软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(9):4218−4241 [doi: 10.13328/j.cnki.jos.007134]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

命令式动态规划类算法程序推导及机械化验证*

左正康[1,2], 孙 欢[1], 王昌晶[1,2], 游 珍[2,3], 黄 箐[2], 王唱唱[1]

[1](江西师范大学 数字产业学院, 江西 上饶 334006)

[2](江西师范大学 计算机信息工程学院, 江西 南昌 330022)

[3](网络化支撑软件国家国际科技合作基地 (江西师范大学), 江西 南昌 330022)

通信作者: 游珍, E-mail: youzhen@jxnu.edu.cn

摘 要: 动态规划是一种递归求解问题最优解的方法, 主要通过求解子问题的解并组合这些解来求解原问题. 由于其子问题之间存在大量依赖关系和约束条件, 所以验证过程繁琐, 尤其对命令式动态规划类算法程序正确性验证是一个难点. 基于动态规划类算法 Isabelle/HOL 函数式建模与验证, 通过证明命令式动态规划类算法程序与其的等价性, 避免证明正确性时处理复杂的依赖关系和约束条件, 提出命令式动态规划类算法程序设计框架及其机械化验证. 首先, 根据动态规划类算法的优化方法 (备忘录方法) 和性质 (最优子结构性质和子问题重叠性质) 描述问题规约、归纳递推关系式和形式化构造出循环不变式, 并且基于递推关系式生成 IMP (minimalistic imperative programming language) 代码; 其次, 将问题规约、循环不变式和生成的 IMP 代码输入 VCG (verification condition generator), 自动生成正确性的验证条件; 然后, 在 Isabelle/HOL 定理证明器中对验证条件进行机械化验证. 算法首先设计为命令式动态规划类算法的一般形式, 并进一步实例化得到具体算法. 最后, 例证所提框架的有效性, 为动态规划类算法的自动化推导和验证提供参考价值.

关键词: Isabelle/HOL; 机械化验证; 动态规划; 命令式; VCG

中图法分类号: TP301

中文引用格式: 左正康, 孙欢, 王昌晶, 游珍, 黄箐, 王唱唱. 命令式动态规划类算法程序推导及机械化验证. 软件学报, 2024, 35(9): 4218–4241. http://www.jos.org.cn/1000-9825/7134.htm

英文引用格式: Zuo ZK, Sun H, Wang CJ, You Z, Huang Q, Wang CC. Program Derivation and Mechanized Verification of Imperative Dynamic Programming Algorithms. Ruan Jian Xue Bao/Journal of Software, 2024, 35(9): 4218–4241 (in Chinese). http://www.jos.org.cn/1000-9825/7134.htm

Program Derivation and Mechanized Verification of Imperative Dynamic Programming Algorithms

ZUO Zheng-Kang[1,2], SUN Huan[1], WANG Chang-Jing[1,2], YOU Zhen[2,3], HUANG Qing[2], WANG Chang-Chang[1]

[1](School of Digital Industry, Jiangxi Normal University, Shangrao 334006, China)

[2](School of Computer Information Engineering, Jiangxi Normal University, Nanchang 330022, China)

[3](National-level International S & T Cooperation Base of Networked Supporting Software (Jiangxi Normal University), Nanchang 330022, China)

Abstract: As a recursive method for finding the optimal solution to a problem, dynamic programming mainly solves the original problem by first solving the subproblems and then combining their solutions. Due to a large number of dependencies and constraints among its subproblems, the validation procedure is laborious, and especially the correctness verification of imperative dynamic programming algorithms is a challenge. Based on the functional modeling and verification of dynamic programming algorithms Isabelle/HOL, this study

