TALplanner: An Empirical Investigation of a Temporal Logic-based Forward Chaining Planner

Patrick Doherty
Dept of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden
patdo@ida.liu.se

Jonas Kvarnström
Dept of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden
jonkv@ida.liu.se

Abstract

We present a new forward chaining planner, TALplanner, based on ideas developed by Bacchus [5] and Kabanza [11], where domain-dependent search control knowledge represented as temporal formulas is used to effectively control forward chaining. Instead of using a linear modal tense logic as with Bacchus and Kabanza, we use TAL, a narrative-based linear temporal logic used for reasoning about action and change in incompletely specified dynamic environments. Two versions of TALplanner are considered, TALplan/modal which is based on the use of emulated modal formulas and a progression algorithm, and TALplan/non-modal which uses neither modal formulas nor a progression algorithm. For both versions of TALplanner and for all tested domains, TALplanner is shown to be considerably faster and requires less memory. The TAL versions also permit the representation of durative actions with internal state.

1. Introduction

Recently, Bacchus and Kabanza et al. [3, 4, 5, 11] have been investigating the use of temporal logics to express search control knowledge for planning. In the approach, they utilize domain-specific control knowledge represented as formulas in a first-order modal tense logic to effectively control a forward-chaining planner. The approach is implemented in the TLplan system. Empirical results for a number of test domains have demonstrated impressive improvements in planning efficiency when compared to many recent planners such as BLACKBOX [14] and IPP [15] (See [5] for comparisons). BLACKBOX and IPP were the best performers in the AIPS'98 planning competition [1].

In the past few years, we have been developing a family of temporal logics for reasoning about action and change in incompletely specified dynamic worlds. These logics are collected under the acronym TAL (Temporal Action Logics) [10, 6, 7, 8, 13, 12]. More recently, we have begun to apply these logics in the WITAS UAV project[1], one aspect of which is the design, specification and implementation of a hybrid deliberative/reactive system architecture for the command and control of an autonomous unmanned aerial vehicle (UAV). A fundamental requirement of the architecture is the ability to generate plans in a timely and/or any-time manner and to be able to dynamically update control knowledge as observations are made during plan generation.

In order to satisfy this requirement, we have begun experimentation with a forward chaining planner in the spirit of TLplan which we call TALplanner. Instead of using a modal tense logic to represent domain-specific control knowledge, we use TAL, which is a first-order logic with explicit time which is used to reason about narratives. In fact, a narrative may be viewed as a plan, and TALplanner generates narratives given goals and control knowledge specified in TAL. Modal formulas can be emulated using macros in a surface language. TAL provides a solid basis for experimentation with forward-chaining planners because of its rich expressivity. Actions with duration, context-dependent actions, non-deterministic actions, delayed effects of actions [13], concurrent actions [12], incompletely specified timing of actions, and side-effects of actions [10] can all be expressed. In addition, robust solutions to the frame [6], ramification [10] and qualification [8] problems exist when reasoning in restricted domains in incompletely specified environments.

As a first step in exploring the design space of planners for our UAV project, we have implemented TALplanner and compared its performance with TLplan. We have used the test domains provided with TLplan (also used

[1] Wallenberg Information Technology and Autonomous Systems Laboratory (WITAS).

in AIPS'98 [1] planning competition) in our comparisons. TALplanner's performance is shown to be markedly better than TLplan both in terms of time and memory. While TLplan is implemented in C, TALplanner is implemented in Java, an interpreted language. Although TALplan/modal is similar to TLplan in many respects, the implementation techniques of both TALplan/modal and TALplan/non-modal are quite different. In addition, both of the TAL-based planners permit the representation of durative actions with internal state. The latter requires modification of the original TLplan algorithm.

The main contribution in this paper is the integration of a well-developed and highly expressive temporal logic for reasoning about action and change with a well-developed and highly promising logical approach to a forward-chaining plan synthesis technique. The intent of this paper is to verify the adequacy of the technique and provide a basis for continued development of the approach in the context of TAL. We demonstrate this by extending the algorithm for durative actions with internal state.

In the remainder of the paper, we will briefly describe TAL and the representation of temporal narratives, the planning task and its representation in TAL, an example, the use of modal and non-modal control formulas, and finally, the empirical test results and comparison with TLplan.

2. TAL: Temporal Action Logics

The basic approach we use for reasoning about action and change is as follows. First, represent a temporal narrative description as a set of narrative statements in a surface language $\mathcal{L}$(ND), which is a high-level language for representing observations, action types (plan operators), action occurrences (instantiations of plan operators), dependency constraints, domain constraints, and timing constraints related to actions and their durations. Second, translate the narrative statements represented in $\mathcal{L}$(ND) into the base language $\mathcal{L}$(FL), which is an order-sorted first-order language with a linear, discrete time structure.[2] Third, minimize potential spurious change via a circumscription axiom which is easily reducible to a first-order formula via syntactic transformation. The intent of the minimization is to deal with the well known frame, qualification and ramification problems. Finally, use the resulting logical theory to reason about the narrative.

Narrative statements in $\mathcal{L}$(ND) are always labeled in a narrative description $\mathcal{N}$ and a set of syntactic restrictions are normally associated with each labeled class of statements. In the following we denote the narrative statements in $\mathcal{N}$ of statement class: action type, observation, dependency constraint, domain constraint, action occurrence, and schedule, as Υ_{acs}, Υ_{obs}, Υ_{depc}, Υ_{domc}, Υ_{occ} and Υ_{scd} with labels acs, obs, dep, dom, occ and scd, respectively. The sets of formulas in $\mathcal{L}$(FL) corresponding to the 2nd-6th narrative statement classes in $\mathcal{N}$ are denoted by Γ_{obs}, Γ_{depc}, Γ_{domc}, Γ_{occ} and Γ_{scd}, respectively. In the translation process *Trans*() from $\mathcal{L}$(ND) to $\mathcal{L}$(FL), an action type in Υ_{acs} corresponding to an action occurrence in Υ_{occ} is instantiated, resulting in a schedule statement in Γ_{scd}.

