Answer-Set Programming in Requirements Engineering

Wenbin Li, David Brown, Jane Huffman Hayes, and Miroslaw Truszczynski

Department of Computer Science, University of Kentucky, Lexington, KY 40506-0633, USA
{wenbin.li,david.b.brown}@uky.edu, {hayes,mirek}@cs.uky.edu

Abstract. [Context and motivation] Requirements form the foundation of software systems. The quality of the requirements influences the quality of the developed software. **[Question/problem]** One of the main requirement issues is inconsistency, particularly onerous when the requirements concern temporal constraints. Manual checking whether temporal requirements are consistent is tedious and error prone and may be prohibitively expensive when the number of requirements is large. **[Principal ideas/results]** We show that answer-set programming tools (ASP) can be successfully applied to detect inconsistencies in software and system requirements. Our assumption is that these requirements are given in a formal requirement specification language called Temporal Action Language ($TeAL$). **[Contribution]** We present a translation from $TeAL$ to the ASP language format accepted by *clingcon*. We show that *clingcon* can analyze requirements for several real software systems, verifying their consistency or identifying inconsistencies. We also examine the performance of the clingcon translation.

Keywords: temporal requirements, requirement engineering, knowledge representation.

1 Introduction

It is well documented in the software engineering literature that software malfunction can frequently be traced back to problems with software or system requirements [10,6,9]. The analysis of requirements for ambiguity, inconsistency, or incompleteness, if performed manually, is labor intensive, tedious, and error-prone. Indeed, a specification of a system may contain so many requirements that it is simply not feasible to check them manually.

We focus our work on consistency checking of temporal requirements. As many software systems support real-time operations, temporal requirements are common. For instance, a mission-critical financial trading system requires that certain transactions occur within a certain amount of time of other transactions (such as posting the proceeds of a stock sale or logging realized dividend payments); an e-commerce system requires that a payment be received a specified time prior to submitting an order for processing; a safety-critical pacemaker system requires that pacing occur within milliseconds of certain detected events. Moreover, as these examples implicitly suggest, high quality of temporal requirements is essential. Errors in specifying, interpreting, or implementing temporal requirements can lead to disastrous consequences. If one or more requirements

C. Salinesi and I. van de Weerd (Eds.): REFSQ 2014, LNCS 8396, pp. 168–183, 2014.

related to the pacing of the heart are in conflict, a negative heart event might not trigger a necessary lifesaving pacing event.

In this paper we show that answer-set programming (ASP), with its program processing tools, can play an important role in the analysis of temporal requirements. The system we are designing addresses the problem in two main steps. First, textual requirements are analyzed to identify temporal requirements and additional relevant information. These are then translated into a high-level temporal requirement representation language called Temporal Action Language or $TeAL$ [15]. $TeAL$ extends the action language AL [5] with dedicated, intuitive syntax to capture temporal constraints in a way that reflects common linguistic patterns. The language was designed to help requirement engineers build correct $TeAL$ representation of the original textual requirements, as well as support partial automation of that step [14]. Second, $TeAL$ theory is translated to the ASP language accepted by the ASP solver *clingcon* [12]. This variant ASP language provides syntax for stating integer constraints that are common in temporal constraints present in software requirements, and *clingcon* is a state-of-the-art solver designed specifically for that language.

With the use of *clingcon*, the consistency of the $TeAL$ theory (and effectively, of the original textual requirements) is verified. This step is the focus of the present paper. In the main contributions, we provide the details of the translation from $TeAL$ to *clingcon* and show its correctness. Practitioners may wonder if formal methods can be applied to non-trivial systems in a timely manner. To address their concerns, we demonstrate the effectiveness of *clingcon* in analyzing several example requirement sets from real software systems and examine the performance of the translation to clingcon.

The paper is organized as follows. In the next section, we present several benchmark requirement documents. They will be used to illustrate our approach. Section 3 provides a brief overview of the language $TeAL$ [14] for representing system specifications. Next, we discuss the translation from $TeAL$ to *clingcon*. Section 6 discusses $TeAL$ representations of benchmark requirement documents and their translations into ASP (*clingcon* input language). That section also presents the results of our benchmark example studies. Our findings are discussed in the last section of the paper, where we also present conclusions and problems for future work.

2 Benchmark Examples of System Requirements

Throughout the paper we will refer to several benchmark examples of requirement documents specifying (fragments of) real software systems. As we are interested in the analysis of temporal requirements, in some cases we modified these examples from their original form by varying durations of actions and including additional temporal constraints. Our objective was to better illustrate both the current functionality of $TeAL$ as well as all aspects of the translation from $TeAL$ to ASP. We describe one of the examples, **CM1**, in detail. We outline the others and provide just one sample requirement (full descriptions at `http://progit.netlab.uky.edu/teal`).

CM1. This example is derived from a requirement document produced by NASA for one of its science instruments. The document was "sanitized" (hence the presence of variables rather than specific constants) and released by NASA for use by the software engineering research community [1].

The Control Component shall send the heart beat message to the Interface of Instrument Control Unit at an interval of E milliseconds. The interface will send the message to the Instrument Control Unit. The Control Component shall process commands within F milliseconds of receipt from the Interface of Instrument Control Unit or the Spacecraft Control Unit. The Instrument Control Unit shall send real-time commands to the Interface of Control Component every B milliseconds. Whenever the Interface of Instrument Control Unit receives a message from the Instrument Control Unit, it verifies the message within J milliseconds. If an error is detected, the message is discarded within K milliseconds, then an error report will be sent to the Control Component, and a NAK message transmitted to the Instrument Control Unit within L milliseconds. If the message is correct, the Interface of Instrument Control Unit shall forward real-time commands to the Control Component within C units of receipt from the Instrument Control Unit.

511Phone. This example is derived from a requirement document for the *Regional Real Time Transit Information System*. These requirements focus on the performance of the 511 System and the data transfers with the transit agencies. They are based on the existing procedures and features of the existing real-time system. [4].

