A Theory of Requirements Capture and Its Applications

Wei Li

Department of Computer Science
Beijing University of Aeronautics and Astronautics
100083 Beijing, P.R. China

Abstract

A theory of requirements capture is developed. Some concepts, such as new law, user's rejection, and reconstruction of a specification are defined; the related theorems are proved. The concepts of capture process and the limit of a capture process are further established. It is proved that, given a user's model and a specification, there is a procedure that all processes generated from the procedure and starting with the specification are convergent, and their limit is the truth of the model. Some computational aspects of reconstructions are studied. Some applications of the theory are briefly discussed, especially, an editor called SpecReviser is introduced.

1 Introduction

Formal approaches to program development usually focus on the problems of how to develop programs from a given specification and how to verify them to meet the specification ([BG77], [BJ 83], [EM 85], [NS 84] and [ST 88]).

In this paper, we study another problem, i.e., how to build a formal specification from some informal requirements. Let us call it the requirements capture. It is the first stage of the software development cycle. A requirement capture is a process describing the evolution of specifications. This process is usually non-trivial since no one can accomplish an appropriate specification of a problem at one stroke. In the process, there is a lot of interactions between the clients and software engineers in order to evaluate and develop an appropriate specification.

The purpose of this paper is to introduce the necessary concepts used in this area, study their computational aspects, prove the related theorems, and show some applications of the theory to software tools. Before going to the technical details, let us study a simple example and see what kind of concepts will be needed in the theory.

Example 1.1 Reverse function.

Consider how a student builds a specification for the reverse function. For simplicity, we assume that a first order language is used as our formal language.

Let L, K, P, Q denote the lists and Nil be the empty list. Let a, b, c denote the elements of a list.

$append : list \times list \rightarrow list$ is the concatenation function.

$reverse : list \times list \rightarrow bool$ is an atomic formula.

At the beginning, the student captured his first law: $A_1 = reverse(Nil, Nil)$ easily. Thus, his specification is

$$\Gamma_1 = \{A_1\} = \{reverse(Nil, Nil)\}.$$

He then found the fact that if K is the reverse of L, then, for any element a, the list $append(\{a\}, K)$ is the reverse of the list $append(L, \{a\})$. He then generalized this fact, and obtained his second law:

$$A_2 \equiv \forall K, L, P : list.(reverse(K, L) \supset reverse(append(P, K), append(L, P))).$$

Since A_2 is logically independent of Γ_1, in other words, neither A_2 nor $\neg A_2$ can be deduced from Γ_1, he added A_2 into Γ_1. Thus, the revised specification became:

$$\Gamma_2 = \{\Gamma_1, A_2\}.$$

After having tried some examples for Γ_2, he realized that Γ_2 is a wrong specification because he found a counter examples: Let:

$$\begin{array}{rcl} L_1 \equiv K_1 & \equiv & c.nil \\ P_1 & \equiv & a.b.nil \\ Q_1 & \equiv & b.a.nil \end{array}$$

then

$$\begin{array}{rcl} append(P_1, K_1) & = & a.b.c.nil \\ append(L_1, P_1) & = & c.a.b.nil \end{array}$$

$reverse(K_1, L_1) = true$, but

$$reverse(append(P_1, K_1), append(L_1, P_1)) = false.$$

L_1, K_1 and P_1 together are called a user's rejection of A_2, and A_2 is said to be rejected by the user. Thus, he deleted A_2 from Γ_2. The revised specification denoted by Γ_3 becomes $\{A_1\}$ again. In fact, a user's rejection is a model (counter example) which falsifies A_2. To continue the process, he then tried the law:

$$A_3 \equiv \forall L, K, P, Q : list.(reverse(L, K) \supset (reverse(P, Q) \supset reverse(append(P, L), append(K, Q)))).$$

Obviously, A_3 is logically independent of $\Gamma_3 = \{A_1\}$. It is called a new law for Γ_3. He added A_3 into Γ_3 and captured Γ_4. The reconstruction of Γ_3 is:

$$\Gamma_4 = \{\Gamma_3, A_3\}.$$

The sequence of specifications: Γ_1, Γ_2, Γ_3, $\Gamma_4, \cdots$ describes the evolution of specifications given by the student. It is called a process of requirements capture for the reverse function. □

The conclusions which we have reached from this example are:

1. The processes of requirements capture for a specific problem can be expressed by a sequence of specifications: $\Gamma_1, \cdots, \Gamma_n, \cdots$, where Γ_{n+1} is a reconstruction of Γ_n. A specification must be revised if a new law or a user's rejection can be found.

