基于设计决定的逐步求精方法及环境*

缪旭 唐稚松

(中国科学院软件研究所)

A METHODOLOGY AND AN ENVIRONMENT FOR STEPWISE REFINEMENT-ACCORDING TO DESIGN DECISIONS.

Miao Xu and Tang Zhisong

(Institute of Software, Academia Sinica)

ABSTRACT

The program development preocess of stepwise specification, transformation and verification consists of two transitions. One is from the informal requirement to the formal specificaiton, the other is from the formal specification to the algorithmic implementation. The key point in the development process is how to make design decisions. In order to maintain and adapt software and to verify, not only should the specification and the implementation be recorded, but also the design decisions made during the development. We give two examples to demonstrate this methodology and how the design decisions take role in it. An environment which includes several subsystems called SPEC, PROT, VERI, and OOMM is implemented to support the above process.

摘 要

逐步描述、变换及证明的软件开发过程包含两个转换，一是从非形式的用户需求到形式描述，一是从形式描述到算法实现。开发过程中的关键是如何做出设计决定。为了更好地维护、重用软件以及程序证明，不仅仅要对软件的形式描述及实现做文档记录，也要记下开发过程中所做的每一步决定。我

* 1989 年8 月26 日收到。

们用两个例子来说明如上这种逐步求精的方法以及设计决定在其中所起的作用，并且我们实现了一个包括SPEC、OOMM、PROT、VERI几个子系统组成的环境来支持上述过程。

§1. 引 言

在设计一个软件系统的过程中，一般来讲，人们无法立刻集中到实现的细节上。即使能够一次编出程序，一次性的程序设计也有许多弊端。一个问题是，这样大的系统很难运转、检查及证明；另一个问题是，时常是得到的产品并不真正解决原来的问题，所解决的，并不是所指定的。

人们很早就认识到了这些弊端，并且产生了逐步求精这一思想，或者叫“自顶向下的程序设计”更广泛的意义下称为“软件工程”。逐步求精的方法提倡把软件开发的整个过程划分为许多小的步骤，从模糊的非形式需求开始到精确定义(但不包含实现细节)的形式描述，再到包含所有实现细节的算法程序，这个过程是逐步设计决定的结果。本文给出了几个例子来说明逐步求精的方法，从下面的例子中我们也可以看出，精化过程的中心点，是如何做出设计决定(design decision)。

本文所有工作都是在XYZ系统内开展的。XYZ系统的一个特殊性质是它的核心语言XYZ/E既能够在统一时序逻辑框架下表示抽象描述，又能够表示算法实现。所有这些都是在统一的时序逻辑的框架下，因此，关于一致性的检查，即每一步能够满足它的上一步，也是在同一个框架之下。除此之外，我们还提出了一种我们称之为XYZ/SDL(SDL 即Specification Design Language) 的语言，这是一种PDL形式的语言，它使表示精化过程中的每一步更加容易。

另外，本文还实现了一个称之为SPEC(Specification) 的环境来支持上述基于设计决定的逐步求精方法。

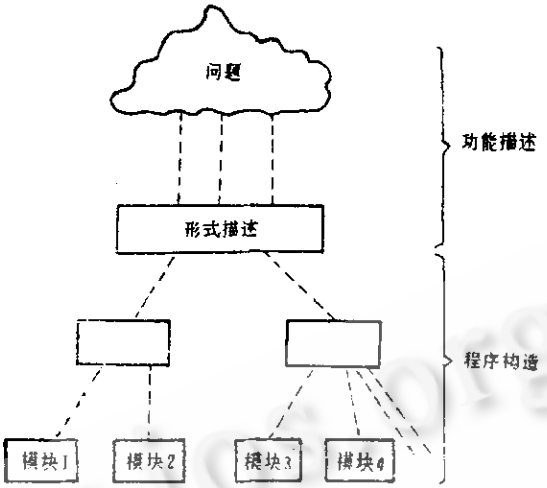
§2. 逐步求精——设计决定(DESIGN DECISIONS) 及支持工具

现在我们来讨论程序开发过程中的实质问题。

在遇到一个问题之后，人们通常的思维方式是，首先抓住主要的或者说最明显的特性：系统要做什么、完成什么样的功能，很可能在这一层次上，需求的描述是非形式和不精确的；逐渐地，人们可以思考得更加完善精确，最后得到所设计的系统精确的形式描述。在得到一个精确定义的形式描述(依旧不包含任何实现细节)之后，人们便开始思考如何实现指定的问题。

