基于程序窗口推理的精化演算*

王云峰[1,2] 李必信[1] 庞军[1] 查鸣[1] 郑国梁[1]

[1](南京大学计算机软件新技术国家重点实验室 南京 210093)

[2](解放军理工大学气象学院 南京 211101)

E-mail: wangyf@seg.nju.edu.cn

摘要 由于数据精化需要针对更大的程序块,所以,它比一般的算法精化更加复杂.在精化演算中过程如何有效地进行数据精化是形式化方法研究中的一个重要内容.该文介绍了相关的基本概念.在精化演算的基础上,构造了一种数据精化算子,并提出一种基于数据精化演算和程序窗口推理的数据精化的方法.

关键词 形式化软件开发方法,精化演算,数据精化,程序窗口推理.

中图法分类号 TP311

程序精化是将程序规约转换成可执行代码的过程.一般将其分为数据精化和算法精化两种形式[1].数据精化是把抽象的数据结构转换为可以高效实现的形式,算法精化是将程序逐步转换为更加便于实现的形式,直至代码.精化演算是一种逐步精化的形式化方法,其理论基础是 Dijkstra 的最弱前置条件理论.精化关系具有自反性和传递性,对于顺序编程的一般结构(如顺序结构、条件语句和迭代语句等),精化关系具有单调性,即如果精化 $A \sqsubseteq C$ 成立,则在任何程序中,可以用 C 代替 A(以 C 实现 A).

这些描述程序开发原则的精化规则极为重要,每一步精化需要利用精化规则加以证明.其中一种即为数据精化:一种数据结构被另外一种数据结构所替换,而整个程序的正确性保持不变.利用数据精化规则可以通过程序规约经过较少的精化步骤开发出程序.

在精化转换过程中,需要处理大量的公式,极为繁琐且易出错,对于数据精化尤其如此.所以,在实际使用精化演算方法时,必须有相应的支撑工具.程序窗口推理(program window inference[2])即为一种用于程序推理与证明工具的推理技术.其思想源于窗口推理(window inference):一种易于处理上下文的定理证明技术.

在一个精化工具中,数据精化的处理比一般的算法精化更加复杂.这是因为,数据精化通常针对的是较大的程序结构,而且需要一步完成.本文首先介绍背景知识,然后构造一种数据精化算子,最后提出一种基于数据精化演算和程序窗口推理的数据精化的方法.

1 基本概念

本节将介绍一些相关的基本概念,如精化演算、窗口推理、程序窗口推理等.

1.1 精化演算

精化演算扩充了 Dijkstra 的卫式命令语言,在可执行的结构中加入规约语句(specification statement).规约和执行程序集成在一种语言里,可以通过在这一语言内的一系列转换来开发程序,对程序的平滑开发起到了重要的作用.在这一开发过程中,初始程序是一种典型的规约语句,最终的程序只包含可执行代码,而开发过程中

* 本文研究得到国家自然科学基金(No.69673006)和国家"九五"重点科技攻关项目基金(No.98-780-01-07-06)资助.作者王云峰,1964 年生,博士生,主要研究领域为形式化方法,面向对象技术.李必信,1969 年生,博士生,主要研究领域为程序理解,面向对象技术.庞军,1976 年生,硕士生,主要研究领域为精化证明工具.查鸣,1975 年生,硕士生,主要研究领域为精化证明工具.郑国梁,1937 年生,教授,博士生导师,主要研究领域为软件工程.

本文通讯联系人:王云峰,南京 210093,南京大学计算机科学与技术系

本文 2000-01-17 收到原稿,2000-04-07 收到修改稿

的程序是两种结构的混合体.