* 基金项目: 国家自然科学基金 (62262031); 江西省自然科学基金 (20232BAB202010, 20212BAB202018); 江西省教育厅科技项目 (GJJ210307, GJJ210333, GJJ2200302); 江西省主要学科学术与技术带头人培养项目 (20232BCJ22013)
本文由“形式化方法与应用”专题特约编辑曹钦翔副教授、宋富研究员、詹乃军研究员推荐.
收稿时间: 2023-09-11; 修改时间: 2023-10-30, 2023-12-13; 采用时间: 2023-12-20; jos 在线出版时间: 2024-01-05
CNKI 网络首发时间: 2024-04-28

avoids dealing with complex dependencies and constraints in proving correctness by verifying the equivalence of imperative dynamic programming algorithms and their programs. Meanwhile, a framework for the design of imperative dynamic programming algorithmic programs and their mechanized verification are proposed. First, according to the optimization method (memo method) and properties (optimal substructure property and subproblems overlapping property) of dynamic programming algorithms, the problem specification is described, the recursive relations are inductively derived, and the loop invariants are formally constructed. Then, the IMP (minimalistic imperative programming language) code is generated based on the recursive relations. Second, the problem specification, loop invariants, and generated IMP code are fed into VCG (verification condition generator) to generate the verification condition for correctness automatically. Additionally, the verification condition is then mechanically verified in the Isabelle/HOL theorem prover. The algorithm is initially designed in the general form of an imperative dynamic programming algorithm and further instantiated to obtain specific algorithms. Finally, the effectiveness of the proposed framework is validated by case studies to provide references for automated derivation and verification of dynamic programming algorithms.

Key words: Isabelle/HOL; mechanized verification; dynamic programming; imperative; verification condition generator (VCG)

动态规划是一种递归求解问题最优解的算法设计方法, 其在求解问题时, 常将原问题拆解为一系列子问题, 并通过计算子问题的解来逐步推导出最终问题的解[1]. 这些子问题之间存在大量依赖关系和约束条件, 导致验证过程复杂. 尤其, 对于命令式动态规划类算法程序的正确性进行验证是一个难点.

形式化方法是基于严格数学基础, 对计算机硬件和软件进行描述、开发和验证的技术[2]. 使用形式化方法可以有效保证动态规划类算法正确性, 近年来已经出现了大量的相关研究工作. 在动态规划类算法形式化推导方面, 文献 [3−5] 关注于 0-1 背包问题及其变形问题的形式化推导, 但未形成适用于一类动态规划问题的通用框架; 文献 [6−8] 则是对一类算法的实现方法和递推关系式的形式化推导, 但缺乏对推导结果的机械验证. 在动态规划类算法形式化验证方面, Bortin 进行了动态规划 CYK 算法的函数式验证[9], Wimmer 等人提供了一个基于 Isabelle/HOL[10]的轻量级框架[11,12], 可以对动态规划类算法进行函数式建模与验证. 上述研究主要集中在使用形式化方法对函数式动态规划类算法进行验证.

然而, 命令式动态规划类算法程序依赖于程序状态导致严重的副作用, 这使得证明命令式动态规划类算法程序的正确性尤其困难. 命令式程序相较于函数式程序有更为广泛的使用, 因此, 对命令式动态规划类算法进行验证变得更加紧迫和重要. 为了验证命令式动态规划类算法, 基于 Wimmer 等人所实现的动态规划类算法 Isabelle/HOL 函数式建模与验证的研究, 本文通过证明命令式动态规划类算法程序与其等价性, 避免了在证明正确性时处理复杂的依赖关系和约束条件, 并进一步提出了命令式动态规划类算法程序设计框架及机械化验证.

具体工作贡献如下.

1) 为了解决命令式动态规划类算法验证困难, 基于 Wimmer 等人实现的动态规划类算法函数式建模与验证, 通过证明命令式动态规划类算法程序与函数式程序的等价性, 提出了一种命令式动态规划类算法程序设计框架, 避免证明命令式算法程序正确性时处理复杂的依赖关系和约束条件.

2) 提出命令式动态规划类算法程序机械化验证方法. 该验证方法自动生成正确性的验证条件, 对其验证采用 Isabelle/HOL 定理证明器, 实现了验证的自动性和机械性, 有效保证算法程序的正确性和可靠性.

3) 基于动态规划类算法设计框架和机械化验证方法, 实例化了 0-1 背包问题和最长公共子序列问题的设计和验证, 从而完成了从一般形式得到两个实例的推导. 检验了所提框架的有效性, 为今后动态规划类算法的自动化推导和验证提供参考价值.