事实上，这是一个依据设计决定的实现细节的逐步增加过程，每一次的设计决定使得实现更精确，更能够有效执行，最终得到一个包括所有细节的可有效执行的程序。

综上所述，我们可以看出，软件开发的整个过程可以认为有两个主要的步骤：一是从用户需求到功能描述，这是一个把需求的非形式描述转为形式描



述的过程，我们称之为功能描述的过程；另一个是从形式描述到有效实现，我们称之为程序构造。

2.1 设计决定 (design decisions)

在得到了一个需求的精确描述之后，人们开始设计一个能够满足所给条件的系统。这里的关键问题是如何做出设计决定。

在整个精化的过程中，设计者要做出许多设计决定，这些设计决定是设计过程中的重要信息，它们是弄清最终的实现如何与起初的需求描述相联系的关键，因此，设计决定是一个中心问题。

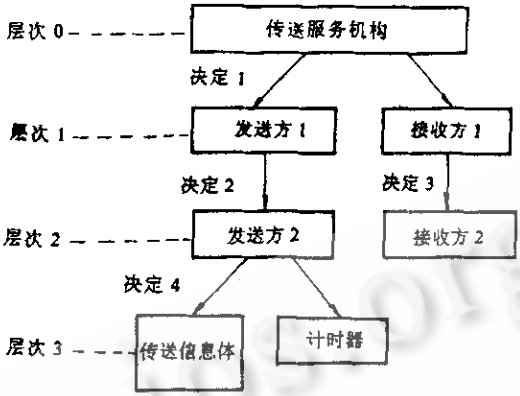
然而，现在的程序设计技术并不注重对于设计过程中所作的设计决定这个关键问题的描述和记录。这些设计信息的丢失导致了许多问题，如因此所带来的难以理解程序如何实现了所需功能以及维护和证明中的重重困难。

我们注重的不仅仅是对问题的描述和实现，更注重设计决定、实现是如何产生的。

总的来说，设计决定主要有两种：一种是在设计过程中把一个系统分为几个子系统的决定，我们称之为分解决定(Separation decisions)。

一个例子是，为了实现发送方的服务功能，并且所给的条件是传送线有时可能发生错误，这样发送方有时需要重传某些信息。显然，发送方不能每一次都重传信息，因此需要知道何时需重传，于是我们设置了一个计时器。这样发送方就分为两部分：一个计时器，一个传送体。

另一种设计决定是，并未做任何分解，而只是对上一步做一些精化，我们称之为“精化决定”。例如在AB协议中，有必要识别每一个信息，因此建立一个“顺序标号域，达到某一精化阶段时，人们发现，用一个无限的标号域是一个浪费，有限的标号域(如“1-2”)也能奏效，因此人们决定用循环的顺序标号域。这就是一种精化决定。



决定1: 分解决定——把整个系统分为一个发送方和一个接收方。

决定2、决定3: 精化决定——发送方2(接收方2)与发送方1(接收方1)会有一些不同，但它能实现所有发送方1(接收方1)能实现的。

决定4: 分解决定——把发送方分解为一个计时器和一个传送信息体。

2.2 支持工具

在逐步精化过程中有两大阶段: 一是从用户需求的非形式描述到所设计系统的形式描述的转换，另一个是从抽象描述到有效的算法实现的转换。

在这里重要的是如何作出这两个转换——从非形式到形式、从抽象到算法。由于缺乏适当的方法导致了许多障碍，使层次设计的过程不自然，也使层次之间的一致性证明更加困难。

因此，需要一种从非形式不断受到形式的精确的转换方法，以及支持这一方法的工具，并且还需要一种能够表示每一个层次的表示工具(符号)。

我们提供了几种工具:

