Towards More Efficient Requirements Formalization: A Study

Wenbin Li, Jane Huffman Hayes, and Miroslaw Truszczynski

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

Abstract. [Context and motivation] Validating natural language requirements is an important but difficult task. Although there are techniques available for validating formalized requirements, the gap between natural language requirements and formalism is huge. **[Question/problem]** As part of a larger piece of work on temporal requirements consistency checking, we developed a front end to semi-automatically translate natural language requirements into an formal language called Temporal Action Language or $TeAL$. This work is based on an underlying assumption that human analysts can assist us in filling in the missing pieces as we translate natural language temporal requirements to $TeAL$. **[Principal ideas/results]** We performed a study to validate this assumption. We found that our front-end tool is more effective and efficient than a manual process. **[Contribution]** We present the design of our front-end and a study that measures the performance of human analysts in formalizing requirements with the help of an automated tool.

Keywords: formal specification, temporal requirements, translation, requirement comprehension.

1 Introduction

Temporal requirements play an important role in the specification, design, and development of software systems. 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 prior to submitting an order for processing. A safety critical pacemaker system requires that pacing occur within milliseconds of certain detected events.

As these examples suggest, errors in specifying, interpreting, or implementing temporal requirements can lead to disastrous consequences. If one or more requirements related to the pacing of the heart are in conflict, a negative heart event might not trigger a required life-saving pacing event. To address such issues, we undertake consistency checking of temporal requirements. This is a labor intensive and tedious task, however. Indeed, it is possible that a specification of a system contains so many temporal requirements (and related contextual requirements) that it is not possible to check them manually. Hence we look to automation for assistance.

Many powerful formal languages and specification techniques have been offered to support temporal consistency checking [5, 11, 14]. Nonetheless, the main challenge for the automation of temporal requirement consistency checking is that they are typically represented as natural language text. There are several reasons for the use of text. Text is highly expressive, there is little to no learning required to use it and, when carefully used, natural text is precise and unambiguous. Yet, fully automated processing of textual requirements remains a distant goal. To take advantage of the power of formal methods in analyzing requirements specified in text, we need ways to translate the natural language requirements into some formal language. However, every attempt to do so runs into the question of whether the formal representation correctly captures the intended meaning of the natural language requirements.

To address this long-standing criticism of formal methods, we have developed an intermediate language to bridge the syntactically significant gap between low-level formalisms and natural language temporal requirements: the Temporal Action Language ($TeAL$) [16]. We developed fully automated methods to translate $TeAL$ to low-level logic formalisms such as answer-set programming (ASP) [18, 19] programs and linear temporal logic (LTL) [12] theories (the translation is the subject of another publication [15]). That reduces the overall problem of consistency checking to that of producing correct $TeAL$ theories from the natural language representations of temporal (and contextual) requirements, and brings up a key question: how can $TeAL$ theories be created more efficiently?

We introduce a semi-automated method that translates natural language to $TeAL$. The efficiency of this method rests on the assumption that humans can assist us during the translation. We performed a study to validate this assumption. We found that our tool is more effective and efficient at the translation task than a manual process.

This paper represents the first study to evaluate human ability to assist with semi-automated translation to a formal language. Measures used in other fields such as foreign language translation have been applied in the study in order to gauge human ability to assist with $TeAL$ translation. The paper is organized as follows. Section 2 briefly describes the formal representations studied. Section 3 presents our approach to natural language temporal requirement translation. Sections 4 and 5 discuss validation and results, respectively. Section 7 and 8 analyze the results and the feedback we collected from participants. Section 8 provides conclusions and a look at future work.

2 Formal Representation of Temporal Requirements

Earlier, we introduced Temporal Action Language ($TeAL$) as a formal language for supporting software requirement analysis [16]. The $TeAL$ language is an extension of Action Language AL [4], a language designed for modeling actions and their effects and for reasoning about ways in which a system can evolve. The $TeAL$ language retains all the features of AL and can also be used to specify temporal constraints. Because $TeAL$ is used to bridge the gap between natural language requirements and low-level logic formalism, we designed its syntax to be as close to natural language as possible to minimize analysts' time and effort. We briefly describe the syntax below (see [16] for a full description).

The basic components of $TeAL$ are actions, fluents, and temporal conditions. Actions change the state of the system. They are performed by agents. For example, the $TeAL$ expression *connect(serA,nodeA)* represents an action to establish a connection to *nodeA*; *serA* is the agent that performs this action. 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 *connected(serA,nodeA)* represents that the server *serA* is connected to the node *nodeA*. Temporal conditions specify temporal relationships on times when events occur. Such events include the start and end of actions as well as the changes of system properties (fluents). In $TeAL$, we use two prompts: **commence** Act and **terminate** Act, to represent the time when action Act starts and successfully finishes. In $TeAL$ one can also relate two consecutive occurrences of the same action to each other. To distinguish between them, $TeAL$ provides the keywords **previous** and **next**, as in: **commence previous** Act and **terminate next** Act. A fluent appearing in temporal conditions represents the time when this fluent becomes true. Similarly, the negation of a fluent in temporal conditions represents the time when this fluent becomes false. Additionally, we view the start of the system as a special event; **startTime** represents when it happens.