2. A new law denoted by A for a given specification Γ is a formula which is logically independent of Γ. The reconstruction of the specification, in this case, is obtained by adding A into Γ.

3. The only way to check whether a specification Γ meets the user's requirements is to *deduce* some result (theorem) B from Γ (including the formulas contained in Γ) using logical inference rules, and to see whether B meets the user's requirements. There are two possibilities:

   (a) Every B matches user's requirements. Then, the specification is accepted.

   (b) There exists a B which meets a user's rejection (a counter-example). Then, the specification must be revised. All formulas contained in Γ have to be checked; those (minimal) subsets which have caused the user's rejection must be found and rectified.

4. A process of requirements capture is non-monotonic if a user's rejection arises after the reconstruction of a specification for a new law. For instance, in the example 1.1 we have: $\Gamma_1 \subseteq \Gamma_2 \not\subseteq \Gamma_3$.

In this paper, the concepts and ideas mentioned in this section will be formalized and expanded step by step.

2 New laws and user's rejections

We assume that the specification language used in the paper is a first order language L defined in [Gall 87]. More precisely, L is a set of strings of symbols formed by a BNF like grammar over an alphabet consisting of logical connectives including $\neg$, $\wedge$, $\vee$, $\supset$, $\forall$, $\exists$, and $\doteq$, a countable set **Var** of variables ranged over by $x, y, z \cdots$, a countable set **FS** of function symbols ranged over by $f, g, h \cdots$, a countable set **CS** of constants ranged over by $a, b, c \cdots$,

and a countable set **PS** of predicate symbols ranged over by the capital letters $P, Q \cdots$.

A and Γ are used to denote a formula and a sequence of formulas respectively. $Th(\Gamma)$ is used to denote the set of all theorems deduced from Γ.

A sequent is a form of $\Gamma \vdash A$. It should be mentioned that we follow [Paul 87], use the symbol $\vdash$ to replace the symbol $\rightarrow$ in [Gall 87], and use $\Gamma \vdash A$ as an object language assertion (see [Paul 87]).

In this paper, we employ the proof rules of sequent calculus given in [Paul 87]. Thus, we will treat a sequence of formulas as a set of formulas if it is needed.

The concepts of validity, satisfiability, falsifiability, provability and consistency used in this paper are defined in [Gall 87]. The proof system (inference rules) is *sound and complete.*

A model **M** is a pair $< M, I >$, where M is a domain and I is an interpretation. We use $\mathbf{M} \models \Gamma$ to denote $\mathbf{M} \models A$ for all A in Γ. Intuitively, a model can be viewed as an instantiation of a specification.

Definition 2.1 Semantic consequence

A is a semantic consequence of Γ denoted by $\Gamma \models A$ iff, for any model **M**, if $\mathbf{M} \models \Gamma$ then $\mathbf{M} \models A$.

Definition 2.2 Specification

A sequence Γ of closed formulas is called a specification, if Γ is consistent and is not empty. The closed formulas contained in Γ are called the laws of Γ.

In terms of mathematical logic, a specification discussed in this paper is a sequence of non-logical axioms, or a closed formal theory. The condition that Γ is not empty means that only the non-logical axioms are interested and can be rejected by the user. The definition of new law is:

Definition 2.3 New law.

A is called a new law for Γ iff there exist two models **M** and **M'** such that

$$\mathbf{M} \models \Gamma \quad \text{and} \quad \mathbf{M} \models A$$

$$\mathbf{M}' \models \Gamma \quad \text{and} \quad \mathbf{M}' \models \neg A$$

Theorem 2.1 A is a new law for Γ iff A is logically independent of Γ, that is neither $\Gamma \vdash A$ nor $\Gamma \vdash \neg A$ is provable.

Proof: straightforward from soundness and completeness. □

Corollary 2.1 If A is a new law for Γ, then both the sequences $\{\Gamma, A\}$ and $\{\Gamma, \neg A\}$ are consistent.

Definition 2.4 User's rejection

Let $\Gamma \models A$. A model $\mathbf{M}$ is a user's rejection of A iff $\mathbf{M} \models \neg A$.

Let $\Gamma_{M(A)} \equiv \{A_i \mid A_i \in \Gamma, \ \mathbf{M} \models A_i, \ \mathbf{M} \models \neg A\}$.

