Encoding Definitional Fragments of Temporal Action Logic into Logic Programming

Marc van Zee[1], Patrick Doherty[2], and John-Jules Meyer[3]

[1] Department of Individual and Collective Reasoning,
University of Luxembourg
[2] Department of Computer and Information Science,
Linköping University
[3] Department of Information and Computing Science,
Utrecht University

Abstract. Temporal Action Logics (TAL) is an expressive class of nonmonotonic temporal logics for reasoning about action and change. In previous work, it has been shown that a very general fragment of the logic can be reduced to first-order logic with equality. Consequently, standard theorem proving techniques can be used to reason in TAL. TAL is intended to be used for robotics. In this case, standard theorem proving techniques are too general and do not provide efficient decision procedures. The goal of this article is to identify a limited subset of TAL that can be directly mapped to a normal logic program. Although quite restrictive, this sets the lower bound on what can be done with direct mappings to logic programs. Discussions concerning extensions to the restricted fragment are also provided.

1 Introduction

Temporal Action Logics (TAL) [2, 4] is a general class of nonmonotonic temporal logics for reasoning about action and change that are based on the Features and Fluents framework of Sandewall [14]. TAL is highly expressive and includes the use of context-dependent durative actions, durational fluents (fluents with default values), ramification constraints, qualification constraints, concurrent and non-deterministic actions. Like several other action formalisms, TAL uses circumscription [11, 12] to solve the frame and the ramification problems, which require nonmonotonicity. In [2], it was shown how the 2nd-order circumscription component could be reduced to a logically equivalent 1st-order component using quantifier elimination. Consequently, standard theorem proving techniques can be used to reason with TAL. This is problematic from an efficiency perspective since in the case of many domains such as robotics, one requires an efficient decision procedure to answer queries. VITAL[4] was an early implementation of a fragment of TAL that uses model generation techniques. This does not scale for large TAL narratives though.

[4] http://www.ida.liu.se/∼jonkv/vital

The generality of TAL is useful in other domains as a natural semantic specification language. For instance, it provides a formal specification of TALplanner. TALplanner[5] [3, 7] is an award-winning forward-chaining planner based on TAL. Both TAL planner and an execution monitoring framework based on TAL have been used in Unmanned Aerial Vehicles [5], developed at Linköping University.[6]

In this paper, the focus is on isolating a restricted fragment of TAL and showing that it can be encoded in a sound manner into normal logic programs. Unfortunately, the fragment for which this works is highly restrictive. The subset of TAL narratives with deterministic single step actions and complete information about initial state can be encoded soundly into logic programs. This in fact makes sense, since any non-determinism in TAL narratives must be excluded from such a direct mapping. Relaxing the single step constraint and complete information at initial state both result in introducing non-determinism and thus multiple minimal models. This, in fact, would apply to any alternative action formalism.

The systematic use of logic programming as a basis for soundly implementing fragments of action formalisms was introduced in the case of the Situation Calculus by Reiter (see [13], Chapter 5). The initial fragment identified by Reiter had similar restrictions for progression to work. This article applies the techniques introduced by Reiter to encode a fragment of TAL soundly into logic programs and analyzes the feasibility of extending the technique to more general fragments of TAL.

An algorithm is developed for the systematic translation of *definitional theories* of TAL into logic programs. The definitional fragment of TAL identified is called TALM. It is then shown that extending this fragment via the use of direct encodings into normal logic programs is not feasible. Any attempt to generalize the TALM fragment results in the introduction of non-determinism into the fragment. The main limitation here is the inability to encode a unique values axiom for features in TAL narratives. This immediately rules out non-deterministic actions, actions with duration and an incomplete initial state.

The rest of this paper is organised as follows: In Section 2 we introduce the basic concepts of TAL, in Section 3 we develop an algorithm to translate definitional theories into logic programs, in Section 4 we reformulate a constrained TAL theory, TALM, as a definitional theory, and finally in Section 5 we discuss the limitations of the translation.

2 Temporal Action Logics

Due to lack of space, we refer the reader to [4] for details about the full syntax and semantics of TAL. In this section, we summarize the main components necessary for understanding the techniques introduced.

In TAL, a scenario (or narrative) can be described in a compact surface language $\mathcal{L}$(ND), which is a high-level macro expandable language consisting of action type specifications (**acs**), dependency constraints (**dep**), domain constraints (**dom**), persistence statements (**per**), observation statements (**obs**), and action occurrence statements

[5] http://www.ida.liu.se/divisions/aiics/aiicssite/projects/talplanner.en.shtml
[6] http://www.ida.liu.se/divisions/aiics/aiicssite/index.en.shtml

(**occ**). To make sure that fluents will not be persistent when they are changed by an action, the reassignment macro R can be used. $R([t_2]\alpha)$ ensures that α will hold at the time point t_2. Consider the following scenario due to Reiter [13] that will be used as a running example.