Given a narrative description Υ in $\mathcal{L}$(ND), the corresponding theory Δ in $\mathcal{L}$(FL) is $\Gamma \wedge \Gamma_{fnd}$, where $\Gamma = \Gamma_{obs} \wedge \Gamma_{occ} \wedge \Gamma_{domc} \wedge \Gamma_{depc} \wedge \Gamma_{scd}$, and Γ_{fnd} are the foundational axioms associated with TAL narratives and contain unique names axioms, temporal structure axioms, etc. Let Γ' be the result of applying a circumscription policy to $\Gamma_{depc} \wedge \Gamma_{scd}$ in Γ to filter spurious change from the theory Δ and $\Delta' = \Gamma' \wedge \Gamma_{fnd}$, then any formula α is entailed by the narrative Υ iff $\Delta' \models \alpha$. See Kvarnström and Doherty [16] for a WWW accessible implementation of TAL, and Doherty et al. [7] for a detailed overview of TAL.

3. The Planning Task

For the planning task, the basic idea is to generate a temporal narrative description $\mathcal{N}_p$ in $\mathcal{L}$(ND), given a goal narrative description $\mathcal{GN}$ in an extended surface language $\mathcal{L}(\text{ND})^*$ as input to the planning algorithm. The goal narrative description contains an intended goal statement Υ_{igoal}, a set of goal domain constraint statements Υ_{gdom} and a set of goal control statements Υ_{gctrl}, in addition to action types, observations, domain and dependency constraints found in standard narrative descriptions. The goal control and goal domain constraint statements represent the domain dependent control knowledge that will be used in the planning algorithm to achieve the intended goal. If the planning algorithm is successful in achieving the goal then the output $\mathcal{N}_p$ of the planner is $(\mathcal{GN} \setminus (\Upsilon_{igoal} \cup \Upsilon_{gctrl} \cup \Upsilon_{gdom})) \cup \Upsilon_{occ}$, where Υ_{occ} is the sequence of action occurrences with timing (planning steps) generated by the planning algorithm. The planning algorithm is sound in the following sense. If the algorithm generates a narrative description $\mathcal{N}_p$ as output given $\mathcal{GN}$ as input then $\Delta'_{\mathcal{N}_p} \models \mathit{Trans}([\mathbf{t}]\Upsilon_{igoal})$, where $\Delta'_{\mathcal{N}_p}$ is the circumscribed logical theory in $\mathcal{L}$(FL) corresponding to $\mathcal{N}_p$, and $[\mathbf{t}]$ is the end timepoint of the last action occurrence in Υ_{occ}. Completeness of the planning algorithm is a more difficult issue and dependent on the nature of the control knowledge used for each domain. Observe that $\Upsilon_{igoal} \cup \Upsilon_{gctrl} \cup \Upsilon_{gdom}$ is only used in the plan synthesis algorithm.

3.1. Extensions to $\mathcal{L}$(ND)

An *atemporal narrative formula* is defined inductively using the set of atemporal atomic expressions and isvalue

[2] In the remainder of the paper, we will use *Trans*() to denote the translation procedure.

expressions in $\mathcal{L}(\text{ND})$ as the base class and using the standard logical connectives, quantifiers and delimiters in the normal manner (see [7]). A *goal expression* has the form $goal(\psi)$, where ψ is any atemporal narrative formula. A *goal domain constraint* in $\mathcal{L}(\text{ND})^*_{\texttt{Goal}}$ has the same syntax as an $\mathcal{L}(\text{ND})$ domain constraint, but may also contain goal expressions. A *goal control formula* in $\mathcal{L}(\text{ND})^*_{\texttt{Control}}$ may contain goal expressions and temporal modal operators applied to atemporal narrative formulas. No other temporal expressions are allowed in the formula.

$\mathcal{L}(\text{ND})$ is extended to $\mathcal{L}(\text{ND})^*$ by adding three new labeled classes of narrative statements used only in goal narratives: intended goal statements (Υ_{igoal}, an atemporal narrative formula, labeled igoal), goal domain constraint statements ($\Upsilon_{gdom} \subseteq \mathcal{L}(\text{ND})^*_{\texttt{Goal}}$, labeled gdom), and goal control statements ($\Upsilon_{gctrl} \subseteq \mathcal{L}(\text{ND})^*_{\texttt{Control}}$, labeled gctrl).

Assume a goal narrative $\mathcal{GN}$ with intended goal $\Upsilon_{igoal} = \{\phi\}$. When recursively evaluating statements in $\Upsilon_{gctrl} \cup \Upsilon_{gdom}$ in the plan synthesis algorithm, the meaning of a goal expression $goal(\psi)$ in such statements is determined by checking whether $\phi \models \psi$ (whether ψ is true in all models (goal states) satisfying ϕ). In the plan synthesis algorithm, it is sometimes necessary to check whether a formula in $\mathcal{L}(FL)$ with possible goal expressions entails another formula in $\mathcal{L}(FL)$ with possible goal expressions. In this case, the formulas are evaluated in the same way as in $\mathcal{L}(FL)$ after evaluating the goal expressions in the manner above and replacing them with *true* or *false*. We use the following notation to describe this evaluation process $Trans^*(\phi) \models^* Trans^*(\psi)$.

3.2. Experimental Methodology

TAL is an expressive logic for reasoning about action and change and has a formal declarative semantics. Consequently, it provides an ideal basis for experimentation with plan synthesis based on the use of declarative domain-dependent control knowledge and for formally verifying the correctness of generated plans. The efficiency of the plan synthesis algorithm is primarily dependent on how efficiently one can check whether a narrative formula is entailed by a partially developed narrative or whether a goal expression is entailed by the intended goal in a narrative, which in turn depends on how well one can reuse previous work done when checking other partially developed narratives. Tradeoffs between efficiency of the plan synthesis algorithm and expressivity of the plan representation language can be studied by placing or relaxing syntactic constraints on the different classes of narrative statements and using analysis tools from from descriptive complexity or applying various model checking techniques, for instance.