· 规约语句

规约语句是精化演算中一种重要的程序表示方法.规约语句提供了一种方便的方法,将抽象规约嵌入程序.通常地,程序 S 规约写为:pre⇒[S]post,规约语句将其表示为 x:[pre,post],其中 x 表示在程序中发生改变的变量列表,其他变量在程序中不发生变化.规约语句的语义定义如下:

$$[x:[\text{pre},\text{post}]]P == \text{pre} \wedge (\forall\ x \cdot \text{post} \Rightarrow P).$$

下面,我们给出一个简单程序的规约语句及其精化:

$$x:[\textbf{true}, x=\max(a,b)] \sqsubseteq \textbf{if}\ a \geqslant b\ x:=a \ \square\ b \geqslant a\ x:=b\ \textbf{fi}.$$

其中$\sqsubseteq$表示精化关系.

1.2 程序窗口推理

原来的焦点是 $\Pi[F]$,它通过转换 F 实现自身的转换.其中被选作子窗口焦点的部分加方框,如$\boxed{F}$,子窗口的假设条件 Γ' 依赖于上下文 $\Pi[\,]$和原来窗口的假设条件 Γ.通常,Γ' 是强于 Γ 的假设.同时,假定子窗口保持新的前序关系$\sqsubseteq'$,则原来的窗口保持前序关系$\sqsubseteq$.当聚焦原来焦点的某一部分时,可以自动形成窗口开启规则.

程序窗口推理系统是窗口推理系统的一种扩充[3],可以显式地处理程序的上下文,更好地支持自动精化.程序窗口推理系统可以处理程序变量,应用精化规则,管理精化过程中产生的各种上下文环境信息.其理论基础具体分为:

· 精化规则.作为该理论的定理模式.

· 精化关系定义.以最弱前置条件为基本语义,定义精化关系.

· 程序变量及其精化的处理方法.

· 应用精化规则的支持策略.对用户隐藏大部分细节,采用较为简单的方法完成转换.

· 与精化相适应的证明界面.

每一步精化均为定理模式(精化规则)的实例应用,其元变量被当前的上下文实例化.下面主要讨论程序窗口推理系统的上下文处理.有关程序变量及其精化处理方法将结合工具中的数据精化另文讨论.

上下文处理

· 前置条件

考虑如下选择命令的第 1 个分支精化的窗口开启规则:

$$\frac{\Gamma \models \boxed{S} \sqsubseteq S'}{\Gamma \models \textbf{if}\ B \to \boxed{S} \ \square\ C \to T\ \textbf{fi} \sqsubseteq \textbf{if}\ B \to S' \ \square\ C \to T\ \textbf{fi}}$$

以及下面的选择命令:

$$\{P\}\ \textbf{if}\ B \to S \ \square\ C \to T\ \textbf{fi}.$$

在 S 的精化过程中,希望能够利用前置条件 P 以及选择条件 B,如果使用上面的窗口开启规则精化 S,无法用到 P 和 B.为此,前置条件需要显式给出,即把上式改写为

$$\{P\}\ \textbf{if}\ B \to \{P \wedge B\} S \ \square\ C \to \{P \wedge C\} T\ \textbf{fi}.$$

这样表示会出现重复信息,尤其是当 P 较为复杂时,上式将极为繁琐.为此,我们引入一种新型窗口:程序窗口,它把前置条件和通常的假设条件放在一起.例如,上述窗口开启规则可以表示为

$$\frac{\Gamma;\textbf{pre}\ P \wedge B \models \boxed{S} \sqsubseteq S'}{\Gamma \quad \textbf{if}\ B \to \boxed{S} \ \square\ C \to T\ \textbf{fi}} \;=\; \textbf{pre}\ P \quad \sqsubseteq \textbf{if}\ B \to S' \ \square\ C \to T\ \textbf{fi}$$

在这一规则中,选择命令的前置条件 **pre** P 自动加上选择条件 B 作为 S 精化的前置条件.标识符 **pre**(与下面的 **inv** 及 **Ival** 一样)作为程序窗口的语法中的关键字.其结合性要强于"⊨"和";",但要弱于逻辑连接符.例如,**pre** $P \wedge B$ 实际上表示 **pre** $(P \wedge B)$.