Example 1 (Simple robot specification). There is one robot called rob and there are two objects, namely a ball and a vase. A robot can only pick up an object if he is not holding anything and if he is next to the object, and he can only drop an object if he is holding it. When a robot drops an object it will fall on the floor. Initially, rob is next to vase and not next to ball. Rob is not holding anything. First, rob picks up vase, after which he walks to ball and drops vase. The narrative specification of this example in the macro language $\mathcal{L}$(ND) is:

acs1 $[t_1, t_2]$ pickup$(r, o) \rightsquigarrow$
$[t_1] \forall_{o_1}[\neg$holding$(r, o_1)] \wedge$ nextto$(r, o) \rightarrow$
$R([t_2]$ holding$(r, o) \wedge \neg$onfloor$(o))$

acs2 $[t_1, t_2]$ walk$(r, o) \rightsquigarrow$
$R([t_2]$ nextto$(r, o)) \wedge \forall_{o_1}[o \neq o_1 \rightarrow R([t_2]\ \neg$nextto$(r, o_1))]$

acs3 $[t_1, t_2]$ drop$(r, o) \rightsquigarrow$
$[t_1]$ holding$(r, o) \rightarrow R([t_2]\ \neg$holding$(r, o) \wedge$ onfloor$(o))$

obs1 $[0]$ nextto(**rob**, **vase**) $\wedge \neg$nextto(**rob**, **ball**)

obs2 $[0] \forall_z \neg$holding(**rob**, z)

obs3 $[0]$ onfloor(**vase**) $\wedge$ onfloor(**ball**)

occ1 $[0, 1]$ pickup(**rob**, **vase**)

occ2 $[1, 2]$ walk(**rob**, **ball**)

occ3 $[3, 4]$ drop(**rob**, **vase**)

Reasoning about a narrative in $\mathcal{L}$(ND) is done by translating it into the base language $\mathcal{L}$(FL)), which is an order-sorted classical first-order language using a linear, discrete time structure. The language uses the ternary predicates $Holds$ and $Occurs$, and the binary predicate $Occlude$. The translation from $\mathcal{L}$(ND) to $\mathcal{L}$(FL) is given by the function $Trans$ [4]. For instance, $Trans([t]f \hat{=} \omega)$ is defined as $Holds(t, f, \omega)$. Similarly, $Trans(R((t, t']f \hat{=} \omega))$ is defined as $\forall t''(t < t'' < t' \rightarrow Occlude(t'', f)) \wedge Holds(t', f, \omega)$. $Occlude(t, f)$ represents that a persistent or durational fluent f is exempt from inertia or default value assumption, respectively, at time t. $Trans([t, t']\Psi)$, where Ψ is an action term, is defined as $Occurs(t, t', \Psi)$, which represents that action Ψ occurs in the interval $[t, t']$.

Consider any narrative $\mathcal{N}$ and let $\mathcal{N}_{per}, \mathcal{N}_{obs}, \mathcal{N}_{occ}, \mathcal{N}_{acs}, \mathcal{N}_{domc}$, and $\mathcal{N}_{depc}$ denote the sets of persistence statements, observation statements, action occurrence statements, action type specifications, domain constraints, and dependency constraints in $\mathcal{N}$ respectively. The TAL domain description (referred to as preferred narrative) $\Delta_{\mathcal{N}}$ is given by

$$CIRC[\Gamma_{occ}; Occurs] \wedge CIRC[\Gamma_{depc} \wedge \Gamma_{acs}; Occlude] \wedge \Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc},$$

where $\Gamma_{per}, \Gamma_{obs}, \Gamma_{occ}, \Gamma_{acs}, \Gamma_{domc}$, and Γ_{depc} are the formulas in $\mathcal{L}$(FL) (first-order logic formulas) obtained by applying $Trans$ on $\mathcal{N}, \mathcal{N}_{obs}, \mathcal{N}_{occ}, \mathcal{N}_{acs}, \mathcal{N}_{domc}$, and $\mathcal{N}_{depc}$

respectively; Γ_{fnd} is the set of foundational axioms in $\mathcal{L}$(FL), containing unique name axioms, unique value axioms, etc., and Γ_{time} is the axiomatization of the particular temporal structure used in TAL.

By proposition 18.2 in [4], the circumscription of $Occurs$ and $Occludes$, respectively $CIRC[\Gamma_{occ}; Occurs]$ and $CIRC[\Gamma_{depc} \wedge \Gamma_{acs}; Occlude]$, can be transformed into first-order definitional forms either through quantifier elimination of predicate completion techniques.