If request*, then transit agency system sends predictions and vehicle location within var1 seconds after receiving data request from the 511 System.*

MODIS. This example is derived from the open source NASA Moderate Resolution Imaging Spectroradiometer (MODIS) documents [2].

Each MODIS standard input data shall be produced every var1 seconds.

UAVTCS. This example is derived from a requirement document for an Unmanned Aerial Vehicle (UAV) Tactical Control System (UAVTCS) of the US Department of Defense for the control of tactical UAVs. [3]

The TCS in the Normal Startup Mode shall initialize the system to the Operation State within 60 seconds from the time power is supplied.

EasyClinic. This dataset describes a variety of artifacts from a small healthcare application. It was developed at the University of Salerno to manage a medical ambulatory [7].

The response time of the service shall be less than A seconds.

iTrust. This dataset is derived from the iTrust project which involves the development of an application through which doctors can obtain and share essential patient information and can view aggregate patient data [7].

An HCP can reassign a previously created lab procedure to a different Lab Technician if the lab procedure is not yet in time.

3 TeAL Overview

$TeAL$ is an extension of the action language AL [5] with expressions to represent temporal constraints. One of our goals in this paper is to demonstrate the feasibility of using ASP tools to address the problem of testing temporal constraints for consistency. The choice of AL is motivated by the availability of effective translations of AL theories into ASP.

The $TeAL$ syntax of temporal constraints follows common linguistic patterns to help analysts construct and revise $TeAL$ theories representing textual requirements. We provide here a brief overview of $TeAL$'s syntax and semantics to facilitate understanding of the main results of the paper concerning the translation from $TeAL$ to ASP. For a detailed description of $TeAL$, we refer to our earlier work [15,14].

Syntax. A $TeAL$ theory is a quadruple $\Delta = \langle SI, AD, TC, IC \rangle$, where SI is the signature, AD is an *action language* (AL) theory [5], TC is the set of *temporal constraints*, and IC is the set of *initial state constraints*. The signature SI contains the names for sorts (for instance, *data* and *agency* in the $TeAL$ representation of the 511Phone example discussed below), the constants that are assigned to sorts (*d1* and *a1* in the same example), and the names of *fluents* and *actions*. As usual, fluents represent atomic (boolean) properties of the system. Complete and consistent sets of (possibly negated) fluents describe the state of the system. For example, the fluent *received(p511,d1,a1)* represents the property that the data item *d1* has been received by the phone system *p511*. Actions change fluents and, consequently, the state of the system. Actions are performed by agents. For example, *send(a1,d1,p511)* represents an action performed by the agent *a1* to send the data item *d1* to the phone system *p511*.

The role of an AL theory AD is to specify the *action domain*, that is, fluents and actions (their preconditions and effects but *not* durations). Namely, AD uses *state constraints* (to specify conditions that must hold in every state), *dynamic causal laws* (to describe the effects of an action when performed in a state), and *executability conditions* (to specify preconditions for an action to be executable). The action language is well known and we do not discuss it in any more detail here.

The component TC in a $TeAL$ theory Δ distinguishes $TeAL$ from AL. It specifies action durations and *temporal constraints* on actions. To refer to time we use a special term $\mathbf{startTime}$ that represents the *initial* time moment with respect to which we interpret Δ. We also refer to time indirectly by means of the *prompts* $\mathbf{commence}\ a$ and $\mathbf{terminate}\ a$ that stand for the times when action a starts and ends, respectively. At present, we assume that once actions are started they terminate successfully. For example, $\mathbf{commence}\ update(p511, d1)$ gives the time when the action $update(p511, d1)$ was initiated. The modifiers $\mathbf{previous}$ and $\mathbf{next}$ can be used with prompts to identify the time moments when the previous (next) prompt occurred. For example, to specify the time when the most recent $update(p511, d1)$ was initiated we may use the expression $\mathbf{commence\ previous}\ update(p511, d1)$. In the present version of $TeAL$, the keywords $\mathbf{previous}$ and $\mathbf{next}$ cannot be nested.

A fluent appearing in a temporal condition represents the time when this fluent has become true. Similarly, the negation of a fluent in a temporal condition represents the time when this fluent has become false [14]. A fluent can change from true to false (or conversely) only because of actions or passage of time. The specification "*a file becomes old if it has not been written to for 10 seconds*" involves the fluent "*old*" (a property of files) that becomes true just because of the passage of time. The passage of time is handled by two special prompts *totrue(fluent)* and *tofalse(fluent)*. We will not discuss this in detail because of space.

Time moments represented by prompts and fluents are connected by temporal relationships *before*, *after*, and *at the same time as* that can also be annotated with specified

quantities of time. $TeAL$ provides several keyword phrases to allow the user to express these relationships. For example, the temporal constraint

$$\textbf{commence}\ update(p511, d1)$$
$$\textbf{noLaterThan}\ 10\ seconds\ \textbf{after}\ received(p511, d1, a1)$$

encodes the constraint: *the phone system starts to update the data within* 10 *seconds after receiving the data*.

These basic temporal constraints, called *temporal conditions in* $TeAL$, can be combined by boolean connectives into more complex ones of the form:

$$\textbf{if}\ \ A_1\ \&\ \ldots\ \&\ A_k, \textbf{then}\ B_1 \mid \ldots \mid B_m \tag{1}$$

where $A_1, \ldots, A_k$ and $B_1, \ldots, B_m$ are temporal conditions or their negations (represented by **not**) and | stands for "or."

$TeAL$ also provides a dedicated syntax to specify the durations of actions:

$$\textbf{duration}\ a\ d\ units$$

where a is an action, d is a positive integer, and $units$ is a time unit. $TeAL$ allows multiple time units, such as *minutes* and *seconds*, but all time units are converted to the smallest unit during the translation.

The fourth part of Δ, IC, defines *constraints* on the initial state. Initial state constraints are of the form:

$$\textbf{initially}\ F \tag{2}$$

where F is a list (conjunction) of fluent literals (intuitively, that must hold in any initial state).

