计算机科学2000Vol. 27№. 3

精化演算支撑工具PRT的研究*)

On A New Support Tool PRT for the Refinement Calculus

杨朝晖 王云峰 郑国梁

(南京大学计算机软件新技术国家重点实验室 计算机科学与技术系 南京210093)

Abstract The refinement calculus for the development of programs from specifications is suited to mechanised support. We review the requirements for tool support of refinement as gleaned from our experience with a number of existing refinement tools, and report on the design and implementation of a new tool to support refinement based on these requirements. The main features of the new tool are close integration of refinement and proof in a single tool (the same mechanism is used for both), good management of the refinement context.

Keywords Refinement calculus, Refinement tool, Theorem proof, Window reasoning

1 引言

精化演算是一种数学表示法和若干规则的集合，用于从程序规约推导出命令式程序。精化是从抽象程序向具体程序转换的过程，其中包含程序的正确性证明。精化的程序开发方法比对已有程序进行验证以保证程序正确性的方法更有效。通过精化演算中的转换规则可以演算出精化的程序。

利用精化演算从规约导出程序的过程由大量步骤构成，非常适合利用机器工具进行辅助。本文对精化工具进行了需求分析和功能分析，研究了一个新的精化工具PRT(Program Refinement Tool)并与现有的一些工具进行了比较。

2 需求分析

应用精化演算导出程序的工具需具有几项功能：

● *规则选择* 每个精化步骤包括从当前步骤选择要精化的部分和相应精化规则的实例。对规则的选择并非唯一，工具应能找出与当前要精化部分匹配的所有规则并让用户决定。

● *规则应用* 一旦选定规则，应用规则可直接进行符号处理并简化公式。

● *实施规则证明* 许多精化规则的使用都有应用条件以确保使用在正确的上下文中。为增强精化结果的可信度，当规则被使用时，应显示其应用条件。这些证明规则为一阶谓词演算的引理，一般不能完全自动化证明，而且经常包括一些相关的浅显证明。所以需要能支持复杂证明的工具。

● *精化步骤的记录* 因为有大量的精化步骤，计算机工具可通过记录和管理精化各阶段间关系及每步中使用的规则与应用条件辅助用户。其管理既要保证完整性，又要能扩展与修改。记录可提供对从规约到结果全过程的跟踪，支持重用与修改推导以适应规约的变化并可利用结构信息在扩展代码与各阶段进行浏览。

对精化工具需求分析的讨论分为五部分：信息描述、定制、精化转换、支持证明、用户界面。

2.1 信息描述

支持精化的工具必须能描述多种不同的信息：规约与代码段；精化规则；应用前提条件；证明表示；精化过程中上述信息间的关系。

规约与代码片段 精化演算使用广谱语言意味着规约与代码要统一表示。表示法要支持谓词逻辑(关于规约的前后置条件)和布尔表达式(用于描述卫式命令)。

精化规则 每个精化规则是表示实例集的模式。每个实例在精化演算中作为一个定理。实例可定义为：目标程序段；相应结果段；应用条件。具体程序段应用

*)本文研究得到国家自然科学基金和国家"九五"攻关项目基金资助。杨朝晖 硕士生，主要研究方向为精化工具。王云峰 博士生，主要研究领域为形式化方法，面向对象技术。郑国梁 博士导师，主要研究领域为软件工程。

规则时、与实例化的目标模式元变量匹配并转换成用元变量值实例化的结果模式。独立元变量可在规则使用时或之后实例化。精化过程中允许未实例化的元变量存于程序中有这样一些好处：元变量可在以后更明确时被实例化；导出规则能被推导和证明，导出规则可能有简化的应用条件、而且扩展了精化演算的规则集。

应用前提条件 应用条件的描述使用与广谱语言相同的逻辑、但增加了一些自然语言的约束。应用条件可包含目标模式与结果模式的元变量，应用精化规则时产生对应的应用条件实例，用于约束规则应用的上下文。

证明表示 如果工具要管理精化中产生的证明规则、就需要表示证明。具体的表示以实施证明的机制为根据，但最好便于在精化变化时进行验证与作相应的改动。

相互关系 精化过程产生大量通过精化关系联系的程序集。精化过程可描述成以初始规约为根的精化树。每个节点表示一段程序，每条边表示一个精化规则的应用。与每条边联系的是实例的参数值、证明过程与证明结果。精化完成后，利用内节点构造程序结构，利用叶节点提供具体命令，通过遍历树可得到程序代码。这种表示没有记录实际的规则应用次序，但可产生较合理的序列。