Example 2 (Robot Specification, Ctd.). The following is a representation of **acs1** in the language $\mathcal{L}$(FL):

$$\begin{aligned}\textbf{acs1'}\quad & Occurs(t_1, t_2, \text{pickup}(r, o)) \rightarrow \\ & \forall_{o_1}[\neg Holds(t_1, \text{holding}(r, o_1), true)] \wedge Holds(t_1, \text{nextto}(r, o), true) \rightarrow \\ & \quad Occlude(t_2, \text{holding}(r, o)) \wedge Occlude(t_2, \text{onfloor}(o)) \wedge \\ & \quad Holds(t_2, \text{holding}(r, o), true) \wedge \neg Holds(t_2, \text{onfloor}(o), true).\end{aligned}$$

The circumscription of the $Occurs$ predicate in the action occurrences (**occ1**, **occ2**, and **occ2**) above is equivalent to the following first-order formula:

$$\begin{aligned}Occurs(t_1, t_2, a) \leftrightarrow & (t_1 = 0 \wedge t_2 = 1 \wedge a = \text{pickup}(\textbf{rob}, \textbf{vase})) \vee \\ & (t_1 = 1 \wedge t_2 = 2 \wedge a = \text{walk}(\textbf{rob}, \textbf{ball})) \vee \\ & (t_1 = 2 \wedge t_2 = 3 \wedge a = \text{drop}(\textbf{rob}, \textbf{vase}))\end{aligned}$$

The circumscription of the $Occlude$ predicate in the action specifications (**acs1**, **acs2**, and **acs3**) above is equivalent to the following set of first-order formulas:

$$\begin{aligned}& Occlude(t, holding(\textbf{rob}, o)) \leftrightarrow \\ & \quad o = \textbf{vase} \wedge \forall_{o_1}[\neg Holds(0, \text{holding}(r, o_1), true)] \wedge \\ & \quad Holds(0, \text{nextto}(r, o), true) \wedge 1 \leq t \leq 2 \\ & Occlude(t, nextto(\textbf{rob}, o)) \leftrightarrow o = \textbf{ball} \wedge t = 2 \\ & Occlude(t, onfloor(o)) \leftrightarrow \\ & \quad o = \textbf{vase} \wedge (t = 1 \wedge \forall_{o_1}[\neg Holds(0, \text{holding}(rob, o_1), true)] \wedge \\ & \quad Holds(0, \text{nextto}(\textbf{rob}, o), true) \vee t = 3 \wedge Holds(2, \text{holding}(\textbf{rob}, o), true)\end{aligned}$$

3 def2P Algorithm: From Definitional Theories to Logic Programs

We begin by providing some background definitions and techniques for the sake of completeness.

3.1 Negation As Failure Semantics: Clark's Completion

The most basic logic programming syntax uses *Horn clauses*, a fragment of first-order logc. A *Horn clause theory* is a set of sentences of the form $A \vee \neg B_1 \vee ... \vee \neg B_n$, which is usually written as $A \leftarrow B_1, ..., B_n$, and also called *Horn rules*. In this, A is the *head* of the Horn clause, and $B_1, ..., B_n$ is the *body*. If the head is of the form $R(t_1, ..., t_n)$, the

Horn clause is said to be *about* the relation symbol R. The Horn clause $\leftarrow B_1, ..., B_n$ is identified with $\mathsf{false} \leftarrow B_1, ..., B_n$ and $A \leftarrow$ is identified with $A \leftarrow \mathsf{true}$, also called a *Horn fact* [6].

Horn clause logic programming generally does not use classical negation, but *negation as failure*. This means exactly what it says: if some fact cannot be derived from the theory (i.e. it *fails*), then we assume that the negation of this fact can be derived from the theory (i.e. it *succeeds*). One of the most widely applicable and often used semantics for negation as failure was given by Clark [1]. This is usually called the (Clark) completion, $comp(P)$, of the original program P. The idea of this result is that we use the 'implied iff': we simply replace all the implications of the Horn clauses with equivalences. The basic result of Clark is that negation as failure is sound for $comp(P)$ for both success and failure. Because of the syntactical form of Horn clauses, $comp(P)$ will be a definitional theory.

Definition 1 (Definitions and Definitional Theories). *A first-order theory is a* definition *iff it has the syntactic form* $(\forall x_1, ..., x_n)(P(x_1, ..., x_n) \equiv \phi)$*, where* P *is an* n*-ary predicate symbol other than equality, and* ϕ *is a first-order formula with free variables among* $x_1, ..., x_n$*. A set of axioms is* definitional *iff its axioms consist of one definition for each predicate symbol, except for equality. The* if-half *of the above definition of* P *is the sentence* $\forall x_1, ..., x_n(P(x_1, ..., x_n) \supset \phi)$.

Theorem 1 (Clark's Theorem [13]). *Suppose* T *is a set of definitions for all predicate symbols except for equality in some first-order language with finitely many predicate symbols, together with the following equality axioms:*

1. *For every pair of distinct function symbols* f *and* g *of the language (including constant symbols):* $f(\overline{x}) \neq g(\overline{y})$.
2. *For every n-ary function symbol of the language:* $f(x_1, ..., x_n) = f(y_1, ..., y_n) \supset x_1 = y_1 \wedge ... \wedge x_n = y_n$.
3. *For every term* $t[x]$ *(other than* x *itself) that mentions the variable* $x : t[x] \neq x$.