Below, we show a $TeAL$ representation of the 511Phone requirement document, shortened due to space limitations. It starts with declarations of sorts, constants, agents, fluents, and actions, including action durations. Next, it specifies the initial conditions as well as the effects and preconditions of actions. Lastly, it specifies temporal constraints (each preceded by the textual constraint it represents).

$\textbf{sort}\ agency;$
$\textbf{sort}\ p511;$
$\textbf{sort}\ data;$
$\textbf{constant}\ agency\ a1;$
$\textbf{constant}\ p511\ phone;$
$\textbf{constant}\ data\ d1;$
$\textbf{agent}\ agency, p511;$

$\textbf{fluent}\ received(p511, data, agency);$
$\textbf{fluent}\ available(data);$
$\textbf{action}\ send(agency, data, p511);$
$\textbf{action}\ update(p511, data);$
$\textbf{duration}\ send(a1, d1, p511)\ 1\ second;$
$\textbf{duration}\ update(p511, d1)\ 1\ second;$

$\textbf{initially}\ available(d1);$
$update(p511, d1)\ \textbf{causes}\ available(d1);$
$\textbf{impossible}\ update(p511, d1)\ \textbf{if}\ \textbf{not}\ received(p511, d1, a1);$

Once the data is sent, it will be received in three seconds.
if terminate $send(a1, d1, p511)$
then $received(p511, d1, a1)$ **noLaterThan** 3 $seconds$ **after**;

The agency shall send data within 60 *seconds after the system starts.*
commence $send(a1, d1, p511)$ **noLaterThan** 60 $seconds$ **after startTime**;

The agency shall send data at least once every 60 *seconds.*
commence $send(a1, d1, p511)$ **noLaterThan** 60 $seconds$
after terminate previous $send(a1, d1, p511)$;

The phone system shall update the data within 10 *seconds after receiving the data.*
commence $update(p511, d1)$
noLaterThan 10 $seconds$ **after** $received(p511, d1, a1)$;

If a piece of data is not updated for 60 *seconds, it shall become unavailable to users.*
if not terminate $update(p511, d1)$
noEarlierThan 10 $seconds$ **before then not** $available(d1)$;

Semantics. We now discuss the semantics of a $TeAL$ theory $\Delta = \langle SI, AD, TC, IC \rangle$. It is largely based on the semantics of AL theories [5]. The key notion is that of a transition system, which is defined based on the AL theory AD. We will denote it by T_Δ. Following Baral and Gelfond [5], we define a *path* of Δ to be a sequence

$$\langle s_0, pr_0; s_1, pr_1; \ldots; s_{k-1}, pr_{k-1}; s_k \rangle$$

such that $s_0 \ldots, s_k$ are states; $pr_0, \ldots, pr_{k-1}$ are sets of prompts;[1] for every expression **initially** F in IC, the state s_0 satisfies F; and for each $i = 0, \ldots, k-1$, $\langle s_i, pr_i, s_{i+1} \rangle$ is an edge in T_Δ. It should be noted that $TeAL$ supports the case that prompts are performed by "*time*" instead of any entities. This allows the representation of "*system changes because of the passage of time*", e.g. "*two seconds after receiving the message, it becomes old*."

Paths of a $TeAL$ theory Δ represent valid evolutions of the system based on actions. They ignore durations of actions and temporal constraints. To take the temporal aspects of $TeAL$ theories into consideration, we define timed paths. Given a $TeAL$ theory Δ, a *timed path with the horizon* h (of Δ) is a sequence:

$$\langle s_0, pr_0, t_0; s_1, pr_1, t_1; \ldots s_{k-1}, pr_{k-1}, t_{k-1}; s_k \rangle, \tag{3}$$

where $\langle s_0, pr_0; s_1, pr_1; \ldots; s_{k-1}, pr_{k-1}; s_k \rangle$ is a path and $0 \leq t_0 < t_1 < t_{k-1} < h$. We assume that all time parameters t_i, $0 \leq i \leq k-1$, and h are scaled to the same time unit and are integers. Intuitively, t_i represents the time when the prompts in the set pr_i are executed, causing the system to change to s_{i+1} in the next time moment.

We now present the semantics of a temporal condition $C = \alpha\ T_E\ \beta$, which reads "α occurs in relation T_E to β." Here α and β are prompts or fluents and T_E is a relation between time points, for instance, **noLaterThan** x seconds **after**. The relation $p, t \models C$, which we read as "C holds (or, is satisfied) on p at time t," is defined as follows (we mention only two representative cases of the definition here and refer to an earlier paper [14] for details):

[1] This is the only difference from the transition system of Baral and Gelfond. In our work, prompts play the role of actions.

1. If $\alpha = prtSymb\ A$ and $\beta = prtSymb\ B$, the condition C states that the A has to be commenced (or terminated, depending on the prompt $prtSymb$) at the time point that is in the relation T_E to the time point of commencing (or terminating, depending on the prompt) the action B (for instance: "forward message 10 seconds after logging message"). Since there is no connection to t, $p, t \models C$ does not depend on t and holds if for every time s, $0 \leq s \leq h(p)$, such that α holds at time s, there is a time s' such that β holds at time s' and s and s' are in the relation T_E, or if no time point s' in the relation T_E with s falls in the range $[0, h(p)]$.
2. If $\alpha = prtSymb\ \mathbf{next}\ A$ and $\beta = prtSymb\ \mathbf{next}\ B$, the condition C states that the time of the next occurrence of $prtSymb\ A$ must be in the relation T_E to the next occurrence of of $prtSymb\ B$. Here the concept of "next" is understood with respect to the time t. Thus, we define $p, t \models C$ if (1) there is no occurrence of $prtSymb\ B$ after t; or if (2) there is an occurrence of $prtSymb\ B$ after t, the first one after t takes place at time s, and there are no times s' within the range $(t, h(p)]$ that are in the relation T_E with s'; or if (3) there is an occurrence of $prtSymb\ B$ after t, the next one after t takes place at time s, there are times s' within the range $(t, h(p)]$ that are in the relation T_E with s' and in one of them the first occurrence of $prtSymb\ A$ after t takes place.