$\mathbf{M}$ is an **ideal** user's rejection of A iff $\Gamma_{M(A)}$ is ***maximal*** in the sense that there is no another user's rejection $\mathbf{M}'$ such that

$$\Gamma_{M(A)} \subset \Gamma_{M'(A)}.$$

An ideal user's rejection of A is denoted by $\overline{\mathbf{M}}(A)$. There may exist many ideal user's rejection of A. Let

$$\mathcal{R}(\Gamma, A) \equiv \{\Gamma_{\overline{M}(A)} \mid \overline{\mathbf{M}} \text{ is an ideal user's rejection of } A\}$$

The user's rejection meets the tradition that whether a specification Γ of a problem is acceptable depends only on whether all deduced results from Γ agree with the user's intuition about the problem; and it has to be rejected if there is a counter example. Anyhow, the acceptance of the specification has nothing to do with the logical inference rules of the first order logic.

The ideal user's rejections model the tradition of scientific research (Occam's razor): if some particular result deduced from a specification is rejected by the user, then only the minimal subsets of laws (contained in the specification) that cause the rejection have to be checked and rectified, and the rests (the maximal subsets) of the laws are *temporarily* retained.

The following definition is a generalized version of revision given in [Gär 88] by Gärdenfors, where the revision is defined in propositional logic.

Definition 2.5 Revision of a specification

Let $\Gamma \vdash A$. A revision Λ of Γ about $\neg A$ is a maximal consistent subset (or subsequence) of Γ, and is consistent with $\neg A$.

Let $\mathcal{A}(\Gamma, A)$ be the set of all revisions of Γ about $\neg A$.

The following theorems show that the proof-theoretic concept of ideal user's rejection is revision, and vice versa.

Theorem 2.2 R-soundness

If $\Lambda \in \mathcal{A}(\Gamma, A)$, then there exists an ideal user's rejection of A, $\overline{\mathbf{M}}$, such that $\Gamma_{\overline{M}(A)} = \Lambda$.

Proof: Since $\Lambda \in \mathcal{A}(\Gamma, A)$ is consistent and is also consistent with $\neg A$, $\{\Lambda, \neg A\}$ is satisfiable. Thus, there exists a model $\mathbf{M}'$ such that $\mathbf{M}' \models \Lambda$ and $\mathbf{M}' \models \neg A$. So, $\mathbf{M}'$ is a user's rejection of A. $\mathbf{M}'$ is maximal, since if there exists another $\mathbf{M}''$ such that $\mathbf{M}'' \models \neg A$ and $\Gamma_{M''(A)} \supseteq \Gamma_{M'(A)} \supseteq \Lambda$, then $\Gamma_{M''(A)} \subseteq \Lambda$ because it is a consistent subset of Γ which is consistent with $\neg A$. □

Theorem 2.3 R-completeness

If $\overline{\mathbf{M}}$ is an ideal user's rejection of A, then there exists a revision Λ of Γ about $\neg A$, such that $\Lambda = \Gamma_{\overline{M}(A)}$.

Proof: Straightforward. □

Corollary 2.2 $\mathcal{R}(\Gamma, A) = \mathcal{A}(\Gamma, A)$

Example 2.1 Let

$$\Gamma \equiv \{A, A \supset B, B \supset C, E \supset F\}$$

be a specification. We have $\Gamma \vdash C$. Assume that C meets a user's rejection, then there are three revisions of Γ about $\neg C$. The $\mathcal{A}(\Gamma, C)$ consists of:

$$\{A, A \supset B, E \supset F\} \quad \{A, B \supset C, E \supset F\} \quad \{A \supset B, B \supset C, E \supset F\}.$$

3 Processes of requirements capture

In this section, we deal with the problem of how to reconstruct a specification when a new law or a user's rejection arises, and will define what a process of requirements capture is.

Definition 3.1 Reconstruction of a specification.

Let A be a new law for Γ. The N-reconstruction of Γ for the new law A is the set $\{\Gamma, A\}$.

Let $\Gamma \models A$ and A have met a user's rejection. An R-reconstruction of Γ for the user's rejection of A is $\Gamma_{\overline{M}(A)}$, where $\overline{\mathbf{M}}$ is an ideal user's rejection of A.

A specification Γ' is a reconstruction of Γ iff Γ' is either an N-reconstruction of Γ for a new law A or an R-reconstruction of Γ for a user's rejection of A.

From theorem 2.3 we know that an R-reconstruction of Γ for the user's rejection of A is a revision of Γ about $\neg A$. Having defined the reconstructions of a specification, we can study the concept of process of requirements capture.

Definition 3.2 Process of requirements capture.

A specification sequence $\Gamma_1, \Gamma_2, \cdots, \Gamma_n, \cdots$ is called a process of requirements capture iff Γ_{i+1} is a reconstruction of Γ_i for $i \geq 1$.