Suppose that P *is a Prolog program obtained from the definitions of* T *by writing the if-halves of all the definitions of* T *as Prolog clauses. Suppose further that* G *is a query goal then,*

1. *Whenever a proper Prolog interpreter succeeds on the goal* G *with answer substitution* θ*, then* $T \models (\forall)G\theta$*. Here,* $(\forall)G\theta$ *denotes the result of universally quantifying all the free variables (if any) of* $G\theta$.
2. *Whenever a proper Prolog interpreter returns failure on the goal* G*, then* $T \models (\forall)\neg G$ *where the quantification is over all free variables mentioned in* G.

The three conditions of this theorem all have to do with the fact that when completing the theory, equality is introduced. This means that it is necessary to provide axioms for these. The first two conditions are simply the *unique names axioms* for the function symbols: they define that two function terms are equal if and only if both their function names and arguments are equal. The last condition is a more technical details that has to do with the "occurs" check of Prolog's unification algorithm, which we will not go into here[7]. We will exlain the concept of a "proper Prolog interpreter" in Section 3.3.

[7] See [13], Chapter 5 for more details.

3.2 Lloyd-Topor Transformations

Although Clark's theorem assumes that a Prolog program P can be obtained from the theory T by writing the if-halves, it does not give any details on how this is done. Because Prolog only uses disjunctions and conjunctions as connectives, we will translate the other connectives (implication, bi-implication and quantifiers) into a form such that it can be parsed by a Prolog interpreter. The same can be said for the query goals.

The Lloyd-Topor transformations [10, 9] is a set of derivation rules for systematically transforming if-halves of definitions of the syntactic form $W \rightarrow A$ into a syntactic form suitable for implementation as Prolog clauses. Here, A must be an atomic formula, but W may be an arbitrary first-order formula, possibly involving quantifiers, in which case we require that the quantified variables of W be different from one another, and from any of the free variables mentioned in W. Originally, the Lloyd-Topor transformations introduce auxiliary predicates when transforming negated existential quantifiers and disjunctions, but Reiter [13] shows that these predicates are only introduced for exposition of the results of Lloyd-Topor and that they can be omitted using a process called *unfolding*. The output of these revised transformations is *a single Prolog executable* formula $lt(W) \rightarrow A$, without introducing new predicates and clauses. Here, $lt(W)$ is a formula Reiter defines inductively on the syntactic structure of W. It is defined as follows:

1. If W is a literal: $lt(W) = W$.
2. $lt(W_1 \wedge W_2) = lt(W_1) \wedge lt(W_2)$.
3. $lt(W_1 \vee W_2) = lt(W_1) \vee lt(W_2)$.
4. $lt(W_1 \rightarrow W_2) = lt(\neg W_1 \vee W_2)$.
5. $lt(W_1 \equiv W_2) = lt((W_1 \rightarrow W_2) \wedge (W_2 \rightarrow W_1))$.
6. $lt((\forall_x)W) = lt(\neg(\exists_x)\neg W)$.
7. $lt((\exists_x)W) = lt(W)$.
8. $lt(\neg\neg W) = lt(W)$.
9. $lt(\neg(W_1 \wedge W_2)) = lt(\neg W_1) \wedge lt(\neg W_2)$.
10. $lt(\neg(W_1 \vee W_2)) = lt(\neg W_1) \vee lt(\neg W_2)$.
11. $lt(\neg(W_1 \rightarrow W_2)) = lt(\neg(\neg W_1 \vee W_2))$.
12. $lt(\neg(W_1 \equiv W_2)) = lt(\neg(W_1 \rightarrow W_2) \wedge \neg(W_2 \rightarrow W_1)))$.
13. $lt(\neg(\forall_x)W) = lt((\exists_x)\neg W)$.
14. $lt(\neg(\exists_x)W) = \neg lt(W)$.

3.3 Allowed Programs

Clark's theorem is not applicable to any Prolog interpreter, but only to *proper* Prolog interpreters. Such an interpreter is one that evaluates a negative literal `not A`, using negation as failure, and moreover, does so *only when (at the time of evaluation) the atom* `A` *is ground*. When `A` is not ground, the interpreter may suspend its evaluation, working on other literals until `A` does become ground, or it may abort its computation. Either way, it never tries to fail on non-ground atoms, because this can result in unsound behaviour [13]. Due to space limitations, we leave out the details[8], but we instead introduce a well-known class of programs that is known to be complete, meaning that it will not cause problems with floundering [15], namely the *allowed programs*. In Section 4 we show that every TALM theory results in an allowed Prolog program.

Definition 2 (Allowed Program, Allowed Query). *A query is said to be* allowed *if every variable which occurs in it occurs in a positive literal of it; a program clause* $A \leftarrow L_1, ..., L_n$ *is allowed if every variable which occurs in it occurs in a positive literal of its body* $L_1, ..., L_n$ *and a program is* allowed *if all of its clauses are allowed.*

[8] For a more detailed discussion see [16], Section 3.2.2.

3.4 the def2P algorithm

The contents of the previous section can be summarized into a single algorithm that translates a definitional theory to a Prolog program such that the program is sound for the theory.