2.2 定制

允许用户在精化工具中根据特殊应用定义新的表示法非常重要。新的表示法既可能是在规约中增加形式化表示，也可能针对目标编程语言修改精化演算语言。这要求工具能扩展精化演算语言语法和应用条件逻辑语法和语义。工具最好能给出正确的精化规则，定义精化演算语言的语义和精化关系并支持证明。

2.3 精化转换

为有效使用规则，需满足几点：能构造规则集使规则易于使用；能识别规则模式与给定代码段的匹配。要能从可用规则集选一规则及其实例参数。选择后，要能应用规则演算出程序段。精化策略是可选的，可根据当前上下文灵活地组合可用规则。

2.4 证明支持

证明器应集成在精化工具中，以随时证明精化产生的规则。证明器与精化工具应有相同界面，使用户只需掌握一种界面概念模型。证明器应能处理应用条件，定制基本逻辑以适应特殊应用并能判断新的精化规则。

2.5 用户界面

精化过程中，用户要观察当前状态及在整个开发过程浏览。用户界面必须能隐藏无关的细节，用户也要能修改记录并观察到相应结果。如果不能完全自动证明规则、就需提供控制和观察证明活动的功能。

精化工具有两种风格的用户界面。一种基于"符号算子"模型、着重于规则应用。另一风格基于"活动文档"模型[1]，强调以类似文本的风格记录与表示精化演算。这种"所见即所得"的风格通过超文本链接提供隐藏信息、将每个证明链接到对应的规则证明。

2.6 设计目标

现在讨论基于以上需求分析的精化工具 PRT。需求分析中的功能作为工具设计的基础、并综合考虑创建工具的实际限制与实用性。设计工具目标为：支持精化规则的选择；尽可能自动化地应用规则；无缝集成到工具中使用户能对规则实施证明；可定制精化和证明结构的显示；各级的设计中都具有通用性。

3 PRT 的研究与分析

3.1 总体结构

从需求分析可以得到一个抽象的结构，如图1所示。图1中，精化引擎应用精化规则，实施精化转换，并产生伴随的证明规则。证明器部分自动化辅助实施证明规则。管理器管理精化各部分之间，规约与其精化之间，规则与其证明之间，及可选的精化间的结构关系。用户界面对上述各功能提供统一的用户界面。

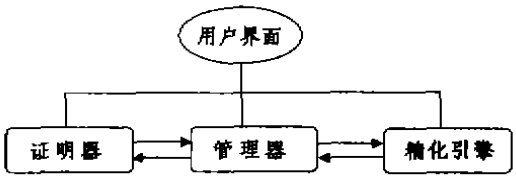

图1

使用 PRT 作为定理证明器时，通过精化引擎、证明器和管理器的集成实现这种结构。精化的完整性依赖精化规则使用的正确性。精化引擎完成应用规则的功能。证明器用于验证规则与实施规则证明。两部分相结合并有相似的操作对精化工具相当有用。这使工具只需建立在一个形式化系统上，能向用户提供统一的界面和概念模型；精化与证明有公共或相关的数据结构，能简化其表示和浏览；规则与精化步骤可共享相关信息。因为规则出现在精化当前状态的上下文中，所以上下文在实施证明规则时随时可用会有很大帮助。

3.2 定理证明器和精化引擎

有一种有效的证明风格：项重写[2]，利用这一模型，从待证明的公式出发，可构造一系列形式对象，每一对象通过保持有效性的关系与正在处理的对象相连。如果该对象序列的最后一个元素有效，其第一个元素也有效。通常，序列中的后继元素通过应用条件重写

规则从前驱元素中演算得到。一般，这种"重写"关系及其对象逻辑的"项结构"允许"子项"的替换。很多定理证明器以这种风格的证明为基础，如著名的通用证明器 Isabelle，证明器 Affirm，Eves 等。定理证明器 Frgo[3]中采用的窗口推理证明技术也以此为基础。

使用基于"项重写"的定理证明器的最大好处在于其证明过程与精化过程相似，两者都包括选取待转换的成分，选择转换规则并进行实例化，验证规则的应用条件，以及用转换结果替换相应片段，两者都依赖相关对象和关系的相同性质：自反性，传递性，单调性。这可减轻用户在精化步骤和证明步骤转换时的负担。

窗口推理 PRT 使用基于"项重写"的窗口推理证明规范[4]，支持在层次证明结构中进行直接面向目标的推理。在任意阶段，证明可包含一层尚未解决的若干子问题及其相应上下文。在此层中的每个节点称为一个窗口。窗口内容包括：一个焦点，作为将要转换的项；一些假设，转换焦点时假定为真；一个关系，指出转换中保持的关系；一个目标，指出转换焦点所期望的结果。