The process is *increasing* iff $\Gamma_n \subseteq \Gamma_{n+1}$ for all n; it is *decreasing* iff $\Gamma_n \supseteq \Gamma_{n+1}$ for all n. The process is *monotonic* iff it is either increasing or decreasing; otherwise it is *non-monotonic*.

Since, for a given user's rejection of A, there may be many R-reconstructions ($\mathcal{A}(\Gamma_n, A)$ contains more than one element), the evolution of a specification should be represented by a *tree*, each branch of which is a process of requirements capture.

Theorem 3.1 Monotonicity

A process of requirements capture $\{\Gamma_n\}$ is increasing iff for any $n \geq 1$, Γ_{n+1} is an N-reconstruction of Γ_n for some new law A.

A process of requirements capture $\{\Gamma_n\}$ is decreasing iff for any $n \geq 1$, Γ_{n+1} is an R-reconstruction of Γ_n for user's rejection of some A.

Theorem 3.2 Non-monotonicity

A process of requirements capture is non-monotonic iff, for some $n > 1$, Γ_{n+1} is an R-reconstruction of Γ_n for a user's rejection, and Γ_n is an N-reconstruction of Γ_{n-1} for a new law; or Γ_{n+1} is an N-reconstruction of Γ_n for a new law, and Γ_n is an R-reconstruction of Γ_{n-1} for a user's rejection.

Proof: Straightforward from the definition. □

The above theorem tell us that non-monotonicity is a characteristic of a process of requirements capture where a user's rejection arises after a new law, or vice versa.

4 The Limits

If a process of requirements capture (or 'capture process' for short) is finite, it means that after finite steps of explorations, we build an appropriate specification. For the infinite capture processes, we need to introduce the concept of limits. In this section, we will build a procedure, and will prove that, given a user's model about a problem and a specification, all processes generated from the procedure and starting with the specification are convergent, and their limit is the truth of the model of the problem. In fact, this theorem is an improved version of the theorem given in [Li 92-2].

In the rest of this paper, we assume that two sentences P and Q are the same sentence iff $P \equiv Q$ (that is $(P \supset Q) \wedge (Q \supset P)$ is a tautology).

Definition 4.1 Limit

Let $\{\Gamma_n\}$ be a capture process. The sequence of formulas:

$$\Gamma^* \equiv \bigcap_{n=1}^{\infty} \bigcup_{m=n}^{\infty} \Gamma_m$$

is called the *upper limit* of the capture process $\{\Gamma_n\}$. The sequence of formulas:

$$\Gamma_* \equiv \bigcup_{n=1}^{\infty} \bigcap_{m=n}^{\infty} \Gamma_m$$

is called the *lower limit* of the capture process $\{\Gamma_n\}$. A capture process is *convergent* iff

$$\Gamma_* = \Gamma^*.$$

The lower limit (and also the upper limit) is called the *limit* of a convergent capture process $\{\Gamma_n\}$. This limit is denoted by $\lim_n \Gamma_n$.

Lemma 4.1 $A \in \Gamma^*$ iff there exist infinite many $\{k_n\}$ such that $A \in \Gamma_{k_n}$. $A \in \Gamma_*$ iff there exists an N such that $A \in \Gamma_m$ for $m > N$.

There are some capture processes which have no limit. Intuitively, such processes indicate that the specification is not an appropriate description of a problem. For example, let $\Gamma_1 = \{A\}$ where A is the statement "tossing a coin and getting tails". Obviously, any capture process starting with Γ_1 has no limit. An appropriate description should be "tossing a coin, the probability of getting tails is 50%".

Definition 4.2 *$\wp$, $L_\wp$ and $\mathbf{M}_\wp$.*
Given a specific problem $\wp$, $L_\wp$ is a subset of L containing countablly infinite (include finite) constant, function and predicate symbols representing the problem $\wp$. A user model of $\wp$ denoted by $\mathbf{M}_\wp$ is a model of $L_\wp$.

Intuitively, a user model of $\wp$ may be considered to be a description of the problem $\wp$ in the real world.

Definition 4.3 $\mathcal{T}_{M_\wp}$
For a given user model $\mathbf{M}_\wp$, $\mathcal{T}_{M_\wp}$ is the set of all sentences of $L_\wp$ valid in $\mathbf{M}_\wp$, i.e.,

$$\mathcal{T}_{M_\wp} \equiv \{A \mid A \in L_\wp,\ FV(A) = \emptyset,\ \mathbf{M}_\wp \models A\}.$$

where $FV(A)$ denotes the set of all free variables occurring in A.