Definition 3 (def2P algorithm). *Suppose $\mathcal{T}$ is a definitional theory . The* def2P *algorithm translates $\mathcal{T}$ to a Prolog program P such that the Prolog resolution algorithm for P is complete for the theory $\mathcal{T}$, and consists of the following steps:*

1. *$\mathcal{T}$ is augmented with Clark's Equality Axioms (see Def. 1), obtaining $\mathcal{T}'$,*
2. *$\mathcal{T}'$ is translated into Lloyd-Topor Normal Form, obtaining $\mathcal{T}'_{norm}$,*
3. *The if-halves of the definitions of $\mathcal{T}'_{norm}$ form the Prolog program P,*
4. *If P falls in the class of* allowed *programs, return P. Else, return* false.

4 A Definitional Theory for TAL

We constrain TAL to integer and positive time, relational and inertial fluents, complete initial state and deterministic, single-step and non-overlapping actions. Moreover, we omit symbolic constants, dependency constraints and domain constraints. Finally we assume a consistent narrative specification. Call this constrained formalism TALM.

Definition 4 (TALM theory). *A* TALM theory *is a constrained TAL theory $\Gamma = \Gamma_{obs} \wedge \Gamma_{occ} \wedge \Gamma_{acs} \wedge \Gamma_{per} \wedge \Gamma_{fnd}$ of the form:*

Γ_{obs}	$Holds(t, f, v)$	*for $t \in \mathbb{N}$, fluent f and value $v \in \{$*true*,* false*$\}$*
Γ_{occ}	$Occurs(t, t+1, a)$	*for $t \in \mathbb{N}$ and action a*
Γ_{acs}	$Occurs(t, t+1, a) \rightarrow \Phi(t, t+1)$	
Γ_{per}	$\neg Occlude(t+1, f) \rightarrow Holds(t+1, f, v) \equiv Holds(t, f, v)$	
Γ_{fnd}	*UNA, CWA, Unique values axioms*	

We obtain a definitional theory for this restricted narrative by providing a definition for the three predicates in the base language $\mathcal{L}$(FL): $Occurs$, $Occlude$ and $Holds$.

Occurs: By definition, only positive occurrences of $Occurs$ predicates are allowed in Γ_{occ}. Each such atomic formula can be put in the logically equivalent form $\forall_{t_1,t_2,a}(t_1 = u_t \wedge t_2 = u'_t \wedge a' = a(\overline{u})) \rightarrow Occurs(t_1, t_2, a')$. Denote such a formula by $\forall_{t_1,t_2,a'} \Psi_i \rightarrow Occurs(t_1, t_2, a')$ where $\Psi_i = (t_1 = u \wedge t2 = u'_t \wedge a' = a(\overline{u}))$. Then the conjunction of ground atomic formulas can be put in the following form: $\forall_{t_1,t_2,a'}(\Psi_1 \vee \Psi_2 \vee ... \vee \Psi_n) \rightarrow Occurs(t_1, t_2, a')$. Denote this formula by $\forall_{t_1,t_2,a} \Upsilon \rightarrow Occurs(t_1, t_2, a)$. By proposition 18.2 in [4], in this case circumscription is equivalent to predicate completion, i.e. $CIRC(\Gamma_{occ}; Occurs)$ is equivalent to:

$$\forall_{t_1,t_2,a} \Upsilon \equiv Occurs(t_1, t_2, a). \tag{1}$$

Occlude: Circumscribing $Occlude$ works in a similar way. The predicate $Occlude$ occurs only in the postcondition of dependency constraints and actions specification formulas. Each member of Γ_{acs} and Γ_{dep} can be transformed syntactically into the logically equivalent form $\forall_{t,f} \Gamma_i(t, f) \rightarrow Occlude(t, f)$. Again, by proposition 18.2 in [4],

$CIRC(\Gamma_{acs} \wedge \Gamma_{acs}; Occlude)$ is equivalent to:

$$\forall_{t,f}[\bigvee_{i=1}^{k} \Gamma_i(t,f)] \equiv Occludes(t,f) \tag{2}$$

Holds: We obtain the definition of the $Holds$ predicate using case distinctions. The following proposition simplifies the TALM narrative.

Proposition 1 (Redundant observations at $t > 0$). *Observations that occur at $t > 0$ in TALM can be inferred from actions and can thus be removed from the narrative.*

Proof. Let Γ be some narrative specification in TALM. Let Γ' be the narrative Γ with all observations at $t > 0$ removed. We have to show that Γ and Γ' have the same models, i.e. $\Gamma \Leftrightarrow \Gamma'$. The truth value of $Occurs$ and $Occlude$ is equivalent for both narratives, because these predicates do not occur in the observations. This means that it suffices to show that $\Gamma \models Holds(t,f,v) \Leftrightarrow \Gamma' \models Holds(t,f,v)$ for any time point t, fluent f and valuation v.