All other cases (different combinations of prompts and fluents for α and β), and the case of temporal conditions $C = \alpha\ T_E$, can be handled similarly.

We say that a temporal condition C *holds* on a timed path p (or is *satisfied* on p), written $p \models C$, if for every t, $0 \leq t \leq h(p)$, $p, t \models C$. These definitions extend in an obvious way to arbitrary temporal constraints as they are simply boolean combinations of temporal conditions.

In principle, in order to check that $p \models C$ holds, one has to check that $p, t \models C$ for every t, $0 \leq t \leq h(p)$. However, for each horizon h and temporal condition C there is a finite set of time points $CP(h, C)$, we call them *checkpoints* for C, with the following property: for every timed path p, $p \models C$ if and only if for every $t \in CP(h(p), C)$ $p, t \models C$ holds. This property implies an algorithm to check whether $p \models C$ holds.

We say that a $TeAL$ theory is *consistent* if for every h there is a timed path p such that $p \models C$, for every temporal constraint C in the theory. Otherwise, the theory is *inconsistent*.

We note that the choice of the horizon h may have a significant effect on whether constraints are satisfied on timed paths. For instance, given two prompts *pr1*, *pr2*, and a timed path p in which *pr1* occurs every 4 seconds starting from time 0, and *pr2* occurs every 5 seconds starting from time 0, it is obvious that the constraint

$$\mathbf{if}\ \ pr1\ \mathbf{then}\ pr2\ \mathbf{noLaterThan}\ 2\ seconds\ \mathbf{after}$$

is satisfied if the horizon is less than 15, but violated on all timed paths with the horizon greater than or equal to 15. The techniques we use later to determine consistency require that the horizon be specified. This example shows that the selection of the right horizon value is important. We comment on this matter later on.

4 From TeAL to *clingcon* Language

We designed a translation from $TeAL$ to clingcon based on two integer parameters: the horizon h (a positive integer) and the number of state changes n (clearly, $n \leq h$), so that each answer set of the translated program represents a valid timed path with the horizon h traversing n states (not counting the initial state) and, conversely, each such path corresponds to an answer set. Given a $TeAL$ theory Δ, we write $\Pi(\Delta)$ for the corresponding clingcon program. $\Pi(\Delta)_{h,n}$ means that the parameters *H* (horizon) and *N* (number of states) are evaluated as *h* and *n*. Additionally, $\Pi(\Delta)$ can be divided into two parts: $\Pi^{ntemp}(\Delta)$, which corresponds to $\langle SI, AD, IC\rangle$, and $\Pi^{temp}(\Delta)$, which corresponds to $\langle TC\rangle$. Therefore, $\Pi^{ntemp}(\Delta)$ represents the system behavior without the temporal constraints (legal sequences of N states when temporal aspects are disregarded). We write $\Pi^{ntemp}(\Delta)_n$ for that program with the parameter N instantiated to n.

Similarly, $\Pi^{temp}(\Delta)$ represents the temporal constraints of Δ. It involves the parameter H that specifies the time horizon within which the constraints are to be considered. In other words, this time horizon determines the space of timed paths on which the constraints are to be satisfied. We write $\Pi^{temp}(\Delta)_h$ for the program $\Pi^{temp}(\Delta)$ with H instantiated to h.

The program $\Pi^{ntemp}(\Delta)_n$ contains all names in SI and a new sort: *state* with a set of constants $\{0 \ldots n\}$, where n is configurable number. In addition, $\Pi^{ntemp}(\Delta)_n$ contains predicates that specify following relations:

- $holds(F, S)$ (fluent *F* is true at state *S*)
- $happen(Pr, S)$ (prompt *Pr* happens at state *S*, and changes the system to the next state)
- $agent(Ag)$ (*Ag* is an agent)
- $act(Act)$ (*Act* is an action)
- $action(Ag, Act)$ (Agent *Ag* performs the action *Act*)
- $dur(action(Ag, Act), Dur)$ (The duration of $action(Ag, Act)$ is *Dur*)
- $prompt(Pr)$ (pr is a prompt, an event that can change the value of a fluent; the available prompts are $com(action(Ag, Act))$, $ter(action(Ag, Act))$, $totrue(F)$ and $tofalse(F)$, the latter two representing the change caused by passage of time)
- $init(F)$ (fluent *F* holds in the initial state)
- $engaged(Ag)$ (agent *Ag* is performing some action)
- $progress(action(Ag, Act), S)$ (agent (Ag) is performing action *Act* in state S)
- $previous(happen(Pr, S1), S)$ (the latest occurrence of prompt *Pr* before state *S* is in state *S1*)
- $next(happen(Pr, S1), S)$ (the earliest occurrence of prompt *Pr* after state *S* is in state *S1*)

$\Pi^{ntemp}(\Delta)_n$ contains rules that represent the state constraints, dynamic causal laws and executability conditions from AD, and the constraints on the initial state from IC. The translation is based on the translation from AL to answer set programming [5]. The use of prompts instead of actions introduces additional constraints in $\Pi^{ntemp}(\Delta)_n$ to specify preconditions and effects of the prompts. Intuitively, starting an action a (that is, executing $\mathbf{commence}\ a$) requires that the action is not in "progress." Moreover, an

agent starting this action must not be "engaged" in the execution of another. Finally, to terminate an action, the action has to be in progress, and terminating an action makes an agent no longer engaged. To model these constraints, w e use predicates $progress$ ($progress(a)$ says that action "a is in progress") and $engaged$ ($engaged(ag)$ says that "agent ag is engaged"). The following rules show how the constraints pertaining to **commence** can be expressed in ASP. To this end, we recall some elements of the ASP (*clingcon* syntax). A rule of the form :- $cond$ expresses a constraint that $cond$ must not hold, an expression kS, where S is a set, represents the constraint that at least k elements in S must be true, and finally, a rule of the form a :- $b, c, \ldots$, says that a can be derived as true if b, $c, \ldots$ have been derived as true.

