A Logical Framework for Evolution of Specifications

Wei Li

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

Abstract

A logical framework of software evolution is built. The concepts of sequence of specifications and the limit of a sequence are established. Some concepts used in the development of specifications, such as new laws, user's rejections, and reconstructions of a specification are defined; the related theorems are proved. A procedure is given using transition systems. It generates sequences of specifications from a given user's model and an initial specification. It is proved that all sequences produced by the procedure are convergent, and their limit is the truth of the model. Some computational aspects of reconstructions are studied; an R-calculus is given to deduce a reconstruction when a specification meets a rejection. An editor called Specreviser is introduced. It is used to develop specifications. The main functions of the editor are given; some techniques used in its implementation are also discussed. Finally, the theory is compared with AGM's theory of belief revision.

1 Introduction

If we observe the history of development of a software system, we will find that the history can be described by a sequence of versions of the software system:

$$\mathbf{V}_1, \mathbf{V}_2, \cdots, \mathbf{V}_n, \cdots,$$

where the version $\mathbf{V}_{n+1}$ is obtained from $\mathbf{V}_n$ either by adding some new pieces of programs, or by correcting some errors, which is done by replacing some pieces of programs of $\mathbf{V}_n$ with some new pieces of programs.

Similarly, the history of specifications of a problem can be described by a sequence of drafts of specifications:

$$\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_n, \cdots,$$

where the draft $\mathbf{S}_{n+1}$ is obtained from $\mathbf{S}_n$ in the following way: Either clients provide some laws, or we ask some questions (propose some laws); the clients may answer the question by "Yes" or "No". The answer "Yes" means that our proposed laws are accepted and will be added into $\mathbf{S}_n$. The answer "No" means that the laws are rejected by the clients; in this case we say that the law has been rejected by facts.

To build a logical framework, let us consider the specification of a software. From the above discussion, we reach the following conclusions:

1. The history of the development of the specification of a program can be described by the sequence of versions of the specification.

2. Each specification in the sequence should be consistent; otherwise, it fails because anything could be deduced, or equivalently, no program can be synthesized from the specification.

3. When clients reject a specification, the specification has to be revised to meet the client's requirement.

4. There should be a procedure to produce the revisions of the specification in an economical way; i.e., the modification of the specification should be as less as possible, and the sequence made up by the revisions should reach an appropriate specification as fast as possible.

The purpose of this paper is to provide a logical framework for describing the history of development of the specification of a program. In section 2, we will introduce the concept of sequence of formal theories to describe the evolution of a specification, and will further introduce a concept called the limit of sequence to model the result of the evolution of a specification. In section 3, we will introduce some concepts to describe the interaction between the specifications and the users. In section 4, we will define a procedure to generate developing sequences. The procedure is defined using a transition system. We will prove that for a given user's model and a specification, all developing sequences generated from the procedure and starting with the given specification are convergent, and their limit is the laws (the set of truth) of the model. In section 5, we provide another procedure to produce reconstructions when a specification is rejected by the user. In section 6, we will give an introduction to an editor called Specreviser which is built based on the logical framework. Finally, we will compare our work with belief revision theory as given in [AGM 85].

2 Sequences and limits

We assume that the formal language which we use is a first order language L defined in [Gall 87]. We use A, Γ and $Th(\Gamma)$ to denote a formula, a sequence of formulas, and the set of all theorems deduced from Γ respectively.

A sequent is of the form $\Gamma \vdash A$. 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 when it is needed.

A model $\mathbf{M}$ is a pair $< M, I >$, where M is a domain and I is an interpretation. Sometimes, we use $\mathbf{M}_\wp$ to denote a model of a specific problem $\wp$, and use $\mathcal{T}_{M_\wp}$ to denote the set of all true sentences of $\mathbf{M}_\wp$. It is obvious that $\mathcal{T}_{M_\wp}$ is countable. We use $\mathbf{M} \models \Gamma$ to denote $\mathbf{M} \models A$ for all A contained in Γ.

The concepts of validity, satisfiability, falsifiability, provability and consistency used in this paper are defined in [Gall 87]. As we know, the proof rules are **sound and complete.**

Definition 2.1 Sequence of specifications

A finite or infinite consistent set (sequence) Γ of closed formulas is called a *specification.* The sentences contained in Γ are called laws.

$\Gamma_1, \Gamma_2, \cdots \Gamma_n, \cdots$ is called a *sequence of specifications*, or sequence for short, if for any n, Γ_n is a specification.

A sequence is increasing (or decreasing) if $\Gamma_n \subseteq \Gamma_{n+1}$ (or $\Gamma_n \supseteq \Gamma_{n+1}$) for all n; otherwise it is non-monotonic.