"$\Rightarrow$": Suppose for some time point t, fluent f and valuation $v \in \{\mathsf{true}, \mathsf{false}\}$ we have that $\Gamma \models Holds(t,f,v)$. We have to show that $\Gamma' \models Holds(t,f,v)$. Suppose that $Holds(t,f,v)$ is not an observation, it follows now directly that $\Gamma' \models Holds(t,f,v)$, because the only difference between Γ and Γ' are observations. Suppose to the contrary that $Holds(t,f,v)$ is an observation. Now, if there is no action specification that implies $Holds(t,f,v)$, then this observation follows from the persistence statement as well, otherwise it will be contradicting with it and the narrative is inconsistent. Therefore, the observation is redundant. On the other hand, if there is an action specification formula that implies $Holds(t,f,v)$, then since the action specification formulas are unchanged in Γ', it follows that $\Gamma' \models Holds(t,f,v)$.

"$\Leftarrow$": Follows directly from the fact that Γ' is a subset of Γ, meaning that everything that is valid in Γ' will be valid in Γ. □

Now, to obtain a definition for the $Holds$ predicate, suppose some time point t, fluent f and value v.

- Suppose $t = 0$. The only formulas that can assign a value to the $Holds$ predicate at $t = 0$ are observations, because at time point 0 no actions can occur since an action has a minimal duration of 1 and a minimal starting time of 0. Moreover, all fluents are assigned a value through the observations because TALM has a complete initial state. So, given that $t = 0$,

$$\forall_{f,v}\left[\bigvee_{i=1}^{n} f = P_i(\overline{u}_i) \wedge v = v_i\right] \equiv Holds(t,f,v). \tag{3}$$

- Suppose $t > 0$. We introduce a second case distinction on $Occlude$:
  - Suppose $\neg Occlude(t,f)$. Using the persistence statement (see Definition (4)) we obtain $\forall_v Holds(t,f,v) \equiv Holds(t-1,f,v)$. So, given that $t > 0$ and $\neg Occlude(t,f)$,

$$Holds(t-1,f,v) \equiv Holds(t,f,v) \tag{4}$$

- Suppose $Occlude(t, f)$. By Eq. (2) we obtain $\Gamma_i(t, f)$. Because actions are deterministic and non-overlapping, and because there will be no observations (Proposition 1) there will be a unique action specification formula that is now true and implies a single $Holds$ statement for the fluent f. Using this we obtain $Holds(t, f, v)$. This means that, given that $t > 0$ and $Occlude(t, f)$,

$$(\Gamma_i(t, f) \wedge v = v_i) \equiv Holds(t, f, v) \tag{5}$$

We now state the completion theorem of the $Holds$ predicate:

Theorem 2 (Completion of Holds). *Formulas (3), (4) and (5) provide necessary and sufficient conditions for the predicate $Holds$:*

$$\begin{aligned} t = 0 \wedge \left[\bigvee_{i=1}^{k} f = P_j(\overline{u}_i) \wedge v = v_i\right] \vee \\ t > 0 \wedge (\neg Occlude(t, f) \wedge Holds(t-1, f, v) \vee \\ Occlude(t, f) \wedge \Gamma_i(t, f) \wedge v = v_i) \\ \equiv Holds(t, f, v) \end{aligned} \tag{6}$$

Proof. Formulas (3), (4) and (5) are the only formulas that make the $Holds$ predicate true in a TALM narrative. We obtain the definition directly by putting these formulas in a disjunction and adding the conditions for each case. □

Fortunately, it turns out that every definitional theory of a TALM narrative fall into the class of allowed programs of Definition 2. We show this in the following theorem.

Theorem 3 (Allowed program). *Suppose Γ is a TALM narrative , which is translated into a definitional theory by using the equivalences above. Let P be the program that is obtained from this theory by applying Clark's Theorem. P falls into the class of* allowed programs*. That is, every variable that occurs in a rule in P occurs in a positive literal of the body of the rule.*

Proof. We have to show that each program clause in P is an allowed program clause. Each clause corresponds to a predicate of the theory, which means that we will have to consider $Occurs$, $Occlude$, and $Holds$. By definition, all literals occurring in the definition of $Occurs$ are positive [4]. This is the same for the definition of $Occlude$ predicate, because the only literals that occur in this definition are $Holds$ literals, and each negated $Holds$ predicate can be translated into an equivalent positive one, using the equivalence $Holds(t, f, \mathsf{true}) \equiv \neg Holds(t, f, \mathsf{false})$. Finally, there occurs one negation in the definition of the $Holds$ predicate, which is in the scope of the $Occlude$ predicate (see Theorem 2):

$$\ldots t > 0 \wedge (\neg Occlude(t, f) \wedge Holds(t-1, f, v) \vee \ldots) \equiv Holds(t, f, v)$$

The variables occurring in the negative literal are t and f, which both occur positively in the $Holds$ predicate that directly follows it. Therefore, $Holds$ is an allowed clause too, because all variables occurring in a negative literal occur in a positive literal in the body. □

We will now demonstrate the translation of our running example. We invite the reader to download the application that was developed along with this paper called TALTranslator. This application can perform the transformation automatically and comes with a user-friendly GUI and several examples[9].