$$\text{:- } 2\{happen(com(action(Ag, Ac)), S) : act(Ac)\}, agent(Ag), state(S). \quad (4)$$

$$happen(totrue(progress(action(Ag, Ac))), S)$$
$$\text{:- } happen(com(action(Ag, Ac)), S), state(S), action(Ag, Ac). \quad (5)$$

$$\text{:- } happen(totrue(progress(Ac)), S), not\ happen(com(Ac), S), state(S). \quad (6)$$

$$happen(totrue(engaged(Ag)), S)$$
$$\text{:- } happen(com(action(Ag, Ac)), S), state(S), action(Ag, Ac). \quad (7)$$

$$\text{:- } happen(totrue(engaged(Ag)), S), agent(Ag), state(S),$$
$$\{happen(com(action(Ag, Ac)), S) : action(Ag, Ac)\}0. \quad (8)$$

$$\text{:- } holds(engaged(Ag), S), happen(com(action(Ag, Ac)), S),$$
$$action(Ag, Ac). \quad (9)$$

The constraints discussed above are not mentioned explicitly in requirement documents. They represent a common (shared) knowledge and must be made explicit in $\Pi^{ntemp}(\Delta)$. Here is yet another example of an implicit constraint that must be included in $\Pi^{ntemp}(\Delta)$: each state must be associated with at least one prompt, because only prompts can change the states of the system. It can be expressed in ASP as follows:

$$1\{happen(Pr, S) : prompt(Pr)\}\text{:- } state(S). \quad (10)$$

Given $\Pi^{ntemp}(\Delta)_n$, the transition diagram T_Δ is constructed according to the rules in Baral and Gelfond's work [5]. We also incorporate prompts and extend the results of *AL* [5] to our theorem.

Theorem 1. *Let Δ be a TeAL theory $\langle SI, AD, IC, TC\rangle$ and n an integer. A sequence $p = \langle s_0, pr_0, \ldots, s_{n-1}, pr_{n-1}, s_n\rangle$ is a valid path in T_Δ if and only if $\Pi^{ntemp}(\Delta)_n$ has an answer set A such that for every i, $1 \leq i \leq n$,*

1. *if f is a fluent, then $f \in s_i$ if and only if $holds(f, i) \in A$*
2. *if pr is a prompt, then $pr \in pr_i$ if and only if $happen(pr, i) \in A$*

Moreover, for every answer set A of $\Pi^{ntemp}(\Delta)_n$ there is a valid path $p = \langle s_0, pr_0, \ldots, s_{n-1}, pr_{n-1}, s_n\rangle$ in T_Δ such that A and p satisfy the two conditions above.

This result establishes a correspondence between valid paths in the transition system T_Δ and answer sets of the program $\Pi^{ntemp}(\Delta)_n$. It opens a way to compute valid

paths of length n in the transition system T_Δ by applying an ASP solver to the program $\Pi^{ntemp}(\Delta)_n$. This is a stepping stone to computing timed paths for Δ, a key element of our approach to checking consistency of temporal constraints that we are to discuss next.

Next, we outline the structure of the program $\Pi^{temp}(\Delta)$. A valid timed path requires that all temporal constraints are satisfied in every time moment on this timed path. However, checking every time moment is infeasible. As we observed in the previous section, it can be replaced by checking the condition on a finite set of *check points*. Given a temporal condition C, the check points for C are chosen so that if C is satisfied (not satisfied, respectively) at the check point t, it is satisfied (not satisfied, respectively) at all time moments in the interval between t and the next check point (or the horizon). Thus, the task of checking satisfiability along a timed path can be reduced to checking satisfiability at every check point.

For instance, let C be: α **noLaterThan** x seconds **after**. We write $occur(\alpha, t)$ for the statement "α happens at time t." If $p \models occur(\alpha, t_1)$ holds, then for every $t' \in [max(0, t_1 - x), t]$, $p, t' \models C$ holds. Additionally, let us suppose that for some $t_2 > t_1 + x$, we have (i) $p \models occur(\alpha, t_2)$, and (ii) for no t_3 such that $t_2 > t_3 > t_1$, $p \models occur(\alpha, t_3)$ holds. Then, for every $t'' \in [t_1 + 1, t_2 - x - 1]$, $p, t'' \not\models C$. It follows that the check points for C are: $\min(t + 1, horizon)$ and $\max(t - x, 0)$, for all t such that $p \models occur(\alpha, t)$ holds.

There are two types of check points. The first type comprises the time moments when the system changes, that is, the *last* time moments when the system is still in its present state. These happen to be the time moments when prompts occur. The second type comprises the time moments when nothing changes in the system, but the *satisfaction of temporal conditions* changes as the result of passing time. For instance, given a timed path p, in which a prompt *pr1* only occurs at second 5, the temporal condition

$$pr1 \textbf{ laterThan } 10 \textit{ seconds } \textbf{before}$$

is satisfied from the 6th to 14th second, and violated at the 15th second.

Each temporal condition is associated with a group of such check points. Each check point has an ID, which is based on its sequence in the timed path, and a value, which is a time moment. Typical answer set solvers will use a relation to represent that *a check point is assigned a time moment*, and the grounding process will generate instances for all possible time moments, which is very inefficient. The key aspect of *clingcon* is that it combines answer set programming with constraint solving. The assignment of time moments to check points, represented as integer variables $time(CP_i)$, where CP_i is a check point, is handled using constraint solving techniques. The rest of the program is constructed according to the standard ASP methodology. This prevents the generation of huge numbers of ground instances of rules.