一般地,一个程序窗口:

$$\Gamma;\mathbf{pre}\ P \models S \sqsubseteq S',$$

与其等价的传统窗口是

$$\Gamma \models \{P\}S \sqsubseteq \{P\}S'.$$

这样,可以将下面一系列的精化:

$$\{P\}S_0 \sqsubseteq \{P\}S_1 \sqsubseteq \ldots \sqsubseteq \{P\}S_n$$

表示为在上下文 **pre** P 下的精化:

$$\mathbf{pre}\ P \models S_0 \sqsubseteq S_1 \sqsubseteq \ldots \sqsubseteq S_n.$$

随着精化的进行,会获得更多的当前的状态信息,环境前置条件通过增加合取谓词不断扩充.虽然在一系列的精化过程中,它依赖于所有的前置条件,但是每一步精化实际上只依赖于该前置条件的一部分.程序窗口推理系统通过管理其环境前置条件,可以自动利用完整的环境前置条件,不致使焦点的表示很零乱.

· 不变式

精化过程中经常用到程序变量的类型信息,在变量声明"**var** $u:T$"的作用范围内的任意一点,当实施证明时都可以有"$u\in T$",对 u 的所有赋值都要求满足"$u\in T$".不变式推广了类型的概念,允许变量的任意性质(甚至变量之间的关系)与类型一样声明.包含局部不变式的抽象程序块记为$|[\mathbf{inv}\ I\cdot S]|$.有类型的局部变量声明 **var** $u:T$ 和无类型的局部变量声明 **var** u 与不变式声明 **inv** $u\in T$ 是相同的.需要注意的是,不能混淆局部不变式和循环不变式.循环不变式只要求在每一次循环的起始点满足,而局部不变式(如变量类型)要求在所有的状态满足.例如,在求最大公因数的规约精化中,引入新变量 w,条件 $w\in N$ 可以看作局部不变式:

$$|[\mathbf{inv}\ w\in N\cdot v,w:[v=gcd(a,b)]]|.$$

局部不变式仅仅包含了 w 的类型信息,不包括其作用域的信息,这一点在下面将予以讨论.

· 变量名

上下文包括作用范围内的变量名、过程名等信息.通常,如果 x,y 是变量集合,对于精化:

$$x:[P]\sqsubseteq y:=e$$

在语法上要求保证 $y\subseteq x$.这一要求指的是变量的名而不是变量的值.在某一特定的精化点,为了提供作用范围内的变量名的信息,引入一个更为形式化的上下文表示,称作 l-value 上下文,记为 **lval**.把这一信息置于上下文中,可以对其自动处理,以免其作为焦点的一部分贯穿每一步精化,从而简化了精化表示.

2 数据精化演算

2.1 数据精化

数据精化可视为程序精化的一个特例:程序中的抽象局部变量被另一组具体变量所替换,而程序的结构尽可能不变.早期的数据精化假定从具体变量存到抽象变量在函数关系[1],则每一具体的程序状态都能映射到抽象的程序状态,这样就允许若干具体的程序状态对应于同一抽象的程序状态.由于数据精化经常会引入临时变量以提高一些操作的效率,所以,上述假定是合理的.

但是,一些有用的数据精化无法用函数关系描述,80 年代中期,开始用抽象和具体变量的"关系"来描述数据精化.90 年代初,"关系"的方法被扩展为用任意程序将抽象和具体的状态空间关联起来.如果该程序将抽象状态空间转换为具体状态空间,则称其为 simulation,如果该程序将具体状态空间转换为抽象状态空间,则称其为co-simulation[4].下面是据此给出的数据精化的形式定义,数据精化关系记为$\prec$.

如果 *sim* 属于 simulation,

定义 1.

$$P\prec P' \text{ iff } sim;\ P\sqsubseteq P';\ sim. \tag{1}$$

同样地,如果 *sim* 属于 co simulation,记为 sim^*.

定义 2.

$$P\prec P' \text{ iff } P;sim^*\sqsubseteq sim^*;P'. \tag{2}$$

其中,操作符";"相当于函数组合.