Time moments represented by actions and fluents are connected by temporal relationships. Given two time moments, *t1* and *t2*, the basic relationship between them can be: "*t1 before/after t2*," or "*t1 and t2 are at the same time*." Additionally, requirements may specify more information, such as "*t1 before t2 for some amount of time*." $TeAL$ provides eight keyword phrases to represent temporal relationships. Most types of temporal relationships specify both time moments explicitly as, for example, in the expression

$$received(server, message, node)\ \mathbf{within}\ 5\ second\ \mathbf{after}\\ \mathbf{terminate}\ send(node, message, server)$$

which encodes the requirement "*the message is received by the server within 5 seconds after it is sent by the node.*" Such elementary relationships between time points are called *temporal conditions*.

The keywords **and**, **or**, and **not**, as well as the **if** ... **then** ... phrase, can be used together with temporal conditions to represent their boolean combinations, called *temporal constraints*. The specific form of a temporal constraint used in $TeAL$ is

$$\mathbf{if}\ A_1\ \mathbf{and}\ \ldots\ \mathbf{and}\ A_k,\ \mathbf{then}\ B_1\ \mathbf{or}\ \ldots\ \mathbf{or}\ B_m; \tag{1}$$

where A_1 **and** ... **and** A_k and B_1 **or** ... **or** B_m are temporal conditions or their negations. An example of a temporal constraint in $TeAL$ is an expression:

$$\mathbf{if\ not\ commence}\ print(server, message)\ \mathbf{within}\ 5\ second\\ \mathbf{after}\ received(server, message, node)\\ \mathbf{then\ terminate}\ send(server, alarm)\ \mathbf{within\ next}\ 2\ second;$$

It captures the constraint "*if a message is not printed within 5 seconds after it is received, the server shall send an alarm within 2 seconds.*"

As mentioned earlier, $TeAL$ is an extension of AL, and it reuses the syntax of AL. In particular, $TeAL$ reuses the expressions of the following three forms:

State constraints	L **if** P;	(2)
Dynamic causal laws	a **causes** E **if** P;	(3)
Executability conditions	**impossible** $pr_1, \ldots, pr_k$ **if** P;	(4)

where E, L, and P are lists of fluents and their negations and pr, $pr_1, \ldots, pr_k$ are prompts. Expression (2) captures the constraint that every state satisfying P must also satisfy L. For example, the $TeAL$ expression

$$powerOn(computer) \textbf{ if } running(computer);$$

says that if a computer is running, the power must be on. Expression (3) specifies the immediate effects of prompts. It says that if prompt *pr* occurs in a state satisfying *P*, then *E* must hold in the next state. For example, the $TeAL$ expression

$$\begin{aligned}&\textbf{terminate } connect(serA, nodeA)\\ &\quad \textbf{causes } connected(nodeA, serA) \textbf{ if } systemOn;\end{aligned}$$

describes what will happen when the action ***connect(serA, nodeA)*** finishes. Finally, expression (4) defines preconditions of prompts. It states that at least one of $pr_1, \ldots, pr_k$ must not occur if *P* holds at this time moment. For example, the $TeAL$ expression

$$\begin{aligned}&\textbf{impossible commence } write(nodeA, serA)\\ &\quad \textbf{if not } connected(serA, nodeA);\end{aligned}$$

specifies that the server ***serA*** and the node ***nodeA*** must be connected (is the prerequisite) for the action ***write(nodeA, serA)*** to be executable.

The expressions above do not specify temporal information. $TeAL$ supports a more general version of these expressions by allowing fluents to be replaced with temporal conditions. This helps analysts to specify much more complicated preconditions and effects such as in the following $TeAL$ expression:

$$\begin{aligned}&\textbf{impossible commence } read(server, message)\\ &\quad \textbf{if not } received(server, message, node) \textbf{ within } 3\ second \textbf{ before } ;\end{aligned}$$

This expression represents that the precondition of starting a read action is "***receiving the message within 3 seconds.***" Similarly, the expression:

$$\begin{aligned}&\textbf{commence } drive(Anne, home, office) \textbf{ causes}\\ &\quad in(Anne, office) \textbf{ within next } 30\ minute;\end{aligned}$$

represents that the effect of the action ***drive(Anne, home, office)*** is that Anne will be in the office in 30 minutes.

To specify durations of actions in $TeAL$, we use expressions

$$\textbf{duration } Action\ d\ unit; \tag{5}$$

To specify the initial state of a system, we use expressions

$$\textbf{initially } F; \tag{6}$$

where F is a list of fluents and their negations (each of the expressions on F is required to hold initially).

3 Translation from Natural Language Requirements

This section addresses prior work in translation as well as our proposed approach.