As already stated, TAL, as a formalism for reasoning about action and change, is highly expressive. Action types can be context dependent or non-deterministic, may have variable duration, and explicit point-based temporal constraints may be used. This permits the representation of many types of plan operators such as those used in STRIPS and ADL. In addition, side-effects represented using dependency and domain constraints, and real concurrency are expressible. Different degrees of incompleteness in the domain specification such as incomplete initial state or observations in states other than the initial state are also expressible. All these properties have a corresponding formal semantics.

Obviously, the expressivity associated with TAL does not automatically induce efficient implementations of plan synthesis algorithms. On the contrary, one must restrict TAL's expressiveness in various ways. For the purposes of this paper, we begin at the lower rungs of the ladder of expressivity.

In the current implementation of TALplanner, the following restrictions apply:

- The set of values a fluent can have in $\mathcal{L}(\text{FL})$ must be finite.
- The initial state of a goal narrative must be completely specified.
- Action types are deterministic, but may be context-dependent.
- No domain or dependency constraints are permitted in narratives.

In addition, we allow:

- Durative actions with internal state changes are permitted.
- Disjunctive goals are allowed.

Note that the TALplanner algorithms currently permit the use of restricted classes of domain and dependency constraints at the risk of generating inconsistent narratives. This could be avoided by introducing a consistency check in the *GoodPlan* algorithm in Section 5.1. Due to the computational complexity this implies, we leave this topic for future research.

3.3. A Goal Narrative Example

In the remainder of this paper, we will use the blocks domain described in Bacchus [5] and used as a test domain in [1].

In TAL, $[t]\ \alpha$ means that α holds at t. Action type specifications of the form $[t_1, t_2]\mathsf{action}(\overline{x}) \rightsquigarrow \bigwedge_{i=1}^{n}([t_1]\alpha_i \rightarrow R([t_2]\beta_i)) \wedge \phi_i$ define what happens when action is invoked with the arguments $\overline{x}$ between times t_1 and t_2: For each i, if α_i holds at t_1, then β_i must hold at t_2. If $1 < i$ then the

action is context dependent and each α_i is a specific precondition (context). ϕ_i denotes a context dependent constraint on the action's duration. For example, for single-step actions, all ϕ_i have the form $t_2 = t_1 + 1$. We define *Preconds*$([t_1, t_2]\mathsf{action}(\overline{x})) = \{\alpha_i \mid 1 \leq i \leq n\}$.

The following *goal narrative* $\mathcal{GN}$ in $\mathcal{L}(\mathrm{ND})^*$ is essentially what is provided as input to the planner.[3] This particular example is quite small, containing only six blocks, `AA–FF`. Other examples tested contain up to 1000 blocks. It contains a specification of action types (acs), observations about the initial state (obs), goal domain constraints (gdom), an intended goal statement (igoal), and goal control statements about the blocks domain (gctrl). Recall that $\mathcal{L}(\mathrm{ND})^*$ is a high-level language designed to support the definition of complex narratives. Many of the operators in the narrative, such as the modal operators, are simply syntactic sugar (macros) and are reduced in the transformation to $\mathcal{L}(\mathrm{FL})$.

acs1 $[t_1, t_2]$ $\mathsf{putdown}(b) \rightsquigarrow [t_1]$ $\mathsf{holding}(b) \rightarrow$
$R([t_2]\ \neg\mathsf{holding}(b) \wedge \mathsf{ontable}(b) \wedge \mathsf{clear}(b) \wedge$
$\mathsf{handempty}) \wedge t_2 = t_1 + 1$

acs2 $[t_1, t_2]$ $\mathsf{pickup}(b) \rightsquigarrow$
$[t_1]$ $\mathsf{ontable}(b) \wedge \mathsf{clear}(b) \wedge \mathsf{handempty} \rightarrow$
$R([t_2]\ \mathsf{holding}(b) \wedge \neg\mathsf{ontable}(b) \wedge \neg\mathsf{clear}(b) \wedge$
$\neg\mathsf{handempty}) \wedge t_2 = t_1 + 1$

acs3 $[t_1, t_2]$ $\mathsf{stack}(b_1, b_2) \rightsquigarrow [t_1]$ $\mathsf{holding}(b_1) \wedge \mathsf{clear}(b_2) \rightarrow$
$R([t_2]\ \neg\mathsf{holding}(b_1) \wedge \neg\mathsf{clear}(b_2) \wedge \mathsf{on}(b_1, b_2) \wedge$
$\mathsf{clear}(b_1) \wedge \mathsf{handempty}) \wedge t_2 = t_1 + 1$

acs4 $[t_1, t_2]$ $\mathsf{unstack}(b_1, b_2) \rightsquigarrow$
$[t_1]$ $\mathsf{on}(b_1, b_2) \wedge \mathsf{clear}(b_1) \wedge \mathsf{handempty} \rightarrow$
$R([t_2]\ \mathsf{holding}(b_1) \wedge \mathsf{clear}(b_2) \wedge \neg\mathsf{on}(b_1, b_2) \wedge$
$\neg\mathsf{clear}(b_1) \wedge \neg\mathsf{handempty}) \wedge t_2 = t_1 + 1$

obs1 $\forall b_1, b_2[[0]\ \mathsf{on}(b_1, b_2) \leftrightarrow b_1 = \mathsf{BB} \wedge b_2 = \mathsf{AA} \vee$
$b_1 = \mathsf{AA} \wedge b_2 = \mathsf{EE}]$

obs2 $\forall b[[0]\ \mathsf{ontable}(b) \leftrightarrow b = \mathsf{EE} \vee b = \mathsf{CC} \vee b = \mathsf{DD} \vee b = \mathsf{FF}]$

obs3 $[0]\ \mathsf{handempty} \wedge \forall b[\neg\mathsf{holding}(b)]$

obs4 $\forall b[[0]\ \mathsf{clear}(b) \leftrightarrow b = \mathsf{BB} \vee b = \mathsf{CC} \vee b = \mathsf{DD} \vee b = \mathsf{FF}]$

igoal1 $\mathsf{ontable}(\mathsf{AA}) \wedge \mathsf{ontable}(\mathsf{BB}) \wedge \mathsf{on}(\mathsf{FF}, \mathsf{EE})$

