Temporal Action Language (TAL): A Controlled Language for Consistency Checking of Natural Language Temporal Requirements
(Preliminary results)

Wenbin Li, Jane Huffman Hayes, and Mirosław Truszczyński

University of Kentucky, USA
wenbin.li@uky.edu hayes,mirek@cs.uky.edu

Abstract. We introduce Temporal Action Language (*TAL*). We design *TAL* as a key component of our approach that aims to semi-automate the process of consistency checking of natural language temporal requirements. Analysts can use *TAL* to express temporal requirements precisely and unambiguously. We describe the syntax and semantics of *TAL* and illustrate how to use *TAL* to represent temporal requirements.

1 Introduction

Requirements such as "*a node should re-identify itself within 10 seconds after making a connection to the server or the server will drop the connection in 2 seconds*" describe temporal dependencies among events. Such *temporal* requirements are common in software projects. Temporal requirements may be inconsistent. Detecting inconsistencies of temporal requirements is essential and should take place before the design phase so that the cost of revisions can be minimized. Automating or partially automating the process is crucial as the task, when performed manually, is time consuming and error-prone.

There has been much research on formal methods for automating the process of requirements analysis [7, 9, 8, 4]. Analyzing temporal constraints automatically requires that they be expressed in a low-level formal language for which good automated reasoning tools are available (such as temporal logic [13] and timed automata [1], which have been used with success to analyze real-time systems [2, 12]). Researchers typically assume that formal representations are already given and focus on methods and tools for analyzing them. Our approach is different as it assumes the requirements are stated in *natural language*, and so addresses the needs of the most typical scenario when software requirements are given as a free-flow narrative (cf. the example above).

Our main contribution is a controlled language called *Temporal Action Language* (*TAL*). We propose it as a key component of a process aiming to minimize the time and effort required to check the consistency of temporal requirements specified in natural language. Translating such requirements faithfully into low-level formal languages is difficult due to the significant "distance" between natural language text and formal expressions in logic, ambiguity common in natural

language descriptions of requirements, importance of implicit information, and insufficient formal method background of analysts. We introduce *TAL* as a bridge between natural language and the low-level target language used for reasoning. This naturally leads to a two-stage process: (1) creating a *TAL* theory that describes the system and (2) detecting conflicts in the *TAL* theory. Each stage can be further decomposed into multiple manageable tasks.

The first stage requires identifying temporal requirements; gathering domain information; making relevant, shared (or commonsense) knowledge explicit; removing ambiguity in requirements; and expressing them in *TAL*. Analyst involvement in the first stage will be necessary. However, we believe natural language processing (NLP) and information retrieval (IR) techniques can effectively assist analysts in the task. The second stage consists of translating *TAL* theory into low-level logic formalism and using its tools to reason about the *TAL* theory. That stage can be fully automated.

We want to use *TAL* as an effective bridge between natural language and a low-level logic, and we designed *TAL* with the following desiderata in mind. First, the syntax of *TAL* must be close to that of natural language because the readability of *TAL* is crucial to the effectiveness and efficiency of the first stage. High readability significantly reduces the time and effort required to verify and validate the *TAL* theories generated in this stage. Second, theories in *TAL* must have a well-defined semantics so that correct automated translations of *TAL* theories into target languages are possible. Third, we want *TAL* to be capable of specifying the temporal constraints that people may find in software requirements. Specifically, we want *TAL* to model the prerequisites and effects of actions, and the time bounds on which actions start and end. Although the overall approach is still under development, we have collected anecdotal evidence suggesting readability of *TAL* theories as well as feasibility of automating translations of *TAL* theories into formal systems.

2 Temporal Action Language *TAL*

Syntax. We use *TAL* to specify temporal constraints on times when events occur. Such events include the start and end of actions and the change of system properties (fluents). We design *TAL* as an extension of *Action Language AL* [3] which allows us to specify actions and fluents, but not temporal information.

A *TAL* theory is a triple (AD,IC,TC) where AD is the set of *action definitions*, IC is the set of *initial constraints*, and TC is the set of *temporal constraints*. The *action definitions* describe the actions by specifying their prerequisites and effects, all expressed as *fluents* (boolean properties). The syntax of AD is that of *AL*. In particular, AD consists of expressions of the following form:

State constraint	L **if** P	(1)
Dynamic causal law	A **causes** L **if** P	(2)
Executability condition	**impossible** $A_1, \ldots, A_k$ **if** P	(3)

where L and P are lists of fluents and their negations, and $A, A_1, \ldots, A_k$ are actions. State constraint (1) says that L holds (every fluent and the negation of a fluent in L holds) in every state in which P holds (in the same sense as L). Dynamic causal law (2) describes the effects of actions. Executability condition (3) specifies the prerequisites of actions. For example:

connect(*serA*, *nodeA*) **causes** *connected*(*nodeA*, *serA*) **if** *systemOn*

says that executing the action *connect*(*serA*, *nodeA*) when the system is on results in *nodeA* and *serA* being connected, and

impossible *identify*(*nodeA*, *serA*) **if** ¬ *connected*(*serA*, *nodeA*)

specifies the prerequisite of the action *identify*(*nodeA*, *serA*).