Prior Work. Our research is closely related to natural language understanding, a major task in natural language processing (NLP) [22]. This task focuses on converting natural language text into formal representations so that programs can handle them. Applications that accept natural language text as input often perform parsing of the text and then represent the parsed text as a logic set. These logic sets can be processed and used to assess the semantics of the text.

Our research concept is very similar to that of $CARL$ [9], an application that integrates natural language parsing techniques with default reasoning to identify inconsistencies in natural language requirements. The parsing task uses the $CICO$ [8] algorithm. The algorithm uses domain-based parsing; it allows analysts to define domain specific rules for the parser.

Natural language processing toolkits such as Stanford parser [13, 7] and $OpenNLP$ library [3] support most of the common NLP tasks, including chunking, parsing, speech tagging, and tokenizing. It should be noted that the Stanford parser also extracts dependencies, which are the grammatical relations between words. This type of information is very useful for our research, the details will be given in the introduction of our proposed approach.

Another NLP task that is important to our research is Semantic Role Labeling (SRL) [10]. The SRL technique detects the semantic arguments of verbs or predicates and the roles of these arguments. For example, given "*a system updates data*," SRL finds the verb *update* with *system* as its agent and *data* as its object. The SRL technique proves to be very useful in extracting actions and fluents from natural language. Such information is necessary for building $TeAL$ theories.

Proposed Approach to Translation. We aim to create a semi-automated approach for checking temporal consistency of requirements given in natural language. Our idea is to translate the requirements into a theory in a low-level formal system, which can be analyzed automatically. As mentioned earlier, the "*distance*" between natural language and low-level formal methods is substantial. We propose to use an intermediate language, $TeAL$, to bridge the gap. Thus, to translate text requirements into a low-level formal system we needs to translate from text to $TeAL$. We present and study one such method in this paper.

The method decomposes the task into four steps, presented below (Figure 1):

- Step 1: extract relevant requirements
- Step 2: identify system information

– Step 3: generate $AlmostTeAL$ statements
– Step 4: build a $TeAL$ theory that models the system

The first three steps are fully automated and generate a collection of $AlmostTeAL$ statements. The last step requires the involvement of an analyst whose task is to convert $AlmostTeAL$ statements to $TeAL$ statements that correctly represent input requirements.

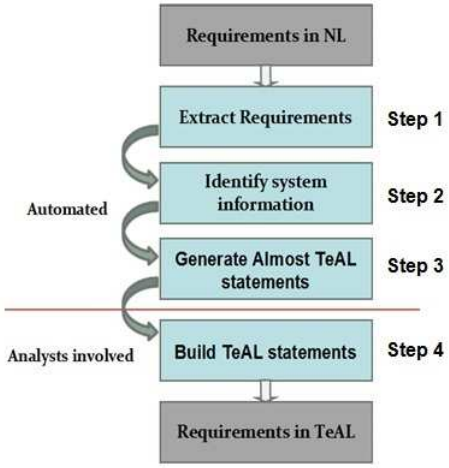

Fig. 1. Steps for Generating TeAL Statements

***Step 1 (extract relevant requirements)*:** Temporal requirements, such as "*the system updates data within 5 seconds after receiving the prediction data*," must be identified and extracted from the collection of requirements. Most temporal requirements contain keywords such as before and within, or patterns such as "*do action every x seconds*." It is viable to detect many, if not all, of the temporal requirements based on these keywords and patterns. The technique described by Nikora [20] and NLP techniques [6] can be used to address this task and have been incorporated into our front-end translator.

Given a set of temporal requirements, we also need to identify non-temporal requirements that are related to them and that might contain relevant system information. The same techniques as listed above can be used here because typically these non-temporal requirements share terms such as entity names with the temporal requirements (not temporal ones that are already found). We employ these techniques in our tool. By the end of this step, the tool has identified all requirements that are necessary for modeling the system.

***Step 2 (identify system information)*:** Given a list of requirements found in Step 1, several types of system elements must be identified: vocabulary and constraints.

The vocabulary consists of the names of objects of the system and their properties. It also includes names of fluents and actions. Our front-end tool uses the seman-

tic role labeling (SRL) technique and Stanford Parser to assist in extracting the vocabulary. As mentioned earlier, SRL finds actions and represents them as predicates such as: *update(system,data)*. Some fluents, such as *received(Receiver,Msg,Sender)*, can also be found in this way. However, SRL cannot detect any fluent from the text "*system is in safe mode,*" while there is a fluent *in(system,safeMode)*. Therefore, our tool uses Stanford Parser to extract fluents such as this one. The parser generates a set of typed dependencies for given texts. Each typed dependency represents a relationship between two words. In this case, the useful dependencies are: $nsubj(is, system)$ and $prep\_in(is, mode)$. These two dependencies illustrate that the system is "*in a mode,*" and this should be modeled as a fluent. Our tool also uses these typed dependencies to decide the types of the semantic arguments. For example, with the typed dependency $prep\_to(sender, receiver)$, our tool decides that the *send* action has an argument whose type is $send\_to$. Typed dependencies are useful because requirements often lack information on all of the arguments. For example, given a requirement "*if . . ., then the node should send the message,*" SRL will consider that the action send only has two arguments: the node that sends, and the message being sent. But if there is another requirement with the typed dependency $prep\_to(sender, receiver)$, the tool can infer that the *send* action has another argument, and the generated action will be:

$$send(node, message, \_)$$

When the tool generates a translation such as this, analysts are alerted that something is missing.