According to its definition, the set $\mathcal{T}_{M_\wp}$ is countable. We represent it as a sequence $\{A_m\}$ (for example, in the lexicographical order).

The following definition gives a procedure to generate the capture process $\{\Gamma_n\}$. The basic idea is: given a user's model $\mathbf{M}_\wp$ and a specification Γ, enumerate all A_i contained in $\mathcal{T}_{M_\wp}$; if A_i is a theorem of Γ_n, then pass it; if A_i is a new law, then take it in, i.e., $\Gamma_{n+1} = \{A_i, \Gamma_n\}$; if $\neg A_i$ is a theorem, that is, $\neg A_i$ has met a user's rejection, then Γ_{n+1} is taken to be an R-reconstruction of Γ_n.

Definition 4.4 Given an user model $\mathbf{M}_\wp$ and a specification Γ, an $\mathbf{M}_\wp$ capture process of Γ denoted by $\{\Gamma_n\}$ is defined as follows:

1. $\Gamma_1 = \Gamma$. Let $\mathcal{T}_{M_\wp} = \{A_m\}$ and $A_{k_1} = A_1$. Let $\{\Theta_n\}$ be a sequence of specifications, and $\Theta_1 = \emptyset$.
2. Γ_{n+1} is defined as below:

(a) If $\Gamma_n \vdash A_{k_n}$ is provable, then put A_{k_n} into Θ_n, that is $\Theta_n := \{\Theta_n, A_{k_n}\}$, and consider $\Gamma_n \vdash A_{k_n+1}$.

(b) If neither $\Gamma_n \vdash A_{k_n}$ nor $\Gamma_n \vdash \neg A_{k_n}$ is provable, then $\Gamma_{n+1} = \{A_{k_n}, \Gamma_n\}$. Let $\Theta_{n+1} = \Theta_n$.

(c) If $\Gamma_n \vdash \neg A_{k_n}$ is provable, then do the following two steps consecutively:

i. $\Gamma_{n+1} = \{\Lambda\}$ where $\Lambda \in \mathcal{A}(\Gamma_n, \neg A_{k_n})$, and $A_{k_i} \in \Lambda$ hold for $i = 1, \cdots, n-1$. Let $\Theta_{n+1} = \Theta_n = \{C_1, ..., C_l\}$. .

ii. $\Gamma_{n+2} = \{A_{k_n}, \Gamma_{n+1}\}$. Let $j = 2$. For $i = 1$ to l do the loop: If $\Gamma_{n+j} \vdash C_i$ is provable, then do nothing. If neither $\Gamma_{n+j} \vdash C_i$ nor $\Gamma_{n+j} \vdash \neg C_i$ is provable, then delete C_i from Θ_{n+j}, and take $A_{k_{n+j}} = C_i$; let $j := j+1$ and construct Γ_{n+j} using the sub-item 2-(b).
Note: Let Γ_m be the last specification constructed by this loop, then either $A_i \in \Gamma_m$ or $\Gamma_m \vdash A_i$ holds for $i = 1, \cdots, k_m$.

It should be mentioned that when an R-reconstruction is taken in response to a user's rejection, some information may be lost. For example, let $\Gamma = \{A \wedge B\}$, we have that both $\Gamma \vdash A$ and $\Gamma \vdash B$ are provable. Assume that A has met a user's rejection, then the maximal consistent subset of Γ which is consistent with $\neg A$ is the empty set. Thus, after the R-reconstruction of Γ for A, B is missing! In order to repair the loss, in the definition we introduce a "stack" called Θ. For any n, Θ_n collects those A_m, $m < k_n$, which can be deduced from Γ_i for some $i < n$. When a user's rejection arises, we execute the sub-item 2-(c)-ii, check all A_m contained in Θ_n, and pick up the lost ones back as new laws. Since, for any n, Θ_n is always finite, the execution of sub-item 2-(c)-ii will always be terminated.

Theorem 4.1 Given an user model $\mathbf{M}_\wp$, and a specification Γ, if Γ does not contain any formula which is logically independent of $\mathcal{T}_{M_\wp}$, then every $\mathbf{M}_\wp$ capture process of Γ denoted by $\{\Gamma_n\}$ is convergent, and

$$Th(\lim_n \Gamma_n) = \mathcal{T}_{M_\wp}.$$

Proof: Let $\{\Gamma_n\}$ be an $\mathbf{M}_\wp$ capture process of Γ. We prove $Th(\lim_n \Gamma_n) = \mathcal{T}_{M_\wp}$ in the following two steps:

1. $\mathcal{T}_{M_\wp} \subseteq Th(\Gamma_*)$. For any $A_i \in \mathcal{T}_{M_\wp}$, by the construction of $\{\Gamma_n\}$, there must exist an n such that either $A_i \in Th(\Gamma_n)$ or $A_i \notin Th(\Gamma_n)$ and $A_i \in \Gamma_{n+1}$.