In terms of mathematical logic, a specification is in fact a sequence of non-logical axioms, or closed formal theories.

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 2.2 Limit of sequence

Let $\{\Gamma_n\}$ be a sequence of specifications. The set of closed formulas:

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

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

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

is called the *lower limit* of the sequence $\{\Gamma_n\}$.

A sequence of specifications is *convergent* iff $\Gamma_* = \Gamma^*$. The limit of a convergent sequence is denoted by $\lim_n \Gamma_n$ and is its lower limit (and also the upper limit).

The meaning of the definition above can be seen from the following theorem:

Lemma 2.1 1. $A \in \Gamma^*$ iff there exist infinitely many k_n such that $A \in \Gamma_{k_n}$.

2. $A \in \Gamma_*$ iff there exists an N such that $A \in \Gamma_m$ for $m > N$.

Proof. Straightforward from the definition. □

Theorem 2.1 If the sequence $\{\Gamma_n\}$ is increasing (or decreasing), then it is convergent and the limit is $\bigcup_{n=1}^{\infty} \Gamma_n$ (or $\bigcap_{n=1}^{\infty} \Gamma_n$).

Example 2.1 Increasing sequence

We have seen many increasing sequences in logic. For example, consider the Lindenbaum theorem: "Every formal theory Γ of L can be extended to a maximal theory." The proof of the theorem is given as follows: Since all sentences of L are countable, they can be listed as: $A_1, A_2, \cdots, A_n, \cdots$. We then define $\Gamma_0 = \Gamma$,

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \bigcup \{A_n\} & \text{if } \Gamma_n \text{ and } A_n \text{ are consistent} \\ \Gamma_{n+1} = \Gamma_n & \text{otherwise} \end{cases}$$

It is obvious that the sequence $\{\Gamma_n\}$ is increasing. Its limit $\bigcup_{n=0}^{\infty} \Gamma_n$ is a maximal theory.

Example 2.2 Sequence without limit

$$\Gamma_n = \begin{cases} \{A\} & n = 2k-1 \\ \{\neg A\} & n = 2k \end{cases}$$

Thus, $\Gamma^* = \{A, \neg A\}$ and $\Gamma_* = \emptyset$. The sequence $\{\Gamma_n\}$ has no limit.

Example 2.3 Random sequence

Let A denote the statement "*tossing a coin and getting tails.*" Γ_n is defined by the result of n^{th} tossing. The sequence $\{\Gamma_n\}$ is a random sequence of A and $\neg A$. Obviously, it has no limit.

Intuitively, such sequence means that the laws contained in the specification perhaps are not appropriate descriptions of the given problem. For example, an appropriate description of the above example should be "*tossing a coin, the probability of getting tails is* 50%."

The main result about the limits is the following: There exists a procedure which, for a given model and a given specification which may be inconsistent with the truth of the model, will produce sequences such that every sequence will start with the given specification, and will be convergent to the same limit which is the set of all laws (the truth) of the model.

3 New law and user's rejection

To build the procedure mentioned above, we need the following concepts:

Definition 3.1 New laws.

A is called a new law for Γ iff there exist two models $\mathbf{M}$ and $\mathbf{M}'$ such that

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

Theorem 3.1 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: The proof is straightforward from soundness and completeness. □

We use $\Gamma \models A$ to denote that A is a semantic consequence of Γ. It means that for any model $\mathbf{M}$, if $\mathbf{M} \models \Gamma$ then $\mathbf{M} \models A$.

The concept of user's rejection is given below:

Definition 3.2 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 $\Gamma_{M(A)} \equiv \{A_i \mid A_i \in \Gamma, \quad \mathbf{M} \models A_i, \quad \mathbf{M} \models \neg A\}$.

$\mathbf{M}$ is called an *ideal* user's rejection of A iff $\Gamma_{M(A)}$ is *maximal* in the sense that there does not exist another user's rejection of A, $\mathbf{M}'$, such that $\Gamma_{M(A)} \subset \Gamma_{M'(A)}$. An ideal user's rejection of A is denoted by $\overline{\mathbf{M}}(A)$.

Since there may exist many ideal rejections of A by facts, we define

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

The user's rejection meets the intuition that whether a specification is acceptable, depends only on whether all its deduced results agree with user's requirements which have nothing to do with the logical inference. In fact, this is the reason that sometimes, we call our theory *open logic* [Li 92].

The ideal user's rejection satisfies the Occam's razor, which says: "*Entities are not to be multiplied beyond necessity.*" Here, it means that if some particular consequence deduced from a specification is rejected, then only the smallest set of laws (contained in the specification) which cause the rejection has to be rectified, and the rest of the laws (a maximal subset) is retained and is assumed to be temporarily correct.