Constraints can be temporal or non-temporal. Temporal constraints often contain patterns for specifying temporal relationships among events. For example, "*do action within x seconds after*" and "*do action every x seconds*" are patterns that are commonly used in temporal requirements. These patterns can be represented by regular expressions such as

$$(PP < ((IN < within)..(CD\$ + NNS)))$$

for the within pattern. We use these regular expressions to extract the temporal constraints.

Additionally, we need to identify the relationships among actions, fluents, and temporal events. For example, we need to find out if a fluent is the precondition or the effect of an action, if two temporal relationships are disjunctive with each other, or whether a temporal relationship is a precondition or not. Our tool uses patterns and typed dependencies mentioned above for this task. The useful dependencies are neg, $conj\_or$, and $conj\_and$, which correspond to negation, disjunction, and conjunction. The patterns

$$(SBAR < ((IN < if)\$ + S))$$

and

$$(SBAR < ((WHADVP < (WRB < when))\$ + S))$$

are used for matching texts of the form "*if something*" or "*when something*." The temporal relationships and fluents that are included in these texts will be marked as preconditions. For example, given the following text: "*If the node receives a command when it*

is not in safe mode, it should report an error in 10 seconds," our tool finds the following information: *receive(node,command)* and *report(node,error)* are actions, *in(node, safe mode)* is a fluent, and

$$report(node, error)\ within\ next\ 10\ seconds$$

is a temporal relationship. In addition, the tool establishes that *receive(node,command)* and *in(node, safe mode)* are to be included in the precondition. Because there is no evidence of negation and disjunction, they are assumed to be conjunctive with each other by default. Besides, if there are other requirements that contain the information of "*receive from somewhere,*" the tool will update the action to:

$$receive(node, command, \_)$$

It should be noted that many non-temporal constraints are treated as "*common sense*" or tacit knowledge, and they will not appear in the requirements. For example, no requirement will specify that "*a message cannot be received if it has not been sent.*" However, such common sense knowledge is necessary for modeling a system. One possible way to further automate the identification of such unspecified information is by using some kind of "*common sense library.*" A possible choice is $ConceptNet$ [17], a commonsense knowledge base that focuses on physical, temporal, and social aspects. It is also possible to use libraries that are domain specific, such libraries should cover the fundamental constraints in the domain.

***Step 3 (identify system information)*:** Our front-end tool builds "$AlmostTeAL$" statements based on the information generated in Step 2. For instance, for each action, the tool analyzes the information extracted in Step 2 to find this action's possible effects and preconditions, connect them with the conjunction or disjunction operator, and use them to construct precondition and effect statements. The tool also analyzes related temporal relationships to organize them into the "*if . . . then . . .*" expressions.

As mentioned above, some data may still be missing in the representation and some data may be unspecified. Given the sample output of Step 2, our tool generates:

$$\textbf{if}\ receive(node, command, \_)\ \textbf{and}\ in(node, safemode)$$
$$\textbf{then}\ report(node, error)\ \textbf{within next}\ 10\ second;$$

***Step 4 (build a $TeAL$ theory that models the system)*:** Analysts need to generate $TeAL$ statements based on the outputs of the front-end tool. More specifically, analysts need to perform the following tasks:

- Read the $AlmostTeAL$ statement to decide what it means.
- Compare the $AlmostTeAL$ statement and its corresponding natural language and generate a correct $TeAL$ statement.

Given the sample output of Step 3, analysts need to remove all errors in that $AlmostTeAL$ statement and complete it to form a $TeAL$ statement:

$$\textbf{if terminate}\ receive(node, command, somewhere)\ \textbf{and}\ in(node, safemode)$$
$$\textbf{then terminate}\ report(node, error)\ \textbf{within next}\ 10\ second;$$

In this case the analysts need to specify whether the constraint concerns the time when actions are commenced or when they are terminated. Also, analysts need to add the arguments for the receive action: the entity the node receives messages from (here denoted by *somewhere*). However, the $AlmostTeAL$ statement is very close to the $TeAL$ statement we want to generate. And it is also close to natural language text, so the analyst's task is manageable and ultimately may even be further automated.

4 Validation

This section addresses validation of the usefulness of $TeAL$ and of the $AlmostTeAL$ tool.

Research Questions. As mentioned earlier, our semi-automated method requires analysts' involvement before correct $TeAL$ theories are generated. This involvement takes place in Step 4, as Steps 1 - 3 are fully automated in our front-end tool that generates $AlmostTeAL$ statements. Once $AlmostTeAL$ output is available, analysts must add missing elements and remove inaccuracies in these statements so that a correct $TeAL$ theory can be passed to the fully automated step of translating into a low-level formal system. The effectiveness and efficiency of this step greatly affects the effectiveness and efficiency of the entire method and is the focus of this paper.