gdom1 $\forall t, b[[t]\ \mathsf{goodtower}(b) \leftrightarrow$
$[t]\ \mathsf{clear}(b) \wedge \neg\mathsf{goal}(\mathsf{holding}(b)) \wedge [t]\ \mathsf{goodtowerbelow}(b)]$

gdom2 $\forall t, b[[t]\ \mathsf{goodtowerbelow}(b) \leftrightarrow [t]\ \mathsf{ontable}(b) \wedge$
$\neg\exists b_2[\mathsf{goal}(\mathsf{on}(b, b_2))] \vee \exists b_2[[t]\ \mathsf{on}(b, b_2) \wedge$
$\neg\mathsf{goal}(\mathsf{ontable}(b)) \wedge \neg\mathsf{goal}(\mathsf{holding}(b_2)) \wedge$
$\neg\mathsf{goal}(\mathsf{clear}(b_2)) \wedge \forall b_3[\mathsf{goal}(\mathsf{on}(b, b_3)) \rightarrow b_3 = b_2] \wedge$
$\forall b_3[\mathsf{goal}(\mathsf{on}(b_3, b_2)) \rightarrow b_3 = b] \wedge$
$[t]\ \mathsf{goodtowerbelow}(b_2)]]$

gctrl1 $\Box\forall b[\mathsf{clear}(b) \wedge \mathsf{goodtower}(b) \rightarrow$
$\bigcirc(\mathsf{clear}(b) \vee \exists b_2[\mathsf{on}(b_2, b) \wedge \mathsf{goodtower}(b_2)])]$

gctrl2 $\Box\forall b[\mathsf{clear}(b) \wedge \neg\mathsf{goodtower}(b) \rightarrow \bigcirc(\neg\exists b_2[\mathsf{on}(b_2, b)])]$

gctrl3 $\Box\forall b[\mathsf{ontable}(b) \wedge \exists b_2[\mathsf{goal}(\mathsf{on}(b, b_2)) \wedge$
$\neg\mathsf{goodtower}(b_2)] \rightarrow \bigcirc(\neg\mathsf{holding}(b))]$

If the goal narrative $\mathcal{GN}$ above is provided as input to the planning algorithm, the output $\mathcal{N}_p$ will be:

[3] A narrative preamble (not included here) is also part of a narrative and contains type information for the fluents and actions in the narrative.

acs1–4, obs1–4: Same as above
occ1 $[0, 1]$ $\mathsf{unstack}(\mathsf{BB}, \mathsf{AA})$
occ2 $[1, 2]$ $\mathsf{putdown}(\mathsf{BB})$
occ2 $[2, 3]$ $\mathsf{unstack}(\mathsf{AA}, \mathsf{EE})$
occ2 $[3, 4]$ $\mathsf{putdown}(\mathsf{AA})$
occ2 $[4, 5]$ $\mathsf{pickup}(\mathsf{FF})$
occ2 $[5, 6]$ $\mathsf{stack}(\mathsf{FF}, \mathsf{EE})$

It is easily observable that $\mathcal{N}_p$ entails the intended goal Υ_{igoal} at time 6.

In Section 5, we describe how the plan sequence Γ_{occ} is generated given an initial goal narrative $\mathcal{GN}$. Before doing this, we will consider the use of *modal* formulas in TAL.

In the rest of the paper, let $\mathcal{GN}$ be a goal narrative.

4. Modal Formulas in TAL

From a semantic perspective, temporal modalities in TAL are simply viewed as a special type of macro-operator in the extended surface language $\mathcal{L}(\mathrm{ND})^*$. Given a formula ϕ in $\mathcal{L}(\mathrm{ND})^*_{\mathtt{Control}}$ containing temporal modal operators, the formula $[\tau]\ \phi$, where τ is the timepoint where ϕ is intended to hold, can be translated into a formula in the base language $\mathcal{L}(\mathrm{FL})$ without temporal modal operators. Modal formulas are used in the following ways:

- TALplan/modal – Control formulas in a goal narrative may contain modal operators. The control formulas are progressed in the TALplan/modal planner using a progression algorithm.
- TALplan/non-modal – Control formulas in a goal narrative may contain modal operators. The control formulas are transformed into a formula in $\mathcal{L}(\mathrm{ND})^*$ without temporal modalities before being used in the TALplan/non-modal planner. This planner is designed in a different manner and contains no progression algorithm.

There are four specific temporal operators, U (until), $\Diamond$ (eventually), $\Box$ (always), and $\bigcirc$ (next), but all of them may be defined in terms of the U operator. As in Kabanza [11], the first three temporal operators can be indexed with closed, open or semi-open intervals. The meaning of a formula containing a temporal operator is dependent on the point n (the current state) in which it is evaluated.

- $\phi\, U_{[\tau,\tau']}\, \psi$ – ϕ must hold from n until ψ is achieved at some timepoint in $[\tau + n, \tau' + n]$.
- $\Diamond_{[\tau,\tau']}\, \phi \equiv \mathit{true}\, U_{[\tau,\tau']}\, \phi$ – Eventually ϕ will be true at a timepoint in $[\tau + n, \tau' + n]$.
- $\Box_{[\tau,\tau']}\, \phi \equiv \neg\Diamond_{[\tau,\tau']}\, \neg\phi$ – ϕ must always be true at all points in the interval $[\tau + n, \tau' + n]$.
- $\bigcirc\phi \equiv \mathit{true}\, U_{[1,1]}\phi$.

The following abbreviations are also used:

- $\phi\, U\, \psi \equiv \phi\, U_{[0,\infty]}\, \psi$, $\Diamond\, \phi \equiv \Diamond_{[0,\infty]}\, \phi$, $\Box\, \phi \equiv \Box_{[0,\infty]}\, \phi$, $[\tau, \tau'] < 0 \equiv \tau < 0 \wedge \tau' < 0$.

The translation function *TransModal* takes a timepoint and a modal control formula as input and returns a formula in $\mathcal{L}(\mathrm{ND})^*$ without temporal modalities as output. In the following, $\mathcal{Q}$ is a quantifier and $\otimes$ is a binary logical connective.