对于任何一个问题的描述都应是一个过程，就是说对一个问题的描述也要经过若干步。许多人抱怨对于问题的形式描述难以读懂，并且在理解上出现二义性。给出一个逐步的描述过程正是要解决这些问题，并且还有助于得到一个正确的形式描述。

在功能的描述阶段中，需要一种能容易的代表非形式的需求和形式描述的符号。XYZ/SDL是一种PDL形式的以XYZ/E语言的合式公式为结构的结构编辑器。它是形式和非形式的结合。用XYZ/SDL我们能够容易地表示精化过程中的每一步，因此是描述问题过程中的有用工具。

SPEC是一个支持逐步精化过程的环境，这个环境提供精化过程中文件的管理，也提供两个专门用于描述每一步的工具。SPEC提供两种形式的文件来进行永久的存贮，一种是通常意义下的文字文件来存贮最终的结果，一种我们称之为“树文件”，用于存贮精化的整个过程。

OOMM是一个用于管理软件开发过程的中间结果及最终结果的模块管理

系统，管理的形式是模块，它提供关于中间阶段的文档的信息保存以及开发过程中的回溯。

这样，利用SPEC 和OOMM, 我们便可以提供保存、查寻及修改所有精化过程中产生的信息的功能。

在XYZ 环境中，还有一个VERI 的子系统，它能够帮助用户做一致性检查，验证某一层次是否满足它的上一个层次。

PROT 子系统是用于构造中间结果的快速原型的，这样便可判断某种解决方案是否可行，否则一味地按照错误的方案走下去是毫无意义的。

另外，XYZ/E 语言本身对子精化过程也是非常有益的。XYZ/E 语言既可以表示问题的抽象描述，也可以表示最终的程序实现，所有这些都是在统一的时序逻辑框架下进行的，这样便可容易地理解程序说明及实现以及进行一致性检查。

XYZ/E 的基本单位是具有一定形式的合式公式，我们称这种合式公式为条件元。条件元具有如下形式:

$$l_b = y \wedge p \Rightarrow @(Q \wedge l_b = z)$$

这里P 是条件，Q 是动作；y, z 分别是定义标号和向前标号；@ 可以是O, 也可以是◇, 也可以有其他的扩充形式，表示在条件成立后，什么时候会发生动作Q。详见参考文献7。

§3. XYZ/SDL

人们已经认识到，许多大的软件系统开发过程中所产生的问题，并非来自于编码的技术上的困难，而是来自于对要解决的问题的描述。因此人们越来越关注软件开发周期的前一阶段。

对于一个软件系统的描述，不是一次完成，而应该是多次精化的结果。

在设计过程中，用来表示中间结果的语言，应当是形式的与非形式的结合，因为精化的过程可以认为是一系列的转换：从非形式到形式，从不精确的程序到可执行的程序，这样人们便希望有一种语言可以同时表示形式的和非形式的，随着设计过程的深入，非形式的部分越来越少，最后全部成为形式的和可执行的。

综上所述，可以看出，我们需要这样一种语言，它可以表示精化过程中的所有层次，从用户最开始的模糊的非形式需求到形式描述，再到算法实现。

我们采用的是一种PDL 形式的语言，我们称之为XYZ/SDL, 并且我们实现了它的语法制导的结构编辑器。

XYZ/SDL 是XYZ/E 语言与非形式的自然语言的语合。XYZ/E 的组成成分是合式公式，而公式是谓词通过逻辑连接词的组合，我们可以把自然语言的一句话看做是一个谓词，因此可以包括在一个公式之内。任何尚不清楚的地方都可先由自然语言来写。下面我们给出一个例子来说明:

这是AB 协议中对某一层次的发送方的描述:

(1) 发送方实现的非形式描述。

(i) 初始: 发送方发送的第一个消息S(0)。

(ii) 发送方发送第i个消息S(i)⇒ 最终接收方接收到S(i)。

(iii) 接收方接收S(i)⇒ 然后存贮S(i)到R(i), 再发送ACK(i)给发送方。

(iv) 接收方发送ACK(i)⇒ 最终发送方接收ACK(i)。

(v) 发送方接收ACK(i)⇒ 然后发送方发送S(i+1)。

2. 将上面非形式描述替换成形式符号, 并根据通道性质(见 §5)