The accuracy of our front-end tool is critical, because it determines how many inaccuracies an analyst needs to detect and correct in order to generate $TeAL$ statements. Our tool is useful if analysts are more effective at generating $TeAL$ with the help of $AlmostTeAL$ statements than they are without the aid of $AlmostTeAL$ statements. To study the effectiveness and efficiency of Step 4, we posit the following research questions:

- RQ1: Does the front-end produce outputs that improve the effectiveness of generating correct $TeAL$ statements?
- RQ2: Does the front-end produce outputs that improve the efficiency of generating correct $TeAL$ statements?

RQ1 and RQ2 are important as they directly evaluate the quality of the method we developed and implemented for generating $AlmostTeAL$ statements in an automated way.

Dependent and Independent Variables. This study uses one independent variable: *Method* (abbreviated as *M*)). There are two levels of this variable: $TeAL$, and $TeAL$ with the assistance of $AlmostTeAL$.

RQ1 addresses the effectiveness of generating $TeAL$ statements. The dependent variables (see TABLE II) that address RQ1 are: Precision ($\mathbf{Prec1}$), Recall ($\mathbf{Rec1}$), and F-measure ($\mathbf{F1}$) of predicates and temporal relationships (*send*, *received*, *within next 10 second*); Precision ($\mathbf{Prec2}$), Recall ($\mathbf{Rec2}$), F-measure ($\mathbf{F2}$) of arguments (e.g., *node*, *message*, *server* as arguments of *send* and *received*), Translation Error Rate ($\mathbf{TER}$)[21], and Translation Difficulty Score ($\mathbf{TDS}$).

The basic structure of $TeAL$ statements is represented by predicates ($Pred$) and temporal relationships ($Temp$). Identifying predicates and temporal relationships is the

key component of our front-end tool because the basic structure of $TeAL$ statements is represented by these two types of information. For instance, *received within 10 seconds after send* is intuitive, though it needs more detail to be a correct $TeAL$ statement.

The measure **Rec1** is defined as the percentage of correct $Pred/Temp$ that are written, while the measure **Prec1** is the percentage of written $Pred/Temp$ that are correct.

$$\mathbf{Rec1} \;=\; \frac{\#\,of\ correct\ Pred/Temp\ written}{\#\,of\ correct\ Pred/Temp}$$

$$\mathbf{Prec1} \;=\; \frac{\#\,of\ correct\ Pred/Temp\ written}{\#\,of\ Pred/Temp\ written}$$

The measure **F1** is a harmonic mean of **Prec1** and **Rec1**:

$$\mathbf{F1} \;=\; \frac{2 * \mathbf{Prec1} * \mathbf{Rec1}}{\mathbf{Prec1} + \mathbf{Rec1}}$$

The above formula puts equal importance to both **Prec1** and **Rec1**.

Our tool also identifies arguments. Arguments are necessary for generating correct $TeAL$ statements. For instance, the example above needs the arguments of *send* and *received*.

Similar to the measures above, **Rec2** defines the percentage of correct arguments that are written, **Prec2** defines the percentage of written arguments that are correct, and **F2** is a harmonic mean of **Prec2** and **Rec2**.

We also use **TER** to measure how close a generated $TeAL$ statement is to the $TeAL$ statement that correctly specifies the system. The measure **TER** is an error metric for machine translation that measures the number of edits required to change a system output into a target text:

$$\mathbf{TER} \;=\; \frac{\#\,of\ edits}{average\ \#\,of\ words\ in\ target\ text}$$

where possible edits include the insertion, deletion, substitution of single words, and shifts of word sequences. We convert each $TeAL$ statement into a sequence of words so that we can use this measure. For instance, we will convert

$$received(node, msg, server)$$
$$\textbf{within}\ 10\ second\ \textbf{after terminate}\ send(server, msg, node)$$

into: *received node msg server within 10 second after terminate send server msg node* and then compare this sequence of words to the answer set to determine how many insertions, deletions, can changes are required.

The measure **TDS** is a rating on a scale from 1 to 5 indicating the participants' subjective opinion about the difficulty of translating from natural language to $TeAL$ with/without $AlmostTeAL$.

The dependent variable that address RQ2 is the average time (**T**) spent on each question. The measure **T**, or **Time**, evaluates the efficiency of the method.

Table 1. Dependent Variables

Variable	Abbr	Scale
Predicate Precision	**Prec1**	[0,1]
Predicate Recall	**Rec1**	[0,1]
Predicate F measure	**F1**	[0,1]
Argument Precision	**Prec2**	[0,1]
Argument Recall	**Rec2**	[0,1]
Argument F measure	**F2**	[0,1]
Translation Error Rate	**TER**	[0,1]
Translation Difficulty Score	**TDS**	{1,2,3,4,5}
Time	**T**	time

Hypothesis. The null hypothesis for RQ1(H_{0RQ1}) is that there is no difference in the **Prec1**, **Rec1**, **F1**, **Prec2**, **Rec2**, **F2**, **TER**, and **TDS** between **TeAL** and **ATeAL**. The alternative hypothesis (H_{1RQ1}) is that there is a difference between the two methods.

