Temporal Composite Actions with Constraints

Patrick Doherty[†]
[†]Linköping University
S-581 83 Linköping, Sweden
email: patrick.doherty@liu.se

Jonas Kvarnström[†]
[†]Linköping University
S-581 83 Linköping, Sweden
email: jonas.kvarnstrom@liu.se

Andrzej Szałas[†,‡]
‡Warsaw University
02-097 Warsaw, Banacha 2, Poland
email: andrzej.szalas@mimuw.edu.pl

Abstract

Complex mission or task specification languages play a fundamentally important role in human/robotic interaction. In realistic scenarios such as emergency response, specifying temporal, resource and other constraints on a mission is an essential component due to the dynamic and contingent nature of the operational environments. It is also desirable that in addition to having a formal semantics, the language should be sufficiently expressive, pragmatic and abstract. The main goal of this paper is to propose a mission specification language that meets these requirements. It is based on extending both the syntax and semantics of a well-established formalism for reasoning about action and change, Temporal Action Logic (TAL), in order to represent temporal composite actions with constraints. Fixpoints are required to specify loops and recursion in the extended language. The results include a sound and complete proof theory for this extension. To ensure that the composite language constructs are adequately grounded in the pragmatic operation of robotic systems, Task Specification Trees (TSTs) and their mapping to these constructs are proposed. The expressive and pragmatic adequacy of this approach is demonstrated using an emergency response scenario.

Introduction

Imagine an emergency response scenario such as the recent earthquake and tsunami in Japan. An emergency response unit has a team of unmanned aerial and ground vehicles at its disposal in addition to ground operations units which can command these robotic teams to assist them with situation assessment and other tasks such as searching for injured civilians or dynamic placement of sensors in areas inaccessible to emergency responders. In order to interact, responders require a means of specifying, scheduling and deploying such teams. An example of a simple sequential mission would be to scan an area for radiation damage and then scan another area for injured civilians within a specified amount of time. One might further assume that additional constraints suitable for the current context and contingencies of the situation might also be required to specify the mission further. For instance, in the case of the nuclear reactor meltdown at Fukushima, one might restrict the use of UAVs in this mission to only those with radiation sensors and military grade hardware to minimize radiation damage to the hardware.

A specification of such a mission would include a temporally constrained sequence of tasks,

$[t_1, t_2]$**monitor-radiation**$(area_1)$; **find-injured**$(area_2)$,

where ";" is used as a sequence operator. In terms of representation, the use of temporal duration introduces some interesting specification issues even for this simple case. For instance, it is obvious that **monitor-radiation**$(area_1)$ has to end before **find-injured**$(area_2)$. It is less obvious whether **monitor-radiation**$(area_1)$ has to start precisely at t_1, whether **find-injured**$(area_2)$ has to end precisely at t_2, or whether gaps should be allowed between actions in a sequence. Similar questions arise for concurrency, conditionals, loops and their combinations when specifying complex temporal tasks.

One of the main issues of focus in this paper is the development of a suitable formal semantics for temporally constrained composite actions that is flexible enough to be used in the field with deployable robotic systems interacting with themselves and with human operators. In this context, tasks may be distributed among multiple robotic systems and their execution is often contextualized with constraints associated with the environmental situations in which they operate.

The approach taken is to formally define temporal composite actions with constraints in Temporal Action Logic (TAL) (Doherty and Kvarnström 2008), a well established logic of action and change. It is then shown how such actions can be mapped into an executable declarative specification of tasks used in a number of deployable UAV systems using Task Specification Trees (TSTs) (Doherty, Landén, and Heintz 2010). TSTs are used in current research with delegation frameworks for cooperative robotics (Doherty and Meyer 2012; Doherty, Heintz, and Landén 2011). They provide a flexible and formal means for representing robotic behaviors, (distributed) tasks, compiled plans, etc. One research goal is to be able to specify such tasks in TAL as composite actions, verify their properties formally using the logical framework and then compile them into executable versions as TSTs. One can also reverse compile a TST into a composite action in TAL and verify its properties formally.

The main representational approach taken is the following. A composite action specification will be characterized recursively using the following construct:

C-ACT ::= with VARS do TASK where CONS

This states that a complex action C-ACT is always specified in a context characterized by a set of variables VARS constrained by a set of constraints CONS. Formally, constraints can be as general as a logical formula, or pragmatically as specific as a set of constraints input to a constraint satisfaction solver. By making the context explicit in terms of a set of constraints, we can use a policy of least commitment in terms of both temporal and other types of constraints. This approach is not only useful in dealing with the temporal issues discussed previously, but may in fact be used explicitly in a robotic system to formally verify or determine the consistency of a (distributed) task through the use of conventional (distributed) constraint satisfaction techniques. This will be considered later in the paper.

Structure and Results

This article presents the following results:

- TAL is extended to represent temporal composite actions with constraints which may include concurrency, sequence, conditionals, loops, while-do, and a concurrent for-each operator. Actions can be recursive and loops can be unbounded. A translation function maps composite actions into elementary actions with constraints.
- To provide a formal semantics for loops and recursion, TAL is extended with a least fixpoint operator. The model theory is supplemented to use strictly arithmetical structures. A proof theory for this fixpoint logic is presented and shown to be sound and complete relative to such structures. The extended logic is denoted as $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$.
- Additionally, it is shown that if one bounds the strictly arithmetical structures, one can specify a formal mapping between syntactically restricted $\mathscr{L}$(FL) theories and deductive databases. A provably polynomial inference method can then be specified.
- A formal grammar and syntax is presented for task specification trees (TSTs). TSTs provide an executable declarative specification language for defining robot behaviors or tasks suitable for individual or distributive execution in multi-robot scenarios. It is then shown how $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$ can be used as a *formal* semantics for TSTs. This is done by providing a formal back and forth mapping between composite actions in $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$ and TSTs.
- We show the flexibility and benefits of the approach by specifying part of an emergency response scenario similar to one used with currently deployable UAV systems (Rudol and Doherty 2008; Doherty and Rudol 2007).

The paper is structured as follows: A brief summary of Temporal Action Logic is first provided. It is then extended with new syntax for fixpoints followed by an extension for specifying composite actions. Composite actions are used in the specification of an emergency response scenario. Task Specification Trees are introduced with a mapping to and from composite actions. This is followed by a presentation of a sound and complete proof theory for $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$ and a syntactic restriction relating inference in $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$ to deductive databases. Related work and conclusions are then provided.

Temporal Action Logic

Temporal Action Logic (TAL) (Doherty and Kvarnström 2001b; 2008) is a well established nonmonotonic temporal logical formalism for representing and reasoning about action narratives. It is also used as a semantic basis for TALplanner (Doherty and Kvarnström 2001b) and TFPOP (Kvarnström 2011). For further details, see (Doherty and Kvarnström 2008).

The $\mathscr{L}$(ND) language. A TAL narrative is specified in the extendible high-level macro language $\mathscr{L}$(ND), which provides an abstract syntax for specifying action types, action occurrences, domain and dependency constraints, observations, etc. This language supports the knowledge engineer when constructing narratives and allows narratives to be specified at a higher abstraction level than logical formulas.

The basic ontology for TAL consists of parameterized features that have values at specific timepoints, $[t]f(\bar{x}) \hat{=} v$, parameterized actions that occur at specific intervals of time, $[t_1,t_2]A(\bar{x})$, and an occlusion operator $X([t]f(\bar{x}))$ which excludes or *occludes* features from a default inertia assumption at explicit timepoints. The value of a feature at a timepoint is denoted by $value(t,f)$.

The $\mathscr{L}$(FL) language. The $Trans()$ function translates $\mathscr{L}$(ND) expressions into $\mathscr{L}$(FL), a first-order logical language that currently uses discrete linear time structures axiomatized as Presburger arithmetic. When adding new constructs to the formalism, the basic idea is to define new expression types in $\mathscr{L}$(ND) and extend the translation function accordingly. This will in fact be done for composite actions.

$\mathscr{L}$(FL) is order-sorted, supporting subsorts. There are a number of sorts for values $\mathscr{V}_i$, including the Boolean sort $\mathscr{B}$ with the constants $\{True, False\}$. $\mathscr{V}$ is a supersort of all such value sorts. There are a number of sorts for features $\mathscr{F}_i$, each one associated with a value sort $dom(\mathscr{F}_i) = \mathscr{V}_j$ for some j. The sort $\mathscr{F}$ is a supersort of all fluent sorts. There is also an action sort $\mathscr{A}$ and a temporal sort $\mathscr{T}$. Generally, t, t' will denote temporal variables, while $\tau, \tau, \tau_1, \ldots$ denote temporal terms. $\mathscr{L}$(FL) currently uses the following predicates:

- $Holds: \mathscr{T} \times \mathscr{F} \times \mathscr{V}$, where $Holds(t,f,v)$ expresses that a feature f has a value v at a timepoint t, corresponding to $[t]\ f \hat{=} v$ in $\mathscr{L}$(ND).
- $Occlude: \mathscr{T} \times \mathscr{F}$, where $Occlude(t,f)$ expresses that a feature f is exempt from the inertia assumption at time t. This corresponds to $X([t]f)$ in $\mathscr{L}$(ND).
- $Occurs: \mathscr{T} \times \mathscr{T} \times \mathscr{A}$, where $Occurs(t_s,t_e,A)$ expresses that a certain action A occurs during the interval $[t_s,t_e]$. This corresponds to $[t_s,t_e]A$ in $\mathscr{L}$(ND).