The second component of a *TAL* theory, *IC*, consists of *initial constraints* defining the initial state of the system. An *initial constraint* is an expression: **initially** L , where L is a fluent.

The presence of the component *TC* in a *TAL* theory is the key feature that distinguishes *TAL* from *AL*. *TC* specifies temporal information including *temporal constraints* and action durations.

A *duration specification* is an expression: **duration** *Act x unit*, where *Act* is an action, x is a positive number, and *units* refers to a time unit such as a millisecond, second, or minute.

Temporal constraints describe temporal relationships among the times when events occur. *Temporal conditions* are the basic component of temporal constraints. A temporal condition models the temporal relationship between the occurrence time of two events. In *TAL*, each action *Act* is associated with two *prompts*: **commence** *Act* and **terminate** *Act*, which represent starting and successfully finishing action *Act*. In *TAL* one can relate two consecutive occurrences of the same action to each other. To distinguish between them, *TAL* provides the keywords **previous** and **next**. A temporal condition is of the form:

⟨*timeReference*⟩ @ ⟨*timeComparator*⟩ [⟨*timeModifier*⟩] ⟨*timeReference*⟩

The expression ⟨*timeReference*⟩ represents the occurrence time of the event. That time can be **startTime** (the time when the system starts), *prompt* (the time when the prompt occurs), a fluent, or its negation (the time when the fluent starts or ceases being true). The expression ⟨*timeComparator*⟩ [⟨*timeModifier*⟩] specifies the temporal relationship between these two time moments. In *TAL*, we use $<$, $\leq$, $=$, $\geq$, or $>$ for ⟨*timeComparator*⟩. The parameter *timeModifier* is optional. It modifies the time t given by the second *timeReference* expression as in "*x seconds* **before** *t*" or "*x milliseconds* **after** *t*," where $x > 0$. For example, in *TAL*, the temporal condition "*serA drops the connection to nodeB 5 seconds after it establishes a connection to nodeA*" can be written as:

commence *dropConn*(*serA*, *nodeB*) @ $= 5$ *seconds* **after**
terminate *estConn*(*serA*, *nodeA*)

The basic form of a *temporal constraint* is:

if A_1 **and** $\ldots$ **and** A_k, **then** B_1 **or** $\ldots$ **or** B_m;

where $A_1, \ldots, A_k$ and $B_1, \ldots, B_m$ are *temporal conditions* or their negations (temporal conditions can be viewed as special temporal constraints with $k = 0$ and $m = 1$). In *TAL*, one can express "*if a connected node does not re-identify itself to the server within 10 seconds after the connection is established, the server shall drop the connection within 2 seconds*" as:

if not terminate *identify*$(nodeA, serA)$ @ ≤ 10 *seconds* **after**
terminate *estConn*$(serA, nodeA)$,
then terminate *dropConn*$(serA, nodeA)$ @ ≤ 2 *seconds*;

Semantics. We base the semantics of a *TAL* theory (AD, IC, TC) on a transition system T_{AD} of the action description component AD. The use of transition systems as the semantics of action language theories was proposed by Gelfond and Lifschitz [6]. That approach applies also to AL [3]. In an AL transition system, states are combinations of fluents and their negations. Arcs between states are labeled with actions because AL assumes that only actions can cause the system to change its state. Since *TAL*'s action description AD is in the syntax of AL, we create the transition system T_{AD} essentially in the same way as in AL but with two modifications. First, the arcs in T_{AD} are labeled with prompts. This is because the prerequisites and effects of actions specified in AD can be viewed as prerequisites of the corresponding **commence** prompts and effects of the corresponding **terminate** prompts. Second, some arcs are labeled with the term *time* as some fluents in *TAL* can change value simply because of time passing (for instance, a message becomes "old" if it is in the queue for more than 20 seconds – no action is required for that).

A sequence $\langle s_0, pr_0, s_1, pr_1, \ldots, s_{x-1}, pr_{x-1}, s_x \rangle$ is a *path* in a transition system T_{AD} if all s_i are states, all pr_i are prompts or *time*, s_0 satisfies all *initial constraints*, and if for each $i = 0, \ldots, x-1$, $\langle s_i, pr_i, s_{i+1} \rangle$ is a transition in T_{AD}. A path in T_{AD} represents a scenario, the evolution of the state of the corresponding system as the result of prompts (time) labeling the arcs, assuming we disregard action durations and temporal constraints.

A path does not show when the events occur. We define a *timed path* as a sequence $\langle s_0, pr_0, t_0, s_1, pr_1, t_1, \ldots, s_{x-1}, pr_{x-1}, t_{x-1}, s_x \rangle$, where $\langle s_0, pr_0, s_1, pr_1, \ldots, s_{x-1}, pr_{x-1}, s_x \rangle$ is a path and for every $i = 0, \ldots, x-1$, $t_i < t_{i+1}$. The times t_i are the times when the system is to progress from state s_i to s_{i+1}. The question of consistency of temporal requirements is that of the existence of arbitrarily long timed paths satisfying all temporal constraints in TC.