Similarly, the null hypothesis for RQ2 (H_{0RQ2}) is that there is no difference in the measure **T** of **TeAL** and **ATeAL**. The alternative hypothesis (H_{1RQ2}) is that there is a difference.

Study Design. We conducted a study that evaluated effectiveness and efficiency with and without $AlmostTeAL$. The study involved thirty four participants, all students in computer science courses at the University of Kentucky. A pre-study questionnaire was given to all the consenting (per IRB regulations) participants in order to gauge prior experience and comfort with requirement analysis and formal languages. Additionally, each participant received a ten minute introduction about the background of the experiment. Participants were also given a fourteen minute training video and a training document. The training video introduced the syntax and semantics of $TeAL$. It focused on the representation of actions, fluents, and temporal relationships. The video includes $AlmostTeAL$ as well. The training document covered everything in the video. The participants were required to watch the video or read the document before the main study task.

After the introduction, the main study assignment was administered. Each participant received a user ID. Each participant received a set of eight questions during the main study task:

– Given a natural language requirement (with/without $AlmostTeAL$), write down its corresponding $TeAL$ statement.

We broke the participants into two groups based on their experience in requirements and formal languages. We randomly divided the participants of each experience level into two groups of the same size. One group wrote $TeAL$ statements with the help of $AlmostTeAL$, another group did not have $AlmostTeAL$ statements.

Participants were asked to complete the tasks in the classroom. They were also asked to record the time they spent on each question. After completing the main study task, participants were asked to submit a hardcopy of the results and complete a post-study questionnaire that asked for their reaction to requirement analysis and formal

languages. The study used examples from two datasets: 511 Regional Real-Time Transit Information System Requirements (511phone) [2] and CM1 [1]. The 511phone dataset presents the system requirements for the Bay Area 511 Regional Real-Time Transit Information System (available open source). The requirements are primarily focused on the performance of the 511 System and data transfers with the transit agencies. The CM1 dataset is a requirement document produced by NASA for one of its science instruments. The document was released by NASA for use by the software engineering research community.

Threats to Validity. Our study was subject to a number of threats to validity, mitigated to the best of our ability. A threat to internal validity is the limited amount of time given to the participants to learn $TeAL$. We were constrained by the amount of time available in the class period. To address this, we separated the training session and the experiment into separate sessions (separate consecutive class periods). This allowed the participants more time to understand $TeAL$ and $AlmostTeAL$ by using the training video and document. Another threat to internal validity is that we created answers for the questions and used them as the golden answer set. Because we designed $TeAL$ and have much experience in creating $TeAL$ statements from natural language requirements, the quality of the golden answer set can be assured.

Our work with student participants represented a threat to external validity. However, these students all have at least three years of background in computer science and they understand the concepts of software engineering and requirements engineering. Their background allows them to perform small tasks of requirement analysis the same as professionals with no significant differences [23]. Another threat to validity deals with our use of two datasets. Though both 511phone and CM1 datasets are from real projects, the study results may differ for different datasets in different domains. One solution is repeating the experiment with other datasets from other domains. The third threat to external validity is the motivation of the participants. Students were given extra credit to participate. This did not ensure that they answered all questions "*seriously*" or thoughtfully. We noticed that two participants read the training document during the experiment before they answered the questions. It is possible that they had not read it before the experiment. This could affect the correctness of their answers and the time it took for them to answer.

Dependent variable issues that threaten construct validity were reduced by the use of standard measures. We address this validity threat by using different sets of measures: precision, recall, F-measure, **TER**, and **TDS**, to analyze different aspects of "the effectiveness of generating $TeAL$." We use **Prec1**/**Rec1**/**F1** to measure the effectiveness of identifying predicates and temporal relationships, **Prec2**/**Rec2**/**F2** for the effectiveness of identifying arguments, **TER** for the edits required from generated $TeAL$ statements to correct answers, and **TDS** for the subjective point of view from participants. Another threat to construct validity is that participants may have guessed the research hypothesis, that is, they may have assumed that $AlmostTeAL$ was the focus of the research with an aim to improve effectiveness and efficiency before they worked on the main study assignment. We addressed this validity threat by not telling them that $TeAL$ and $AlmostTeAL$ are our research areas.

5 Results

Table 2 presents the results of the study whether using **ATeAL** is more effective than generating **TeAL** expressions directly (RQ1) and whether using **ATeAL** is more efficient than using **TeAL** directly (RQ2).