利用这样的窗口，通过连续地在若干子问题上移动焦点完成该问题的证明。对于一个子问题，通过相应的假设，一焦点开启新的窗口。在这一窗口中，可以通过转换消解这一子问题，或者通过开启新窗口将其进一步分解。在一子问题解决之后，该窗口关闭，关闭的方法是：对于一个证明，将焦点还原为 TRUE，对于精化，则将规约转换为代码。这样，最终的证明可以表示为由若干窗口和转换组成的一种层次结构，由此构成了从最初问题到所需结果之间的正确转换。

使用这种规范，解决一个问题只需解决它的所有子问题。对于一个子问题，通过相应的假设，焦点开启新的窗口。在这个窗口内，可通过转换或分解进一步打开新的窗口处理问题。当子问题被解决，例如证明焦点为真或规约精化为代码，窗口将关闭。最终的证明表示为由众多窗口和转换组成的层次结构并一起构成了从初始问题到所求结果的正确转换。

在标准的定理证明器中，需要保持的关系一般为等价或蕴含，但别的关系也可能出现。当使用窗口推理进行精化转换时，是"被精化"关系。通过在前置条件下直接打开窗口并且弱化的操作可以替代象"弱化前置条件"这样的精化规则。

PRT 是通过对 Ergo 定理证明器扩展而建立的，使用窗口推理证明规范。Ergo 可扩充，通过具有理解力的策略语言支持自动证明，证明简单规则，无需用户干预，自动调用策略。

建立程序窗口推理逻辑 精化的标准处理以高阶逻辑为基础。程序变量作为域和其值的域明确区分。状态作为将变量映射到值的函数。表达式和谓词将状态分别映射到值和布尔值。程序起到谓词转换子的作用，而无需最弱前置条件算子。

精化演算的一个优点是大部分过程只需使用条件为一阶逻辑的精化规则，而且不需引用所有的状态。一些特别的逻辑条件用自然语言描述，在另外一些逻辑条件下，程序变量就象逻辑变量一样（例如，"$x=0\Rightarrow x\leqslant 1$"）。因为对所有的状态规则的逻辑条件一般隐式地表示为全称量词，所以可以缩小为一阶概念。

使用高阶方式描述精化，全称量词必须明确，各种标志符的表示也必须相互区分，作为表达式和谓词的参数的状态也不能随便隐藏，另外，这种方式使得形式化证明精化规则、清楚的推导状态和关系成为可能。使用没有清楚描述状态的一阶逻辑则不能做到。

程序窗口推理以模态逻辑为基础，这种模态逻辑与高阶逻辑语义有关。由于使用模态逻辑避免了高阶的结构，程序变量在规则的应用条件中可象模态逻辑的变量一样。使用约束状态集的模态算子可以区分标志符的不同出现。因为状态是谓词的参数，精化规则可以被证明。

程序窗口推理理论 Ergo 具有广泛的理论基础，覆盖了经典逻辑、集合论、代数和象序列这样的结构，为支持精化，使用程序窗口推理的理论被加进来。程序窗口推理理论提供对处理程序变量、应用精化规则和管理各种精化中的上下文的支持。它包括：

- 通过最弱前置条件函数(wp)得出的精化关系的定义。
- 作为理论中定理模式的精化规则。
- 应用于精化关系的窗口的打开和关闭。
- 处理程序变量和变量替换的机制。
- 应用精化规则的支持策略，对用户隐藏了许多细节并以简单的方式进行替换。
- 适合精化的证明界面。

每个精化步骤就是定理模式实例的应用，同时根据当前上下文对元变量进行相应实例化。Ergo 的理论认为最终程序是初始规约的精化结果。

程序变量 是有限常量集的模型。由于 Ergo 内嵌了不被模态逻辑使用的替换和量词的概念，不能直接使用逻辑变量。表达式替换程序变量表示为谓词的一个模态函数，由能计算可能发生替换的策略支持。类似地，可用全称和存在量词约束程序变量。

应用精化规则 在 Ergo 中，公理或定理被理解为直接推导规则。规则的应用通过实例化以匹配要使用的情形。其余模式变量暂时不被实例化。这些元变量可在以后推导的任意阶段实例化。对精化而言，应用规则时完全实例化更方便。所以每个精化规则都用一个参