(a) For the first case, by definition 4.4, there must be an l such that $A_i \in Th(\Gamma_m)$ for $m \geq l$, since $\mathcal{T}_{M_p}$ is consistent. For each $m \geq l$, there is a finite subset of Γ_m denoted by $\Delta_m = \{B_{m_1}, \cdots, B_{m_j}\}$ and $\Delta_m \vdash A_i$.

For each k, $1 \leq k \leq j$, either $B_{m_k} \in \bigcap_{n=l}^{\infty} \Gamma_n$ or $(\bigcap_{n=l}^{\infty} \Gamma_n) \vdash B_{m_k}$, otherwise by definition 4.4, $B_{m_k} \in \Gamma$, and there must exist an $m' \geq m$ such that $\Gamma_{m'}$ contains $\neg B_{m_k}$, thus $B_{m_k} \notin Th(\Gamma_{m'})$ which is a contradiction. Thus $A_i \in Th(\bigcap_{m=l}^{\infty} \Gamma_m)$ holds. Therefore

$$A_i \in Th(\bigcup_{n=1}^{\infty} \bigcap_{m=n}^{\infty} \Gamma_m) = Th(\Gamma_*).$$

(b) For the second case, by the definition 4.4, $A_i \in \Gamma_m$ for any $m > n$. Thus,

$$A_i \in \bigcup_{n=1}^{\infty} \bigcap_{m=n}^{\infty} \Gamma_m = \Gamma_*.$$

2. $\Gamma^* \subseteq \mathcal{T}_{M_p}$. Assume that there is a closed formula A such that $A \in \Gamma^*$ and $A \notin \mathcal{T}_{M_p}$. There are only two possibilities:

   (a) neither $\mathcal{T}_{M_p} \vdash A$ nor $\mathcal{T}_{M_p} \vdash \neg A$ is provable, but this contradicts that Γ does not contain any logically independent formula w.r.t. $\mathcal{T}_{M_p}$.

   (b) $\neg A \in \mathcal{T}_{M_p}$. This is also impossible, if so, there must be i such that $\neg A = A_i$, then there must be n such that $\neg A \in \Gamma_n$. Thus $\neg A \in \Gamma_m$ for all $m > n$. this is $\neg A \in \Gamma^*$, a contradiction.

Thus, we have

$$\Gamma^* \subseteq \mathcal{T}_{M_p} \subseteq Th(\Gamma_*)$$

and so $\Gamma_* = \Gamma^*$. Hence, the theorem has been proved. □

It is not difficult to see that the theorem still holds without the condition: Γ does not contain logically independent laws of $\mathcal{T}_{M_p}$, because the similar techniques given in the above proof can be used to eliminate those laws from Γ. Furthermore, if we replace the sub-items 2-(a) and 2-(b) in definition 4.4 by the following sentence: if A is consistent with Γ then $\Gamma_{n+1} = \{A_{k_n}, \Gamma_n\}$, the theorem also holds.

The proof of the theorem shows that the capture processes will always converge on the same limit, if they are generated in the way that for any Γ_n which has met a user's rejection of some A, we *arbitrarily* choose a maximal consistent subset of Γ_n which is consistent with $\neg A$ and contains $A_{k_1}, \cdots, A_{k_n}$. Thus, the uniqueness of revision discussed in [Gär 88] seems unnecessary.

In practice, since a user usually may not know the mathematical model of a problem at the beginning of a capture process, the theorem cannot be applied.

However, to build a reconstruction does not need the user to know the whole model of the problem. There is a *criterion of modification*. It says that a theory must be modified if its logical consequences fail to agree with the practice; and a statement must be replaced by another one if neither itself or its negation agrees with the practice. If the user follows the criterion, then he can approach an appropriate specification of the problem by making the reconstructions given in the proof of the theorem.

We can view a program written in a language (or a block of machine codes) as a specification. If so, all the versions of a software in their given order form a capture process, since for every version, its succedent version is obtained from this version either by adding some new pieces of programs (new laws) or by correcting some mistakes (which can be decomposed to making some R-reconstructions followed by taking some N-reconstructions). Then, the above theorem says that a software will eventually approach the goal by producing the versions. Therefore, an important task for software engineers is to discover those procedures that their generated capture processes are convergent as fast as possible.