Formulas can be defined inductively using the standard rules, connectives and quantifiers of first-order logic.

Foundational Axioms and Circumscription. When a narrative is translated, macro expansion generates appropriate *Occlude* formulas for each action and dependency constraint (causal) formula as exemplified below. Additional foundational axioms such as unique names and domain closure axioms are appended (when required). A filtered circumscription policy is then used as a basis for solving the frame, ramification and qualification problems, where both *Occlude* and

Occurs are minimized relative to specific partitions in the narrative theory. Although 2nd-order in nature, the circumscription axioms used are reducible to equivalent 1st-order formulas, and can often be replaced using predicate completion. Therefore, classical first-order theorem proving techniques can be used for reasoning about TAL narratives.

The circumscription policy ensures that actions occur only when stated and that fluents are occluded only when explicitly affected by an action or dependency constraint. An additional axiom states that only occluded fluents are permitted to change value. Dependency constraints can be used to explicitly specify indirect effects of actions, providing a means of dealing with the ramification problem. A modular representation of qualifications is achieved through a combination of fluents with default values and dependency constraints affecting such fluents whenever an action is qualified. See (Doherty and Kvarnström 2008) for details.

Elementary Actions. A narrative specification in $\mathscr{L}$(ND) includes *action type specifications*, which declare a named elementary action. The basic structure is as follows:

$$[t_1,t_2]A(\bar{v}) \rightsquigarrow (\Gamma_{pre}(t_1,\bar{v}) \rightarrow \Gamma_{post}(t_1,t_2,\bar{v})) \wedge \Gamma_{cons}(t_1,t_2,\bar{v}), \quad (1)$$

stating that if $A(\bar{v})$ is executed during the interval $[t_1,t_2]$, then $\Gamma_{pre}(t_1,\bar{v})$ is its preconditions, $\Gamma_{post}(t_1,t_2,\bar{v})$ its postconditions and $\Gamma_{cons}(t_1,t_2,\bar{v})$ specifies logical constraints associated with the action during its execution. As an example, the following defines the elementary action **fly-to** that will later be used in an emergency response scenario (the R macro denotes assignment at a timepoint or during an interval):

$$\begin{array}{l}
[t,t']\textbf{fly-to}(uav,newx,newy) \rightsquigarrow \\
\quad [t]\mathsf{fuel}(uav) > \mathsf{fuel\text{-}usage}(uav,\mathsf{x}(uav),\mathsf{y}(uav),newx,newy) \rightarrow \\
\quad R([t+1]\ \mathsf{hovering}(uav) \mathrel{\hat{=}} False) \wedge \\
\quad R((t,t']\ \mathsf{x}(uav) \mathrel{\hat{=}} newx \wedge \mathsf{y}(uav) \mathrel{\hat{=}} newy \wedge \\
\qquad \mathsf{fuel}(uav) \mathrel{\hat{=}} value(t,\mathsf{fuel}(uav) - \\
\qquad \mathsf{fuel\text{-}usage}(uav,\mathsf{x}(uav),\mathsf{y}(uav),newx,newy))) \wedge \\
\quad t'-t = value(t,\mathsf{flight\text{-}time}(uav,\mathsf{x}(uav),\mathsf{y}(uav),newx,newy))
\end{array}$$

Its translation into $\mathscr{L}$(FL) is the following, where semantic attachment is used for greater and minus.

$$\begin{array}{l}
\forall t,t',uav,newx,newy[\\
Occurs(t,t',\textbf{fly-to}(uav,newx,newy)) \rightarrow (\\
\quad Holds(t,\mathsf{greater}(\mathsf{fuel}(uav), \\
\qquad \mathsf{fuel\text{-}usage}(uav,\mathsf{x}(uav),\mathsf{y}(uav),newx,newy))) \rightarrow \\
\quad Holds(t+1,\mathsf{hovering}(uav),False) \wedge \\
\quad Holds(t',\mathsf{x}(uav),newx) \wedge Holds(t',\mathsf{y}(uav),newy) \wedge \\
\quad Holds(t',\mathsf{fuel}(uav),value(t,\mathsf{minus}(\mathsf{fuel}(uav), \\
\qquad \mathsf{fuel\text{-}usage}(uav,\mathsf{x}(uav),\mathsf{y}(uav),newx,newy))) \wedge \\
\quad t'-t = value(t,\mathsf{flight\text{-}time}(uav,\mathsf{x}(uav),\mathsf{y}(uav),newx,newy)) \wedge \\
\quad \forall u[t < u \leq t' \rightarrow Occlude(t,\mathsf{x}(uav)) \wedge \\
\qquad Occlude(t,\mathsf{y}(uav)) \wedge Occlude(t,\mathsf{fuel}(uav))] \wedge \\
\quad Occlude(t+1,\mathsf{hovering}(uav)))
\end{array}$$

[1]

Elementary actions will be used as the basic building blocks when we extend $\mathscr{L}$(ND) to support composite actions, and their syntax and semantics will remain the same as in TAL. First, however, we extend the base language $\mathscr{L}$(FL) to support fixpoints, an elegant and succinct way of expressing unbounded loops and recursion in logic.

[1] *Occlude* formulas generated by expanding the R macro.

Adding Fixpoints to TAL. The fixpoint extension to TAL will be denoted as TALF and the fixpoint extension to the language $\mathscr{L}$(FL) as $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$. Fixpoint logic (Arnold and Niwiński 2001) strikes a nice balance between 1st-order and 2nd-order logic. The ability to represent loops, recursion and inductive definitions is essential in the context of reasoning about action and change, yet the increase in expressivity is conservative enough to still allow relatively efficient inference techniques. This is shown later when a proof theory for TALF is presented with soundness and completeness results.

$\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$ is obtained by extending $\mathscr{L}$(FL) to allow fixpoint formulas of the form

$$\mathrm{LFP}\ X(\bar{x}).\big[\Gamma(X,\bar{x},\bar{z})\big] \quad (2)$$

to appear within formulas in $\mathscr{L}$(FL) provided that all occurrences of X in Γ are positive. The meaning of (2) is provided by the following Kleene characterization of fixpoints:

$$\mathrm{LFP}\ X(\bar{x}).\big[\Gamma(X,\bar{x},\bar{z})\big] \equiv \bigvee_{i\in\omega} \Gamma^i(False,\bar{x},\bar{z}), \quad \text{where}$$

$$\Gamma^i(False,\bar{x},\bar{z}) \stackrel{\text{def}}{=} \begin{cases} False & \text{for } i=0 \\ \Gamma(\Gamma^{i-1}(False,\bar{x},\bar{z}),\bar{x},\bar{z}) & \text{for } i>0. \end{cases}$$

Composite Actions

We now extend $\mathscr{L}$(ND) to support *composite action type specifications*, which declare a named composite action:

$$[t,t']\underline{\mathsf{comp}}(\bar{v}) \rightsquigarrow A(t,t',\bar{v})$$

where $\underline{\mathsf{comp}}(\bar{v})$ is a *composite action term* such as $\underline{\mathsf{monitor\text{-}pattern}}(x,y,dist)$, consisting of an action name and a list of parameters, and $A(t,t',\bar{v})$ is a *composite action expression* where only variables in $\{t,t'\} \cup \bar{v}$ may occur free. A composite action expression (C-ACT) supports common constructs such as sequences $(A;B)$ and concurrency $(A \,||\, B)$, and is defined as follows:

$$\begin{array}{lll}
\text{C-ACT} & ::= & [\tau,\tau']\ \text{with}\ \bar{x}\ \text{do TASK where}\ \phi \\
\text{TASK} & ::= & [\tau,\tau']\ \text{ELEM-ACTION-TERM} \mid \\
 & & [\tau,\tau']\ \text{COMP-ACTION-TERM} \mid \\
 & & (\text{C-ACT; C-ACT}) \mid \\
 & & (\text{C-ACT} \,||\, \text{C-ACT}) \mid \\
 & & \text{if}\ [\tau]\ \psi\ \text{then C-ACT else C-ACT} \mid \\
 & & \text{while}\ [\tau]\ \psi\ \text{do C-ACT} \mid \\
 & & \text{foreach}\ \bar{x}\ \text{where}\ [\tau]\ \psi\ \text{do conc C-ACT}
\end{array}$$

where $\bar{x}$ is a potentially empty sequence of variables, ϕ is a TAL logic formula, ELEM-ACTION-TERM is an elementary action term such as **fly-to**(uav,x,y), COMP-ACTION-TERM is a composite action term, and $[\tau]\ \psi$ is a TAL formula referring to facts at a single timepoint τ.