Definition 3.3 Acceptable modification
Let $\Gamma \vdash A$. An acceptable modification Λ of Γ by $\neg A$ is a maximal subset of Γ with the property that Λ is consistent with $\neg A$.
Let $\mathcal{A}(\Gamma, A)$ be the set of all acceptable modifications of $\neg A$.

Theorem 3.2 $\mathcal{A}(\Gamma, A) = \mathcal{R}(\Gamma, A)$.

Proof: Let us prove $\Rightarrow$.
Suppose $\Lambda \in \mathcal{A}(\Gamma, A)$. It is consistent with $\neg A$, so there is a model $\mathbf{M}'$ such that $\mathbf{M}' \models \Lambda$ and $\mathbf{M}' \models \neg A$. Thus, $\mathbf{M}'$ is a rejection of A by facts. $\mathbf{M}'$ is maximal, since if there exists another $\mathbf{M}''$ such that $\mathbf{M}'' \models \neg A$ and $\Gamma_{M''(A)} \supset \Gamma_{M'(A)}$, then $\Gamma_{M''(A)} \in \mathcal{A}(\Gamma, A)$; but this is impossible. □

Example 3.1 Let $\Gamma \equiv \{A, A \supset B, B \supset C, E \supset F\}$
We have $\Gamma \vdash C$. The $\mathcal{A}(\Gamma, C)$ consists of:

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

The following definition tells us how to reconstruct a specification, when we meet a new law or a user's rejection.

Definition 3.4 Reconstruction
Let A be a theorem of Γ. An E-reconstruction of Γ for the theorem A is Γ itself.
Let A be a new law for Γ. An N-reconstruction of Γ for the new law A is the sequence $\{\Gamma, A\}$.
Let $\Gamma \models A$ and A be rejected by the user. An R-reconstruction of Γ for the user's rejection of A is Δ, where $\Delta \in \mathcal{R}(\Gamma, A)$.
Γ' is a reconstruction of Γ iff Γ' is an E- or an N- or an R-reconstruction.

It is obvious that R-reconstruction is not unique. If Δ is an R-reconstruction of Γ for a user's rejection of A, then Δ is a maximal subset of Γ and is consistent with $\neg A$.
The N-reconstruction and R-reconstruction are similar to the expansion and maxichoice contraction in [AGM 85] respectively. The minor difference is that all the concepts in AGM theory are proof-theoretic and are defined for the logical closure $Th(\Gamma)$ (called *belief sets* in [AGM 85]). In contrast, the concepts given here are model-theoretic, and defined for a specification (formal theory) Γ. The key difference is that we are interested in how to build the convergent sequences.

4 The limit of developing processes

Having given the general concepts, we study the problem of how to describe the evolution of specification of a program.

Definition 4.1 Developing process
A sequence of specifications $\Gamma_1, \Gamma_2, \cdots, \Gamma_n, \cdots$ is a developing process, if Γ_{i+1} is a reconstruction of Γ_i for $i \geq 1$.

It should be mentioned that if $\mathcal{A}(\Gamma_n, A)$ contains more than one element, then there are many R-reconstructions of Γ_n. Thus, the evolution of a specification should be represented by a tree, each branch of which is a developing process.

Theorem 4.1 1. A developing process $\{\Gamma_n\}$ is increasing (or decreasing) iff for all $n \geq 1$, Γ_{n+1} is an N-reconstruction (or R-reconstruction) of Γ_n.

2. A developing process is non-monotonic iff N- and R-reconstructions occur alternatively.

Proof: Straightforward from the definition. □

Let us now give the procedure mentioned above. We assume that

1. $\mathcal{T}_0$ is a given countable consistent set of sentences which we accept, and is denoted by $\{A_m\}$.

2. Γ is a given specification. It may be inconsistent with $\mathcal{T}_0$, and is to be used as the initial specification of the developing process.

The basic idea of building the procedure is: Take $\Gamma_1 = \Gamma$. Γ_{n+1} is constructed recursively as follows: If $\Gamma_n \vdash A_i$, then take $\Gamma_{n+1} = \Gamma_n$; if A_i is a new law for Γ_n, then Γ_{n+1} is the N-reconstruction of Γ_n for A_i, i.e., $\{A_i, \Gamma_n\}$; if $\Gamma_n \vdash \neg A_i$, that is, $\neg A_i$ has met a user's rejection (A_i is to be accepted), then Γ_{n+1} is an R-reconstruction of Γ_n.