Table 2. Mean values of **Prec1**, **Rec1**, and **F1**, and of **Prec2**, **Rec2**, and **F2**

	Prec1	**Rec1**	**F1**	**Prec2**	**Rec2**	**F2**
TeAL	84.13%	85.63%	84.58%	65.25%	58.31%	60.96%
ATeAL	89.39%	89.28%	89.11%	84.89%	83.28%	83.97%

Specifically, Table 2 shows the mean values of precision (**Prec1**), recall (**Rec1**), and F-measure (**F1**) for predicates and temporal relationships. When **ATeAL** is used, the results are better in all aspects than when **TeAL** is used alone. However, the results are very close in this part of the study. The values of **Prec1**, **Rec1**, and **F1** also illustrate that participants performed well in capturing the general structure of **TeAL** statements, but the possibility of incorrect or missing predicates/temporal relationships cannot be ignored, no matter what target language is used.

Table 2 also shows the mean values of precision (**Prec2**), recall (**Rec2**), and F-measure (**F2**) for arguments. The **ATeAL** method is better than **TeAL** for 20% in precision and 25% in recall. The results show that it was much more difficult for the participants to generate correct and complete arguments without the help of $AlmostTeAL$.

Table 3. Mean values of **TER**, **TDS** and **T**

	TDS	**TDS**	**T**
TeAL	52.75%	3.38	282 sec
ATeAL	25.11%	4.33	167 sec

Table 3 shows the mean values of **TER**, **TDS** and the mean values of time **T** spent on each question. The results show that the participants wrote better **TeAL** with the help of $AlmostTeAL$ statements: the number of edits required from the generated **TeAL** statements to the correct **TeAL** was halved. The results on **TDS** illustrate that the participants generally felt more comfortable and found it easier to write **TeAL** statements with $AlmostTeAL$ statements presented. Finally, participants reduced time spent by 40% with the help of $AlmostTeAL$ statements.

6 Discussion

Based on the results above, it is clear that the $AlmostTeAL$ statements generated by our front-end tool improve the process of generating $TeAL$ statements in both effectiveness and efficiency.

Figure 2 compares between **TeAL** and **ATeAL** with regard to the objective measures concerning effectiveness: **Prec1**,**Rec1**,**F1**,**Prec2**,**Rec2**,**F2**, and **TER**.

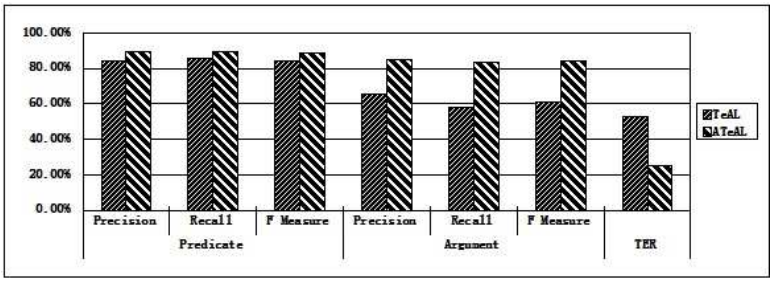

Fig. 2. Results of Objective Measures

Though there were practical differences in the **Prec1** and **Rec1** of our study, the differences were not statistically significant. The high **Rec1** and **Prec1** values (84%-89%) show that one possibility is that these elements can be identified without the help of $AlmostTeAL$. Yet the performance of **ATeAL** is still slightly better than **TeAL**.

The results of **Prec2** and **Rec2** show that participants had a hard time in identifying arguments without $AlmostTeAL$: they missed about 40% of arguments, while 35% of the arguments they identified were incorrect. The $AlmostTeAL$ statements greatly improved both precision and recall to 83%-84%. The differences in the **Prec2** and **Rec2** measures are extremely significant. It appears that $AlmostTeAL$ finds more correct arguments than the participants. Additionally, the missing pieces in $AlmostTeAL$ can remind participants what information to look for when they read natural language requirements. Participants also reduced time spent by 40% and halved their error rate with the help of $AlmostTeAL$ statements. The differences in the **TER** and **T** measures are also extremely significant. The decrease of **TER**, together with the increase of **Prec2** and **Rec2**, proves the effectiveness of **ATeAL**.

Additionally, the feedback from participants proves that they prefer **ATeAL** to **TeAL**. On the one hand, 56% of the participants thought it was difficult to write $TeAL$ statements without any hints (**TDS** ≤ 3); on the other hand, 83% of the participants felt the presence of $AlmostTeAL$ provides useful information (**TDS** ≥ 4).

Returning to the questions of interest, based on the study we found that:

- RQ1: Yes. The $AlmostTeAL$ statements generated by the automated method help analysts to produce $TeAL$ with fewer errors. We can reject the null hypothesis in favor of the alternative (H_{0RQ2}).
- RQ2: Yes. The $AlmostTeAL$ statements generated by the automated method reduces the time needed for this process. We can reject the null hypothesis in favor of the alternative (H_{1RQ2}).