Init ⇒ ◇ {sender} ACCEPT CACK (-1))

{sender} ACCEPT (ACK(j)) ⇒ ◇ ({receiver} ACCEPT (scj+1))

receiver ACCEPT (sci)) ⇒ ◇ (R(i) = S(i) ∧ {sender } ACCEPT CACK (i)))

§4. SPEC ——一个支持上述方法的小环境

SPEC是XYZ环境中的一个子系统, 用来对问题进行描述(specification), 即问题的文字描述。这里我们所说的描述(Specification)不仅仅指对问题的非形式需求的描述, 也指最终的算法实现, 这是由于XYZ/E语言本身的特性, 它能在统一的时序逻辑框架下表示两者。SPEC的设计目标是用来支持逐步精化的过程, 所采取的一部分途径是通过保存精化过程的整个历史以及管理对历史的修改。为了使用户都方便地书写不同阶段地描述, SPEC还提供了两个特殊的工具: XYZ/SDL的结构编辑器及XYZ/BE的结构编辑器, 通过调用XYZ环境下的其它子系统也可以构造中间结果的快速原型以及验证某一层次是否满足它的上一层次。

4.1 片断(Section)

全部的精化过程包括许多步, 或称为层次。每一层都是对问题的部分完全的描述。随着精化过程的逐步开展, 才会变得精确可执行, 直到最后成为一个可执行程序。

在SPEC系统中, 我们用片断来模拟每一个层次, 在一个片断中书写一个层次的内容, 因此片断是SPEC系统内的编辑单位, 正象文件系统中的文件。随着精化过程的展开, 我们建立了相应的树结构(一个片断是一个结点), 这棵树便记录了整个精化过程的历史。

SPEC提供"创建一个片断", "编辑一个片断"、"重画一个片断"(在屏幕上重新显示片断的内容)等功能, 也可以建立片断树, 修改这棵树, 如插入一个结点、附加一个子树、删除一个结点或一棵子树的功能。

4.2 文件

在SPEC中, 我们提供两种文件: 一种我们称之为树文件, 是用来存贮整个精化的历史; 另一种为文字文件, 就是一般意义下的文件, 可以用来存贮精化的最终结果。

4.3 两种特殊的编辑器

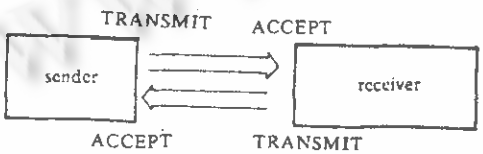
SPEC提供两个特殊的编辑器，一个是XYZ/SDL的语法制导的结构编辑器，利用这个编辑器可以书写任何形式的公式，另一个是XYZ/BE的语法制导的结构编辑器。在精化过程的开始阶段，要将非形式的需求描述转换成形式描述，因此需要利用XYZ/SDL的结构编辑器。当转换到算法程序后，则需要利用XYZ/BE的结构编辑器。这两个编辑器都提供结构化的输入方式以及文字输入方式，并提供语法检查功能。

§5. 例子

例子1. AB协议

我们的第一个例子是对关于AB协议的逐步求精：从非形式的服务需求到真正可执行的XYZ/E书写的算法程序。我们把整个过程分为几个层次(level)，每一个层次是它上一层次的进一步精化，且比它的上层包含了更多的细节。

我们所采取的模型如图所示



通道性质假设:

对于从发送方(sender)到接收方(receiver)的通道，我们有:

(1). channel-1:

□ ~ {sender} TRANSMIT (S(i)) ⇒ ~ ◇ {receiver} ACCEPT (S(i))

/* Nothing comes out that was not put in. */

(2). channel-2:

□◇ {sender} TRANSMIT (S(i)) ⇒ ◇ {receiver} ACCEPT (S(i))

对于从receiver到sender的通道情形类似。

1. 描述层次0 (level 0):

(1) 决定：将用户的非形式需求加以形式化描述。

(2) 所给需求：从发送方发出的一串信息能正确地被接收方接收，既没有丢失、附加，也没有顺序的改变。

(3) 形式化描述:

service - 1:

∀ n: (|S|=n<N) ⇒ ◇ (|R|=n)