When an R-reconstruction is taken in response to a user's rejection in the n^{th} stage of the developing process, there are two things which we should notice:

1. Only those R-reconstructions containing all new laws accepted before the n^{th} stage are interesting. We introduce a sequence Δ to store the accepted new laws for the steps concerning N-reconstructions.

2. Some information may be lost. For example, consider $\Gamma = \{A \wedge B\}$, both $\Gamma \vdash A$ and $\Gamma \vdash B$ are provable. Assume that A has met a user's rejection, then the maximal 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, we introduce a sequence Θ to collect those A_m for the steps concerning E-reconstructions. After each R-reconstruction, the procedure checks all A_m contained in Θ, and picks up the lost ones back as new laws. Since, at any developing stage j, Θ is always finite, the checking will be terminated.

In order to make the notation easy to understand, we describe the procedure using a transition system [Plo 82] and [Li 82]. We introduce the quadruple:

$$< \mathcal{T}, \Gamma, \Theta, \Delta >$$

to denote the configuration. $head(\mathcal{T})$ and $tail(\mathcal{T})$ are defined as usual. We introduce the operator $\star$ to denote the concatenation of a finite sequence with another (finite or infinite) sequence:

$(A_1, \cdots, A_n) \star (B_1, B_2, \cdots, B_n, \cdots) \equiv (A_1, \cdots, A_n, B_1, B_2, \cdots, B_n, \cdots)$

Definition 4.2 Procedure

Let the initial state of the configuration be

$$< \mathcal{T}_0, \Gamma, \emptyset, \emptyset > .$$

Let $\Gamma_1 = \Gamma$. Γ_{n+1} is defined by the following three rules recursively:

$$\frac{\Gamma \vdash head(\mathcal{T})}{< \mathcal{T}, \Gamma, \Theta, \Delta > \longrightarrow < tail(\mathcal{T}), \Gamma, \Theta * \{head(\mathcal{T})\}, \Delta >} \quad (1)$$

$$\frac{\Gamma \nvdash head(\mathcal{T}) \quad \Gamma \nvdash \neg head(\mathcal{T})}{< \mathcal{T}, \Gamma, \Theta, \Delta > \longrightarrow < tail(\mathcal{T}), head(\mathcal{T}) * \Gamma, \Theta, \Delta * \{head(\mathcal{T})\} >} \quad (2)$$

$$\frac{\Gamma \vdash \neg head(\mathcal{T}) \quad \Delta \subset \Lambda \quad \Lambda \in \mathcal{A}(\Gamma, \neg head(\mathcal{T}))}{< \mathcal{T}, \Lambda, \Theta, \Delta > \longrightarrow < \{head(\mathcal{T})\} * \Theta * tail(\mathcal{T}), \Lambda, \Theta, \Delta >} \quad (3)$$

The sequence $\{\Gamma_n\}$ is called a developing process of $\mathcal{T}_0$ and Γ if it is generated by the above procedure.

Theorem 4.2 Let $\mathbf{M}_\varphi$ be a given model, and let Γ be a specification. Every developing process of $\mathcal{T}_{M_\varphi}$ and Γ denoted by $\{\Gamma_n\}$ is convergent. and

$$\lim_{n \to \infty} Th(\Gamma_n) = \mathcal{T}_{M_\varphi}.$$

Proof: Let us prove the case that Γ does not contain logically independent laws of $\mathcal{T}$. Techniques similar to those given in the following proof can be used to prove the theorem without this restriction.

Let $\{\Gamma_n\}$ be a developing process of $\mathbf{M}_\varphi$ and Γ. We prove $\lim_n Th(\Gamma_n) = \mathcal{T}_{M_\varphi}$ in the following two steps:

1. $\mathcal{T}_{M_\varphi} \subseteq (Th(\Gamma))_*)$. For any $A_i \in \mathcal{T}_{M_\varphi}$, 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 4.2, there must be an l such that $A_i \in Th(\Gamma_m)$ for $m \geq l$, since $\mathcal{T}_{M_\varphi}$ 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 4.2, $B_{m_k} \in \Gamma$, 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 $A_i \in \bigcap_{m=l}^{\infty} Th(\Gamma_m)$ holds. Therefore

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

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

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

(b) $\neg A \in \mathcal{T}_{M_\varphi}$. 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

$$Th(\Gamma)^* \subseteq \mathcal{T}_{M_\varphi} \subseteq Th(\Gamma)_*$$

and so $Th(\Gamma)_* = Th(\Gamma)^*$. Hence, the theorem has been proved. □

The $\mathcal{T}_{M_\varphi}$ can be viewed as the set of laws characterizing a specific problem $\mathbf{M}_\varphi$ in the real world; then the above theorem says that we can start from any given specification (which may be inconsistent with $\mathbf{M}_\varphi$) and produce versions of specifications using the above procedure; all these sequences generated are convergent to the set of laws characterizing the problem.

