Nonmonotonic reasoning and planning is made possible in our theorem prover through an assumption-based argumentation system. The set of *abducibles* consists of negated occlusion, action occurrences, temporal constraints, and positive or negative holds formulas, depending on the current reasoning task [13]. These are allowed to be assumed rather than proven, as long as they are not counter-explained or inconsistent. As an example, consider the following natural deduction proof fragment, explained below (where the justifications in the right margin denote row numbers, (P)remises, (H)ypotheses, and additional background (K)nowledge).

1	(= (value 12:00 (location uav1)) base)	P
2	(∧ (= (start i37) 12:00) (= (finish i37) 13:00))	P
3	┌ (¬ (Occlude i37 (location uav1)))	H
4	(= (value 13:00 (location uav1)) base)	1-3,K
5	(= helipad2 helipad2)	K
6	(Believes uav1 (start i38) '(= helipad2 helipad2))	5
7	(Possible uav1 i38 '(fly helipad2))	6,K
8	┌ (Schedule uav1 i38 (fly helipad2))	H
9	(Committed uav1 (start i38) '(Occurs uav1 i38 (fly helipad2)))	8,K
10	(Occurs uav1 i38 (fly helipad2))	7,9,K
11	(= (value (finish i38) (location uav1)) helipad2)	10,K
12	(→ (¬ (Occlude i (location uav1))) (Disjoint i i38))	10,K
13	(Disjoint i37 i38)	3,12

The UAV is located at base at noon, as in Row 1. Suppose it needs to remain at the same location at 1 p.m. One way of proving this would be by using the persistence formula in Section 3. The location fluent is only persistent if it is not occluded. While uav1 has no knowledge about whether it is occluded or not, (¬ Occlude) is abducible and may be (tentatively) *assumed*. The effect of making non-occlusion abducible is to implement a default persistence assumption. Row 2 introduces a fresh interval constant and Row 3 indicates the assumption using a Copi style (described e.g. by Pelletier [17]) vertical line in the margin.

Suppose further that uav1 also needs to visit helipad2. The only way of proving this would be to use the fly action defined in Section 4. A backward modus ponens rule adopts (Occurs uav1 i38 (fly helipad2)) as a sub goal. Backward chaining again, on the action occurrence axiom in Section 3, causes (Possible uav1 i38 '(fly helipad2)) and (Committed uav1 (start i38) '(Occurs uav1 i38 (fly helipad2))) to become new sub goals. These are again specified by the fly action specification. The first of these sub goals is satisfied by Row 6 and the fact that helipad2 is both an identifier and a fly-zone. The commitment goal in Row 9 is satisfied by Row 5, the fact that helipad2 is a viable argument to the fly procedure, and Row 8, which assumes that uav1 schedules the procedure call. The implementation of the proof rule that adds Row 8 performs the actual scheduling by updating an internal data structure. It is still possible to backtrack, removing the assumption in Row 8, as long as the procedure call has not yet been executed, i.e. if it is scheduled to occur at some future time or if execution has not yet reached this

point. This could happen if something causes the theorem prover to reconsider flying to helipad2, or if scheduling the flight causes a conflict with some other assumption that was made previously. In such cases the procedure call would be removed from the internal data structure as well.[3] Finally, having proved the robot's ability and commitment to flying to helipad2 Row 10 concludes that the flight will occur, with the effect that uav1 ends up at helipad2 in Row 11.

Flying should occlude the location fluent in any intersecting interval. This would most naturally be expressed by ($\rightarrow$ (Intersect i i38) (Occlude i (location uav1))). But, as noted in Section 4, we use the contrapositive form instead. The reason is the need for consistency checking when assumptions have been made. It is well known that the problem of determining consistency of a first-order theory is not even semi-decidable. Our theorem prover uses its forward rules to implement an *incomplete* consistency check (more on this below), and the contrapositive form makes these forward rules applicable. Row 12, which is an effect of the fly action, together with the assumption in Row 3 trigger the forward modus ponens rule, adding the disjointness constraint in Row 13. This enforces a partial ordering of the two intervals to avoid any conflict between the persistence of the UAV's location, and its moving about. Another forward inference rule consists of a constraint solver that determines whether the set of temporal constraints is consistent. If it is impossible to order i37 and i38 so that they do not intersect in any way, then an inconsistency has been detected and the prover needs to backtrack, perhaps cancelling the most recent assumption or removing the action that was last added to the schedule.