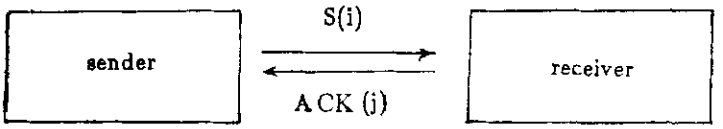
/* R follows S when both growing */

这里R, S分别是发送方和接收方的缓冲区，N是它的最大长度。

service - 2:

∀ n < |R| R(n)=S(n)

/* R is an initial subarray of s. */



2. 描述层次1(level 1)

(1) 决定: 用发送方和接收方两部分来实现上述功能

(2) 描述: middle stage – 1:

Init ⇒ ◇ {sender} ACCEPT (ACK(-1))

/* 我们假定在最开始时, 发送方已接收了(ACK(-1)) */

middle stage – 2:

{sender} ACCEPT (ACK(j)) ⇒ ◇ {receiver} ACCEPT (S(j+1))

middle stage – 3:

{receiver} ACCEPT (S(i)) ⇒ ◇ (R(i) = S(i) ∧ {sender} ACCEPT (ACK(i))

一直到这一层次, 我们都未把发送方与接收方的描述分开处理。事实上, 这一层次依旧是一个发送方和接收方的混合描述, 因此它是联结服务层描述及分开发送方描述和接收方描述的桥梁, 这也就是为什么我们称之为中间阶段(middle stage)描述的原因。

3. 描述层次2(lever 2):

(1) 决定: 分别描述发送方和接收方

(2) 分别的描述:

sender – 1.1:

Init ⇒ ◇ ACCEPT (ACK(-1))

这时ACCEPT的动作者(发送者)被省略。

sender – 1.2

ACCEPT (ACK(j)) ⇒ ((~ ACCEPT (ACK (j+1)) ⇒ ◇ TRANSMIT (S(j+1))) Until ACCEPT (ACK (j+1)))

/* 在发送方接收到第j个消息的ACK(承认信息)之后, 如果没收到第j+1个消息的ACK, 发送方就会发送第j+1个消息, 这一运行直到接收到第j+1个消息ACK */

receiver – 1.1:

ACCEPT (S(i)) ⇒ ◇ (TRANSMIT (ACK(i)) ∧ R(i) = S(i))

/* 在接收方接收到第i个消息之后, 发送对这一消息认可的消息。*/

receiver – 1.2:

TRANSMIT (ACK(j)) ⇒ ∀ K:0 <= k <= j : R(k) = S(k)

/* 如果接收方发送了对于第i个消息的认可信息, 那就意味着, 它已经正确地接收了这个消息之前的所有消息。*/

4. 描述层次3 (level 3): 将描述变成算法。

(1) 决定:

我们引进两个变量。"last-confirmed" 是一个用来记录一个顺序号的变量, 具有这个顺序号的消息以及前面的消息都已被认可, 并且这之后的消息尚未被承认。另一个变量 "next" 用来记录下一个要发送消息的顺序号, 并且这

个消息尚未被承认。发送方保持一个一位的窗口来贮存就要发送的信息，当这位所指示的消息被发送后，窗口就向前移动一个单位。我们用“lb=label i”来代表某一状态，“lb=start”代表起始状态。又如，lb=l1∧0 <next-last-confirmed ≤ 1∧ ~ ACCEPT (ACK(next)) 代表某一个状态，在这个状态下具有“next”这个顺序号的消息即将被发送或已被发送，尚未接收到认可消息仍要被发送。

(2). 算法描述:

lbs 是发送方的状态控制变量。

sender – 2.1:

lbs=start ⇒ ◇ (last-confirmed=-1∧ next=0∧lbs=l1)

/* 在初使状态下，假定已收到S(-1)的认可消息，因此发送方要发送S(0) */

sender – 2.2:

lbs=l1∧(O<next-last-confirmed<=1)∧ ~ ACCEPT(ACK(next))⇒ ◇ (TRANSMIT (S(next))∧lbs=l2)