The procedure also shows that when we deal with a user's rejection of A at any stage n, we can **arbitrarily** choose one maximal set of Γ_n which is consistent with $\neg A$. If it contains Δ which is the set of new laws $A_k \in \mathcal{T}_{M_\varphi}$ accepted in the 1^{st} to n^{th} stage, then the developing process will always converge to $\mathcal{T}_{M_\varphi}$. Therefore, we do not need the unique maximal consistent subset (which is required in [Gär 88]).

5 A calculus of R-reconstruction

From the definition given in section 3, we know that the key point of building the reconstructions for a specification Γ is to find the maximal subsets of Γ which are consistent with some given formula A. The following two theorems show that this is not an easy task.

Theorem 5.1 1. If a specification Γ contains finite propositions only, then building a reconstruction of Γ is an NP-hard problem.

2. If a specification Γ is a set of first order formulas, then building a reconstruction of Γ is an undecidable problem.

Proof: The first is directly from Cook's theorem [GJ 79]; the second is from Gödel's second incompleteness theorem [Shoen 67]. □

More detailed results about complexity for propositional logic can be found from [EG 92].

According to the theorem in the last section, when $\Gamma \vdash A$ and A has met a user's rejection, any R-reconstruction of Γ for A (containing Δ) can be chosen, and the limit of developing process does not changed. In this section we define a calculus to produce one R-reconstruction. It is called R-calculus, and is defined using transformation rules. The purpose of the rules is to delete the sentences in Γ which contradict $\neg A$. We assume that Γ contains finite sentences, and use the form:

$$\Delta | \Gamma$$

to denote a configuration which is read as Δ overrides Γ. In particular, if $\Delta = A$, then it means that either $\Gamma \vdash \neg A$ and $\neg A$ has met a user's rejection (i.e, A has to be accepted), or A and Γ are consistent. We use

$$\Delta \mid \Gamma \Longrightarrow \Delta' \mid \Gamma'$$

to mean that the configuration $\Delta \mid \Gamma$ is transformed to $\Delta' \mid \Gamma'$. $\Longrightarrow^*$ is used to denote the transition closure as usual. In particular, the form

$$\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma$$

means that $\Delta \mid \Gamma, A$ is transformed to $\Delta \mid \Gamma$, and A is called a deleted formula in the transformation. In this case, A contradicts Δ. The rules of R-calculus are given below:

Definition 5.1 R-calculus

Structural rules

Contraction

$$\Delta \mid A, A, \Gamma \Longrightarrow \Delta \mid A, \Gamma \qquad A, A, \Delta \mid \Gamma \Longrightarrow A, \Delta \mid \Gamma$$

Exchange

$$\Delta \mid A, B, \Gamma \Longrightarrow \Delta \mid B, A, \Gamma \qquad A, B, \Delta \mid \Gamma \Longrightarrow B, A, \Delta \mid \Gamma$$

Logical rules

R-$\wedge$ left rule:

$$A \wedge B, \Delta \mid \Gamma \Longrightarrow A, B, \Delta \mid \Gamma$$

R-$\wedge$ right rule:

$$\frac{\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \wedge B, \Gamma \Longrightarrow \Delta \mid \Gamma} \qquad \frac{\Delta \mid B, \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \wedge B, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

R-$\vee$ left rule:

$$A \vee B, \Delta \mid \Gamma \Longrightarrow A, \Delta \mid \Gamma \qquad A \vee B, \Delta \mid \Gamma \Longrightarrow B, \Delta \mid \Gamma$$

R-$\vee$ right rule:

$$\frac{\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma \qquad \Delta \mid B, \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \vee B, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

R-$\supset$ left rule:

$$A \supset B, \Delta \mid \Gamma \Longrightarrow \neg A, \Delta \mid \Gamma \qquad A \supset B, \Delta \mid \Gamma \Longrightarrow B, \Delta \mid \Gamma$$

R-$\supset$ right rule:

$$\frac{\Delta \mid \neg A, \Gamma \Longrightarrow \Delta \mid \Gamma \qquad \Delta \mid B, \Gamma \Longrightarrow \Gamma}{\Delta \mid A \supset B, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

R-$\neg$ left rule:

$$\neg A, \Delta \mid A, \Gamma \Longrightarrow \neg A, \Delta \mid \Gamma$$

R-$\neg$ right rule:

$$A, \Delta \mid \neg A, \Gamma \Longrightarrow A, \Delta \mid \Gamma$$

R-$\forall$ left rule:

$$\forall x.A, \Delta \mid \Gamma \Longrightarrow A[y/x], \forall x.A, \Delta \mid \Gamma$$