Timing and Constraints. An essential feature of our approach is that like elementary actions, every *part* of a composite action C-ACT is annotated with a temporal interval during which it is executed. For example, the expression

$$\begin{array}{l}
[t_1,t_2]\text{with}\ uav,t_3,t_4,t_5,t_6\ \text{do} \\
\quad \big([t_3,t_4]\textbf{fly-to}(uav,x,y); \\
\qquad [t_5,t_6]\textbf{collect-video}(uav,x,y)\big) \\
\quad \text{where}\ [t_1]\mathsf{has\text{-}camera}(uav)
\end{array}$$

denotes a composite action where two elementary actions take place in sequence within the interval $[t_1,t_2]$.

The with-do-where construct provides a very flexible means

of constraining variables as desired for the task at hand. In essence, "$[t_1,t_2]$with $\bar{x}$ do TASK where ϕ" states that there exists an instantiation of the variables in $\bar{x}$ such that the specified TASK is executed within the interval $[t_1,t_2]$ in a manner satisfying ϕ, which may be a conjunction of temporal, spatial and other types of constraints. Above, this is used to pick a UAV that has a camera rather than an arbitrary UAV.

Though the sequence operator (;) could implicitly constrain the two actions **fly-to** and **collect-video** to cover the entire temporal interval $[t_1,t_2]$, our aim is to maximize flexibility. Therefore the implicit constraints associated with each new composite construct are only as strong as needed to make it semantically meaningful. In particular, sub-actions are only constrained to occur *somewhere within* the execution interval of a composite action, and gaps are permitted between sub-actions – but the actions in a sequence must occur in the specified order without overlapping in time. This is formally specified in the *TransComp* function below.

Should stronger temporal constraints be required, they can be introduced in a where clause. For example, $t_1 = t_3 \wedge t_4 = t_5 \wedge t_6 = t_2$ would disallow gaps in the sequence above. Furthermore, variations and extensions to the constructs proposed here, such as gapless sequences, can easily be added without modifying the $\mathscr{L}(\mathrm{FL_{FP}})$ base logic.

Translation into $\mathscr{L}(\mathbf{FL_{FP}})$. A composite action expression is translated to $\mathscr{L}(\mathrm{FL_{FP}})$ by the following extension to the $Trans()$ function from (Doherty and Kvarnström 2008), which calls a new translation function $TransComp(\tau,\tau',T)$ to translate the task T. The intended meaning is that T occurs *somewhere* within the interval $[\tau,\tau']$.

$$Trans\big([\tau,\tau']\text{with } \bar{x} \text{ do } T \text{ where } \phi\big) \stackrel{\text{def}}{=} \exists \bar{x}\big[TransComp(\tau,\tau',T) \wedge Trans\big(\phi\big)\big]$$

If the task is a call to an elementary action $\mathbf{elem}(\bar{v})$, then *TransComp* calls the standard $Trans()$ function. Calls to named composite actions are discussed later.

$$TransComp(\tau,\tau',[\tau_1,\tau_2]\mathbf{elem}(\bar{v})) \stackrel{\text{def}}{=} Trans\big([\tau_1,\tau_2]\mathbf{elem}(\bar{v})\big) \wedge \tau \le \tau_1 < \tau_2 \le \tau'$$

Two potentially concurrent actions are simply constrained to occur within the given interval $[\tau,\tau']$. A sequence of two actions must additionally occur in the stated order. An if/then/else statement is translated into a conjunction of conditionals, where both the timepoint τ_c at which the condition is checked and the execution interval of the selected action (A_1 or A_2) must be within $[\tau,\tau']$.

$$\begin{aligned}
&TransComp(\tau,\tau',\big([\tau_1,\tau_2]A_1 \;||\; [\tau_3,\tau_4]A_2\big)) \stackrel{\text{def}}{=} \\
&\quad Trans\big([\tau_1,\tau_2]A_1\big) \wedge Trans\big([\tau_3,\tau_4]A_2\big) \wedge \\
&\quad \tau \le \tau_1 \le \tau_2 \le \tau' \wedge \tau \le \tau_3 \le \tau_4 \le \tau' \\
&TransComp(\tau,\tau',\big([\tau_1,\tau_2]A_1;[\tau_3,\tau_4]A_2\big)) \stackrel{\text{def}}{=} \\
&\quad Trans\big([\tau_1,\tau_2]A_1\big) \wedge Trans\big([\tau_3,\tau_4]A_2\big) \wedge \\
&\quad \tau \le \tau_1 \le \tau_2 \le \tau_3 \le \tau_4 \le \tau' \\
&TransComp(\tau,\tau',\text{if } [\tau_c]F \text{ then } [\tau_1,\tau_2]A_1 \text{ else } [\tau_3,\tau_4]A_2) \stackrel{\text{def}}{=} \\
&\quad Trans\big([\tau_c]F\big) \rightarrow Trans\big([\tau_1,\tau_2]A_1\big) \wedge \\
&\quad Trans\big([\tau_c]\neg F\big) \rightarrow Trans\big([\tau_3,\tau_4]A_2\big) \wedge \\
&\quad \tau \le \tau_c \le \tau' \wedge \tau_c \le \tau_1 \le \tau_2 \le \tau' \wedge \tau_c \le \tau_3 \le \tau_4 \le \tau'
\end{aligned}$$

A *concurrent foreach* statement allows a variable number of actions to be executed concurrently. An example is given in the next section, where all available UAVs with the ability to scan for injured people should do so in parallel. Below, $\bar{x}$ is a non-empty sequence of value variables. For all instantiations of $\bar{x}$ satisfying $[\tau_c]F(\bar{x})$, there should be an interval within $[\tau_1,\tau_2]$ where the composite action $A(\bar{x})$ is executed.

$$\begin{aligned}
&TransComp(\tau,\tau',\text{foreach } \bar{x} \text{ where } [\tau_c]F(\bar{x}) \text{ do conc } [\tau_1,\tau_2]A(\bar{x})) \stackrel{\text{def}}{=} \\
&\quad \forall \bar{x}[Trans\big([\tau_c]F(\bar{x})\big) \rightarrow Trans\big([\tau_1,\tau_2]A(\bar{x})\big)] \wedge \\
&\quad \tau \le \tau_c \le \tau_1 \le \tau_2 \le \tau'
\end{aligned}$$

A while loop is translated into a least fixpoint. Informally, the LFP parameter u represents the time at which the previous iteration ended, and is initially given the value τ as seen in the final line below. In each iteration the temporal variable t_c is bound to the timepoint at which the iteration condition F is tested, which must be at least u and at most τ'. If the condition holds, the variables $[t_1,t_2]$ are bound to an interval where the inner action A is executed (similarly constrained to be in $[t_c,\tau']$), the action occurs, and the next iteration may start no earlier than t_2, specified by $X(t_2)$.

$$\begin{aligned}
&TransComp(\tau,\tau',\text{while } [t_c]F \text{ do } [t_1,t_2]A) \stackrel{\text{def}}{=} \\
&\quad \tau \le \tau' \wedge \mathrm{LFP}\ X(u).\big[\\
&\qquad \exists t_c[u \le t_c \le \tau' \wedge \\
&\qquad\quad (Trans\big([t_c]F\big) \rightarrow \exists t_1,t_2[t_c \le t_1 \le t_2 \le \tau' \wedge \\
&\qquad\qquad Trans\big([t_1,t_2]A\big) \wedge X(t_2)])] \\
&\quad \big](\tau)
\end{aligned}$$

Assume a composite action is named using a statement such as $[t,t']\underline{\text{comp}}(\bar{x}) \rightsquigarrow A(t,t',\bar{x})$. A named action can be called in two places: As part of a composite action expression, where one composite action calls another, and at the "top level" of a narrative, where one states that a specific composite action occurs. We therefore extend both *Trans* and *TransComp*:

$$\begin{aligned}
&Trans\big([\tau_1,\tau_2]\underline{\text{comp}}(\bar{a})\big) \stackrel{\text{def}}{=} \\
&\quad \mathrm{LFP}\ Y(t,t',\bar{x}).\big[Trans\big(A'(t,t',\bar{x})\big)\big](\tau_1,\tau_2,\bar{a}) \\
&TransComp(\tau,\tau',[\tau_1,\tau_2]\underline{\text{comp}}(\bar{a})) \stackrel{\text{def}}{=} \\
&\quad Trans\big([\tau_1,\tau_2]\underline{\text{comp}}(\bar{a})\big) \wedge \tau \le \tau_1 \le \tau_2 \le \tau'
\end{aligned}$$

where $A'(t,t',\bar{x})$ is $A(t,t',\bar{x})$ with all occurrences of $\underline{\text{comp}}$ replaced with Y. This use of fixpoints permits direct recursion, where an action may call itself. Full support for mutually recursive action definitions is added through simultaneous fixpoints. We omit this in the paper for brevity and readability.

Relaxed Syntax. Omitting "with $\bar{x}$ do" is considered equivalent to specifying the empty sequence of variables and omitting where ϕ is equivalent to specifying where TRUE. Also, the ; and || constructs are easily extended to allow an arbitrary number of actions, as in $(A;B;C;D)$.