给定"具体数据结构"和"抽象数据结构"之间的抽象关系为 AI,*sim* 和 *sim*° 的语义可以用最弱前置条件的形式来定义:

对于任意一个谓词 φ,可以如下定义 *sim* 和 *sim*°:

$$|[sim]|\varphi \triangleq \exists\ a \cdot \mathrm{AI} \wedge \varphi, \tag{3}$$

$$|[sim^{\circ}]|\varphi \triangleq \forall\ a \cdot \mathrm{AI} \Rightarrow \varphi. \tag{4}$$

实际上,对状态转换程序加上某些约束,可以保证数据精化关系在不同的程序结构中满足分配律,在数据精化过程中维持抽象程序的结构不变. 例如,若 *sim* 属于 co-simulation,满足"析取性"和"严格性"(disjunctive and strict),则数据精化满足以下规则:

$$(S_1 \prec S'_1) \wedge (S_2 \prec S'_2) \Rightarrow S_1; S_2 \prec S'_1; S'_2. \tag{5}$$

2.2 数据精化的演算

· 程序块

由于局部变量的作用范围是一块语句,所以数据精化可以描述成块语句的转换. 故数据精化可以看作是算法精化的特例:精化仅限于块语句.

块语句表示为

$$|[\mathbf{var}\ a \,|\, \mathrm{Init} \cdot P]|, \tag{6}$$

即为数据精化的对象,其中 **var** a 为引入程序的局部变量,根据谓词 Init 对其进行初始化.

块语句的语义以最弱前置条件定义:对于任意一个不包含自由变量 a 的谓词 φ,

$$|[\,|[\mathbf{var}\ a \,|\, \mathrm{Init} \cdot P]|\,]|\varphi \triangleq \forall\ a \cdot \mathrm{Init} \Rightarrow \Box P\, \Box \varphi. \tag{7}$$

· 数据精化算子

在对块语句进行数据精化时,将抽象和具体变量的关系表示为抽象不变式 AI,对于谓词和程序分别引入精化算子 $\mathrm{D_{AI}}$ 和 D_{AI},将块语句的数据精化表示为

$$|[\mathbf{var}\ a \,|\, \mathrm{Init} \cdot P]| \sqsubseteq |[\mathbf{var}\ c \,|\, \mathrm{D_{AI}Init} \cdot D_{\mathrm{AI}}P]|. \tag{8}$$

任意给定一个谓词 φ,$\mathrm{D_{AI}}$ 及其对偶算子 $\mathrm{D^{*}_{AI}}$ 定义为

$$\mathrm{D_{AI}}(\varphi) \triangleq \exists\ a \cdot \mathrm{AI} \wedge \varphi, \tag{9}$$

$$\mathrm{D^{*}_{AI}}(\varphi) \triangleq \forall\ a \cdot \mathrm{AI} \Rightarrow \varphi. \tag{10}$$

根据第 2.1 节以及数据精化的谓词转换子的"对偶和邻接[5]"概念,我们有

$$P \prec P' \text{ iff } sim; P; sim^{\circ} \sqsubseteq P'. \tag{11}$$

这样可以将 $(sim; P; sim^{\circ})$ 看作"最小"(最抽象)精化,则对于语句 P 有

$$sim; P; sim^{\circ} \sqsubseteq D_{\mathrm{AI}}(P). \tag{12}$$

2.3 数据精化算子的应用

针对不同的程序结构,可以递归程序语句的精化算子,根据数据精化定理对相应的程序语句进行精化. 根据 R. Ruksenas 等人[6]的结果,下面给出一些典型程序语句的数据精化算子.

2.3.1 规约语句

对于规约语句 P—$a, g:[\mathrm{pre}, \mathrm{post}]$,从式(12)可以得到下面的公式:

$$D_{\mathrm{AI}}(a, g:[\mathrm{pre}, \mathrm{post}]) \sqsubseteq c, g:[sim\ \mathrm{pre}, sim\ \mathrm{post}]. \tag{13}$$

对于 P,则得到如下的数据精化:

$$a, g:[\mathrm{pre}, \mathrm{post}] \prec c, g:[sim\ \mathrm{pre}, sim\ \mathrm{post}]. \tag{14}$$