基于以上分析, 本文给出了命令式动态规划类算法程序推导及机械化验证, 其中设计框架和机械化验证方法如后文图 1 所示. 首先, 命令式动态规划类算法的设计使用动态规划的备忘录性质描述问题规约, 动态规划的最优子结构性质和子问题重叠性质归纳递推关系式以及动态规划的最优子结构性质构造动态规划类算法的循环不变式[13], 并且生成递推关系式对应的 IMP 代码[10]; 然后将上述生成的问题规约、IMP 代码和循环不变式输入 VCG[10], 自动生成验证条件; 最后利用 Isabelle/HOL 定理证明器对生成的验证条件进行验证, 保证证明的可靠性.

1 相关工作

动态规划是一种高效的算法设计方法. 它通过将原始问题拆分为子问题, 并利用已经计算过的子问题的结果, 避免了重复计算, 从而显著提高了计算效率. 近年来, 针对动态规划算法的研究, 主要包括设计、形式化推导以及验证这 3 个方面.

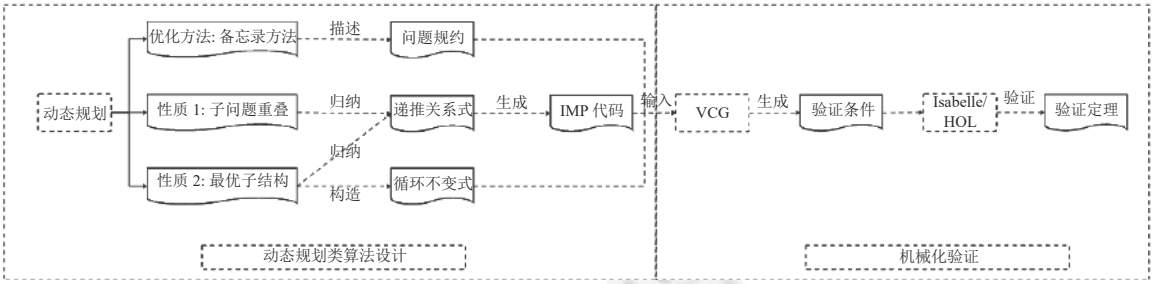

图 1 命令式动态规划类算法程序设计框架及机械化验证

● 动态规划算法的设计. 文献 [14] 主要工作是通过优化递推关系式, 达到提高动态规划类算法时间效率的目的, 也探讨动态规划类算法的应用. 文献 [15] 使用并行化技术对动态规划类算法进行优化, 通过交互式的逐步求精方法, 开发了由 SMT 验证求精的应用条件. 文献 [16] 介绍如何在 Apache Spark 上高效执行动态规划算法, 总结 4 种动态规划类算法的设计方法, 并解释如何将这些算法映射到 Spark 框架上. 文献 [17] 基于半环结构提出了一种可解释动态规划类算法的方法, 借助半环的代数模型构建 Haskell 库, 实现动态规划类算法, 通过保留执行过程的结果, 来解释做出的最优决策选择. 上述文献关注动态规划一类算法的设计优化, 主要通过测试来检验算法正确性. 然而, 测试只能发现部分错误, 由于无法对算法的所有可能情况做到全覆盖, 因此无法证明算法无错, 要确保算法的正确性仍需要增加严格的形式化方法验证.

● 动态规划算法的形式化推导. 文献 [3−5] 针对经典 0-1 背包问题以及若干变形问题进行手工形式化推导, 并没有推导一类动态规划算法的形式化方法. 文献 [6,7] 使用数学符号形式化推导动态规划类算法的递推关系式, 并且通过具体的问题展示如何根据问题的特点和要求, 建立状态转移方程, 最终得到问题的最优解. 文献 [8] 针对自上而下和自底向上两种动态规划类算法进行形式化推导, 并且介绍了动态规划算法在优化问题中的应用. 其中, 自上而下方法结合了分治和备忘录的思想, 通过存储实例的解决方案来优化算法性能; 自底向上方法通过在每一步中生成新的子实例来避免了递归调用的内存和时间开销, 不需要对所有可能的子实例进行解决、存储和组合. 文献 [6−8] 是对动态规划一类算法关于实现方法和递推关系式的形式化推导, 缺少对算法正确性进行机械化验证.