For some restrictions on the input theory we are able to guarantee completeness of the nonmonotonic reasoning [13]. But in the general case, when one cannot guarantee completeness of the consistency checking, we might conceivably fail to discover that one of the assumptions is unreasonable. This would not be a cause of unsoundness, since we are still within the sound system of natural deduction, but it might result in plans and conclusions that rest on impossible assumptions. A conclusion Φ depending on an inconsistent assumption would in effect have the logical form $\perp\rightarrow \Phi$, and thus be tautological and void. This is to be expected though. Since consistency is not even semi-decidable, the most one can hope for is for the agent to continually evaluate the consistency of its assumptions, improving the chances of them being correct over time, while regarding conclusions as tentative. [21].

6 Generated Plans

By applying the natural deduction theorem prover to the TAL formalization we are able to automatically generate plans for the robot cooperation scenario. We present the proof goals and the resulting plans below.

Let us initially place the crate and the UAV (carrying nothing) at base at 12:00:

(= (value 12:00 (location crate15)) base)
(= (value 12:00 (location uav1)) base)
(= (value 12:00 (carrying uav1)) nil)

The goal is to have crate15 delivered to the storage named store23 at some future time point:

Show ($\exists$ t (= (value t (location crate15)) store23))

The UAV uses theorem proving to reason backwards from this goal approximately like what follows. "For the crate to be at store23 someone must have put it there. I could put it there myself if I was located at store23 carrying crate15. But I can't think of any way to satisfy the fly-zone precondition of flying to store23. Though my knowledge of ground vehicles suggests a completely different possibility. My goal would also be satisfied if both the crate's location and store23 were in drive-zones, and some ground vehicle had committed to the goal. In fact, helipad2 is a drive-zone, and it is also a fly-zone, so I can go there and drop the crate off. Before going there I should attach crate15, which is right here next to me. Then I'll decided upon some particular ground robot, say, ugv3, and request that it adopts the goal that crate15 is at store23."

While the robots are not nearly as self aware as this monologue suggests, it corresponds roughly to the search space for the following plan:

```
(Schedule uav1 i1 (attach crate15))
(Schedule uav1 i2 (fly helipad2))
(Schedule uav1 i3 (detach crate15))
(Schedule uav1 i4
  (request ugv3
    '(= (value (finish i5) (location crate15)) store23)))
(Before i1 i2)
(Before i2 i3)
(Before i3 i4)
(Before i4 i5)
```

The UAV executes its plan, including sending the goal request to ugv3. We switch to look inside the mind of the UGV as it tries to prove that the requested formula is satisfied. Suppose that half an hour has passed and that the UGV happens to be at some other storage building, carrying nothing:

(= (value 12:30 (location ugv3)) store14)
(= (value 12:30 (carrying ugv3)) nil)

The UGV will have to drive to the crate in order to pick it up and deliver it to store23. But ugv3 does not know crate15's location, and scheduling a drive to (value 12:30 (location crate15)) is prevented by the Identifier requirement on the drive action argument. The restriction is necessary since trying to find the coordinate of (value 12:30 (location crate15)) will certainly not generate any results given

[3] The link between theorem proving and action execution is an interesting topic. The mechanism described above is one approach, but we are currently investigating alternatives.

the robot's area map. The plan should instead involve finding a standard identifier for crate15's current location and looking *that* up in the map.