数表注释。其模式变量一般要求明确地实例化。这些参数都有元类型以控制如何获得实例和验证，例如给用户提示。应用规则时可以让参数不被实例化，保持使用元变量的灵活性。

Ergo 中公理和定理也能解释为条件推理规则。规则使用时需要实施条件。一些条件（特别是大部分语法条件）通过由一张可扩充的表驱动自动使用策略来实施。其余规则通过以下方式之一控制：缺省方式是剩下的规则记录为假设，但必须在当前窗口关闭前实施；规则实施也可延迟，在定理中被记录为假设，用户可在以后作为单独的证明实施；最后的选择是规则一产生就必须实施并由用户决定当前选择。

上下文 在确定精化的使用和实施规则时要用到这样几种上下文：l-value 上下文—获得有关变量名和过程名的信息，用于实施规则。不变式上下文—获得要维持的变量值的信息。不变式用于合并类型信息、维护隐含的精化逻辑。前置条件上下文—获得有关初始变量值的假定，变量值可由先前的赋值和验证得到。

这三种上下文在 Ergo 的假设表中用 lval、inv 和 pre 表示。窗口打开和关闭规则自动管理假设表，从上下文推导并操作新假设。

4 现有工具比较

现在已有一些精化工具：Red 是牛津大学研制的工具[5]，HOL 是基于高阶逻辑（HOL）定理证明的工具[6]，Centipede 是处理精化图的工具[7]，Proxac 是可直接用于精化演算的通用转换工具，Cogito 是包括从 Z 进行精化的方法和一套工具[8]。下面讨论它们的差异并与 PRT 比较。

形式化程度 HOL 和 Cogito 通过经典逻辑深刻地描述了规约和程序。精化规则均由最初定理证明而来，这保证了精化开发的高可靠性，也便于使用从传统数学得到的结果。但另一方面，这种深刻的描述使得在精化层次的表示和形式化较麻烦。别的工具把规约和程序当作语法上可操作的无需解释的项，精化规则不能被证明，精化过程中也没有数学定理的形式化表示。

PRT 将命令、谓词、程序和逻辑变量视为不同的语法类，规约、程序和逻辑公式的语法与传统的相近。状态被描述成模态逻辑中的可能空间，允许进行高度形式化描述，保留了精化前提为一阶逻辑的形式，所以仍然可以使用标准的数学结果。其不足在于使用的"目标构建"逻辑是一种全新的逻辑，其合理性尚待证明。

证明规则的支持 所有的工具都提供对证明精化规则的支持，区别在于支持种类的不同。RRE 尝试进行完全自动化的证明。HOL 和 Cogito 对证明提供适当的策略辅助，而非全自动化证明。Red 和 Centipede 不直接支持证明，但提供对外部证明工具的链接。

PRT 使用程序窗口推理逻辑支持实施证明规则。由于大部分规则的应用条件只需证明其一种状态，因而通常可忽略其形式，应用从传统逻辑得出的结果实施证明。因为精化和证明逻辑的紧密结合使得这些活动比较相似，所以可以缩小过程模型和界面风格方面的区别，更便于用户理解。使用相同的精化和证明引擎可加强精化正确性，因为在语义上这些活动是等价的。

用户界面 Proxac，RRE 和 Centipede 注重可用性，提供丰富的图形界面，便于用户使用，但难以记录用于非线性浏览、修改和重用的推导步骤。别的工具具有简单和强大的命令行驱动界面，支持用户和机器可读的脚本，这些脚本可进行文本编辑和反馈到重用的工具中。

PRT 的精化和证明引擎是命令行驱动的，有两种用户界面原型支持证明和浏览。

通用性 Proxac 简单，易于增加规则，而且无需预定义即可操作程序。PRE 中新的、未证明规则可加入系统、但难以增加新的程序结构。HOL 和 Cogito 理论上具有一定通用性，为增加新的程序结构，要提供结构的定义并需要定义相应的高层策略以使用新规则和操纵新结构。Red 支持新结构，通过组合已存在的规则推导精化规则，但不能引入新的程序结构或新增规则。

PRT 通过定义新结构的最终前置条件语义和定义这些结构如何影响程序窗口推理的上下文来进行扩展。新的精化规则可以是假设并能用精化的定义和程序结构最弱前置条件语义证明。以包括一阶谓词逻辑、代数和集合论的扩充程序库为基础可定义应用理论。