对于谓词 φ,$sim\varphi$ 可以看作 $\mathrm{D_{AI}}(\varphi)$,对于 $sim^{\circ}\varphi$ 和 $\mathrm{D^{*}_{AI}}(\varphi)$ 同样如此. 这样,式(14)可以改写为

$$a, g:[\mathrm{pre}, \mathrm{post}] \prec c, g:[\mathrm{D_{AI}}(\mathrm{pre}), \mathrm{D_{AI}}(\mathrm{post})], \tag{15}$$

这是非常有用的规则,因为利用它可以从抽象的规约语句中演算得到具体的规约语句. 实际上,式(8)与 Morgan[4]等人的结果是一致的.

2.3.2 赋值语句和上下文语句

对于非确定性赋值语句 post$[a,g\backslash a',g']$,其数据精化公式为

$$D_{\mathrm{AI}}(\mathrm{post}[a,g\backslash a',g'])=\forall\, a\cdot \mathrm{AI}\Rightarrow(\exists\, a'\cdot \mathrm{AI}[a,g\backslash a',g']\wedge \mathrm{post}[a,g\backslash a',g']).$$

上下文语句包括断言$\{\varphi\}$和假设$[\varphi]$.程序中的断言$\{\varphi\}$表明条件φ在程序该点满足.对应地,假设$[\varphi]$表示条件φ在相应的程序点成立,但是假设的正确性需要在程序中加以证明.对于这两种上下文语句,D_{AI}定义为

$$D_{\mathrm{AI}}(\{\varphi\})\triangleq\{\mathrm{D}_{\mathrm{AI}}(\varphi)\},$$

$$D_{\mathrm{AI}}([\varphi])\triangleq[\mathrm{D}_{\mathrm{AI}}(\varphi)].$$

2.3.3 顺序语句

对于顺序语句,数据精化算子满足分配律:

$$D_{\mathrm{AI}}(S_1;S_2)=D_{\mathrm{AI}}(S_1);D_{\mathrm{AI}}(S_2).$$

它与式(5)是一致的.

2.3.4 条件语句、选择语句、循环语句

对于条件语句,数据精化算子也满足分配律,数据精化算子可以移入条件语句,但是需要在每一具体的条件分支中加入一条假设语句.如卫式命令$g\rightarrow S$可以数据精化为

$$D_{\mathrm{AI}}(g\rightarrow S)=\mathrm{D}_{\mathrm{AI}}(g)\rightarrow[\mathrm{D}^{*}_{\mathrm{AI}}(g)];D_{\mathrm{AI}}(S).$$

对于选择语句和循环语句,在数据精化时同样需要加入假设$[\mathrm{D}^{*}_{\mathrm{AI}}(g)]$.

对于条件语句:

$$D_{\mathrm{AI}}(\textbf{if } g \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi})=\textbf{if } \mathrm{D}_{\mathrm{AI}}(g) \textbf{ then } [\mathrm{D}^{*}_{\mathrm{AI}}(g)];$$
$$D_{\mathrm{AI}}(S_1) \textbf{ else } D_{\mathrm{AI}}(S_2) \textbf{ fi}$$

对于选择语句:

$$D_{\mathrm{AI}}(\textbf{if } (g_1\rightarrow S_1)\,[]\ldots[]\,(g_n\rightarrow S_n) \textbf{ fi})=\textbf{if } \mathrm{D}_{\mathrm{AI}}(g)\rightarrow[\mathrm{D}^{*}_{\mathrm{AI}}(g)];$$
$$D_{\mathrm{AI}}(S)\,[]\ldots[]\,\mathrm{D}_{\mathrm{AI}}(g)\rightarrow[\mathrm{D}^{*}_{\mathrm{AI}}(g)];D_{\mathrm{AI}}(S) \textbf{ fi}$$