7 Feedback

We get several comments from the post-study questionnaire about $TeAL$. There are positive comments such as: *The syntax and order of arguments felt natural*" and "$TeAL$

provides a consistent and precise structure for interpreting requirements and relationships. There are also comments that point out problems, such as: "*It was a little unclear how much was always required to be strict about things,*" "*it was hard to be certain about if I was successfully stating things in perfect* $TeAL$*,*" and "*Sometimes I wasn't sure what words to use.*" These comments remind us to further improve the effectiveness of step 2 and step 3, as better $AlmostTeAL$ statements will solve/partially solve these problems. Additionally, we are considering providing other information together with $AlmostTeAL$, such as reminding analysts that certain parts of the $AlmostTeAL$ statements are incomplete, or presenting a list of possible values for arguments.

8 Conclusion and Future Work

This work tackles a fundamental problem of requirements engineering. Requirements are most often given as natural language text and so are prone to ambiguities, incompleteness, and inconsistencies. To manually analyze requirements for correctness is hard and error-prone itself. The solution is in automation of the process. However, the distance between a natural language and a low-level formal one for which automated reasoning tools are available is large. We proposed to bridge the gap by means of an intermediate-level formal language $TeAL$. We use our translator tool to generates expressions in "$AlmostTeAL$" that are close to the correct ones in $TeAL$ so that we can significantly ease the analyst task to produce final correct $TeAL$ results. The effectiveness of the proposed approach largely depends on the help that $AlmostTeAL$ can provide. We performed an experiment to study this problem and provided evidence that suggests that using $AlmostTeAL$ in the process of translation improved efficiency of analysts and helps with accuracy.

We leaves several interesting questions for the future. First, we plan to enhance our translator tool by developing modules of common (tacit) knowledge that we expect will improve the accuracy of the translation process in Steps 1-3. Second, we are looking for methods to demonstrate possible incompleteness and ambiguity in $AlmostTeAL$. Finally, our experience with the front-end translator to $AlmostTeAL$ demonstrates it can be enhanced to provide analysts with feedback on obvious problems with the requirements (some entities never defined, missing terms, etc.). Thus, the quality of the input requirements can be improved even before they are translated into low-level formalism for consistency analysis. We plan to explore this direction in depth.

Acknowledgment This work is funded in part by the National Science Foundation grant CCF-0811140. This work was previously sponsored by NASA grant NNG05GQ58G. We thank the anonymous participants. We thank Mark Hays for statistics assistance.

References

1. CM-1 Dataset PROMISE Website, `http://promisedata.org/promised/trunk/promisedata.org/data/cm1-maintain/cm1-maintain.txt`, accessed: 2013-4-18

2. 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: 2013-4-18
3. Baldridge, J.: The opennlp project. URL: http://opennlp. apache. org/index. html,(accessed 2 February 2012) (2005)
4. Baral, C., Gelfond, M.: Reasoning agents in dynamic domains. In: Logic-based artificial intelligence, pp. 257–279. Springer (2000)
5. Cimatti, A., Giunchiglia, E., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Integrating bdd-based and sat-based symbolic model checking. In: Frontiers of Combining Systems, pp. 49–56. Springer (2002)
6. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: The detection and classification of non-functional requirements with application to early aspects. In: Requirements Engineering, 14th IEEE International Conference. pp. 39–48. IEEE (2006)
7. De Marneffe, M.C., MacCartney, B., Manning, C.D., et al.: Generating typed dependency parses from phrase structure parses. In: Proceedings of LREC. vol. 6, pp. 449–454 (2006)
8. Gervasi, V.: The cico domain-based parser (2001)
9. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Transactions on Software Engineering and Methodology (TOSEM) 14(3), 277–330 (2005)
10. Gildea, D., Jurafsky, D.: Automatic labeling of semantic roles. Computational linguistics 28(3), 245–288 (2002)
11. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering 23(5), 279–295 (1997)
12. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and reasoning about systems. Cambridge University Press (2004)
13. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. pp. 423–430. Association for Computational Linguistics (2003)
14. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 134–152 (1997)
15. Li, W., Brown, D., Hayes, J.H., Truszczynski, M.: Answer-set programming in requirements engineering. In: Requirements Engineering: Foundation for Software Quality, pp. 168–183. Springer (2014)
16. Li, W., Hayes, J.H., Truszczyński, M.: Temporal action language (tal): a controlled language for consistency checking of natural language temporal requirements. In: NASA Formal Methods, pp. 162–167. Springer (2012)
17. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of english: The penn treebank. Computational linguistics 19(2), 313–330 (1993)
18. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm, pp. 375–398. Springer (1999)
19. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence 25(3-4), 241–273 (1999)
20. Nikora, A.P., Balcom, G.: Automated identification of ltl patterns in natural language requirements. In: Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on. pp. 185–194. IEEE (2009)
21. Snover, M., Dorr, B., Schwartz, R., Micciulla, L., Makhoul, J.: A study of translation edit rate with targeted human annotation. In: Proceedings of association for machine translation in the Americas. pp. 223–231 (2006)
22. Spyns, P.: Natural language processing. Methods of information in medicine 35(4), 285–301 (1996)

23. Tichy, W.F., Padberg, F.: Empirical methods in software engineering research. In: Software Engineering-Companion, 2007. ICSE 2007 Companion. 29th International Conference on. pp. 163–164. IEEE (2007)