R-$\forall$ right rule:

$$\frac{\Delta \mid A[t/x], \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid \forall x.A, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

R-$\exists$ left rule:

$$\exists x.A, \Delta \mid \Gamma \Longrightarrow A[t/x], \exists x.A, \Delta \mid \Gamma$$

R-$\exists$ right rule:

$$\frac{\Delta \mid A[y/x], \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid \exists x.A, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

In the quantifier rules, x is any variable and y is any variable free for x in A and not free in A unless $y = x$ ($y \notin FV(A) - \{x\}$). The term t is any term free for x in A. In both R-$\forall$ left and R-$\exists$ right rules, the variables y does not occur free in the lower sequent.

R-$\neg$ substitution rules:

$$A, \Delta \mid \Gamma \Longrightarrow A', \Delta \mid \Gamma \qquad \Delta \mid A', \Gamma \Longrightarrow \Delta \mid A, \Gamma$$

In the $\neg$ substitution rules, A and A' are defined below:

A	$\neg(B \wedge C)$	$\neg(B \vee C)$	$\neg\neg B$	$\neg(B \supset C)$	$\neg\forall x.B$	$\neg\exists x.B$
A'	$\neg B \vee \neg C$	$\neg B \wedge \neg C$	B	$B \wedge \neg C$	$\exists x.\neg B$	$\forall x.\neg B$

R-Γ consistency rule:

$$\frac{\Gamma \vdash \neg A}{\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma}.$$

$\Delta \mid \Gamma$ is called a terminated configuration if no one of the above rules can be applied to.

Informally, the right rules are used to delete those formulas which occur in the right hand side of a configuration and contains a formula occurring in the left side of the configuration as a component; the left rules are used to decompose the formulas occurring in the left hand side of a configuration. For the R-calculus, we can prove the following theorem:

Theorem 5.2 If $A \mid \Gamma \Longrightarrow^* \Delta' \mid \Gamma'$ and $\Delta' \mid \Gamma'$ is a terminated configuration, then Γ' is an R-reconstruction of Γ for A.

Proof: The theorem can be proved using the techniques given in chapter 5 of [Gal 87], since the theorem is, in fact, talking about a kind of completeness. We omit the proof since it is too long.

Let us study some examples to see how to use the rules.

Example 5.1 Consider the example given in section 3. Let

$$\Gamma \equiv \{A, A \supset B, B \supset C, E \supset F\} \text{ and } \Gamma' \equiv \{A, A \supset B, E \supset F\}.$$

$\Gamma \vdash C$ is provable. Assume that C has met a user's rejection. According to the definition, Γ' is an R-reconstruction of Γ. Using the R-calculus we have:

$$\frac{\neg C \mid \neg B, \Gamma' \Longrightarrow^3 \neg C \mid \Gamma' \qquad \neg C | C, \Gamma' \Longrightarrow^2 \neg C | \Gamma'}{\neg C | B \supset C, \Gamma' \Longrightarrow^1 \neg C | \Gamma'}.$$

where $\Longrightarrow^1$ is by R-$\supset$ right rule; $\Longrightarrow^2$ is by the Axiom; and $\Longrightarrow^3$ is by R-Γ consistency since $\Gamma' \vdash \neg\neg B$.

Example 5.2 Let $\Gamma \equiv \forall x.A(x), \Gamma'$. It is proved that

$$\Gamma \vdash \neg\exists x.\neg A(x).$$

Assume that $\neg\exists x.\neg A(x)$ has been rejected, i.e., We have to accept $\exists x.\neg A(x)$. Using R-calculus we have:

$$\exists x.\neg A(x) | \forall x.A(x), \Gamma' \Longrightarrow^1 \neg A(t/x), \exists x.\neg A(x) | \forall x.A(x), \Gamma'$$

and

$$\frac{\neg A(t/x), \exists x.\neg A(x) | A(t/x), \Gamma' \Longrightarrow^3 \neg A(t/x), \exists x.\neg A(x) | \Gamma'}{\neg A(t/x), \exists x.\neg A(x) | \forall x.A(x), \Gamma' \Longrightarrow^2 \neg A(t/x), \exists x.\neg A(x) | \Gamma'}$$

Where $\Longrightarrow^1$ is by R-$\exists$ left rule, $\Longrightarrow^2$ by R-$\forall$ right rule, and $\Longrightarrow^3$ by the Axiom.