对于循环语句:

$$D_{\mathrm{AI}}(\textbf{do } (g_1\rightarrow S_1)\,[]\ldots[]\,(g_n\rightarrow S_n) \textbf{ od})=\textbf{do } \mathrm{D}_{\mathrm{AI}}(g)\rightarrow[\mathrm{D}^{*}_{\mathrm{AI}}(g)];$$
$$D_{\mathrm{AI}}(S)\,[]\ldots[]\,\mathrm{D}_{\mathrm{AI}}(g)\rightarrow[\mathrm{D}^{*}_{\mathrm{AI}}(g)];D_{\mathrm{AI}}(S) \textbf{ od}$$

2.3.5 块语句

对于块语句,数据精化满足如下公式:

$$D_{\mathrm{AI}}(|[\textbf{var } b\,|\,\mathrm{Init}\cdot Q]|)=|[\textbf{var } b\,|\,\mathrm{D}^{*}_{\mathrm{AI}}(\mathrm{Init})\cdot D_{\mathrm{AI}}(Q)]|.$$

3 程序窗口推理中的数据精化

在对程序进行部分精化时,其上下文信息是非常重要的.上面介绍的窗口推理和程序窗口推理提供了处理这种上下文信息的方法.但是,均未包含数据精化.由于数据精化需要针对更大的程序块,所以,它比一般的算法精化更加复杂.

在这一节,我们提出一种利用程序窗口推理进行数据精化的方法,其中的数据精化演算是基于第2.2节的"数据精化算子".

3.1 窗口开启规则

由于数据精化仅仅针对块语句进行,所以只需对块语句给出窗口开启规则.由于程序不能同时处于抽象和具体的状态,所以对于抽象变量的替换必须一步完成.

对于块语句式(8),有如下的窗口开启规则:

$$\frac{H;\ \textbf{pre } P[c\backslash c']\wedge \mathrm{D}_{\mathrm{AI}}(\mathrm{Init});\textbf{lval } L[c\backslash c']\wedge c\in \mathrm{Var};\textbf{inv } \mathrm{Inv}[c\backslash c']\wedge c\in T'\wedge \mathrm{AI}\models \boxed{P}\prec D_{\mathrm{AI}}(P)}{H;\ \textbf{pre } P;\textbf{lval } L;\textbf{inv } \mathrm{Inv}\models\ |[\textbf{var } a;T'\,|\,\mathrm{Init}\cdot \boxed{P}]|\sqsubseteq|[\textbf{var } c;T'\,|\,\mathrm{D}_{\mathrm{AI}}(\mathrm{Init})\cdot D_{\mathrm{AI}}(P)]|}. \tag{16}$$

其中AI为表示抽象和具体变量之间关系的不变式.利用这一规则,块语句的转换主要集中在根据"抽象命令"演

算"具体命令",而其他复杂的谓词全部归结到 **pre** an 和 **inv** 上下文中,如具体变量的初始条件 $D_{AI}(Init)$及其 AI 等.

符号$\prec$代表抽象和具体命令之间的数据精化关系(见第 2.1 节),它具有自反性和传递性.为了在子窗口中保持相同的精化关系$\sqsubseteq$,将$([\overline{P}]\prec D_{AI}(P))$等价地写为$D_{AI}([\overline{P}])\sqsubseteq P'$,则式(16)可以写为

$$\frac{H;\ \mathbf{pre}\ P[c\backslash c']\wedge D_{AI}(Init);\mathbf{lval}\ L[c\backslash c']\wedge c\in Var;\mathbf{inv}\ Inv[c\backslash c']\wedge c\in T'\wedge AI\models D_{AI}([\overline{P}])\sqsubseteq P'}{H;\mathbf{pre}\ P;\mathbf{lval}\ L;\mathbf{inv}\ Inv\models|[\mathbf{var}\ a:T\mid Init\cdot[\overline{P}]]|\sqsubseteq|[\mathbf{var}\ c:T'\mid D_{AI}(Init)\cdot P']|}. \tag{17}$$

这一规则与算法精化具有相同的形式.

3.2 焦点转换规则

对于不同的程序结构,需要根据不同的数据精化定理进行精化(转换).数据精化算子的应用可以用焦点转换规则来描述,其中,采用 inv 上下文信息集中表示转换规则的应用条件,便于在精化过程中加以证明.