We assume that ugv3 believes that uav1 knows where crate15 is, and that whatever location that is, it is a drive-zone (although see Section 7 for a discussion of this):

```
(∃ x (∧ (Believes uav1 12:30
           '(= (value 12:30 (location crate15)) ,x))
         (Identifier x)))
(DriveZone (value 12:30 (location crate15)))
```

The task is then to prove the content of uav1's request:

```
Show (= (value (finish i5) (location crate15)) store23)
```

The resulting plan makes use of the request and informRef speech act combination to formulate a question corresponding to "what is crate15's location". Furthermore, while this question will equip the robot with a standard identifier, this identifier is not yet known at the time the plan is being constructed. Rather than scheduling the drive procedure call, ugv3 instead plans to *request itself* to carry out the driving after having asked uav1 about crate15's location. At the time at which this request is managed, the required information will be available for scheduling the actual drive procedure call. The rest should be a simple matter of going to store23 to drop crate15 off at its goal:

```
(Schedule ugv3 i6
  (request uav1
    '(Occurs uav1 i7
       (informRef ugv3
         (value (start i5) (location crate15))))))
(Schedule ugv3 i8
  (request ugv3
    '(Occurs ugv3 i9
       (drive (value (start i5) (location crate15))))))
(Schedule ugv3 i10 (attach crate15))
(Schedule ugv3 i11 (drive store23))
(Schedule ugv3 i12 (detach crate15))
(Before i6 i7)
(Before i7 i8)
(Before i8 i9)
(Before i9 i10)
(Before i10 i11)
(Before i11 i12)
```

Let us switch our attention back to uav1 and see what it plans to do about ugv3's request for information. The UAV's current state is described by:

```
(= (value 12:30 (location crate15)) helipad2)
(= (value 12:30 (location uav1)) helipad2)
(= (value 12:30 (carrying uav1)) nil)
```

The proof goal is defined by the incoming request:

```
Show (Occurs uav1 i7
       (informRef ugv3
         (value (start i5) (location crate15))))
```

Since uav1 has first hand knowledge about crate15's location it schedules an inform procedure call according to the definition of the informRef speech act:

```
(Schedule uav1 i7
  (inform ugv3
    '(= (value (start i5) (location crate15)) helipad2)))
```

Switching our focus back to ugv3 we find that it has received the formula that uav1 informed it about:

```
(= (value (start i5) (location crate15)) helipad2)
```

This puts ugv3 in a position where it can prove the content of its request to itself:

```
Show (Occurs ugv3 i9
       (drive (value (start i5) (location crate15))))
```

The result is that the missing drive procedure call is inserted at the right place in the execution schedule with the standard identifier plugged in as its argument:

```
(Schedule ugv3 i9 (drive helipad2))
```

Once at helipad2, the rest of the scheduled actions will have the robot attaching crate15, driving to store23, and dropping the crate off to satisfy the goal and complete the scenario.

7 Limitations and Future Work

The work presented in this paper is far from a complete solution to the robot cooperation scenario. One unsolved question regards our assumption that ugv3 believes that uav1 knows where crate15 is. Maybe there ought to be some commonsense knowledge that would allow it to defeasibly infer uav1's knowledge from the fact that it delegated a goal that directly involved that knowledge. One might suspect that this is but one instance of a more general problem of reasoning about who is likely to know what in which situations. An alternative solution would be to have the UAV reason about the fact that ugv3 needs to know where the crate is to be able to move it to its destination. The UAV could then pro-actively inform the UGV about the crate's location before requesting the UGV to move it.

An ad hoc move that we were forced to make was to remove the informRef speech act from the UAVs knowledge base while generating the first plan. While this particular action is not needed for that particular plan, the UAV clearly ought to have access to all its actions at all times. The reason for our move has to do with the fact that uav1 must attempt to solve the goal itself before considering delegating it. What makes our scenario interesting is that it is not possible to solve without cooperation. But uav1 can not know that trying to deliver crate15 by itself is futile until it has explored all alternative ways of doing so. Unfortunately, the informRef speech act made for a rather unwieldy search space, which was more than our theorem prover had time to explore while we cared to wait. This prevented

uav1 from giving up on the prospect of managing the delivery by itself within a reasonable amount of time. We suspect that as the agents are equipped with more knowledge and actions, more possibilities will open up in the theorem prover's search space, and the need for some kind of heuristic to help guide search will increase.