Inputs: A formula $\gamma \in \mathcal{L}(\mathrm{ND})^*_{\texttt{Control}}$ and a timepoint n where γ is intended to be evaluated.
Output: A formula in $\mathcal{L}(\mathrm{ND})^*$ without temporal modalities.

procedure *TransModal*(n, γ)
if $\gamma =$ *goal*(ϕ) **then return** *goal*(ϕ)
if γ contains no modalities **then return** $[n]\ \gamma$
if $\gamma = \mathcal{Q}x.\phi$ **then return** $\mathcal{Q}x.$*TransModal*(n, ϕ)
if $\gamma = \phi \otimes \psi$ **then**
  return *TransModal*$(n, \phi) \otimes$ *TransModal*(n, ψ)
if $\gamma = \neg\phi$ **then return** $\neg$*TransModal*(n, ϕ)
if $\gamma = (\phi\, U_{[\tau,\tau']}\psi)$ **then return** $(\exists[t : n + \tau \leq t \leq n + \tau']$ (*TransModal*$(t, \psi) \wedge \forall[t' : n \leq t' < t]$*TransModal*$(t', \phi)))$

The algorithm *TransModal* provides the *meaning* of the temporal modalities in TAL, a linear, discrete temporal logic, which correspond to their intuitive meaning in a linear tense logic.

5. TALplanner: A Forward Chaining Planner

The algorithm described below is based on a combination of those found in Bacchus [5] and Kabanza [11]. The distinction is that the algorithms are modified for the TAL family of logics and the notion of a narrative. In addition, the cycle check is done in a different place. The implementation of the algorithm also differs. Two of the major differences are the use of lazy evaluation in the node expansion algorithm in TALplanner and the method used to evaluate formulas in states. The method used is the same as that used in VITAL [16].

5.1. The TALplan/modal Algorithm

Inputs: An initial goal narrative $\mathcal{GN}$, a sentence $\alpha \in \mathcal{L}(\mathrm{ND})^*_{\texttt{Goal}}$, and a sentence $\gamma \in \mathcal{L}(\mathrm{ND})^*_{\texttt{Control}}$.
Output: A narrative plan $\mathcal{N}_p$ which entails the goal α.

procedure *TALplan/modal*$(\alpha, \gamma, \mathcal{GN})$
$\texttt{acc} \leftarrow \{\}$ // Accepted final states
$\texttt{Open} \leftarrow \{(\gamma, 0, 0, \mathcal{GN})\}$
while $\texttt{Open} \neq \emptyset$ **do**
  $(\phi, \tau, \tau', \mathcal{GN}) \leftarrow$ *Choose*$(\texttt{Open})$
  $\texttt{Open} \leftarrow \texttt{Open} \setminus (\phi, \tau, \tau', \mathcal{GN})$
  $\phi^+ \leftarrow$ *Progress_goal*$(\phi, \tau, \tau', \mathcal{GN})$
  if $\phi^+ \neq$ *false* **then**
    if *GoodPlan*$(\alpha, \tau', \mathcal{GN})$ **then**
      return $\mathcal{GN} \setminus (\Upsilon_{igoal} \cup \Upsilon_{gctrl} \cup \Upsilon_{gdom})$
    if (state at time τ' for $\mathcal{GN}$) $\notin \texttt{acc}$ **then**
      $\texttt{acc} \leftarrow \texttt{acc} \cup \{$(state at time τ' for $\mathcal{GN}$)$\}$
      $\texttt{Open} \leftarrow \texttt{Open} \cup \{(\phi^+, \tau_1, \tau_2, \mathcal{GN}') \mid$
        $(\mathcal{GN}', [\tau_1, \tau_2]\ \texttt{a}) \in$ *Expand*$(\tau', \mathcal{GN})\}$

Different implementations of *Choose* provide different search algorithms. For example, in the empirical tests we use depth-first search. Different implementations of *Expand* provide for the possibility of using different lookahead, decision-theoretic and filtering mechanisms for choice of actions. Different implementations of *GoodPlan* provide for different criteria for evaluating plans which satisfy the goal, in terms of resource usage etc.

Inputs: A timepoint τ and a goal narrative $\mathcal{GN}$.
Output: A set of pairs $(\mathcal{GN}'_i, [\tau, \tau'_i]\ \texttt{a}_i)$, where for all i, $\mathcal{GN}'_i = \mathcal{GN} \cup \{[\tau, \tau'_i]\ \texttt{a}_i\}$ (that is, the old narrative with a new action occurrence added).

procedure *Expand*$(\tau, \mathcal{GN})$
$\texttt{Succ} \leftarrow \{\}$
for all $\texttt{a}(\overline{x}) \in$ *ActionTypes*$(\mathcal{GN})$ **do**
  for all $[\tau, \tau']\ \texttt{a}(\overline{c}) \in$ *Instantiate*$(\tau, \texttt{a}(\overline{x}))$ **do**
    for all $\phi \in$ *Preconds*$([\tau, \tau']\ \texttt{a}(\overline{c}))$ **do**
      if *Trans*$(\mathcal{GN} \setminus (\Upsilon_{igoal} \cup \Upsilon_{gctrl} \cup \Upsilon_{gdom})) \models$ *Trans*$([\tau]\ \phi)$
      then $\texttt{Succ} \leftarrow$
        $\texttt{Succ} \cup \{(\mathcal{GN} \cup \{[\tau, \tau']\ \texttt{a}(\overline{c})\}, [\tau, \tau']\ \texttt{a}(\overline{c}))\}$
return Succ

Inputs: A sentence $\alpha \in \mathcal{L}(\mathrm{ND})^*_{\texttt{Goal}}$, a timepoint τ and a goal narrative $\mathcal{GN}$.
Output: *true* iff $\mathcal{GN}$ satisfies α at timepoint τ.

procedure *GoodPlan*$(\alpha, \tau, \mathcal{GN})$
if *Trans*$(\mathcal{GN} \setminus (\Upsilon_{igoal} \cup \Upsilon_{gctrl} \cup \Upsilon_{gdom})) \models$ *Trans*$([\tau]\alpha)$ **then return** *true*
else return *false*

5.1.1. Progression of modal formulas. Assume that an action a occurs in interval $[\tau, t_2]$ where the state associated with timepoint τ is the current state from which one is progressing. Assume further that $\mathcal{GN}$ is a partial narrative containing the action occurrence $[\tau, t_2]$ a and where there are no other action occurrences $[\tau', t'_2]$ a$'$ in $\mathcal{GN}$ where $t'_2 > \tau$. The following algorithm can be used for progressing a formula ϕ from τ to t_2.