例如,对于顺序结构 $S_1;S_2$,相应的转换规则可以写为

$$\frac{H;\mathbf{pre}\ P[c\backslash c']\wedge D_{AI}(Init);\mathbf{lval}\ L[c\backslash c']\wedge c\in Var\models\mathbf{inv}\ Inv[c\backslash c']\wedge c\in T'AI}{S_1;S_2\prec D_{AI}(S_1);D_{AI}(S_2)}. \tag{18}$$

利用程序窗口推理,可以简化数据精化算子的演算.例如,对于卫式命令:

$$D_{AI}(\{P\};g\rightarrow S)=\{D_{AI}(P)\};D_{AI}(g)\rightarrow([D_{AI}^{*}(g)];D_{AI}(S))$$

在程序窗口推理中可以简化为

$$D_{AI}(\mathbf{pre}\ P\wedge g\vdash S)=D_{AI}(\mathbf{pre}\ P\wedge g)\vdash D_{AI}(S).$$

3.3 与算法精化的比较

本文将数据精化当作算法精化的特例处理,但是,二者的窗口开启规则和窗口转换规则还是有所不同的.

3.3.1 窗口开启规则

对于数据精化仅有一条针对变量声明的块语句的窗口开启规则.其焦点针对块语句中所有的命令,而非其中的部分命令.由于程序不能同时处于抽象和具体状态,所以,对于抽象变量的替换必须一步完成.当焦点移进块语句时,抽象不变式 AI 以及由其他转换得到的谓词如 D_{AI}都归结到新窗口的上下文信息中.

3.3.2 窗口转换规则

对于算法精化,焦点的转换是根据 Morgan[7]的精化演算规则,将程序语句精化成不同的结构(如迭代、选择、循环等结构).而对于数据精化,焦点转换一般不改变命令的结构,它只是用数据精化算子从基于较抽象的数据结构的命令演算出基于较为具体的数据结构的命令.

4 结 语

本文提出了一种基于数据精化演算和程序窗口推理的数据精化方法.该方法支持演算方式的数据精化:根据抽象程序(或规约)自动构造具体的程序.在精化过程中,程序窗口推理是一种管理复杂性的有效方法.有效地管理和利用程序上下文信息,可以简化精化过程.在下一步的工作中,我们将进一步研究数据精化算子,对其特例进行分析,同时,利用程序窗口推理技术进一步简化数据精化的实现过程.

参考文献

1 Hoare C A R. Proofs of correctness of data representation. Acta Informatica, 1972,1(4):271~281

2 Nickson R, Hayes I. Supporting contexts in program refinement. Technical Report, 96~29, Software Verification Research Centre, University of Queensland, 1995

3 Nickson R, Utting M. A new face for Ergo: adding a user interface to a programmable theorem prover. Technical Report, TR-95-42, Software Verification Research Centre, University of Queensland, 1995

4 Gardiner P H B, Morgan C C. A single complete rule for data refinement. Formal Aspects of Computing, 1993,5(4):367~382

5 Wright J von. A lattice-theoretica base for program refinement [Ph. D. Thesis]. Abo Akademi University, 1990

6 Rimvydas Ruksenas. A tool for data refinement. Technical Report, TUCS-TR-119, Turku Centre for Computer Science, 1997

7 Morgan C C. Programming from Specifications. 2nd edition. London: Prentice-Hall Inc., 1994

On Refinement Calculus with Program Window Inference

WANG Yun-feng[1,2] LI Bi-xin[1] PANG Jun[1] ZHA Ming[1] ZHENG Guo-liang[1]

[1](*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)
[2](*Meteorology College PLA University of Technology Nanjing* 211101)

Abstract Data refinement is hard to deal with in a refinement tool compared with ordinary algorithmic refinement, since data refinement usually has to be done on a large program component at once. So it is important to learn how to perform data refinement effectively. In this paper, the background is introduced first, then the data refinement calculator is constructed. Finally an approach is proposed for data refinement which is based on data refinement calculus and program window inference.

Key words Formal software development method, refinement calculus, data refinement, program window inference.