As we mentioned, using the above R-calculus, only *one* R-reconstruction can be deduced. For example, we can deduce the R-reconstructions $\{A, A \supset B, E \supset F\}$, but not the other two: $\{A, B \supset C, E \supset F\}$ and $\{A \supset B, B \supset C, E \supset F\}$ given in Example 3.1.

To obtain all R-reconstructions, we believe that a new rule has to be introduced. To do so, we need to define an order:

Definition 5.2 For every right rule of R-calculus, the formula (in the lower transition) to which the rule is applied is called the principle formula, the formulas introduced in the upper transitions are called the side formulas. An order $\prec$ is defined recursively as below:

1. If P is a side formula and Q is the principle formula in an instance of some right rule of R-calculus, then $P \prec Q$.
2. If $P \prec Q$ and $Q \prec R$, then $P \prec R$.

The new rule is defined as:

R-selection rule

$$\frac{\Delta \mid P, Q, \Gamma \Longrightarrow \Delta \mid Q, \Gamma \quad Q \prec P}{\Delta \mid P, Q, \Gamma \Longrightarrow \Delta \mid P, \Gamma}$$

Having defined the R-selection rule, we have naturally reached the following conjecture:

> *For a give $A \mid \Gamma$ where $\Gamma \vdash \neg A$ and A has to be accepted, using R-calculus and R-selection rule we can deduce all R-reconstructions.*

In fact, the order $\prec$ satisfies the entrenchment relation given by Gärdenfors and Makinson. The R-calculus, the order $\prec$ and the R-selection rule together make the R-reconstructions deducible.

6 The Specreviser

There is a new editor called Specreviser which is built based on the above theory [Li 93]. A prototype of the editor has been implemented on a Sun workstation. To explain its functions, let us study the following example:

Example 6.1 Alarm control

Assume that we have acquired the laws of an Alarm control from the clients. It includes four laws:

1. An Alarm will be raised within 20 seconds, when the device senses an abnormal condition.
2. The alarm will continue as long as it is abnormal.
3. The alarm lasts for at least 3 seconds.
4. The alarm will stop in 5 seconds after the device is recovered.

Our task is to write a specification for these laws. We assume that the first and the fourth items are crucial. We call them **marked** laws. This means that they cannot be changed even if they contradict with some other laws. For simplicity, we use:

$ab(x)$ to denote that at time x the device is abnormal, and
$raisealarm(x)$ to denote that at time x alarm is raised, and
$workalarm(x)$ to denote that at time x the alarm is active,
$(x \leq t \leq x + y)$ to denote $(x \leq t) \wedge (t \leq x + y)$, and $t \in [x, y)$ denotes $x \leq t < y$.

The user's first step in developing a specification might include the following (not necessarily correct) representations of these laws:

1. The first requirement:

$$(1^*). \qquad \forall x.\exists y.y < x \wedge \forall t.(y \leq t < x \wedge \neg ab(t)) \wedge ab(x) \supset (\exists z.((x \leq z \leq x + 20) \wedge raisealarm(z)).$$

This formula says that: 1). At time x the device becomes abnormal. 2). There exists time $y < x$, such that in the period $[y, x)$ the device was normal. 3). Alarm will be raised in the period $[x, x + 20]$. 4). It is a **marked** requirement denoted by a *!

2. The second requirement:

   (2^*). $\forall t.workalarm(t) \supset ab(t)$.

   This formula means that abnormal status of the device is a necessary condition of working status of the alarm.

3. The third requirement:

   (3^*). $\forall x.raisealarm(x) \supset \forall t.(x \leq t \leq x + 3 \supset workalarm(t))$

   It means that: 1).If at time x the alarm is raised. 2).then in the time interval $[x, x + 3]$ the alarm works. 3).This requirement is **marked** by a *.

Having written the third law, we may think that our specification by now is working well. In fact, a careful reader has already found that the specification **fails** because law (3^*) contradicts with law (2)!

The trouble can be seen more clearly through the following counter example: Assume that at time t_1 the device became abnormal; at time $t_2 = t_1 + 5$ the alarm is raised; and then at time $t_3 = t_1 + 6$ the device is recovered. Then, according to the third law, in the time interval $[t_1 + 6, t_1 + 8]$, the alarm is working, meanwhile the device is normal.

Since the third law is **marked**, we have to change the formula representing of the second law. It could be:

$$(2') \qquad \begin{array}{l} \forall x.ab(x) \wedge (\exists t.x < t \wedge raisealarm(t) \supset \\ \exists z.\forall y.(t < y \leq z \wedge ab(y) \wedge workalarm(y))) \end{array}$$

It says that: 1) If at x the device is abnormal and at time $t > x$ the alarm is raised; 2) then there exists $z > t$ such that in the time interval $[t, z]$ the alarm does not stop as long as the device is abnormal; Law $2'$ is consistent with laws 1 and 3.