Extensions. We have now defined a small core of important composite actions. This is not an exhaustive list, and additional macros can easily be added. For example, to wait for a formula ϕ to become true, we can define $[\tau,\tau']$wait-for(ϕ) as $Trans\big(\forall u[\tau \le u < \tau' \rightarrow [u]\neg\phi] \wedge [\tau']\phi \wedge \tau' > \tau\big)$.

Preserving Solutions to the Frame, Ramification and Qualification Problems

As discussed earlier, the TAL solution to the frame, ramification and qualification problems uses a filtered circumscription policy involving the minimization of both action occurrences and spurious change. We will now show that this so-

lution carries over directly to TALF. In the following, we assume familiarity with circumscription and common notation used to denote circumscription policies (Lifschitz 1991).

A TAL narrative consists of a logical theory that can be partitioned into specific types of formulas. Let Γ denote the translation of a narrative $\mathscr{N}$ in $\mathscr{L}$(ND) into $\mathscr{L}$(FL) using the *Trans* function,

$$\Gamma = \Gamma_{\mathsf{per}} \wedge \Gamma_{\mathsf{obs}} \wedge \Gamma_{\mathsf{dom}} \wedge \Gamma_{\mathsf{occ}} \wedge \Gamma_{\mathsf{dep}} \wedge \Gamma_{\mathsf{acs}},$$

where Γ_{per}, Γ_{obs}, Γ_{occ}, Γ_{acs}, Γ_{dom}, and Γ_{dep} denote the persistence formulas, observation formulas, action occurrence formulas, action type specifications, domain constraint formulas, and dependency constraint formulas in Γ, respectively.

Filtered circumscription is used to minimize *Occurs* in Γ_{occ} and *Occlude* in $\Gamma_{\mathsf{dep}} \wedge \Gamma_{\mathsf{acs}}$ as follows:

$$\Gamma_1 = \Gamma_{\mathsf{per}} \wedge \Gamma_{\mathsf{obs}} \wedge \Gamma_{\mathsf{dom}} \wedge \mathit{CIRC}[\Gamma_{\mathsf{occ}}; \mathit{Occurs}] \wedge \mathit{CIRC}[\Gamma_{\mathsf{dep}} \wedge \Gamma_{\mathsf{acs}}; \mathit{Occlude}]. \quad (3)$$

Let $\Gamma_2 = \Gamma_{\mathsf{fnd}} \wedge \Gamma_{\mathsf{time}}$, where Γ_{fnd} and Γ_{time} denote additional foundational and temporal axioms. For any narrative $\mathscr{N}$ in TAL whose translation into $\mathscr{L}$(FL) is Γ, a preferred narrative theory in the base logic $\mathscr{L}$(FL) is defined as $\Delta_{\mathscr{N}} = \Gamma_2 \wedge \Gamma_1$. We say that a formula α in the base logic $\mathscr{L}$(FL) is *preferentially entailed* by $\mathscr{N}$ iff $\Delta_{\mathscr{N}} \models \alpha$. It can also be shown that both circumscription formulas are reducible to logically equivalent first-order formulas by showing that *Occurs* and *Occlude* only appear positively in $\mathit{CIRC}[\Gamma_{\mathsf{occ}}; \mathit{Occurs}]$ and $\mathit{CIRC}[\Gamma_{\mathsf{dep}} \wedge \Gamma_{\mathsf{acs}}; \mathit{Occlude}]$, respectively.

It is straightforward to show that the solutions to the frame, ramification and qualification problems are preserved with the addition of composite actions to $\mathscr{L}$(ND) and the extension of $\mathscr{L}$(FL) with fixpoints to $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$. *Trans* and *TransComp* only introduce changes to the narrative partition Γ_{occ} of action occurrence formulas. Rather than just a conjunction of *Occurs* atoms, Γ_{occ} can now consist of boolean combinations of *Occurs* atoms and fixpoint formulas.

Lemma 1 Let $\Gamma = \Gamma_{\mathsf{occ}} \wedge \Gamma'$ be the translation of a narrative $\mathscr{N}$ in $\mathscr{L}$(ND) into $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$ using *Trans*() and *TransComp*(). The *Occurs* predicate only occurs positively in Γ_{occ}.

Proof. By structural induction. ◁

Lemma 2 The circumscription of *Occurs* in Γ_{occ} can be expressed as follows:

$$\mathit{CIRC}[\Gamma_{\mathsf{occ}}(\mathit{Occurs}); \mathit{Occurs}] \equiv \Gamma_{\mathsf{occ}}(\mathit{Occurs}) \wedge \neg\exists \bar{x}[\mathit{Occurs}(\bar{x}) \wedge \Gamma_{\mathsf{occ}}(\lambda \bar{y}(\mathit{Occurs}(\bar{y}) \wedge \bar{x} \neq \bar{y}))] \quad (4)$$

Proof. By Lemma 1 and the following proposition (Lifschitz 1991, p. 316): If $A(P,Z)$ is positive relative to P, then the circumscription $\mathit{CIRC}[A(P,Z); P; Z]$ is equivalent to

$$A(P,Z) \wedge \neg\exists x,z[P(x) \wedge A(\lambda y(P(y) \wedge x \neq y), z)].$$

◁

Note that the formula on the rhs of the equivalence in Lemma (2) can be a fixpoint formula in the general case.

Lemma 3 Let $\Gamma_{\mathsf{occ}} \wedge \Gamma'$ be the translation of a narrative $\mathscr{N}$ in $\mathscr{L}(ND)$ into $\mathscr{L}(FL_{FP})$ using *Trans*() and *TransComp*(). Assume that there is a finite bound on timepoints and other domains used in the narrative translation. Then $\mathit{CIRC}[\Gamma_{\mathsf{occ}}(\mathit{Occurs}); \mathit{Occurs}]$ is reducible to a logically equivalent 1st-order formula. ◁

The Scenario, Formalized

Several composite actions that can be useful in disasters such as the Fukushima scenario will now be considered. In particular, multiple UAVs will be used to monitor radiation and scan for injured people in a regular grid around a damaged reactor. Note that our focus is on demonstrating the $\mathscr{L}$(ND) composite action constructs and some aspects of the actions are necessarily simplified for expository reasons.

We assume a set of elementary actions whose meaning will be apparent from their names and from the explanations below: **hover-at**, **fly-to**, **monitor-radiation**, **collect-video**, and **scan-cell**. Each elementary action is assumed to be defined in standard TAL and to provide suitable preconditions, effects, resource requirements and (completely or incompletely specified) durations. For example, only a UAV with suitable sensors can execute **monitor-radiation**.

In the following composite action, a UAV hovers at a location $(x_{\mathrm{uav}}, y_{\mathrm{uav}})$ while using its on-board sensors to monitor radiation and collect video at $(x_{\mathrm{targ}}, y_{\mathrm{targ}})$.

$[t,t']$monitor-single$(uav, x_{\mathrm{uav}}, y_{\mathrm{uav}}, x_{\mathrm{targ}}, y_{\mathrm{targ}}) \rightsquigarrow$
$[t,t']$with t_1,t_2,t_3,t_4,t_5,t_6 do (
  $[t_1,t_2]$**hover-at**$(uav, x_{\mathrm{uav}}, y_{\mathrm{uav}})$ ||
  $[t_3,t_4]$**monitor-radiation**$(uav, x_{\mathrm{targ}}, y_{\mathrm{targ}})$ ||
  $[t_5,t_6]$**collect-video**$(uav, x_{\mathrm{targ}}, y_{\mathrm{targ}})$
) where $t_1 = t_3 = t_5 = t \wedge t_2 = t_4 = t_6 = t'$

The constraints in the where clause model a requirement for these particular actions to be synchronized in time and for the UAV to hover in a stable location throughout the execution of monitor-single. These constraints could easily be relaxed, for example by stating that hovering occurs throughout the action but monitoring occurs in a sub-interval.

The following action places four UAVs in a diamond pattern to monitor a given location at a given distance, counted in grid cells. The UAVs involved are not specified as parameters, but are chosen freely among those available UAVs that are equipped for surveillance and have radiation sensors.

$[t,t']$monitor-pattern$(x,y,dist) \rightsquigarrow$
$[t,t']$with $s_1,\ldots,w_4,uav_1,uav_2,uav_3,uav_4$ do (
  ($[s_1,s_2]$**fly-to**$(uav_1, x+dist, y)$;
    $[s_3,s_4]$monitor-single$(uav_1, x+dist, y, x, y)$) ||
  ($[u_1,u_2]$**fly-to**$(uav_2, x-dist, y)$;
    $[u_3,u_4]$monitor-single$(uav_2, x-dist, y, x, y)$) ||
  ($[v_1,v_2]$**fly-to**$(uav_3, x, y+dist)$;
    $[v_3,v_4]$monitor-single$(uav_3, x, y+dist, x, y)$) ||
  ($[w_1,w_2]$**fly-to**$(uav_4, x, y-dist)$;
    $[w_3,w_4]$monitor-single$(uav_4, x, y-dist, x, y)$) ||
) where
  $[t]$surveil-equipped$(uav_1) \wedge$ has-radiation-sensors$(uav_1) \wedge$
  $[t]$surveil-equipped$(uav_2) \wedge$ has-radiation-sensors$(uav_2) \wedge$
  $[t]$surveil-equipped$(uav_3) \wedge$ has-radiation-sensors$(uav_3) \wedge$
  $[t]$surveil-equipped$(uav_4) \wedge$ has-radiation-sensors$(uav_4) \wedge$
  $s_3 = u_3 = v_3 = w_3 \wedge s_4 = u_4 = v_4 = w_4 \wedge$
  $s_4 - s_3 \geq$ minduration

Four sequences are executed in parallel. Within each sequence, a specific UAV flies to a suitable location and then

monitors the target. We require the target to be monitored simultaneously by all four UAVs ($s_3 = u_3 = v_3 = w_3$ and $s_4 = u_4 = v_4 = w_4$), while $s_4 - s_3 \geq$ minduration ensures this is done for at least the specified duration.