The speech acts themselves are subject to some limitations. One is our disregard of any physical preconditions to communication such as geographical closeness constraints. Our robots are assumed to have a radio link at all times. Another limitation is that we do not consider indirect speech acts. This seems reasonable as long as we are thinking of communication between our robots. But there is no denying that many of the speech acts uttered by *humans* are indirect. A human UAV operator uttering "Could you make sure that crate15 is in store23?" is presumably requesting the UAV to make sure the goal is satisfied rather than querying about its ability to do so. Another serious limitation is our assumption that other agents always accept requests. Some rejected requests could reasonably be handled during plan execution through re-planning or plan repair. But others should be considered already during planning and would result in conditionally branching plans or plans with loops that repeat requests until accepted.

A future development could be the inclusion of composite actions, which would make it possible to explicitly represent informRef as a macro action that includes an inform speech act. This is in contrast to our current formalization where inform is only a procedure call and not a stand alone action. Another possibility for development exists with regards to the execution schedule mechanism. While we think that it is a promising method for integrating planning and execution, the description of its workings that we have provided here is rather sketchy and needs further elaboration. In particular we would like to take advantage of our integrated temporal constraint solver to calculate action durations and schedule actions at explicit clock times.

Finally, an agent architecture based exclusively on logical reasoning raises efficiency concerns. Both plans in our running example were automatically generated by the theorem prover in 2 minutes and 35 seconds on a Pentium M 1.8 GHz laptop with 512 MB of RAM. That might or might not be reasonable, depending on the application. But, in either case, it was admittedly a small problem, which begs the question of whether the architecture will scale up to real-world problems. Alas, we do not yet know. But there are at least some reasons to be optimistic.

One reason is, as already mentioned, the use of a temporal constraint solver for reasoning with time. More generally, one can view special purpose algorithms as additional natural deduction rules that make certain types of inferences efficient. Another reason is the choice of an interruptible algorithm for nonmonotonic reasoning. In a real-time setting the agent can act, at any time, to the best of its knowledge given the reasoning it has performed up to that point.

But most encouragingly, achieving satisfactory performance in certain domains is already possible. E.g., our theorem prover was applied to UAV surveillance and quickly generated plans for realistic size problems [11]. Furthermore, the agent architecture was used to the control the characters in a computer game that requires real-time interaction [12]. We believe computer games to be a particularly suitable domain for empirical studies of logical agents on the road from tiny benchmark problems towards larger real-world applications.

8 Conclusions

We have described a scenario involving communication and cooperation between two robots. The solution required one robot to plan to delegate a goal through communication using a request speech act. The other robot had to plan to achieve knowledge preconditions, again through communication using a nested request and informRef speech act. These speech acts were formalized in an extension of Temporal Action Logic that includes syntactic belief and commitment operators, which were made possible through the use of a quotation mechanism. The formalization made it possible to generate a plan involving both cooperation and communication using automated theorem proving. Finally, a novel scheduling mechanism provided a tightly coupled integration between planning and the execution of generated plans.

The formalization used quotation, which seems most befitting of a logicist framework. The robots' explicit representation of beliefs as formulas in a knowledge base motivates their representation of others' beliefs as quoted formulas. Further benefits may be gained by using quotation in the context of speech acts. A fuller theory of communication will presumably also include locutionary acts, i.e. the actual utterances that encode messages between agents. These are most naturally thought of as strings consisting of quoted formulas from the agents' knowledge bases.