$\Pi^{temp}(\Delta)_h$ contains rules that set up check points. The following rules are applied to all check points:

$$\$domain(0..horizon). \tag{11}$$

$$1\{map(C, S) : check(C)\}1 \text{ :- } state(S). \tag{12}$$

$$\text{:- } map(C1, S1), map(C2, S2), S1 > S2, not\ time(C1) > time(C2). \tag{13}$$

$$\text{:- } check(C1), check(C2), C1 > C2,$$
$$time(C1) < horizon, time(C1) \leq time(C2). \quad (14)$$
$$checkhappen(Pr, C) \text{ :- } happen(Pr, S), map(C, S). \quad (15)$$
$$checkholds(F, C) \text{ :- } holds(F, S), map(C, S). \quad (16)$$

We use the relation $map(C, S)$ to represent that state S is mapped to check point C. Rule (11) states that the range of check points must be within the horizon. Rule (12) states that only one state can be mapped to a check point. Rules (13) and (14) state that the time assignment of states and check points must be based on their sequence in the path. Rules (15) and (16) use two new relations: $checkhappen$ and $checkholds$. They are the "*check point*" version of the $happen$ and $holds$ relations that are used above. These two rules mean that whatever happens or holds in a state must happen or hold in its corresponding check point.

The *Temp* module also contains rules for specifying check points for each temporal condition. Using the temporal condition *C* above, *Temp* contains the following rules:

$$exist(cp1, C1) \text{ :- } check(C2),\ time(C2)\$ == time(C1) + 1, check(C1),$$
$$checkhappen(\alpha, C1). \quad (17)$$
$$\text{:- } checkhappen(\alpha, C1),\ not\ exist(cp1, C1),$$
$$horizon >= time(C1) + 1. \quad (18)$$

Rule (17) uses the relation $exist(cp1, C1)$ to define that for any check point $C1$ (when α occurs), there exists another check point immediately after it. Rule (18) means that if α occurs at a check point, and the horizon is large enough, then there must be another check point as defined by $exist(cp1, C1)$. $\Pi^{temp}(\Delta)_h$ also contains similar rules for the check points $t - x - 1$.

$\Pi^{temp}(\Delta)_h$ uses the relation $sat(C, arguments, \textbf{\textit{CP}})$ to represent that "*the temporal condition C is satisfied on the check point CP*". The $arguments$ are the actions and fluents involved in C. Let C be the example above, $Temp$ contains the following rules:

$$sat(C, \alpha, CP1) \text{ :- } checkhappen(\alpha, CP2), CP2 > CP1,$$
$$time(CP2) - time(CP1) <= x. \quad (19)$$
$$\text{-}sat(C, \alpha, CP1) \text{ :- } not\ sat(C, \alpha, CP1),\ horizon >= time(CP1) + x. \quad (20)$$

Rule (19) defines when $p, time(CP1) \models C$, and rule (20) defines when $p, time(CP1) \not\models C$.

Given a temporal constraint of the form (1), $\Pi^{ntemp}(\Delta)_n$ uses the following rule to check that for each check point, the temporal constraint is satisfied.

$$\text{:- } sat(A_1, args, \textbf{\textit{CP}}), \ldots, sat(A_k, args, \textbf{\textit{CP}}),$$
$$\text{-}sat(B_1, args, \textbf{\textit{CP}}), \text{-}sat(B_m, args, \textbf{\textit{CP}}),\ check(\textbf{\textit{CP}}). \quad (21)$$

This rule means that for each check point *CP*, if all the temporal conditions $A_1, \ldots, A_k$ are satisfied on *CP*, then at least one of $B_1, \ldots, B_m$ shall be satisfied on *CP* as well.

Rules of type (21) complete the description of $\Pi^{temp}(\Delta)$ and so, also of $\Pi(\Delta)$. The following result establishes the correspondence between valid timed paths for a $TeAL$ theory Δ and answer sets of the program $\Pi^{temp}(\Delta)_{n,h}$.

Theorem 2. *Let Δ be a $TeAL$ theory and h and n integers such that $0 < n \leq h$. A sequence $p = \langle s_0, t_0, pr_0, \ldots, s_{n-1}, t_{n-1}, pr_{n-1}, s_n \rangle$ is a valid timed path of Δ if and only if $\Pi(\Delta)_{h,n}$ has an answer set A such that for every i, $1 \leq i \leq n$,*

- *if f is a fluent, then $f \in s_i$ if and only if $holds(f, i) \in A$*
- *if pr is a prompt, then $pr \in pr_i$ if and only if $happen(pr, i) \in A$*
- *there is j, $1 \leq j \leq n$, such that $map(j, i) \in A$ and $time(j) = t_i$.*

Moreover, for every answer set A of $\Pi(\Delta)_{n,h}$ there is a valid timed path $p = \langle s_0, t_0, pr_0, \ldots, s_{n-1}, t_{n-1}, pr_{n-1}, s_n \rangle$ such that A and p satisfy the conditions above.

This theorem shows that timed paths of horizon h and n state changes can be computed by ASP tools such as *clingcon*, thus providing useful information concerning consistency of temporal requirements. We discuss this issue in detail in the next section.

5 Tools Developed for Processing $TeAL$ Theories

Theorem 1 suggests an approach to testing consistency of temporal requirements represented by a $TeAL$ theory D. First, one constructs the program $\Pi(D)$, as described in the previous section. It involves two integer parameters representing the horizon and the number of states. Instantiating these parameters with specific values h and n of these parameters (we recall that we must have $0 < n \leq h$), yields the program $\Pi(D)_{h,n}$ that we then process with the *clingcon* solver.

If for every $n = 1, \ldots, h$, the program $\Pi(D)_{h,n}$ has no answer sets, then the transition system T_D has no valid timed path of the horizon h. In other words, there is no way to implement the system so that it runs for h time units. This indicates that the requirements are inconsistent.

If, on the other hand, for some n, $1 \leq n \leq h$, $\Pi(D)_{h,n}$ has answer sets, it means that the requirements are consistent, as long as we only consider running the system for h time units. This is not an absolute guarantee of consistency. It may be that the problem with the requirements shows up only for some larger values of the horizon. For instance, the temporal constraint "$Prompt$ **noLaterThan** 10 $second$ **after startTime**" is not satisfied by a timed path p if *Prompt* does not occur on p within the first 10 seconds. However, if p has a horizon *less* than *10* then, according to our definition, this temporal constraint is satisfied on p even if *Prompt* does not occur on p. This is because paths of horizon shorter than 10 cannot be used as counterexamples to the constraint — there is always a possibility that should the path be extended, the *Prompt* would occur on it and the constraint would hold. Thus, to demonstrate a problem with this requirement, paths with the horizon of at least 10 must be considered. In general, the larger the horizon for which valid timed paths can be found, the stronger the assurance of consistency.