As flying does not need to be synchronized, the intervals for the **fly-to** actions are only constrained *implicitly* through the definition of a sequence. For example, the translation ensures that $t \leq s_1 \leq s_2 \leq s_3 \leq s_4 \leq t'$, so that each **fly-to** must end before the corresponding monitor-single.

All grid cells must also be scanned for injured people. The following generic action makes use of all available UAVs with the proper capabilities, under the assumption that each such UAV has been assigned a set of grid cells to scan. An assignment could be generated by another action or provided as part of the narrative specification.

$[t,t']$scan-with-all-uavs$() \rightsquigarrow [t,t']$with u,u' do
  foreach uav where $[t]$can-scan(uav) do conc
    $[u,u']$scan-for-people(uav)

As shown below, each UAV involved in this task iterates while there remains at least one grid cell that it has been assigned ("owns") and that is not yet scanned. In each iteration, it scans one of those cells: The "with" clause states that there exist coordinates (x,y) that are scanned, while the "where" clause ensures that the selected coordinates do belong to the given UAV and have not already been scanned. Also in each iteration, the variable t_c is bound to the time at which the condition is tested and u,u' are bound to the timepoints at which the inner action is performed.

$[t,t']$scan-for-people$(uav) \rightsquigarrow [t,t']$with u,u' do
$[u,u']$while $[t_c]\exists x,y[$owns$(uav,x,y) \wedge \neg$scanned$(x,y)]$ do
  $[u,u']$with x,y do
    $[u,u']$**scan-cell**(uav,x,y)
 where $[t_c]$owns$(uav,x,y) \wedge \neg$scanned(x,y)

Finally, we can define a small mission to occur within the interval $[0,1000]$, where scanning may utilize the entire interval while the grid cell $(20,25)$ is monitored at a distance of 3 cells between time 0 and time 300.

$[0,1000]$(
  $[0,1000]$scan-with-all-uavs$()$ ||
  $[0,300]$monitor-pattern$(20,25,3)$
)

Task Specification Trees and Robotics

The composite action extension to TAL is intended to deal with a number of representational and pragmatic issues which arise in research with complex autonomous robotics systems. In the field, use of robotic systems requires a well-specified representational formalism for specifying high-level missions which are easily understood by ground operators. Such missions often have explicit temporal deadlines and resource constraints which require the use of scheduling. Additionally, generic mission patterns often need to be constrained dynamically and incrementally relative to environmental and contingent conditions in the field. Thus, constraints offer a natural way to enhance missions. Also, one often wants to analyze mission properties both during pre- and post-mission phases. Since a composite action is a theory in $\mathscr{L}(\mathrm{FL}_{\mathrm{FP}})$, questions about missions become queries relative to an inference mechanism. Composite actions in TAL provide just such a well-specified representational formalism for each of these requirements.

Although it is essential to have an abstract, formally grounded mission specification, it is equally essential to tightly ground such specifications into the actual operations of robotic systems. Current practice in robotic systems employs the use of what are informally called 3-layered architectures (Gat 1998; Doherty et al. 2004), consisting of a control, reactive and deliberative layer. Many of these systems are reactive concentric in the sense that (reactive) tasks or behaviors consist of both calls to deliberative capabilities such as a planner or reasoner in addition to calls to control modes which interface to sensors and actuators in a robotic system. In some systems, there is very little deliberation and these task behaviors acquire the role of compiled plans which need to be executed.

In recent work with delegation-based frameworks for cooperative robotics (Doherty and Meyer 2012; Doherty, Heintz, and Landén 2011), a formal task specification language for robotics was proposed that represents tasks as Task Specification Trees (TSTs) (Doherty, Landén, and Heintz 2010; Landén, Heintz, and Doherty 2010). TSTs map directly to executable procedural representations in a robotic architecture. The UAV robotic architecture itself has a FIPA (FIPA-AAS 2002) compatible agent layer implemented using JADE (F. Bellifemine and Poggi 2005) with FIPA ACL (FIPA-ACL 2002) used as the communication protocol. Each node in a TST can be delegated to a robotic system and each node can be directly executed in such systems. Such representations have to be highly flexible, sharable, dynamically extendable during runtime and distributed in nature. Viewed in the context of composite actions in TAL, a TST may be characterized as consisting of temporal parameters + control + constraints + elementary actions. Constraint solving techniques are in fact used to determine whether a TST can be consistently executed in a single robotic system or distributed among multiple systems.

We will now provide a brief description of TSTs and show how composite actions in TAL provide a formal semantics for TSTs. The basic formal relationship is that the set of traces (models) for a composite action in TAL constrain the space of execution traces for the corresponding TST. During execution, the trace that results from the execution of the TST in the robotic system is a member of the set of legal traces (models) for the composite action in question.

Syntax. The syntax of a TST specification is as follows:

```
TST   ::= NAME '(' VARS ')') '='
          (with VARS)? TASK (where CONS)?
TSTS  ::= TST | TST ';' TSTS
TASK  ::= ACTION | GOAL | call NAME '(' ARGS ')' |
          sequence TSTS | concurrent TSTS |
          if [VAR] COND then TST else TST |
          while [VAR] COND TST |
          foreach VARS where [VAR] COND do conc TST
VAR   ::= <variable name>
VARS  ::= VAR | VAR ',' VARS
```

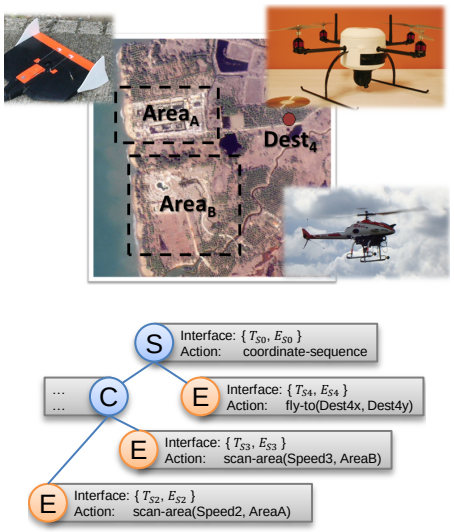

Figure 1: Example mission with corresponding TST.

ARG	::=	VAR | VALUE
ARGS	::=	ARG | ARG ',' ARGS
CONS	::=	<constraint> | <constraint> **and** CONS
VALUE	::=	<value>
NAME	::=	<node name>
COND	::=	<FIPA ACL query message requesting the value of a boolean expression>
GOAL	::=	<goal statement name($\bar{x}$)>
ACTION	::=	<elementary action call name($\bar{x}$)>

Each TST corresponds to a specific named and parameterized node. The parameters, VARS, specify a node *interface* which can be accessed from the outside. As exemplified below, the first two parameters always specify the start and end time for the node. As in TAL composite actions, these can be constrained relative to each other using constraints in a **where** clause. A separate node type exists for each different TASK. For example, a sequence node specifies sequential execution of its children.

Some node types use FIPA ACL queries, which can be TAL formulas sent to a theorem proving module such as the deductive database-based inference mechanism presented in this paper or the one in (Magnusson and Doherty 2008).

Example. Consider a small scenario where the mission is to first scan Area_A and Area_B, and then fly to Dest_4 (Figure 1). In the associated TST, nodes marked S and C are sequential and concurrent nodes, respectively, while nodes marked E are elementary action nodes. The corresponding TST specification associates each node with a task name τ_i. There are two composite actions, *sequence* (τ_0) and *concurrent* (τ_1), and three elementary actions of type **scan-area** and **fly-to**:

$\tau_0(T_{S_0},T_{E_0}) =$
with $T_{S_1},T_{E_1},T_{S_4},T_{E_4}$ **sequence**
$\tau_1(T_{S_1},T_{E_1}) =$
with $T_{S_2},T_{E_2},T_{S_3},T_{E_3}$ **concurrent**
$\tau_2(T_{S_2},T_{E_2}) =$ **scan-area**$(T_{S_2},T_{E_2},Speed_2,Area_A)$,
$\tau_3(T_{S_3},T_{E_3}) =$ **scan-area**$(T_{S_3},T_{E_3},Speed_3,Area_B)$
where $cons_{\tau_1}$,
$\tau_4(T_{S_4},T_{E_4}) =$ **fly-to**$(T_{S_4},T_{E_4},Dest_4x,\ Dest_4y)$
where $cons_{\tau_0}$

$cons_{\tau_0} = T_{S_0} \le T_{S_1} \le T_{E_1} \le T_{S_4} \le T_{E_4} \le T_{E_0}$
$cons_{\tau_1} = T_{S_1} \le T_{S_2} \le T_{E_2} \le T_{E_1}$ **and** $T_{S_1} \le T_{S_3} \le T_{E_3} \le T_{E_1}$

Translation. A translation from TAL composite actions into TSTs is specified below. In this translation, temporal constraints implicit in a composite action must be "lifted" into the nearest *enclosing* **with** clause in a TST, to make them available to a constraint solver. The *TransTST*() function therefore relies on a *TransTask*() function, corresponding closely to *TransComp*(), which returns a tuple containing both a TST TASK and a constraint.