● 动态规划算法的验证. Bortin 使用 Isabelle/HOL 定理证明器定义出备忘录表的方式[9], 对典型动态规划 CYK (Cocke-Younger-Kasami algorithm) 算法进行形式化定义和机械化验证, 辅助函数和验证代码有千余行. Bortin 针对一个典型动态规划问题进行验证, 而 Wimmer 等人对于一类动态规划问题进行研究[11,12]. Wimmer 等人介绍基于 Isabelle/HOL 定理证明器验证动态规划类算法结果是最优解的正确性, 并且设计了带有自动记忆功能的递归函数, 给出自动验证正确性的框架, 具体实现见文献 [12]. 文献 [9,11,12] 是主要集中在使用形式化方法对函数式动态规划算法进行验证, 然而命令式程序相较于函数式程序逻辑结构更复杂, 且依赖于程序状态导致严重的副作用, 因此验证命令式动态规划类算法程序更加困难.

针对以上问题, 本文基于 Wimmer 等人所实现的动态规划类算法函数式建模与验证, 通过证明命令式动态规划类算法程序与其等价性, 提出了命令式动态规划类算法程序设计框架及机械化验证. 避免在证明正确性时处理复杂的依赖关系和约束条件, 解决了命令式代码验证困难的问题. 其中验证条件采用 VCG 自动生成, 验证过程采用 Isabelle/HOL 定理证明器对验证条件进行机械化验证; 基于命令式动态规划类算法设计框架和机械验证, 实例

化了 0-1 背包问题和最长公共子序列问题的设计和验证, 完成了从一般形式得到具体实例的推导, 验证了所提框架的有效性, 为未来动态规划类算法的自动化推导和验证提供了有价值的参考.

2 动态规划类算法的设计框架

本节主要介绍命令式动态规划类算法程序设计框架. 首先根据动态规划类算法的备忘录方法对问题规约进行描述, 包括前置条件和后置条件; 其次依据动态规划类算法的子问题重叠性质和最优子结构性质归纳递推关系式; 然后根据动态规划类算法的最优子结构性质构造循环不变式; 最后基于递推关系式生成相应的 IMP 代码[10], 完成命令式动态规划类算法程序设计.

该设计框架基于 Wimmer 等人所实现的动态规划类算法函数式建模, 并且文献 [12] 直接证明了函数式求解结果满足最优解. 本节通过在后置条件和循环不变式中分别设计命令式动态规划类算法程序运行结束和过程中的解与相应函数式程序解是相等的, 间接验证了命令式动态规划类算法程序 IMP 代码的正确性.

2.1 问题规约描述

动态规划法将待求解原问题 P 分解成若干个相互重叠的子问题, 记为 $P1$, $P2$,…, Pn, 每个子问题对应决策过程的一个阶段. 在解决问题的过程中, 为了避免重复计算, 动态规划法引入了备忘录的概念, 用于存储计算结果, 避免重复计算子问题[1]. 备忘录将已经求解子问题 Pi 的最优解 $f(Pi)$, 以二维最优决策表 m 的方式进行存储, 这个最优解可以是最大值、最小值、最长路径、最短路径等不同形式, 根据具体问题而定.

如图 2 所示, 在使用递归算法自顶向下对最优决策表 m 进行求解时, 按照具体算法的步骤, 从左到右、从上到下依次填写最优决策表 m. 其中, i 表示阶段从 0 开始到 u, j 表示状态从 0 开始到 v, 最终 i, j 分别到达求解最后目标问题 $m[u, v]$ 的阶段 u 和状态 v, u=*length* $m-1$, v = *length* $(m[i-1])-1$[18,19]. i, j 分别作为外循环变量和内循环变量遍历二维最优决策表 m, 最终外循环变量 i 遍历最优决策表跳出循环, 即结束时 i=*length* m.