Example 3 (Robot specification, continued). The definition of the $Holds$ predicate for the fluent onfloor is (slightly simplified for readability):

$$
\begin{aligned}
&Holds(t, onfloor(o), v) \leftrightarrow \\
&\quad t = 0 \wedge v = \mathsf{true} \vee \\
&\quad t > 0 \wedge \\
&\qquad (o = \textbf{vase} \wedge (t = 1 \wedge \forall_{o_1}[Holds(0, \text{holding}(\textbf{rob}, o_1), false)] \wedge \\
&\qquad Holds(0, \text{nextto}(\textbf{rob}, o), true) \wedge v = \mathsf{false} \vee \\
&\qquad t = 3 \wedge Holds(2, \text{holding}(\textbf{rob}, o), true) \wedge v = \mathsf{true}) \vee \\
&\quad \neg Occlude(t, \text{onfloor}(o)) \wedge Holds(t-1, \text{onfloor}(o), v))
\end{aligned}
$$

Next, we input this definitional theory into the def2P algorithm (Def. 3). Since Prolog provides the equality axioms (step 1), we can directly apply the Lloyd-Topor transformations. What follows is the Prolog code for the fluent onfloor(o) (note that Prolog uses "`;`" for disjunction):

```
holds(T,onfloor(O),V) :-
  T=0, V=true ;
  T>0, ( O=vase, T=1, holds(0, holding(rob,O1),false),
    holds(0,nextto(rob,O),true), v=false;
    t=3, holds(2, holding(rob,O), true), v=true);
    not occlude(T, onfloor(O)), T2 = T-1, holds(T2, onfloor(O),V) ).
```

5 Relaxing the Constraints

In the previous sections we have introduced a restricted fragment of TAL, which we referred to as TALM. We then showed how it can be translated into a logic program sound for the narrative in question.

In this section we will consider if, and to what extent the constraints on TALM can be relaxed while still being able to use direct mappings of TAL narratives into logic programs. We discuss non-deterministic actions, concurrent actions, actions with duration and an incomplete initial state. Almost all the restrictions that we will discuss (except for concurrent actions) have one property in common: relaxing each of them will result in a non-deterministic narrative. This means essentially that a narrative can have multiple interpretations.

The unique values axioms associated with the foundational axioms in TAL for any narrative are crucial when discussing non-deterministic narratives, because it rules out narratives in which a fluent has two values at the same time point. This implies that rather than modelling several alternative values for a fluent in one mode, several minimal models are required instead. The unique values axioms follow:

$$\forall_{t,f} \exists_v Holds(t, f, v) \tag{7}$$

$$\forall_{t,f,v_1,v_2}[v1 \neq v2 \supset \neg(Holds(t, f, v_1) \wedge Holds(t, f, v_2))] \tag{8}$$

[9] Download from http://icr.uni.lu/marc/TALTranslator.rar

In the previous section we did not explicitly encode these axioms into the definitional theory, because each narrative in TALM is fully deterministic: it will have a unique model in which each fluent has exactly one value. Satisfaction of the axioms is implicit in the encoding of TALM narratives into a logic program.

Theorem 4 (Unique Model). *Each TALM narrative Γ has a unique model m in which each fluent has exactly one value per time point.*

Proof. We show that for each narrative Γ in TALM, the interpretations of the predicates $Holds$, $Occlude$ and $Occurs$ are unique. Because these are the only predicates occurring in the narrative, it follows directly that the narrative has a unique model. Suppose some narrative Γ,

- For $Occurs$: The interpretation of $Occurs$ is determined only by the action occurrences. This interpretation is unique because the assignments of the action occurrences are non-deterministic and non-overlapping.
- For $Occlude$: Similar to $Occurs$.
- For $Holds$: This follows directly from the case distinction used when constructing the definition of the $Holds$ predicate (see Section 2).

□

5.1 Extending TALM with Non-determinism

Any relaxation of the restrictions on TAL narratives associated with TALM introduce one form or another of non-deterministic choices of fluent values at time points. This implies that in the general case, Theorem 4 no longer holds. For any relaxation of TALM to be sound relative to encoding into a logic program, the unique values axioms would have to become an explicit part of the the definitional theory. Unfortunately, this is not possible.

Observe that Equation 8 is equivalent to,

$$v_1 \neq v_2 \rightarrow \neg Holds(t, f, v_1) \vee \neg Holds(t, f, v_2),$$

It is in general not possible to bring such a formula into a definitional form. To obtain a definition of a predicate P, we require a set of formulas of the form,

$$(\Phi_1 \rightarrow P) \wedge ... \wedge (\Phi_i \rightarrow P),$$

which can then be combined to $(\Phi_1 \vee ... \vee \Phi_i) \rightarrow P$.