Let p be a timed path and t a time in the time range of p (not greater than the time of the last state change). It is straightforward, albeit tedious, to specify when B holds at time t on p. For instance, let B stand for *prompt*1 @ $=$ x seconds **after** *prompt*2. If the most recent occurrence of *prompt*1 before or at t is at time t' and at time $t' + x$ there is an occurrence of *prompt*2, then we say that the condition holds at B. There are several such cases to cover. We omit details due to space limits. Next, we define a temporal constraint to be satisfied on p at time t if at least one temporal condition in the consequent of

C evaluates to true whenever all temporal conditions in the antecedent evaluate to true (interpreting t as "now"). Finally, we say that a temporal constraint C holds on a path p if C holds at every time t within the range of p.

Consistency Checking. Consistency of a *TAL* theory means the existence of arbitrarily long timed paths. It guarantees that there is no inconsistency in temporal requirements. A weaker notion of *bounded consistency* means the existence of a timed path with a given bound on its time range. It guarantees that no inconsistency in temporal requirements can exhibit itself prior to the bound. The larger the bound, the more accurately the notion approximates that of consistency. Other interesting questions are whether an event can (or will) occur within a given time bound, or whether a system can (will) satisfy a certain property while running. Since *TAL* is a formal system, a promising approach to decide (bounded) consistency and related questions is to develop translations to low-level logic systems and use automated reasoning tools that are available for them.

3 Validation

To date, we studied bounded consistency and experimented with translations of the *TAL* representation of the problem of existence of a timed path of bounded length into answer-set programming (ASP) [10, 11]. We selected ASP because it is well suited for modeling search problems and has fast solvers [5]. We created an example scenario with multiple temporal requirements and represented it as a *TAL* theory. We manually translated the *TAL* theory into an answer-set program. We used a solver, *clingcon* [5], to process it. We found that when the requirements were consistent, *clingcon* returned at least one answer set that represents a valid scenario. Upon modifying the requirements to make them inconsistent, *clingcon* did not return any timed paths. The experiment shows the feasibility of reasoning about consistency of temporal requirements given in *TAL* by translating them to low-level target logics and then using automated reasoning tools.

We also wrote sixteen natural language temporal requirements and their corresponding *TAL* statements. We selected four people with various computer science backgrounds, from working in industry for years as a requirements engineer to having a bachelor's degree in computer science. We briefly introduced the syntax of *TAL* to them and asked them to rate the similarity in meaning of the natural language statements and their formal *TAL* representations (we used a scale from 0, completely different, to 5, exactly the same). The mean rating for the sixteen pairs was 4.76. The result shows that the participants found *TAL* statements to be unambiguous and easy to understand.

4 Discussion and Future Work

This paper presents the language *TAL* for specifying temporal requirements. It extends *AL* by allowing users to specify temporal dependencies among events

using *temporal conditions* and *constraints*. The syntax of *TAL* is close to natural language and, based on anecdotal evidence, easy to follow. The semantics are based on the concepts of transition systems and timed paths. Checking for (in)consistency of temporal requirements is reduced to creating *TAL* expressions from natural language requirements since, once the *TAL* representation is available, it can be processed in a fully automated way.

Future work includes automating the translation from *TAL* to ASP and creating tools based on natural language processing and information retrieval to assist analysts in generating *TAL* theory based on requirements given in natural language. We will also study temporal logics and timed automata as possible target formalisms. Finally, we will perform systematic experiments to validate the scope and feasibility of the approach.

Acknowledgment

This work is funded in part by the National Science Foundation under NSF grant CCF-0811140 and JPL grant 1401954.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
3. Baral, C., Gelfond, M.: Reasoning agents in dynamic domains. Logic-based artificial intelligence pp. 257–279 (2000)
4. Dutertre, B., Stavridou, V.: Formal requirements analysis of an avionics control system. IEEE Transactions on Software Engineering SE 23, 267–278 (1997)
5. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. International Conference on Logic Programming (ICLP) pp. 235–249 (2009)
6. Gelfond, M., Lifschitz, V.: Action languages. Electronic Transactions on Artificial Intelligence (ETAI) 2, 193–210 (1998)
7. Heitmeyer, C.: Software cost reduction. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering. John Wiley & Sons, 2nd edn. (2002)
8. Klein, M.: An exception handling approach to enhancing consistency, completeness and correctness in collaborative requirements capture. Concurrent Engineering Research and Applications 5, 37–46 (1997)
9. Lamsweerde, A.V., Darimont, R., Letier, E.: Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering 24(11), 908–926 (1998)
10. Marek, V., Truszczynski, M.: Stable models and an alternative logic programming paradigm. The Logic Programming Paradigm: a 25-Year Perspective pp. 375–398 (1999)
11. Niemela, I.: Logic programs with stable model semantics as a constraint paradigm. Annals of Mathematics and Artificial Intelligence 25, 241–273 (1999)
12. Olderog, E.R., Dierks, H.: Real-Time Systems. CUP (2008)
13. Pnueli, A.: The temporal logic of programs. Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS) pp. 46–57 (1977)