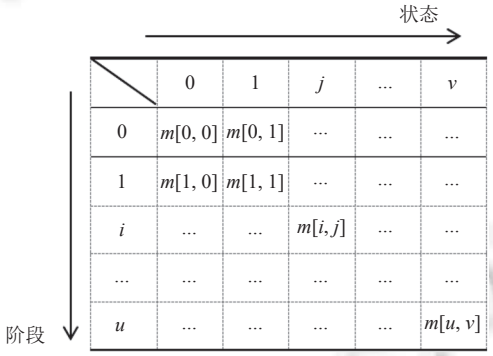

图 2 动态规划类算法求解过程

针对动态规划类算法, 借助形式化的方法来描述解决这类问题的程序规约<P, Q>, 其中 P 和 Q 分别是该算法程序的前置条件和后置条件[20].

(1) 前置条件 P 是程序执行之前满足的条件, 即在执行动态规划算法之前, 必须满足的条件. 最优决策表 m 作为动态规划类算法的备忘录, 其行数要满足大于等于 1.

(2) 后置条件 Q 表示程序终止时应满足的条件, 即在执行动态规划算法之后, 得到的结果应该满足的条件. 算法终止的时候, 设对应的外循环变量 x 等于跳出循环的值 *length* l, 即 x=*length* l, 表示各阶段状态的所有子问题均已求解. 文献 [12] 给出关于函数式动态规划算法的定义 (设为 G), 并且直接证明定义的函数求解结果满足最优解. 因此, 本文主要通过证明函数 G 与命令式动态规划类算法的等价性来间接确保命令式动态规划算法的正确性, 即证明命令式动态规划类算法求解最优决策表 m 的最后目标问题 $m[u, v]$ 与函数式定义的对应位置 $G\ u\ v\ \overline{\lambda}$ 相等, 以

此表示最终得到的结果满足最优解, 从而证明了命令式动态规划类算法的正确性.

综上所述, 基于二维最优决策表 m 作为动态规划类算法的备忘录, 描述可得在 Isabelle/HOL 中命令式动态规划类算法程序规约如下.

■ 前置条件 P: $length\ m \geqslant 1$.

■ 后置条件 Q: $x = length\ l \wedge m!u!v = G\ u\ v\ \overline{\lambda}$.

其中, m 表示存储动态规划类算法结果的备忘录, 采用的二维最优决策表的形式存储; x 表示遍历 m 的外循环变量, x=$length\ l$ 表示最终计算结束跳出循环时 x 的取值为 $length\ l$; u 和 v 分别表示最后目标问题的阶段和状态; $m!u!v$ 表示在 Isabelle/HOL 中对二维最优决策表 m 最后目标问题的取值; $\overline{\lambda}$ 表示文献 [12] 定义函数式动态规划类算法时除去递归变量的其他变量; $G\ u\ v\ \overline{\lambda}$ 表示通过函数式定义对于最后目标问题的求解结果.

2.2 递推关系式归纳

动态规划类算法通过划分原问题为若干个子问题, 并利用最优子结构和重叠子问题两个性质进行求解. 动态规划的重叠子问题指在解决原问题 P 的过程中, 会遇到相同的子问题 Pi. 而最优子结构性质指问题 P 的最优解 $f(P)$ 由其子问题 Pi 的最优解 $f(Pi)$ 组合而成. 为了避免相同子问题的重复求解, 动态规划类算法对每一个子问题只计算一次, 并将计算结果保存在备忘录中, 以获得较高的求解效率. 递推关系式 (recurrence relation, RC) 是根据最优子结构性质以及已经求解过的子问题结果, 得出整个问题的最优解. 这个过程是从次小的问题开始, 通过将问题划分为多个子问题, 一步步向较大的问题进行转化, 最终得到原问题 P 的最优解 $f(P)$.

根据动态规划类算法的两个性质, 可知动态规划是多阶段最优化决策解决问题的过程, 是将过程分成若干个互相联系的阶段, 在它的每一阶段都需要做出决策. 解决问题时, 一般由初始状态 (阶段 i=1) 开始, 通过对中间阶段决策的选择, 达到结束状态 (阶段 i=$length\ m$). 这些决策形成了一个决策序列, 同时确定了完成整个过程的一条最优的活动路线. 依据动态规划多阶段决策的性质, 递推关系式反映阶段之间的依赖关系. 本文提出了动态规划类算法的递推关系式归纳步骤.