Unfortunately the unique values axiom does not provide us with such a construction, and we can also not transform it in a direct manner into one. This means that we cannot hope to express the unique values axiom and at the same time maintain a transformation to an equivalent definitional theory. Therefore, we generally cannot allow non-deterministic actions in TALM, because it will possibly lead to an inconsistent model in which a fluent has two values at one time point. This immediately rules out the possibility to extend TALM with any form of non-determinism, including non-deterministic actions, multi-step actions, and an incomplete initial state, because each of these extensions introduce a form of non-determinism into the theory. This is not surprising as it applies to most any approach to modelling action and change.

5.2 Concurrent actions

Much work in reasoning about action and change has been done under the assumption that there is a single agent performing sequences of non-overlapping actions. The use of explicit time points in TAL enables the direct specification of narratives where action execution intervals are partly or completely overlapping, whether those actions are performed by a single agent or by multiple agents. No special concurrency operators are required to do this.

Independent Concurrent Actions: Extending TALM with independent concurrent actions involving disjoint sets of features is unproblematic. Since the effects of the actions do not interfere with each other (i.e. two actions change the same fluent at the same time), the expected effects will take place. Such concurrent actions will not introduce multiple models. We omit the proof of this proposition because it is trivial. The consequence of this is that the unique value axioms can be omitted, which means that it is possible to extend TALM with independent concurrent actions.

Dependent Concurrent Actions: In the case where actions affecting the same fluents occur concurrently, concurrent actions that affect the value of the same fluent can never assign different values to this fluent, because this would lead to an inconsistent narrative. Otherwise, both actions assign the same value to the fluent. In either case, it is not necessary to add the unique values axioms to the theory because there are no multiple models introduced. Again, the proof of this is trivial and is omitted. Thus, concurrent actions are allowed as long they do not assign different values to the same fluent at the same time. Or in other words: concurrent actions that assign different values to the same fluent are not allowed.

6 Conclusion

The aim of this paper was to isolate a definitional fragment of the highly expressive TAL formalism and show how it can be soundly encoded into a normal logic program. This was done by defining the fragment TALM and providing an algorithm, def2P, that encodes TAL narratives in this fragment into logic programs that can be shown to be sound and complete relative to this particular fragment. One would of course like to find more general reasoning techniques that relax some of the restrictions associated with TALM narratives. At the very least, one would require a mapping to disjunctive logic programs such as Answer Set Programs. Lee and Palla [8] have recently demonstrated that it is possible to reformulate more general fragments of TAL into an answer set program. Consequently the SAT-based implementation techniques used there can be applied to TAL. One of the authors is currently looking into the translation of TAL theories to Satisfiability Module Theory (SMT) programs. Another appropriate extension would include the use of constraints to more efficiently model the temporal constraints associated with more general TAL narratives.

7 Acknowledgments

Marc van Zee is funded by the National Research Fund, Luxembourg.

References

1. Clark, K.L.: Negation as failure. In: Logic and Databases. pp. 293–322 (1977)
2. Doherty, P.: Reasoning about action and change using occlusion. In: 11th European Conference on Artificial Intelligence,1994. John Wiley and Sons (1994)
3. Doherty, P., Kvarnström, J.: Talplanner : A temporal logic based forward chaining planner. Annals of Mathematics and Artificial Intelligence 30(1-4), 119–169 (2000)
4. Doherty, P., Kvarnström, J.: Temporal action logics. In: Handbook of Knowledge Representation, pp. 709–757. No. 3 in Foundations of Artificial Intelligence, Elsevier (2009)
5. Doherty, P., Kvarnström, J., Heintz, F.: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. Autonomous Agents and Multi-Agent Systems 19(3), 332–377 (Dec 2009)
6. Hodges, W.: Logical features of horn clauses. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.) Handbook of logic in artificial intelligence and logic programming (vol. 1), pp. 449–503. Oxford University Press, Inc., New York, NY, USA (1993)
7. Kvarnström, J.: TALplanner and other extensions to Temporal Action Logic. Ph.D. thesis, Linkping UniversityLinkping University, Department of Computer and Information Science, The Institute of Technology (2005)
8. Lee, J., Palla, R.: Reformulating temporal action logics in answer set programming. In: AAAI 2012 (2012)
9. Lloyd, J.W.: Foundations of logic programming. Springer-Verlag New York, Inc. (1987)
10. Lloyd, J., Topor, R.: Making prolog more expressive. Journal of Logic Programming 1, 225–240 (1984)
11. Mccarthy, J.: Circumscription—a form of non-monotonic reasoning. Readings in nonmonotonic reasoning pp. 145–152 (1987)
12. McCarthy, J.: Applications of circumscription to formalizing common sense knowledge. Artificial Intelligence 59, 23–26 (1986)
13. Reiter, R.: Knowledge in action: logical foundations for specifying and implementing dynamical systems. MIT Press, Cambridge, Mass. (2001)
14. Sandewall, E.: Features and fluents (vol. 1): The representation of knowledge about dynamical systems. Oxford University Press, Inc., New York, NY, USA (1994)
15. Shepherdson, J.: Negation as failure, completion and stratification. In: in Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 5. Oxford Science Publications (1998)
16. van Zee, M.: Implementing Temporal Action Logics. Master's thesis, Utrecht University (2013)