We built a tool, *TeALTrans* that implements the approach to consistency testing outlined above (full description and the source code at `http://progit.netlab.uky.edu/teal`). To compute answer sets of programs $\Pi(D)_{h,n}$ the tool uses *clingcon*, an ASP solver integrated with a specialized integer constraint solver *gecode* [16]. Delegating solving linear-integer constraints to

gecode gives *clingcon* a significant performance advantage over "pure" ASP solvers such as *clasp* [11]. The latter compile all integer constraints into boolean ones, which results in a dramatic blow-up of the theory size.

The lack of the absolute guarantee of consistency is a limitation due to our choice of ASP tools for processing. A more traditional approach to checking consistency of temporal requirements based on LTL [13] and model checking tools such as *Nusmv* [8] in theory does not suffer from these difficulties. We designed a translation from $TeAL$ to *Nusmv* and studied the effectiveness of this approach, too.

6 Study Results

We studied the correctness and efficiency of our tool using the six benchmark examples described in Section 2. For each, we created its corresponding $TeAL$ representation. We recall that the temporal constraints in our examples involve constants (parameters). Consistency of the constraints depends on specific values one chooses for these parameters. For each benchmark problem, we considered four parameter settings: (1) *underconstrained-relaxed* or *UR*, the temporal constraints leave much room for the system to evolve, they are "easy" to satisfy; (2) *underconstrained-tight* or *UT*, the constraints are still satisfiable but they significantly restrict the ways in which the system can evolve; (3) *overconstrained-barely* or *OB*, the constraints are inconsistent, but a small relaxation of some of them would make them consistent; and (4) *overconstrained-much* or *OM*, the constraints are significantly overconstrained and no small relaxations make them consistent. Finally, we studied three values for the horizon: $h = 50, 100,$ and 200 and set the time-out limit at 7200 seconds.

The following table shows the results of our study. For each of the problems, it shows the number of constraints, the parameter settings (UR, UT, OB, and OM), and the running time. For problems that are consistent, the table also shows the number of states, n, for which the constraints were shown to be satisfiable. There were no time-outs when the theories were consistent. For overconstrained cases, the tool timed out several times (for one problem for h=50, for five problems for $h = 100$, and for all problems for $h = 200$). Whenever the tool timed-out, we show in the table the last value of n, for which inconsistency was successfully demonstrated.

As mentioned above, the choice of the horizon h may affect our confidence in the determination that the requirements are consistent, and in general the larger the horizon, the stronger the evidence of consistency. However, there is a flip side to this observation. As the results show, the larger the horizon, the more computationally intensive the computing task becomes. This is because the number of possible values for the number of states grows with h. Estimating a value for the number of states, with which the constraints are consistent, is difficult. So our tool considers all of them in turn starting with $n = 1$. If *clingcon* finds an answer set, we assert that the $TeAL$ theory does not contain inconsistency within the horizon and terminate. Otherwise, we proceed to the next value of n or terminate (and declare inconsistency) if $n = h$.

Our results also show that if the $TeAL$ theory is consistent, the consistency could be established within the time limit imposed (even for $h = 200$). This is a strong indication of the practical potential of our tool.

Table 1. Results of the study; six problems, 4 parameter settings

Example	# Constraints	Type	Horizon		
			50	100	200
CM1	23	UR	395 sec, 9 states	1139 sec, 17 states	2151 sec, 34 states
		UT	429 sec, 9 states	1353 sec, 17 states	2328 sec, 34 states
		OB	5962 sec	> 2 hours, 40 states	> 2 hours, 37 states
		OM	5913 sec	> 2 hours, 40 states	> 2 hours, 37 states
511Phone	11	UR	564 sec, 9 states	2551 sec, 18 states	3571 sec, 35 states
		UT	572 sec, 9 states	2732 sec, 18 states	3691 sec, 35 states
		OB	> 2 hours, 42 states	> 2 hours, 38 states	> 2 hours, 36 states
		OM	> 2 hours, 42 states	> 2 hours, 38 states	> 2 hours, 36 states
MODIS	10	UR	204 sec, 7 states	589 sec, 12 states	1787 sec, 20 states
		UT	221 sec, 7 states	594 sec, 12 states	1901 sec, 20 states
		OB	4212 sec	7009 sec	> 2 hours, 47 states
		OM	4204 sec	6878 sec	> 2 hours, 47 states
UAVTCS	13	UR	681 sec, 9 states	1677 sec, 17 states	4104 sec, 33 states
		UT	696 sec, 9 states	1783 sec, 17 states	4143 sec, 33 states
		OB	5813 sec	> 2 hours, 42 states	> 2 hours, 35 states
		OM	5771 sec	> 2 hours, 42 states	> 2 hours, 35 states
iTrust	12	UR	606 sec, 7 states	1574 sec, 13 states	3945 sec, 24 states
		UT	601 sec, 7 states	1591 sec, 13 states	4043 sec, 24 states
		OB	6042 sec	> 2 hours, 37 states	> 2 hours, 25 states
		OM	5906 sec	> 2 hours, 37 states	> 2 hours, 25 states
EasyClinic	10	UR	306 sec, 8 states	1025 sec, 14 states	2775 sec, 29 states
		UT	323 sec, 8 states	1236 sec, 14 states	2834 sec, 29 states
		OB	6194 sec	> 2 hours, 38 states	> 2 hours, 31 states
		OM	6275 sec	> 2 hours, 38 states	> 2 hours, 31 states

The situation is different when the theory is inconsistent. It takes a long time for the tool to determine inconsistently. The reason is obvious and related to the discussion above. If the $TeAL$ theory is inconsistent, then for *each* number of states, n, $1 \leq n \leq h$, *clingcon* will attempt to determine consistency (that is, find an answer set) and eventually fail. However, especially when n is large, the grounding bottleneck reappears (variables representing states have to be instantiated). This makes it hard for *clingcon* to handle large values of n.