/* 如果S(next)在窗口之内，并且尚未被承认，发送方就要发送它，之后发送方进入lb=l2的状态，在这一状态下，发送方等待接收这一消息的认可信息。*/

sender – 2.3:

lbs=l2∧ACCEPT(ACK(j))∧(j>last-confirmed)⇒ ◇(last-confirmed=j∧next=j+1∧lbs=1)

sender – 2.4:

lbs=l2∧ ~ ACCEPT(ACK(j))⇒ ◇lbs=l1

/* 如果发送方处于lb=l2的状态，并且没收到任何承认信息，则回到lb=l1的状态(这意味着发送方要重新发送) */

同样地，接收方没有变量last-acked用来记录已经被认可了的消息的顺序号，在顺序号之前都已被认可。

接收方的控制变量是lbr.

receiver – 2.1:

lbr=start ⇒ ◇ lbr=l1

receiver – 2.2:

lbr=l1∧ACCEPT(S(i))∧(i>last-acked) ⇒ ◇ (TRANSMIT ((ACK(i)))∧R(i)=S(i)∧last-acked =i∧lbr=l1)

5. 描述层次4 (level 4):

(1). 决定。精化层次3——加入一个计时设施start(i), cancel(i) 和 time out(i) 是关于第(i)个消息的时序变量。

(2). 发送方的描述:

sender –3.1:

lbs=start ⇒ ◇ (last-confirmed=-1∧next= O∧lbs=1)

sender–3.2:

lbs=l1∧(O<next-last-confirmed<1) ⇒ ◇ (TRANSMIT(s(next))∧start (next)=tt∧next =next+1∧lbs=l2)

sender–3.3:

lbs=l2∧time out(i)=tt ⇒ ◇ (TRANSMIT(S(i))∧start (i)=tt∧lbs=l2)

sender–3.4:

lbs=l2∧ACCEPT(ACK(j))∧j>last-confirmed ⇒ ◇ (cancel (j) = tt∧last-confirmed

=j∧next=j+1∧lbs=l1)

发送方发送S(next)之后，就启动设在这一消息上的计时器，并且修改next的值。如果发送方没收到S(next)的承认信息，则此时变量last-confirmed还没有被改变，而next已经被加1,因此next-last-confirmed=2。S(next)这一消息未落在窗口内(只有<next-last-confirmed≤1时才发送S(next)这一消息),因此这时发送方不发送S(next),如果一段时间内一直没收到承认信息，则计时器报时，发送方重发这一消息。

(3). 计时器的描述:

add a timing devices:

timer – 1:

start (i) ⇒ (◇ cancel (i) ll ~ canel (i) until timeout (i))

timer – 2:

timeout (i) ⇒ □ (start (i) ff ∧ timeout (i) = ff ∧ cancel (i) =ff)

接收方未变。

6. 描述层次5 (level 5):

(1). 决定: 对信息的顺序号进行求精，用循环顺序码来标识信息。

(2). 描述:

发送方:

sender – 4.1:

lbs=start ⇒ ◇ (last-confirmed=-1∧next=0∧lbs=l1)

sender – 4.2:

lbs=l1∧(0<next-last-confirmed<=1) ⇒ ◇ (TRANSMIT (S (next), n)∧ start (n) = tt ∧ next = next+1∧lbs=l2)

sender – 4.3:

lbs = l2∧timeout(i')=tt⇒ ◇ (TRANSMIT (S(i), i')∧start(i')=tt∧lbs=l2)

sender – 4.4

lbs=l2∧ACCEPT (ACK(j'))∧ j >last-confirmed⇒ ◇ (cancel (j')=tt∧last-confirmed= j∧next=j+1∧lbs=l1)

where n=next mod 2;

i=last-confirmed+(last-confirmed+i') mod 2

j=last-confirmed+(last-confirmed+j') mod 2.