5 Computational aspects of revisions

From the definitions and theorems about new laws and ideal user's rejections, we know that the key point for building a reconstruction of specification Γ is to find a maximal consistent subset of Γ which is consistent with some given formula A. The following two theorems show that it is not an easy task.

Theorem 5.1 For propositional logic, if every specification contains finite laws (propositions) only, then building a reconstruction of a specification is an NP-hard problem.

Proof: The proof follows directly from Cook's theorem ([GJ 79]).

Theorem 5.2 For the first order language L, building a reconstruction of a specification is a undecidable problem.

Proof: It is proved by Gödel's second incompleteness theorem [Shoen 67].

In computer science, we are interested in what circumstances a reconstruction can be efficiently built. In this section, we give a simple case. Assume that the language L is restricted to be without equality $\doteq$.

Definition 5.1 $\Gamma_{-\{A\}}$ *and* $\Gamma_{+\{A\}}$.

Assume that A is contained in Γ. $\Gamma_{-\{A\}}$ is the subsequence of Γ obtained by eliminating all A_i containing any predicate symbols in common with A. $\Gamma_{+\{A\}}$ is the subsequence of Γ obtained by eliminating all A_j contained in $\Gamma_{-\{A\}}$.

Definition 5.2 Simple user's rejection

Given $\Gamma \models A$, a user's rejection is called simple iff there exists a model $\mathbf{M}$ such that

$$\mathbf{M} \models A_i \qquad \text{for all } A_i \text{ in } \Gamma_{-\{A\}} \qquad \text{and} \qquad \mathbf{M} \models \neg A.$$

Lemma 5.1 If $\Gamma \models A$ and A has met a simple user's rejection, then any R-reconstruction of Γ for A contains $\Gamma_{-\{A\}}$.

The concept of simple user's rejection allows us to apply the well-known Davis-Putnam procedure to find the maximal consistent subsets. In fact, we can use the following procedure to build a R-reconstruction:

formula: a well-formed first order formula
F-sequence: sequence of closed formals

```
procedure R-REVISE(Γ: F-sequence; A: formula; var   T: sequence);
  begin
    T := Γ_{-{A}}
    for every B contained in Γ_{+{A}} do
        if {T, ¬A, B} is consistent
        then T := {T, B}
        else skip
    endfor
  end
```

Recently, Gu claimed that he built some efficient algorithms based on optimization theory to solve SAT problem [Gu 92]. It may bring about some hope to build reconstruction practically for a large class of problems.

6 Specification revisers

At the end of section 4, we briefly discussed the role of theorem 4.1 in the software development. Since a new version of a software is always made by some editors, the theory suggests that for the different levels of specifications we need different editors. The editors which we have used are syntax directed editors and synthesizers (including type and proof editors). They can check the syntactic or static errors of the inputs automatically, and they work interactively with the user.

The theory given in this paper inspires us to build some editors for the requirement capture. Let us call them specification reviser. It checks the consistency of the newly added laws with respect to a specification which has been stored. It also detects the fallibility of a specification when a user's rejection exists, and points out the rejected laws. The editor will further provide all possible

R-reconstructions for a rejected law, and ask the user to select their preferred revision (theorem 4.1 guarantees that the user will eventually approach the goal). It allows the user to mark some laws which cannot be rejected, even if some facts contradict them. Thus, the editor will direct the user to capture an appropriate specification while keeping the consistency of revised specifications at every stage in a process. Finally, it works interactively with the user. Here, we give an outline of the main procedures of the editor called SpecReviser [LW 92]:

function MINIMAL(A: formula, Γ: F-sequence);

For a given formula A, and a consistent F-sequences Γ, the function MINIMAL outputs *all* the sets $\Gamma - \Lambda$ where $\Lambda \in \mathcal{A}(\Gamma, \neg A)$. If every such set contains marked laws and A is not marked, then Γ is not revised and outputs "A contradicts the marked laws". If every such subset contains marked laws and A is marked, then outputs "Please remark the marked laws".

```
procedure SPEC-REVISER(A: formula; var Γ: F-sequence);
  var Δ: F-sequence;
  begin
    Δ := ∅
    loop do
        begin
          input a formula A;
          if CONSISTENT(Γ, A)
            then begin
                 Γ := {Γ, A};
                 output:"A is accepted";
                 if A is marked then Δ := {Δ, A}
                 end
            else begin
                 output:"A is not consistent with Γ";
                 MINIMAL(A, Γ);
                 the user makes a choice, say Γ';
                 Γ:=Γ − Γ';
                 if A is marked then Δ := {Δ, A}
                 end;
        end
    end loop
  end
```