Our results suggest two practical steps to address the problem. First, for all problems and overconstrained parameter settings (when the constraints are inconsistent), the inconsistency demonstrated itself already when $h = 50$. Thus, if a tool times out with a particular value of h, one might try smaller values of h. If the theory is inconsistent, the tool might now succeed in determining that. Secondly, one might run the tool until it times out. If the last value of n for which the computation succeeded with n is sufficiently large (for instance, at least $h/5$), one might take this as an indication of a *possible* problem with the requirements.

The results also show that changing the parameter combinations from *UR* to *UT* does not affect the time for computing answer sets. Similarly, there seems to be no such effect when we change from *OB* and *OM* (but here we have fewer data points to draw conclusions).

Next, the study shows that the number of constraints in the $TeAL$ theory has much impact on the effectiveness of our tool as does the value of h. The example system with the largest number of constraints, CM1, does not turn out to be more difficult than the others. Finally, the study demonstrates the correctness of our tool. In all cases, the results produced by the tool were consistent with our "manual" analysis of the problems.

We also performed experiments based on LTL, but our translation of the six problems into the input format of *Nusmv* resulted in theories that *Nusmv* could not handle (timed-out in *all* cases).

7 Discussion, Conclusions, and Future Work

We presented an approach for analyzing software requirements using answer-set programming (ASP). We presented a translation from the $TeAL$ language for describing temporal requirements to ASP and stated results establishing the correctness of the translation. We used several benchmark examples taken from real software systems to test the correctness and efficiency of our tool.

The results we presented indicate the potential of our approach to assist requirement engineers verify the consistency of temporal requirements. In the six examples that we studied, the tool provided strong evidence of consistency, whenever the requirements were consistent. With one exception, it also was able to detect inconsistency when the theory was inconsistent (with the choice of $h = 50$).

It has to be noted that when we determine consistency, we do not obtain an absolute proof of consistency but only a proof of consistency within the specified horizon. For hard problems, when the tool times out even with smaller values of h, we similarly only obtain support to the claim of inconsistency but not an absolute proof. This is a limitation of our approach.

A more traditional approach to checking consistency of temporal requirements based on LTL [13] and model checking tools such as *Nusmv* [8] in theory does not suffer from these difficulties. However, our paper shows that ASP tools that give rise to an approach based on the parameters h and n are more effective. While the results do not always provide absolute assurances of consistency (inconsistency), by appropriately choosing the parameters we can obtain some balance between the strength of the guarantee we get and the time in which we compute this guarantee.

Our future work involves improving the efficiency of our tool. One possible approach is to estimate the lower bound and upper bound on the number of states n for which it is sufficient to run *clingcon*. Another direction is to study the completeness threshold for the horizon h, that is, find the value of h such that consistency with respect to h gives an absolute guarantee of consistency (for every $TeAL$ theory such a threshold exists). Finally, we intend to work on optimizations to our current translation.

At present, when we report inconsistency, we provide no indication which requirements cause the problem. We will develop extensions to the present tool that will suggest to the analyst likely combinations of requirements that might be responsible for the inconsistency.

References

1. CM-1 Dataset PROMISE Website,
`http://promisedata.org/promised/trunk/promisedata.org/data/cm1-maintain/cm1-maintain.txt`
(accessed: April 18, 2013)

2. MODIS Science Data Processing Software Requirements Specification Version 2, SDST-089, GSFC SBRS (November 1997), http://www.fas.org/irp/program/collect/uav_tcs.htm
3. UAV Tactical Control System (May 2010), http://www.fas.org/irp/program/collect/uav_tcs.htm
4. Regional Real-Time Transit Information System System Requirements Version 3.0 (2012), http://www.mtc.ca.gov/planning/tcip/Real-Time_TransitSystemRequirements_v3.0.pdf (accessed: April 18, 2013)
5. Baral, C., Gelfond, M.: Reasoning Agents in Dynamic Domains. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 257–279. Kluwer Academic Publishers, Norwell (2000)
6. Boehm, B., Papaccio, P.: Understanding and Controlling Software Costs. IEEE Transactions on Software Engineerin 14(10), 1462–1477 (1988)
7. Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., Panichella, S.: On the Role of the Nouns in IR-based Traceability Recovery. In: IEEE 17th International Conference on Program Comprehension, ICPC 2009, pp. 148–157. IEEE (2009)
8. Cimatti, A., Giunchiglia, E., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Integrating BDD-based and SAT-based Symbolic Model Checking. In: Armando, A. (ed.) FroCos 2002. LNCS (LNAI), vol. 2309, pp. 49–56. Springer, Heidelberg (2002)
9. Firesmith, D.: Specifying Good Requirements. Journal of Object Technology 2(4), 77–87 (2003)
10. Firesmith, D.: Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. Journal of Object Technology 6(1), 17–33 (2007)
11. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *clasp*: A Conflict-Driven Answer Set Solver. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 260–265. Springer, Heidelberg (2007)
12. Gebser, M., Ostrowski, M., Schaub, T.: Constraint Answer Set Solving. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 235–249. Springer, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-02846-5_22
13. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press (2004)
14. Li, W., Hayes, J.H., Truszczyński, M.: Temporal Action Language (TAL): a Controlled Language for Consistency Checking of Natural Language Temporal Requirements. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 162–167. Springer, Heidelberg (2012)
15. Li, W., Truszczynski, M., Huffman Hayes, J., Brown, D.B.: Temporal Action Language. University of Kentucky Computer Science Department Technical Report (2012-521-12) (2012)
16. Schulte, C., Lagerkvist, M., Tack, G.: Gecode. Software Download and Online Material at the Website (2006), http://www.gecode.org