$TransTask(\tau,\tau',[\tau_1,\tau_2]\textbf{elem}(\bar{v})) \stackrel{\text{def}}{=}$
$\langle \textbf{elem}(\tau_1,\tau_2,\bar{v}), \tau \le \tau_1 \le \tau_2 \le \tau' \rangle$
$TransTask(\tau,\tau',[\tau_1,\tau_2]\underline{\text{comp}}(\bar{v})) \stackrel{\text{def}}{=}$
$\langle \textbf{call } \underline{\text{comp}}(\tau_1,\tau_2,\bar{v}), \tau \le \tau_1 \le \tau_2 \le \tau' \rangle$
$TransTask(\tau,\tau',([\tau_1,\tau_2]A_1;[\tau_3,\tau_4]A_2)) \stackrel{\text{def}}{=}$
$\langle (\textbf{sequence } TransTST([\tau_1,\tau_2]A_1), TransTST([\tau_3,\tau_4]A_2)),$
$\tau \le \tau_1 \le \tau_2 \le \tau_3 \le \tau_4 \le \tau' \rangle$
$TransTask(\tau,\tau',([\tau_1,\tau_2]A_1 \mid\mid [\tau_3,\tau_4]A_2)) \stackrel{\text{def}}{=}$
$\langle (\textbf{concurrent } TransTST([\tau_1,\tau_2]A_1), TransTST([\tau_3,\tau_4]A_2)),$
$\tau \le \tau_1 \le \tau_2 \le \tau' \wedge \tau \le \tau_3 \le \tau_4 \le \tau' \rangle$
$TransTask(\tau,\tau',\text{if } [\tau_c]F \text{ then } [\tau_1,\tau_2]A_1 \text{ else } [\tau_3,\tau_4]A_2) \stackrel{\text{def}}{=}$
$\langle (\textbf{if } [\tau_c]F \textbf{ then } TransTST([\tau_1,\tau_2]A_1)$
$\textbf{else } TransTST([\tau_3,\tau_4]A_2)),$
$\tau \le \tau_c \le \tau' \wedge \tau_c \le \tau_1 \le \tau_2 \le \tau' \wedge \tau_c \le \tau_3 \le \tau_4 \le \tau' \rangle$
$TransTask(\tau,\tau',\text{while } [t_c]F \text{ do } [t_1,t_2]A) \stackrel{\text{def}}{=}$
$\langle \textbf{while } [t_c]F \ \ TransTST([t_1,t_2]A),$
$\tau \le t_c \le \tau' \wedge \tau \le t_1 \le t_2 \le \tau' \rangle$
$TransTask(\tau,\tau',\text{foreach } \bar{x} \text{ where } [\tau_c]F(\bar{x}) \text{ do conc } [\tau_1,\tau_2]A(\bar{x})) \stackrel{\text{def}}{=}$
$\langle \textbf{foreach } \bar{x} \textbf{ where } [\tau_c]F(\bar{x}) \textbf{ do conc } TransTST([\tau_1,\tau_2]A(\bar{x})),$
$\tau \le \tau_c \le \tau_1 \le \tau_2 \le \tau' \rangle$

TransTST() translates a full TAL C-ACT into a TST node. A composite action type specification is translated as follows, given that $TransTask(\tau,\tau',T) = \langle T',\psi \rangle$:

$TransTST([t,t']\underline{\text{comp}}(\bar{v}) \rightsquigarrow [t,t']\text{with } \bar{x} \text{ do } T \text{ where } \phi) \stackrel{\text{def}}{=}$
$\underline{\text{comp}}(t,t',\bar{v}) = \textbf{with } \bar{x} \ \ T' \ \textbf{ where } \phi \textbf{ and } \psi$

Finally, a C-ACT can also occur nested inside another composite action. Since all TST nodes must be named, a new name must be automatically generated. Then,

$TransTST([\tau,\tau']\text{with } \bar{x} \text{ do } T \text{ where } \phi) \stackrel{\text{def}}{=}$
$\text{name}(\tau,\tau') = \textbf{with } \bar{x} \ \ T' \ \textbf{ where } \phi \textbf{ and } \psi$

TSTs can be translated back into TAL composite actions in a very similar manner, with the current exception of GOAL nodes which will in the future be handled through planning (Doherty and Kvarnström 2001a; Kvarnström 2011).

Reasoning in Fixpoint TAL

We now present a proof system for TALF which is complete relative to a class of arithmetical structures. The idea is to reduce reasoning about fixpoints in TALF to classical first-order reasoning over a fixed *arithmetical* structure, modeling the application domain. We follow the line of research on relative completeness initiated in (Cook 1978) and further developed in numerous papers. The closest to the approach

presented in this section are (Harel 1978) and (Szałas 1991). Below we adjust the results of (Szałas 1991). For a sequent calculus closer to the implementation see (Szałas 1996).

Let us discuss the class of interpretations we deal with. First, we assume such interpretations contain the sort ω of natural numbers and the constants 0, 1 and functions $+$, $*$. Next, note that in applications one deals with potentially infinite data types whose elements are finitely represented: Any symbol, number, list, tree, etc., is finite. We formulate this condition as a technical assumption that for each sort there is a relation "encoding" its elements as natural numbers. Let us now formally define the considered structures.

Definition 4 A first-order interpretation I is called *strictly arithmetical* (*s-arithmetical*, in short) provided that:

- I contains the sort ω of natural numbers and the constants 0, 1 and functions $+$, $*$, with the standard interpretation;
- for each sort s of I there is an effective binary relation e_s such that for each x of sort s there is exactly one $i \in \omega$ with $e_s(x,i)$ true in I. ◁

It is worth emphasizing that TAL structures, as used in applications, are always strictly arithmetical.

Definition 5 A proof system P is *s-arithmetically sound* (*complete*), provided that for any s-arithmetical interpretation I and any formula ϕ,

$$Th_I \vdash \phi \text{ implies (is implied by) } I \models \phi, \tag{5}$$

where Th_I denotes the set of all classical first-order formulas valid in I. ◁

The above definition differs from relative completeness (Cook 1978) and arithmetical completeness (Harel 1978) in the class of admissible interpretations. In particular, the class of s-arithmetical interpretations is a proper subclass of the arithmetical interpretations of (Harel 1978).

Note that the set of classical formulas validated by an s-arithmetical interpretation is highly undecidable. On the other hand, the complexity of reasoning remains no worse than of TALF itself. The major advantage of the approach is that we reduce reasoning to classical logic by providing useful and intuitive proof rules reflecting reasoning based on invariants and backward induction. These rules are provably strongest in the considered context in the sense that they guarantee s-arithmetical completeness. In particular, they are stronger than those considered in (Magnusson and Doherty 2008). Also, in practice one does not use *all* formulas valid in I, but only some of them and perhaps not that complex. In particular, such formulas may not involve multiplication, so we frequently end up in decidable Presburger arithmetics. What is important is that when bounds on time are set so that structures become finite, we may still improve the complexity even to PTIME when formulas are suitably restricted.

Definition 6 By proof system P_{TAL} for TALF we shall mean the proof system containing the following axioms and inference rules:

- all instances of classical propositional tautologies;
- $\forall x[\phi(x)] \to \phi(t)$, where t is a term;
- the following inference rules:

$$\frac{\phi,\ \phi \to \psi}{\psi} \qquad \frac{\phi \to \psi}{\forall x\,\phi(x) \to \forall x\,\psi(x)} \tag{6}$$

$$\frac{\Gamma(\phi) \to \phi,\ \phi \to \psi}{\big(\text{LFP } X.\big[\Gamma(X)\big]\big) \to \psi} \tag{7}$$

$$\frac{\phi \to \exists n\,\psi(n),\ \psi(n+1) \to \Gamma(\psi(n)), \neg\psi(0)}{\phi \to \text{LFP } X.\big[\Gamma(X)\big]}, \tag{8}$$

where n is a variable not appearing in Γ. ◁

Adapting the results of (Szałas 1991) gives the following:

Theorem 7 The proof system P_{TAL} for TALF is s-arithmetically sound and s-arithmetically complete. ◁

Note that Definition 5 refers to a single arithmetical structure. A natural question is what happens when the theory one deals with can have more than one model. In this case, at least one considers a fixed model for arithmetics $\langle \omega, 0, +, *, = \rangle$ to represent time structure. Therefore, rather than considering in (5) the theory Th_I one considers a theory consisting of two parts:

- Th_ω – an interpreted classical first order theory of arithmetics $\langle \omega, 0, +, *, = \rangle$ formalizing the TALF time structure
- Th_{real} – the translated narrative theory

The theory $Th_\omega \cup Th_{real}$ is then to be taken as the underlying theory. Relative soundness is obviously still preserved while relative completeness may be lost. However, the proof system P_{TAL} defined in Definition 6 still remains powerful.