Our philosophy is based on the principle that logic is an intelligent agent's "language of thought". The formalization of the speech acts are similar to their corresponding semantics proposed in the literature. But unlike many other approaches that view the semantics as normative, such as agent communication languages, we put the formulas in our agents' heads where the agents can reason with them using theorem proving. In fact, our use of a prefix notation for formulas makes the correspondence between the theory in this paper and its Lisp implementation *exact*, save for some logical symbols that are not available for use as Lisp identifiers. Through this approach we hope to construct an agent architecture based on logical planning with a level of flexibility that would be difficult to match using agent programming languages.

References

[1] James Allen. *Natural Language Understanding*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1988.

[2] Ernest Davis and Leora Morgenstern. A first-order theory of communication and multi-agent plans. *Journal of Logic and Computation*, 15(5):701–749, 2005.

[3] Patrick Doherty. Reasoning about action and change using occlusion. In *Proceedings of the Eleventh European Conference on Artificial Intelligence ECAI'94*, pages 401–405, 1994.

[4] Patrick Doherty and Jonas Kvarnström. Temporal action logics. In Vladimir Lifschitz, Frank van Harmelen, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007.

[5] Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald McKay, James McGuire, Richard Pelavin, Stuart Shapiro, and Chris Beck. Specification of the KQML agent-communication language. Technical Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA, July 1993.

[6] Foundation for Intelligent Physical Agents. FIPA communicative act library specification. `http://www.fipa.org/specs/fipa00037/`, 2002.

[7] Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, June 1992.

[8] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[9] Jonas Kvarnström. VITAL: Visualization and implementation of temporal action logics. `http://www.ida.liu.se/~jonkv/vital/`, 2007.

[10] Martin Magnusson. *Deductive Planning and Composite Actions in Temporal Action Logic*. Licentiate thesis, Linköping University, September 2007. `http://www.martinmagnusson.com/publications/magnusson-2007-lic.pdf`.

[11] Martin Magnusson and Patrick Doherty. Deductive planning with inductive loops. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 528–534. AAAI Press, 2008.

[12] Martin Magnusson and Patrick Doherty. Logical agents for language and action. In Christian Darken and Michael Mateas, editors, *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference AIIDE-08*. AAAI Press, 2008.

[13] Martin Magnusson, Jonas Kvarnström, and Patrick Doherty. Abductive reasoning with filtered circumscription. In *Proceedings of the 8th Workshop on Nonmonotonic Reasoning, Action and Change NRAC 2009*. UTSePress, 2009. Forthcoming.

[14] Robert Moore. Reasoning about knowledge and action. Technical Report 191, AI Center, SRI International, Menlo Park, CA, October 1980.

[15] Leora Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 867–874, 1987.

[16] Leora Morgenstern. *Foundations of a logic of knowledge, action, and communication*. PhD thesis, New York, NY, USA, 1988. Advisor: Ernest Davis.

[17] Francis Jeffry Pelletier. A brief history of natural deduction. *History and Philosophy of Logic*, 20:1–31, 1999.

[18] C. Raymond Perrault and James F. Allen. A plan-based analysis of indirect speech acts. *Computational Linguistics*, 6(3-4):167–182, 1980.

[19] C. Raymond Perrault, James F. Allen, and Philip R. Cohen. Speech acts as a basis for understanding dialogue coherence. In *Proceedings of the 1978 workshop on Theoretical issues in natural language processing*, pages 125–132, Morristown, NJ, USA, 1978. Association for Computational Linguistics.

[20] John Pollock. Natural deduction. Technical report, Department of Philosophy, University of Arizona, 1999. `http://www.sambabike.org/ftp/OSCAR-web-page/PAPERS/Natural-Deduction.pdf`.

[21] John L. Pollock. *Cognitive Carpentry: A Blueprint for how to Build a Person*. MIT Press, Cambridge, MA, USA, 1995.

[22] Lance J. Rips. *The psychology of proof: deductive reasoning in human thinking*. MIT Press, Cambridge, MA, USA, 1994.

[23] Erik Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems*, volume 1. Oxford University Press, 1994.

[24] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.