Inputs: A timepoint τ corresponding to the current state, a timepoint $t_2 \geq \tau$ corresponding to the successor state, a formula $\phi \in \mathcal{L}(\mathrm{ND})^*_{\texttt{Control}}$ labeling the current state, and a goal narrative $\mathcal{GN}$.
Output: A new formula ϕ^+ labeling the successor state.

Algorithm *Progress_goal*$(\phi, \tau, t_2, \mathcal{GN})$

1. $\tau = t_2 : \phi^+ = \phi$
2. ϕ contains no temporal modalities:
   if *Trans*$^*(\mathcal{GN} \setminus \Upsilon_{gctrl}) \models^*$ *Trans*$^*([\tau]\ \phi)$ **then** $\phi^+ \leftarrow$ *true*
   else $\phi^+ \leftarrow$ *false*
3. $\phi = \neg\phi_1 : \phi^+ \leftarrow \neg Pg(\phi_1, \tau, t_2, \mathcal{GN})$
4. $\phi = \phi_1 \otimes \phi_2 : \phi^+ \leftarrow Pg(\phi_1, \tau, t_2, \mathcal{GN}) \otimes Pg(\phi_2, \tau, t_2, \mathcal{GN})$

5. $\phi = \phi_1 U_{[\tau_1,\tau_2]} \phi_2$: **if** $[\tau_1, \tau_2] < 0$ **then** $\phi^+ \leftarrow$ *false*
else if $0 \in [\tau_1, \tau_2]$ **then** $\phi^+ \leftarrow Pg(\phi_2, \tau, t_2, \mathcal{GN}) \vee$
$(Pg(\phi_1, \tau, t_2, \mathcal{GN}) \wedge Pg(\phi_1 U_{[\tau_1-1,\tau_2-1]} \phi_2, \tau+1, t_2, \mathcal{GN}))$
else $\phi^+ \leftarrow Pg(\phi_1, \tau, t_2, \mathcal{GN}) \wedge$
$Pg(\phi_1 U_{[\tau_1-1,\tau_2-1]} \phi_2, \tau + 1, t_2, \mathcal{GN})$

The result of *Progress_goal* (abbreviated *Pg*) is simplified using the rules $\neg$*false* = *true*, (*false* $\wedge \alpha$) = ($\alpha \wedge$ *false*) = *false*, (*false* $\vee \alpha$) = ($\alpha \vee$ *false*) = α, $\neg$*true* = *false*, (*true* $\wedge$ α) = ($\alpha \wedge$ *true*) = α, and (*true* $\vee \alpha$) = ($\alpha \vee$ *true*) = *true*.

Since $\bigcirc$, $\Box$, and $\Diamond$ can be defined in terms of U, the algorithm above suffices, although adding the following cases to the algorithm might be useful for clarity and efficiency:

6. $\phi = \Diamond_{[\tau_1,\tau_2]}\ \phi_1$: **if** $[\tau_1, \tau_2] < 0$ **then** $\phi^+ \leftarrow$ *false*
else if $0 \in [\tau_1, \tau_2]$ **then** $\phi^+ \leftarrow$
$Pg(\phi_1, \tau, t_2, \mathcal{GN}) \vee Pg(\Diamond_{[\tau_1-1,\tau_2-1]} \phi_1, \tau + 1, t_2, \mathcal{GN})$
else $\phi^+ \leftarrow Pg(\Diamond_{[\tau_1-1,\tau_2-1]} \phi_1, \tau + 1, t_2, \mathcal{GN})$
7. $\phi = \Box_{[\tau_1,\tau_2]}\ \phi_1$: **if** $[\tau_1, \tau_2] < 0$ **then** $\phi^+ \leftarrow$ *false*
else if $0 \in [\tau_1, \tau_2]$ **then** $\phi^+ \leftarrow$
$Pg(\phi_1, \tau, t_2, \mathcal{GN}) \wedge Pg(\Box_{[\tau_1-1,\tau_2-1]} \phi_1, \tau + 1, t_2, \mathcal{GN})$
else $\phi^+ \leftarrow Pg(\Box_{[\tau_1-1,\tau_2-1]} \phi_1, \tau + 1, t_2, \mathcal{GN})$
8. $\phi = \bigcirc \phi_1$: **if** $\tau + 1 = t_2$ **then** $\phi^+ \leftarrow \phi_1$
else $\phi^+ \leftarrow Pg(\phi_1, \tau + 1, t_2, \mathcal{GN})$

The following theorem provides a semantic justification for the use of the progression algorithm.

Theorem 1 *Trans*$^*(\mathcal{N}) \models^*$ *Trans**(*TransModal*$(n, \phi))$ *iff* *Trans*$^*(\mathcal{N}) \models^*$ *Trans**(*TransModal*$(t_2,$ *Progress_goal*$(\phi, n, t_2, \mathcal{N})))$

5.1.2. Action Duration and Internal State. In [5], Bacchus and Kabanza use a first-order version of LTL [9], linear temporal logic, and restrict their algorithm to single step actions. In [4], they use a first-order version of MITL [2], metric interval temporal logic, where actions may have duration. In the latter case, although actions have duration, they have no internal state. In other words, a plan step may have duration, but there are no states or state changes between the initiation state and the effect state. This is reflected in the model structure for the logics and the progression algorithm. On the other hand, TAL actions with duration have internal state and one can express change in fluent values within an action's duration. Consequently, the *Progress_goal* algorithm proposed in [4] has to be modified to reflect progression within an action's duration. In this case, we replace the original step 5 in [4] (reformulated for TAL):

5. $\phi = \phi_1 U_{[\tau_1,\tau_2]} \phi_2$: **if** $[\tau_1, \tau_2] < 0$ **then** $\phi^+ \leftarrow$ *false*
else if $0 \in [\tau_1, \tau_2]$ **then** $\phi^+ \leftarrow Pg(\phi_2, \tau, t_2, \mathcal{GN}) \vee$
$(Pg(\phi_1, \tau, t_2, \mathcal{GN}) \wedge \phi_1 U_{[\tau_1,\tau_2]-(t_2-\tau)} \phi_2)$
else $\phi^+ \leftarrow Pg(\phi_1, \tau, t_2, \mathcal{GN}) \wedge \phi_1 U_{[\tau_1,\tau_2]-(t_2-\tau)} \phi_2$

with its modification for actions with internal state in step 5 of the current algorithm (Section 5.1.1). Steps 6–7 are also modified accordingly.