If the domains and timepoints are finite and bounded, as in Lemma 3, then we have the following theorem which can be proved by a direct application of Theorem 6.1 (Doherty, Łukaszewicz, and Szałas 1996).

Theorem 8 Assume that there is a finite bound on timepoints and other domains used in the narrative translation. Then the proof system obtained from P_{TAL} defined in Definition 6 by removing rule (8) and adding the axiom $\Gamma(\text{LFP } X.\big[\Gamma(X)\big]) \equiv \text{LFP } X.\big[\Gamma(X)\big]$ is sound and complete for TALF. ◁

Querying Bounded TAL Structures

Let us continue to assume that time is bounded, i.e., there is a constant $c \in \omega$ such that for any considered timepoint t we have $0 \le t \le c$, and only finitely many fluents, say n, are allowed. Then one can represent any such TAL (so TALF, too) structure as a deductive database. That is, one has:

- an *extensional database* consisting of facts (positive ground literals) of the form $Holds(t,f,v)$, $Occlude(t,f)$ and $Occurs(t_s,t_e,a)$;
- an *intensional database* consisting of relations defined by means of fixpoints, i.e., of the form

  $$R(\bar{x}) \equiv \text{LFP } X(\bar{x}).\big[\phi(X(\bar{x}))\big]$$

  such that $\bar{x}$ consists of all free variables in $\phi(X(\bar{x}))$.

For example, the formula:

$$R(t,x,y) \equiv \text{LFP } X(t,x,y).\big[Holds(t,f(x,y),True) \vee \exists z[Holds(t,f(x,z),True) \wedge X(t,z,y)]\big]$$

defines $R(t,x,y)$ to be the transitive closure of the fluent f at time t. That is, $R(t,x,y)$ holds when $f^*(x,y)$ holds, where f^* is the transitive closure of f computed at timepoint t.

Assuming that a TAL structure is given, e.g., generated by a partial expansion of a composite action, one may be interested in whether a particular elementary or composite action $\phi(\bar{x})$ is scheduled for execution within a given time interval, say $[\tau,\tau']$, i.e., whether all elementary actions have their occurrences defined and all associated constraints are satisfied. Such queries can easily be formulated by using *Trans*:

$$Trans\big([\tau,\tau']\phi(\bar{x})\big). \tag{9}$$

If there are no free variables in (9), the query returns *True* (if A is scheduled) or *False* (otherwise). If (9) contains free variables, the query returns tuples satisfying the query, such as timepoints when the action is scheduled.

It is well known that relations defined using fixpoints are computable in PTIME w.r.t. $(c+n+m)$, where m is the number of objects other than timepoints and fluents stored in the database (see, e.g., (Abiteboul, Hull, and Vianu 1996)), so both constructing such relations and querying them using first-order or fixpoint formulas is in PTIME. Moreover, such queries capture PTIME when the database domain is linearly ordered, which can safely be assumed for TAL structures.

Alternatively, one can replace fixpoint definitions by a more comfortable rule language with fixpoint semantics such as Semi-Horn Query Language, SHQL (Doherty, Łukaszewicz, and Szałas 1999). Other rule languages, like stratified DATALOG$^\neg$, can also be applied (for a survey see, e.g., (Abiteboul, Hull, and Vianu 1996)), but SHQL allows for quite immediate translations of fixpoints. Any definition of the least fixpoint, say $\text{LFP}\, X(\bar{x}).\big[\Gamma(X,\bar{x},\bar{z})\big]$, can be expressed by the following rule:

$$\Gamma(X,\bar{x},\bar{z}) \rightarrow X(\bar{x}).$$

For example, the translation of the **while** loop appearing in the definition of scan-for-people(uav) is:

$$\begin{aligned}&TransComp\big(t,t',\\ &\quad \text{while } [t_c]\exists x,y[\mathsf{owns}(uav,x,y)\wedge\neg\mathsf{scanned}(x,y)] \text{ do}\\ &\qquad [u,u']\text{with } x,y \text{ do } [u,u']\textbf{scan-cell}(uav,x,y) \text{ where}\\ &\qquad\quad [t_c]\mathsf{owns}(uav,x,y)\wedge\neg\mathsf{scanned}(x,y)\big) =\\ &t\le t'\wedge \text{LFP}\, X(u).\big[\phi\big](t),\end{aligned}$$

where ϕ is

$$\begin{aligned}&\exists v[u\le v\le t'\wedge\\ &\ (\exists x,y[Holds(v,\mathsf{owns}(uav,x,y))\wedge\neg Holds(v,\mathsf{scanned}(x,y))]\rightarrow\\ &\ \ \exists t_1,t_2[v\le t_1\le t_2\le t'\wedge\\ &\ \ \ \exists x,y[Trans\big(t_1,t_2,\textbf{scan-cell}(uav,x,y)\big)]\wedge\\ &\qquad Holds(t_c,\mathsf{owns}(uav,x,y))\wedge\\ &\qquad \neg Holds(t_c,\mathsf{scanned}(x,y))\wedge X(t_2)])]\end{aligned}$$

and $Trans\big(t_1,t_2,\textbf{scan-cell}(uav,x,y)\big)$ is the standard TAL translation of the **scan-cell** action. Therefore, the SHQL rule expressing the least fixpoint of this translation is:

$$\phi \rightarrow X(u)$$

in addition to the fact $X(t)$ expressing the application of the fixpoint to timepoint t.

Related Work

Logics of action and change have been studied extensively (Gelfond and Lifschitz 1993; Sandewall 1994; Shanahan 1997; Reiter 2001; Thielscher 2005; Doherty and Kvarnström 2008), yet there has been relatively less research which focuses specifically on composite actions in all their generality. An early precursor is the use of strategies in the situation calculus (McCarthy and Hayes 1969). A more extensive and recent study of complex actions is the Golog (Levesque et al. 1997) framework which is also based on a dialect of the situation calculus (McCarthy 1963; Reiter 2001). Customizability of Golog programs has recently been considered in the context of semantic web services (McIlraith and Son 2002). Focus in this section will be placed primarily on Golog and its extensions. Although a full comparison of this rich body of work is not possible to do within the confines of this section, a number of comparative observations can be made related to the respective logics and their semantics while saving comparisons with the pragmatics of use in robotics for a future publication.

Like earlier logics in the Features and Fluents framework (Sandewall 1994), TAL uses a macro language that can be expanded into standard formulas in a base logic. TALF leverages this approach by introducing new composite action constructs in $\mathscr{L}$(ND) and providing a translation into standard elementary actions in $\mathscr{L}(\text{FL}_{\text{FP}})$. The Golog framework uses a similar approach, where complex action expressions are macros which can be expanded into standard formulas in the situation calculus. Primitive actions are defined in the usual situation calculus manner using action preconditions, effect axioms and successor state axioms which are used to deal with the frame problem, in contrast to the TAL solution which is based on minimization of occlusion.

The approach to representing complex actions in Golog draws considerably on the intuitions from dynamic logic (Harel 1978) where primitive actions, test actions, sequences of actions, nondeterministic choice of actions, choice of action arguments, and iteration are used in their construction. The nondeterministic operators are essential in Golog and are used to define conditional and while loop expressions. TALF does not have explicit nondeterministic expressions but instead defines concurrency, conditional and while-do operators directly and with the help of fixpoints. Nondeterministic choice of action arguments is easily expressible using the with VARS do TASK where CONS construct in TALF. Additionally, procedures are also introduced in Golog to be able to call complex actions recursively. TALF uses the names in composite action type specifications and provides a recursive semantics directly in terms of fixpoints.

In the situation calculus, $do(a,s)$ returns the situation resulting from executing action a in situation s. In Golog, this is generalized for complex actions using an abbreviation, $Do(\delta,s,s')$, which states that s' is *one* possible terminating situation when the complex action δ is executed in s. Since complex actions can be non-deterministic, there may be other terminating situations and consequently, alternative execution traces. The Golog interpreter is responsible for (non-deterministically) choosing actions to construct an execution trace satisfying the background theory. An execution trace is a sequence of primitive actions which can then be passed to an execution module in a robotic or other system.