对上下文管理的支持 PRE 维持封装了上下文的结构并用于证明。HOL 和 Cogito 中，上下文是结构定义的一部分。别的工具不表示上下文。

PRT 使用程序窗口推理，有强大的表示法表示各种上下文，包括隐式前置条件、类型、不变式和变量与别名。当焦点进行推导时，自动更新窗口规则的上下文信息。

结束语 本文对精化工具进行了需求分析和功能分析，讨论了作为定理证明器和精化引擎基础的窗口推理系统和用于程序精化推理的程序窗口推理系统。同时分析了设计目标、总体结构、精化与证明的表示方法等问题。然后研究一个符合要求的精化演算支撑工具 PRT。PRT 的特点在于：

- 证明和精化的紧密结合；
- 管理精化与证明上下文的新的逻辑与证明规范（程序窗口推理机制）；

（下转第46页）

活的GC称为异步GC，它可以被其他线程中断；②运行中的线程要创建对象，但空间不够，此时激活的GC为同步GC，它是不可中断的。

垃圾回收可分为两个阶段。一是标记过程：GC线程遍历存储空间，标记所有可被系统中运行线程访问到的对象；二是对象回收过程，将未标记的对象（垃圾对象）占用的内存释放到内存空闲链中。

标记过程要遍历整个对象存储空间，这是一项耗时费力的工作。遍历从根集开始，根集的选取应保证正确性和高效性。正确性要求对每一个活对象（应用程序可访问到的对象），都存在一个根集到该对象的引用序列。Java芯片系统中的根集包括系统类表入口和线程队列。为了提高遍历的效率，我们在对象的组织上考虑了对检查的支持，在对象域的组织上，我们将引用域连续存放，这样就很容易识别出引用域。由于线程栈中也会出现引用，我们在对象类中对栈结构作了描述，遍历时，通过线程对象找到当前线程栈上的对象，而后就很容易识别引用域。这样比较容易标记所有线程可引用到的对象。

一般情况，垃圾回收标记过程应不可中断，因为中断期间运行的应用程序可能改变对象之间的引用关系。芯片系统中允许中断异步垃圾的回收，为保证正确性，芯片系统提供硬件的同步支持。在程序执行对象域写指令时，硬件自动检查对象头中的标志位。当标志位满足一定条件时，引起一次陷入，执行相应的处理方法，完成同步操作。

芯片系统的垃圾回收通过异步垃圾回收，有效地降低了同步垃圾回收进行的频度，从而提高了系统的实时性；而且通过由分配引起的同步回收，解决了异步垃圾回收浮动垃圾的问题，从整体上提高了系统性能。

总结 Java芯片系统面向嵌入式系统，其设计和实现具有较强的针对性，存储管理也不例外。我们在了解Java存储管理要求之后，结合Java芯片系统和应用特点，设计了存储管理方案，并进行了实现。由于针对性强，设计时可以比较灵活，且效果良好。

参考文献

1 Lindolm T, Yellin F. The Java Virtual Machine Specification. Addison-Wesley, 1996

2 Wilkinson T J. Kaffe 0.8.4 Source Code. Available at: http://www.kaffe.org

3 Agesen O, Detlets D. Finding References in Java Stacks. Available at: http://www.sun.com, also in OOPSLA'97 Workshop on Garbage Collectionand Memory Menagement, 1997

4 朱海滨. 面向对象. 长沙：国防科技大学出版社

5 Wilson P. Uniprocessor Garbage Collection Techniques

（上接第50页）

● 可扩展的定理基础，支持Z工具包并允许修改以适应新的应用领域；

● 灵活的用户界面，支持构造与程序推导的重用。

本文还对现有的精化工具进行了比较。现有工具大多不支持数据精化，自动化程度尚低，而且不支持大型复杂的软件精化，因此在精化工具的研究方面，还有许多工作要做。

参考文献

1 Carrington D, et al. Structured presentation of refinements and proofs: [TR-95-46]. Software Verification Research Centre, The University of Queensland

2 Hsiang J, et al. The term rewriting approach to automated theorem proving. Journal of Logic Programming, 1992

3 Utting M, Whitwell K. Ergo user manual: [TR-93-19]. Software Verification Research Centre, The University of Queensland, 1994

4 Robinson P, et al. Formalizing a hierarchical structure of practical mathematical reasoning. Journal of Logic and Computation, 1993

5 Vickers T. An overview of a refinement editor. In: Proc. of the 5th Australian Soft. Eng. Conf., 1990

6 von Wright J. Program refinement by theorem prover. In Till [Ti194]

7 Back R J R, et al. A program refinement environment. Ser A 139, Abo Akademi, 1992

8 Bloesch A, et al. The Cogito methodology and system. IEEE Computer Society Press, 1994