5.2. The TALplan/non-modal Algorithm

The non-modal version of the TALplan algorithm does not progress control formulas. Instead, it translates them into control formulas with no temporal modal operators using the *TransModal*$^+$ algorithm (see below).

Inputs: An initial goal narrative $\mathcal{GN}$, a sentence $\alpha \in \mathcal{L}(\text{ND})^*_{\texttt{Goal}}$, and a sentence $\gamma \in \mathcal{L}(\text{ND})^*_{\texttt{Control}}$.
Output: A narrative plan $\mathcal{N}_p$ which entails the goal α.

procedure *TALplan/non-modal*$(\alpha, \gamma, \mathcal{GN})$
  `acc` $\leftarrow \{\}$ // Accepted final states
  `Open` $\leftarrow \{($*TransModal*$^+(0, \gamma), 0, 0, \mathcal{GN})\}$
  while `Open` $\neq \emptyset$ **do**
    $(\phi, \tau, \tau', \mathcal{GN}) \leftarrow$ *Choose*(`Open`)
    `Open` $\leftarrow$ `Open` $\setminus (\phi, \tau, \tau', \mathcal{GN})$
    if not *Trans*$^*(\mathcal{GN} \setminus \Upsilon_{gctrl}) \models^*$ $\neg$*Trans*$^*(\phi(\tau'))$ **then**
      if *GoodPlan*$(\alpha, \tau', \mathcal{GN})$ **then**
        return $\mathcal{GN} \setminus (\Upsilon_{igoal} \cup \Upsilon_{gctrl} \cup \Upsilon_{gdom})$
      if (state at time τ' for $\mathcal{GN}$) $\notin$ `acc` **then**
        `acc` $\leftarrow$ `acc` $\cup$ {(state at time τ' for $\mathcal{GN}$)}
        `Open` $\leftarrow$ `Open` $\cup$ $\{(\phi, \tau_1, \tau_2, \mathcal{GN}') \mid$
          $(\mathcal{GN}', [\tau_1, \tau_2]$ `a`$) \in$ *Expand*$(\tau', \mathcal{GN})\}$

One of the main advantages of using TALplan/non-modal is that since it does not store multiple progressed control formulas, the algorithm uses much less memory (see Section 6 and Table 1).

The translation function *TransModal*$^+$ is similar to *TransModal*. The differences are due to the fact that formula progression is not used in TALplan/non-modal. The function takes a timepoint and a modal control formula as input and returns a formula in $\mathcal{L}(\text{ND})^*$ without temporal modalities as output. In the following, $\mathcal{Q}$ is a quantifier and $\otimes$ is a binary logical connective.

Assuming $\mathcal{GN}$ is a goal narrative in $\mathcal{L}(\text{ND})^*$ with at least one action occurrence, t_* is the ending timepoint of the last action occurrence in $\mathcal{GN}$.

Inputs: A formula $\gamma \in \mathcal{L}(\text{ND})^*_{\texttt{Control}}$ and a timepoint n where γ is intended to be evaluated.
Output: A formula in $\mathcal{L}(\text{ND})^*$ without temporal modalities, parameterized by t_*.

procedure *TransModal*$^+(n, \gamma)$
if $\gamma =$ *goal*(ϕ) **then return** $n \le t_* \rightarrow$ *goal*(ϕ)
if γ contains no modalities **then return**
$n \le t_* \rightarrow [n]\ \gamma$
if $\gamma = \mathcal{Q}x.\phi$ **then return**
$n \le t_* \rightarrow \mathcal{Q}x.$*TransModal*$^+(n, \phi)$
if $\gamma = \phi \otimes \psi$ **then return**
$n \le t_* \rightarrow$ (*TransModal*$^+(n, \phi) \otimes$ *TransModal*$^+(n, \psi)$)
if $\gamma = \neg\phi$ **then return** $n \le t_* \rightarrow \neg$*TransModal*$^+(n, \phi)$
if $\gamma = (\phi\, U_{[\tau,\tau']} \psi)$ **then return** $(n + \tau' \le t_*) \rightarrow$
$(\exists[t : n + \tau \le t \le n + \tau']$ (*TransModal*$^+(t, \psi) \wedge$
$\forall[t' : n \le t' < t]$ *TransModal*$^+(t', \phi)))$

6. Empirical Results

We have tested TALplanner in the blocks world as well as in the movie, gripper, logistics, mystery, mystery prime and grid domains from Round 1 of the AIPS 1998 Planning Competition [1]. We compared the results to TLplan in three of the domains: The blocks world, the logistics world, and the movie world. For all of these domains, the general setup (operators, predicates and control rules) and many of the actual problems (initial state, goal, and objects in the domain) have been taken from the TLplan distribution.

All tests were run on the same 333 MHz Pentium II computer running Windows NT 4.0 SP3, using 256 MB of memory. For TLplan, we used the precompiled version that can be downloaded from `http://www.lpaig.uwaterloo.ca/~fbacchus`. For TALplanner, we used JDK 1.2 (`http://java.sun.com`). In all cases, we made sure that the computer was very lightly loaded and that it was never swapping.

Note that for this experiment, TALplanner tried different operators in exactly the same order as TLplan, in order to avoid random differences in the amount of time and memory needed by the planners.

For the standard blocks world, we created ten different test scenarios using between 25 and 1000 blocks. In all cases, TLplan used the world definition and control rules from `domains/Blocks/4OpsBlocksWorld.tlp` in the TLplan distribution, and TALplanner used the corresponding TAL world definition and control rules seen in the example scenario in Section 3.3.