There have been many extensions to the Golog framework

that require extending the basic ontology used in the situation calculus. The main extensions of relevance to this comparison are those involving time (Reiter 1998b), durative actions (Pinto 1994; Reiter 1998b) and concurrency (Baier and Pinto 1999; Reiter 1998a). These concepts are already ontologically grounded in the original TAL formalism with no changes required for composite actions. Since actions are instantaneous in the situation calculus, one models a durative action as a process which is both started and terminated with an instantaneous action. In order to specify temporal duration explicitly, one adds a standard interpretation of the reals with appropriate operators in addition to axioms specifying the occurrence time of instantaneous actions and start time of situations. With these extensions, one can represent interleaved concurrent actions with duration and the Golog interpreter can be modified accordingly. ConGolog (Giacomo, Lespérance, and Levesque 2000) uses an interleaving view of concurrency, while TConGolog (Baier and Pinto 1999) is an extension which can model true concurrency. This requires further extensions to the ontology where sets of instantaneous actions which are restricted to starting at the same time are introduced. The cc-Golog language introduces continuously changing fluents whose values may trigger events (Grosskreutz and Lakemeyer 2003).

Sequential (or concurrent) *temporal* Golog programs require a temporal reasoning component to reason about temporal constraints. One choice is to use ECLIPSE, a logic programming language with constraint solving capability. When such a program is executed, it generally does not result in a fully instantiated sequence of actions but instead yields occurrence times representing all feasible solutions to the temporal constraints. Using a strategy of minimizing action duration in sequence, a specific schedule for the actions is generated and can be sent to a robotic system for execution. In TALF, the approach is somewhat different. Composite actions are compiled into TSTs and the generation and execution of elementary actions is done dynamically. During runtime, a TST calls a constraint solver to either check feasibility of all constraints associated with it, or to generate a specific schedule using various optimization strategies. During the execution of the TST, if any constraints are violated a temporal-logic based execution monitor (Doherty, Kvarnström, and Heintz 2009) identifies failure, and failure mitigation strategies are then employed which may involve additional calls to the constraint solver at that point in time.

In its full generality, 2nd-order logic is required to characterize the meaning of Golog programs. This is due to the need to formally characterize recursion and nondeterministic iteration. In TALF, the use of a fixpoint logic suffices for characterizing recursion and iteration. In fact, it is probably adequate for the situation calculus too, since the requirement there is to formally specify transitive closure and inductive sets. Even though second-order logic is undecidable, one can develop a sound and complete proof theory for the fixpoint logic described here.

Although the motivations for developing TALF and Golog have some overlap, the work described here is actually derived from an analysis of a deployed UAV system which uses a software architecture with TSTs as a central part. TSTs are in fact used as a basis for current work with cooperative robotics (Doherty, Heintz, and Landén 2011; Doherty et al. 2010). We consider the back and forth incremental development of both robotic systems and their formal correlates to be an appealing methodology which contributes to both formal and pragmatic development of robotic systems in a highly productive and novel manner.

Conclusions

The focus of this paper has been to formally characterize temporal composite actions with constraints in TAL by using a fixpoint semantics in addition to providing a sound and complete proof theory for these additions. Additionally, through the use of TSTs, the pragmatic relation between a logical specification language for composite actions and an executable representation for robotic tasks was presented. Future work will include additional development of the cooperative robotic framework with appropriate extensions to the logical specification language. Additionally, effort will be placed on generalization of our planners for composite tasks and their tighter integration with the TST-based software architecture and execution monitoring systems.

Acknowledgments

This work is partially supported by grants from the Swedish Research Council (VR) Linnaeus Center CADICS, VR grant 2009-3857, the ELLIIT Excellence Center at Linköping-Lund in Information Technology, the Swedish National Aviation Engineering Research Program NFFP5, the LinkLab center for future aviation systems, and SSF, the Swedish Foundation for Strategic Research.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1996. *Foundations of Databases*. Addison-Wesley.

Arnold, A., and Niwiński, D. 2001. *Rudiments of μ-calculus*. Studies in logic and the foundations of mathematics.

Baier, J., and Pinto, J. 1999. Integrating true concurrency into the robot programming language GOLOG. In *Proc. Intl. Conf. of the Chilean Computer Science Society*, 179–186.

Cook, S. 1978. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing* 7(1):70–90.

Doherty, P., and Kvarnström, J. 2001a. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence* 30:119–169.

Doherty, P., and Kvarnström, J. 2001b. TALplanner: A temporal logic-based planner. *AI Magazine* 22(3):95–102.

Doherty, P., and Kvarnström, J. 2008. Temporal action logics. In Lifschitz, V.; van Harmelen, F.; and Porter, F., eds., *The Handbook of Knowledge Representation*. Elsevier.

Doherty, P., and Meyer, J.-J. C. 2012. On the logic of delegation: Relating theory and practice. In *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*.

Doherty, P., and Rudol, P. 2007. A UAV search and rescue

scenario with human body detection and geolocalization. In *Proc. Australian Joint Conference on Artificial Intelligence*.

Doherty, P.; Haslum, P.; Heintz, F.; Merz, T.; Nyblom, P.; Persson, T.; and Wingman, B. 2004. A distributed architecture for autonomous unmanned aerial vehicle experimentation. In *Proceedings of DARS*.

Doherty, P.; Kvarnström, J.; Heintz, F.; Landén, D.; and Olsson, P.-M. 2010. Research with collaborative unmanned aircraft systems. In *Proc. Dagstuhl WS on Cognitive Robotics*.

Doherty, P.; Heintz, F.; and Landén, D. 2011. *Agent-Oriented Software Engineering XI: Revised Selected Papers*, volume 6788. Springer-Verlag. chapter A Delegation-Based Architecture for Collaborative Robotics, 205–247.

Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Journal of Autonomous Agents and Multi-Agent Systems* 19(3):332–377.

Doherty, P.; Landén, D.; and Heintz, F. 2010. A distributed task specification language for mixed-initiative delegation. In *Proceedings of PRIMA*.

Doherty, P.; Łukaszewicz, W.; and Szałas, A. 1996. A reduction result for circumscribed semi-Horn formulas. *Fundamenta Informaticae* 28(3-4):261–271.

Doherty, P.; Łukaszewicz, W.; and Szałas, A. 1999. Declarative PTIME queries for relational databases using quantifier elimination. *J. of Logic and Computation* 9(5):739–761.

F. Bellifemine, F. Bergenti, G. C., and Poggi, A. 2005. JADE – a Java agent development framework. In *Multi-Agent Programming – Languages, Platforms and Applications*. Springer.

FIPA-AAS. 2002. *FIPA Abstract Architecture Specification*. Foundation for Intelligent Physical Agents.

FIPA-ACL. 2002. *FIPA Communicative Act Library Specification*. Foundation for Intelligent Physical Agents.

Gat, E. 1998. Three-layer architectures. In D. Kortenkamp, R. P. B., and Murphy, R., eds., *Artificial Intelligence and Mobile Robots*. MIT Press. chapter 8.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17:301–321.

Giacomo, G. D.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1-2):109.

Grosskreutz, H., and Lakemeyer, G. 2003. cc-Golog – An action language with continuous change. *Logic Journal of IGPL* 11(2):179–221.

Harel, D. 1978. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer-Verlag.

Kvarnström, J. 2011. Planning for loosely coupled agents using partial order forward-chaining. In *Proc. ICAPS*.

Landén, D.; Heintz, F.; and Doherty, P. 2010. Complex task allocation in mixed-initiative delegation: A UAV case study (early innovation). In *Proceedings of PRIMA*.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3):59 – 83.

Lifschitz, V. 1991. Circumscription. In Gabbay, D. M.; Hogger, C. J.; and Robinson, J. A., eds., *Handbook of Artificial Intelligence and Logic Programming*, volume 3. Oxford University Press. 297–352.

Magnusson, M., and Doherty, P. 2008. Deductive planning with inductive loops. In *Proc. KR*, 528–534.

McCarthy, J., and Hayes, P. 1969. *Some philosophical problems from the standpoint of artificial intelligence*. Number 4 in Machine Intelligence. Edinburgh U. Press. 463–502.

McCarthy, J. 1963. Situations and actions and causal laws. Technical report, Stanford University, CA.

McIlraith, S., and Son, T. C. 2002. Adapting Golog for composition of web services. In *Proc. KR*.

Pinto, J. 1994. *Temporal Reasoning in the Situation Calculus*. Ph.D. Dissertation, University of Toronto.

Reiter, R. 1998a. Natural actions, concurrency and continuous time in the situation calculus. In *Proc. KR*.

Reiter, R. 1998b. Sequential, temporal Golog. In *Proc. KR*.

Reiter, R. 2001. *Knowledge in Action – Logical Foundations for specifying and Implementing Dynamical Systems*. MIT Press.

Rudol, P., and Doherty, P. 2008. Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery. In *Proc. IEEE Aerospace Conf.*

Sandewall, E. 1994. *Features and Fluents. The Representation of Knowledge about Dynamical Systems*. Oxford University Press.

Shanahan, M. 1997. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia.* MIT Press.

Szałas, A. 1991. On strictly arithmetical completeness in logics of programs. *Theoretical Computer Science* 79:341.

Szałas, A. 1996. On natural deduction in first-order fixpoint logics. *Fundamenta Informaticae* 26:81–94.

Thielscher, M. 2005. *Reasoning Robots - The Art and Science of Programming Robotic Agents*. Springer.