接收方:

receiver – 4.1

lbr=start ⇒ ◇ (last-acked=-1∧lbr=l1)

receiver – 4.2

lbr=l1∧ACCEPT (S(i'))∧(i>last-acked)⇒ ◇ TRANSMIT (ACK(i'))∧R(i)=S(i)∧last-acked =i∧(br=l1)

where i=last-acked+(last-acked+i')mod2

在这一层次之前，我们所写的TRANSMIT (S(i))是TRANSMIT(S(i), i)的简写。

7. 描述层次6 (level 6):

(1) 决定: 把上述描述精化为可执行的XYZ/E程序。

(2) 算法程序:

sender:

sender – 5.1:

lbs=start ⇒ O(last-confirmed=-1∧next=0∧lbs=l1);
sender - 5.2:
lbs=l1∧(0<next-last-confirmed<=1) ⇒ O(n=next mod 2∧lbs=l2);
sender - 5.3:
lbs=l2⇒ O(TRANSMIT(s(next),n)∧start(n)=tt∧next=next+1∧lbs=l3);
sender - 5.4:
lbs=l3∧time out(i')=tt⇒ O(i=last-confirmed+(i'+last-confirmed)mod 2∧lbs=l4);
sender - 5.5
lbs=l4 ⇒ O (TRANSMIT(S(i), i')∧start(i')=tt∧lbs=l3);
sender - 5.6:
lbs=l3∧ACCEPT(ACK(j'))⇒ O (j=last-confirmed + (j' + last-confirmed) mod 2∧lbs=l5);
sender - 5.7:
lbs=l5∧j>last-confimed⇒ C(concel(j')=tt∧last-confirmed=j∧lbs=l1);
receiver:
receiver - 5.1
lbr=start ⇒ O (last-acked=-1∧lbr=l1);
receiver - 5.2
lbr=l1∧ACCEPT(ACK(i')) ⇒ O (i=last-acked+(last-acked+i') mod 2∧lbr=l2);
receiver - 5.3
lbr=l2∧(0<i-last-acked<=1 ⇒ O (TRANSMIT(ACK(i))∧R(i)=S(i)∧lbr=l1);

例子2. 电梯系统

这个例子的所有描述部分都已略去，我们只给出整个精化过程中所作的所有设计决定，由此可得到精化过程的轮廓，详细的描述请见Techn. Rept. LS-CAS-XYZ-89-5

1. 描述层次1

设计决定：把用户的非形式需求形式化。

2. 描述层次2

设计决定：精化上一层次。在电梯移动之后，确定下一状态是什么，设变量loc来记录电梯当前所在层数。

3. 描述层次3

决定：把整个电梯系统分为两个进程："电梯控制进程"和"数据管理进程"。在上一层次的描述中，有这样一个谓词："第j层有乘客请求。"电梯如何知道这一点呢？这是来自于乘客请求，因此电梯需要有一个查询户请求的机制。"数据管理进程"负责对乘客请求的查询，并且向"电梯控制进程"报告。

4. 描述层次4

决定：电梯外每层都有两个按钮，但在这一层之前，我们假定只有一个按钮。在这一层我们加入这一细节。我们用cal-up和cal-down两个数组来存贮电梯外部各层次的向上、向下请求。"up(i)"和"down(i)"是相应的要传送的信息

5. 描述层次5

决定：在乘客的请求被服务了之后，这些请求应被删除。我们用三个数组来存贮上述三种请求。在"电梯控制进程"收到up(i)这一信息之后，U(i)为真；服务之后，再把U(i)置为假。

外面的呼叫板在接到乘客的请求后，向“数据管理进程”发消息，“floor”、“up”或“down”，然后“数据管理进程”负责修改数组“F”“U”、“D”里的内容，待服务完毕，再置为假。

6. 描述层次6

决定：把上述描述写成XYZ/E。

附录A. AB协议的证明

由于篇幅所限，这里只给出层次2满足层次1的证明，其他请见Techn. Rept IS-CAS-XYZ-89-5

层次2满足层次1:

从层次2的描述中得到层次1中的第一式，即middlestage-1是很明显的，下面我们先来证明从层次2中可推出层次1的第二式，即middlestage-2。然后再证middlestage-3

(1) {sender}ACCEPT(ACK(j) ⇒ ((~ {sender}ACCEPT(ACK(j+1)) ⇒ ◇ {sender} TRANSMIT(S(j+1)) Until {sender} ACCEPT(ACK(j+1)))

sender – 1.2

(2) {sender}ACCEPT(ACK(j))

middlestage–2 的前件

下面我们来考虑(1)式中的A Until B

第一种情形：B最终成立，即，

(3) ◇ {sender ACCEPT (ACK(j+1))