Table 1 contains the results. The *Worlds* column contains the number of worlds that were created by TLplan. This is equal to the number of narratives that were added to `Open` in TALplanner. *Plan* shows the length of the resulting plan, and the remaining columns show times (in seconds) and memory usage (in kilobytes).

For the logistics world, in which objects can be transported within cities by truck and between airports by airplane, we tested the 30 scenarios from the TLplan distribution that were originally from [1]. TALplanner was always faster than TLplan, in several cases 30–75 times faster. The movie world, which does not use control formulas, gave similar results, mainly as a result of lazy evaluation in TALplanner. Due to space limitations, the exact results of these and additional domains will be presented in a forthcoming report.

In each case, TALplanner outperformed TLplan, both for modal and for non-modal control formulas. For the smaller scenarios, TALplanner needs more memory than TLplan, since it needs the Java Virtual Machine. However, TALplanner itself uses less memory than TLplan. Therefore, TALplanner could handle larger scenarios than TLplan, which could not handle the three largest scenarios without swapping.

In comparing TALplanner with TLplan, we believe that the primary reason for the speedup is simply due to various code optimizations used in TALplanner. More interesting is the comparison between the modal and non-modal versions of TALplanner. Using modal control formulas was slightly slower than non-modal formulas for the smallest scenarios and for one of the largest scenarios. For most scenarios, modal control formulas were faster than non-modal. We conjecture that the reason for this is the use of optimization techniques related to evaluating universally quantified control formulas where a node in the search space has many children. In the non-modal version the same evaluation has to be done for each child, whereas for the modal version similar evaluations need only be done once. The non-modal version also uses less memory than the modal version since there is no need to store additional progressed formulas.

7. Conclusions

Bacchus [5] and Kabanza [11] have proposed a novel progressive planning technique based on the use of a modal tense logic to represent domain-dependent knowledge for controlling search. The technique has proven to be extremely successful for several application domains. The work presented in this paper, without claiming any novelty in terms of the basic technique, verifies the results shown by TLplan. We do this by introducing a new forward chaining planner, TALplanner. Two versions were considered, TALplan/modal which is based on the use of *modal* formulas and a progression algorithm, and TALplan/non-modal which does not use modal formulas or a progression algorithm. TALplan/modal is most similar to the work of Bacchus and Kabanza, while TALplan/non-modal is considerably different in design. We showed how temporal modalities could be emulated in TAL, a nonmonotonic temporal logic for reasoning about action and change. Both versions of TALplanner were empirically tested against TLplan and in both cases and for all domains, TALplanner was shown to be considerably faster and required less memory. The novelty of our approach is the integration of TAL with the forward chaining plan paradigm. Due to its considerable expressivity, TAL is an ideal candidate for continuing research with TALplanner, especially in the context of planning in incompletely specified dynamic environments with variable time constraints for acting. We demonstrated this by extending the original algorithms so planning can be done using durative actions with internal state. We are currently working on extensions to TALplanner related to the use of nondeterministic plan operators, concurrent plan operators, dependency and domain constraints, and incomplete initial states.

	Blocks	Plan length	Worlds created	TLplan		TALplan/modal		TALplan/non-modal	
				time (s)	memory (k)	time (s)	memory (k)	time (s)	memory (k)
1	25	16	344	0.110	3104	0.110	6640	0.060	6612
2	50	70	2295	1.963	5672	1.603	6792	1.302	6644
3	70	106	4361	7.501	8752	4.677	7516	4.406	6644
4	100	160	8945	37.254	14912	14.441	7772	14.060	6644
5	140	232	17829	185.497	27532	39.246	9372	41.940	7208
6	280	580	74297	4297.750	104464	394.196	21308	474.012	8536
7	460	580	178697	32303.100	178884	3208.159	39528	1899.992	9840
8	640	1228	365069			5862.197	68464	7679.733	14284
9	820	1908	463779			10487.159	95620	12837.629	18732
10	1000	2232	718281					25028.509	24264

Table 1. Blocks world results

8. Acknowledgments

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

References

[1] AIPS98. Artificial Intelligence Planning Systems: 1998 Planning Competition. `http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`, 1998.

[2] R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctiality. In *10th ACM Symposium on Principles of Distributed Computing*, pages 139–152, 1991.

[3] F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 141–153. ISO Press, 1996.

[4] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[5] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 1998. Submitted for publication.

[6] P. Doherty. Reasoning about action and change using occlusion. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 401–405, 1994.

[7] P. Doherty, J. Gustafsson, L. Karlsson, and J. Kvarnström. TAL: Temporal Action Logics, language specification and tutorial. *Linköping Electronic Articles in Computer and Information Science*, 3(15), 1998. Submitted to ETAI. `http://www.ep.liu.se/ea/cis/1998/015/`.

[8] P. Doherty and J. Kvarnström. Tackling the qualification problem using fluent dependency constraints: Preliminary report. In *Proceedings of the 5th International Workshop on Temporal Representation and Reasoning (TIME'98)*, 1998.

[9] E. A. Emerson. *Handbook of Theoretical Computer Science*, volume B, chapter Temporal and Modal Logic. MIT, 1990.

[10] J. Gustafsson and P. Doherty. Embracing occlusion in specifying the indirect effects of actions. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning*, pages 87–88, San Francisco, 1996. Morgan Kaufmann Publishers.

[11] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95:67–113, 1997.

[12] L. Karlsson and J. Gustafsson. Reasoning about concurrent interaction, 1998. Accepted for publication in *Journal of Logic and Computation*.

[13] L. Karlsson, J. Gustafsson, and P. Doherty. Delayed effects of actions. In *Proceedings of the 13th European Conference on Artificial Intelligence*, 1998.

[14] H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. System available at `http://www.research.att.com/~kautz`.

[15] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *European Conference on Planning*, pages 273–285, 1997. System available at `http://www.informatik.uni-freburg.de/~koehler/ipp.html`.

[16] J. Kvarnström and P. Doherty. VITAL. An on-line system for reasoning about action and change using TAL, 1997. Available at `http://anton.ida.liu.se/vital/vital.html`.