7 Related work

From a view point of logic, our theory (see [Li 91, 92-1 and 92-2]) is close to the *theory of knowledge in flux* given by Gärdenfors in the sense of setting up a

"theory for modeling the dynamics of epistemic states" (see [Gär 88]). For example, he defined the proof-concepts: expansions of Γ by A and the revisions of Γ by A, whose corresponding model-theoretic concepts are our N-reconstruction of Γ for a new law A (with a minor difference) and R-reconstructions of Γ for a user's rejection of A respectively. For defining the revisions, he introduced the principle of informational economy which is essentially the same as Occam's razor which we use to set up ideal user's rejection.

The main differences between Gärdenfors's theory and ours are:

1. He set up a formal theory of expansions and revisions of belief sets. In contrast, we built a theory of sequences and limits of belief sets (or specifications called in this paper).
2. He studied all the concepts in propositional logic, and define them proof-theoretically. In contrast, we study these concepts in first order logic, and define them both proof-theoretically and model-theoretically. We introduced the model-theoretic concepts such as new laws and user's rejections to describe the interactions between logical inference (logical information) and user's practice (empirical information).
3. We introduced some concepts and obtained some results which are new to his theory. For example, we defined the capture processes and the limit of a capture process, advanced to build procedures for generating capture processes, and proved that the processes, generated from the procedure in section 4, are convergent, and their limit is the truth of the user's model of the given problem. The proof of the about limit theorem shows that the uniqueness of revision needed in his theory seems unnecessary.

It should be mentioned that much work remains to be done. For example, when a user's rejection arises, there are, in practice, many ways to revise a specification . For instance, we could change the concepts, or replace the formal language or even change the paradigm of thinking. Here, we only provide a simple solution – the R-reconstructions.

Finally, it should also be pointed out that the concepts and results proved in this paper can be applied to any formal languages with a complete proof system, and that the theory has many other potential applications, such as machine learning, knowledge base maintenance and diagnostic techniques.

Acknowledgement

I would like to thank Günter Hotz and Reinhard Wilhelm for the encouragements, and thank Zhou Chaochen and referees for correcting some mistakes in the earlier versions of the paper. The work is supported by the National High-Tech Programme of China.

References

[BG 77] Burstall, R. and Goguen J.A., Putting theories together to make specifications, Proc. 5th. IJCAI Cambridge, Mass, 1045-1058 (1977).

[BJ 83] Bjørner, D., and Jones, C., Formal specification and Software Development, Prentice Hall International, 1983.

[EM 85] Ehrig, H. & Mahr, B., Fundamentals of Algebraic Specification, Springer-Verlag, 1985.

[Gall 87] Gallier, J.H., Logic for Computer Science, foundations of automatic theorem proving. John Wiley & Sons, 1987, 147-158, 162-163, 197-217.

[Gär 88] Gärdenfors, P., Knowledge in Flux, The MIT Press, 1988.

[GJ 79] Garey, M.R. and Johnson, D.S., Computers and Intractability, Freeman and Company, 1979.

[Gu 92] Gu, J., An Invited Lecture at the Institute of Computing Technology, Academia Sinica, Beijing, 1992.

[Li 91] Li, W., An Introduction to Open Logic System, Invited paper, Proceedings of International Conference for Young Computer Scientists, 1991.

[Li 92-1] Li W., Towards a Theory of Epistemic Processes, International Workshop on Automated Reasoning, IFIP WG 12.3, Beijing, North-Holland, July, 1992.

[Li 92-2] Li, W., An Open Logic System, Scientia Sinica, Series A, 10, 1992.

[LW 92] Li, W. and Wang, Y., An Introduction to Specreviser, Technical report of BUAA, CSE-92-10-5, 1992.

[NS 84] Nordström, B., and Smith, J., Propositions and specifications of programs in Martin-Löf's type theory, BIT 24 (1984), pp 288-301.

[Paul 87] Paulson, L., Logic and Computations, Cambridge University Press, 1987, 38-50.

[Shoen 67] Shoenfield, J.R., Mathematical Logic, Addison-Wesley, Reading, Mass, 1967, 74-75.

[ST 88] D. Sannella and A. Tarlecki, Toward formal development of programs from algebraic specifications: implementations revisited. Acta Informatica 25,233-281(1988).