根据通道假定的第一式

(4) ◇ {receiver} TRANSMIT (ACK(j+1))

(3) channel-1

(5) ◇ {receiver} ACCEPT (S(j+1))

第2种情形：B永远不真，即，

(6) □ ~ {sender} ACCEPT (ACK(j+1))

(即□ ~ B)

根据Until的性质、A Until成立，且有□ ~ B，则有□A，因此，我们有

(7) □(~ {sender} ACCEPT (ACK(j+1)) ⇒ ◇ {sender} TRANSMIT (S(j+1)))

(8) □◇ {sender} TRANSMIT (S(j+1))

(6)、(7)

再由通道的第2条假定

(9) ◇ {receiver} ACCEPT (S(j+1))

(8) channel-2

综合这两种情形，我们可以得到

{sender} ACCEPT (ACK(j)) ⇒ ◇ {receiver} ACCEPT (S(j+1))

即middlestage-2

下面我们来证middlestage-3

(1) {receiver} ACCEPT (S(i)) ⇒ ◇ ({receiver} TRANSMIT(ACK(i))∧R(i)=S(i))

(2) {receiver} ACCEPT (S(i))

middlestage-3 前件

(3) ◇ {receiver} TRANSMIT (ACK(i))

(1), (2)

我们分以下两种情形来考虑

第一种情形:

如果有◇ {sender} ACCEPT (ACK(i)) 我们便有

{receiver} ACCEPT (S(i)) ⇒ ◇ {sender} ACCEPT (ACK(i))

即 middlestage-3

第二种情形

如果不成立, 即,

□ ~ {sender} ACCEPT (ACK(i))

与上面的证明类似我们可以得到,

□◇{sender}TRANSMIT (S(i))

另一方面, 我们有通道假定2,

□◇{sender}TRANSMIT(S(i))⇒ ◇ {receiver} ACCEPT (S(i))

从上面两个式子, 我们有

□◇{receiver} ACCEPT (S(i))

□◇{receiver} TRANSMIT (ACK(i))

再利用通道假定2

◇ {sender} ACCEPT (ACK(i))

综合上面两种情形, 我们得:

{receiver} ACCEPT (S(i)) ⇒ ◇ {sender} ACCEPT (ACK(i))

参考文献

[1] Feng Yulin: Hierarchical Protocol Analysis by Temporal Logic. J. of Comput. Sci & Technol. Vol. 3, No. 1, 1988.

[2] H. Barringer, R. kuiper: Hierarchical Temporal Logic Specification of Concurrent systems. presented at STL/SERC work-shop on the analysis of concurrent systems. Cambridge. Sep, 1983.

[3] H. Barringer, R. Kuiper: Hierarchical Development of Concurrent Systems in Temporal Logic Framework. Lecture Notes of computer Sciences. 1986.

[4] Manna, A. Pnueli: Verificaiton of Concurrent Programs, Part 1: The Temporal Logic Framework.

[5] L. Lamport: What good Is Temporal Logic. Proceedings of IF IP Congress, 1983.

[6] Tang, CS. Towards A Unified Logic Basis for Progamming Language. Proceedings of IF IP Congress, 1983.

[7] C. S. Tang, An Introduction to XYZ System. Inst. of Software, Academia Sinica, IS-CAS-XYZ-88-1.

[8] Zhou Chaochen, Specifying Communicating System with Temproral Logic, Draft 1986.

[9] Zhou Shenzong, The method of Designing and Verifying Concurrent Systems in the Temporal Logic Framework Master Thesis of Inst. of Software, Academia Sinica, 1985.

[10] Peter Lee, et al, Senmantically Based Program Design Environment: The Ergo project in 1988. CMU-CS-88-118.

致 谢

本文作者应感谢XYZ小组的同志们。这小组的学术环境及热烈的讨论对本文的形成有很大的助益。特别应提到的是冯玉琳博士、林惠民博士，他们都曾为本文提出极好的意见，另外，张永光同志对本工作的实现也给予了很大的帮助，在此致谢。本文第一作者作为冯、林两博士的学生，对他们的悉心指教在此表示谢意。