This example shows that when we build a specification, usually it is very difficult for a user to check whether his specification is consistent, especially when the specification is large — for example, it contains more than 300 laws. In fact, there is a need to build some software tools to **interactively** check the consistency of the specification in every stage of its development.

Our Specreviser is a software system to meet the need. It works like an editor. The new versions of the specification are produced by Specreviser.

Like syntax-directed editors and synthesizers (including type and proof editors), the Specreviser works interactively with the user. It not only points out the syntactic or static errors of the law entered, but also checks the consistency of the newly added laws with respect to the specification which has already been stored.

The Specreviser detects the fallibility of the specification stored when a user's rejection arises. It will point out all the minimal subsets of laws which cause the rejection.

The Specreviser will further provide all possible reconstructions to cope with the user's rejection, and ask the user to select their preferred revision.

We have proved that a user can choose one any of the possible solutions, and the revised specification will eventually reach all the laws characterizing the problem. The difference between an expert and inexperienced user is that the latter may take more steps to reach the goal.

The Specreviser allows the user to mark some laws to mean that they are crucial and cannot be rejected. If a newly added law contradicts some marked laws, then it cannot be added into the specification; if it is also marked, then the Specreviser will point out the contradiction and ask the user to re-mark the laws.

The Specreviser will direct the user in creating an appropriate specification while maintaining the consistency of the specification at each stage of the developing process.

In practice, a user usually may not know the mathematical model of a problem at the beginning of a developing process. However, to build a reconstruction does not require the user to know the whole model of the problem. The famous Chinese Philosopher Confucius claimed a criterion which says that "*truth comes from practice.*" Here it means that a theory must be modified if its logical consequences fail to agree with our observations and experiments; and a statement must be replaced by another if neither itself nor its negation agrees with practice. If one follows this criterion, then he builds his model during development and eventually reaches an appropriate specification by making the reconstructions.

7 Related work

As we have mentioned that the concepts which are similar to the reconstructions were first given in the AGM theory ([AGM 85]), where they call them expansion, contraction and revision. The differences between their approach and ours are the following:

1. AGM focused on setting up a formal theory of revisions of belief sets, i.e., the postulates of revisions. In contrast, our goal is to build a theory of *sequences* of formal theories to model the evolution of knowledge. The belief set is a special specification satisfying $\Gamma = Th(\Gamma)$.
2. AGM introduced the proof-theoretic concepts of expansion, contraction, and revision and studied their properties. In contrast, we use the model-theoretic concepts such as new law and rejection by facts to describe the interactions between logical inference (logical information) and human experience (empirical information). We also set up the relations between the model-theoretic concepts and the proof-theoretic concepts.
3. We introduced the concept of limit of sequences to model the result of the evolution of specifications. We gave a procedure to build developing processes from a given model and a specification. We further proved that the developing processes generated from the procedure are started at the specification and are convergent to the truth of the model.
4. The theorem shows that *at any stage* n of a developing process, we can choose any R-reconstruction (containing Δ) and the process will always have the same limit. Therefore, the unique revision (R-reconstruction) required by AGM is not necessary. A precise description of the uniqueness

of revisions may be expressed as: For a given model and a specification, **the limit** of all developing processes generated by the procedure is unique.

5. We have introduced the R-calculus to deduce the R-reconstructions when a specification meets a rejection.

Finally, it should be pointed out that the concepts and results given in this paper can be applied to any formal languages with a complete proof system; and provide a framework for knowledge base maintenance, machine learning, software engineering, and diagnostic techniques.

Acknowledgement

I would like to thank Zhang Yuping for helpful discussions. The work is supported by Chinese Science Foundation and Chinese High-Tech Programme.

References

[AGM 85] Alchourrón, C.E., Gärdenfors, R. and Makinson, D., On the logic of theory change: partial meet contraction and revision functions, The Journal of Symbolic Logic, Vol.50, No.2, June, 1985.

[EG 92] Eiter, T. and Gottlob, G., On the complexity of propositional knowledge base revision, Artificial Intelligence, Vol. 57, October, 1992.

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

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

[Li 92] Li, W., An Open Logic System, Scientia Sinica, Series A, Oct, 1992 in Chinese, March, 1993 in English.

[Li 93] Li W., A Theory of Requirement Capture and its Applications, TAPSOFT'93, LNCS 668, 1993, Springer-Verlag.

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

[Plo 82] Plotkin, G., An structural approach to operational semantics, Lecture notes, Arhus University, Denmark, 1982.

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