步骤 RC_1. 定义阶段和状态. 阶段是问题求解过程的一个时段, 状态表示分析对象在某个阶段的状况, 用变量 i 表示阶段变量, 变量 j 表示状态变量.

步骤 RC_2. 确定决策. 当各阶段的状态取定以后, 就可以做出不同的决定, 从而确定允许进入到当前 i 阶段 j 状态的阶段和状态分别对应 i'和 j'; 主要分为 3 种情况: i 减少, j 不变, 即 i'<i, j'=j; i 不变, j 减少, 即 i'=i, j'<j; i 减少, j 也减少, 即 i'<i, j'<j.

步骤 RC_3. 给出允许决策集合. 决策的取值往往要限制在一定的范围内, 用 $D(i, j)$ 表示允许进入到当前 i 阶段 j 状态的允许决策集合. 则有 $(i', j') \in D(i, j)$.

步骤 RC_4. 定义函数.

● 辅助函数. 当前 i 阶段 j 状态下可能的取值不是允许决策之间通过取值直接得出, 还需要根据具体问题允许决策通过辅助函数 $X(i, j)$ 来得出.

● 依赖函数. 依赖关系用函数 t 表示, 即表示允许决策与当前 i 阶段 j 状态之间的取值存在怎样的依赖关系.

● 最优化函数. 用于明确问题的优化目标, 即在问题中达到的最佳结果, 可以表示为 max (最大) 或者 min (最小), 用最优化函数 opt 表示.

● 最优指标函数. 用于衡量所选定策略优劣的数量指标称为指标函数, 第 i 阶段第 j 状态的最优指标函数记作 $F(i, j)$.

● 判断函数. 对于具体问题需要分情况讨论对应的递推关系式, 该分情况指的是关于 i 和 j 的某种关系进行判断. 设关于 i 的函数是 Ji, 关于 j 的函数是 Jj, 且关于 i 和 j 的函数是 $Judge$, 不同情况关于 i 和 j 的函数分别是 $Judge_1, Judge_2$ 等.

步骤 RC_5. 归纳动态规划的递推关系式.

$$Judge(Ji(i),Jj(j)) \to F(i,j) = \underset{(i',j')\in D(i,j)}{\text{opt}} t(F(i',j'),X(i,j)).$$

递推关系式和最优决策表的关联如下.

● j 表示第 i 阶段的某个状态, 使用最优指标函数记作 $F(i,j)$ 对最优决策表 $m[i,j]$ 进行求解.

● 函数 t 用来表示 i 阶段 j 状态的最优指标函数 $F(i,j)$ 和 i'阶段 j'状态的最优指标函数 $F(i',j')$ 之间的递推关系, 即用于表示最优决策表 $m[i,j]$ 和 $m[i',j']$ 的递推关系式, $m[i',j']$ 是 $m[i,j]$ 的规模更小子问题.

2.3 循环不变式构造

循环不变式 (loop invariant, INV) 指的是每次迭代之前和之后都有效的断言, 这一特性对于验证循环程序具有至关重要的作用. 给定一个循环 while(C){S}, 一个前置条件 P 和一个后置条件 Q, 当验证 while(C){S}涉及找到一个循环不变式 ρ, 它可以从前置条件得出结论, 并隐含后置条件[20]. 循环程序的验证必须满足以下 3 个条件, 其中第 2 个是描述循环的 Hoare 三元组[21]:

$$\mathrm{P} \Rightarrow \rho, \quad \{\rho \wedge \mathrm{C}\}\ \mathrm{S}\ \{\rho\}, \quad \rho \wedge \neg \mathrm{C} \Rightarrow \mathrm{Q}.$$

在第 2.1 节中已经给出动态规划类算法的前置条件 P 和后置条件 Q, 前置条件是 $length\ m \geqslant 1$, 后置条件是 $x = length\ l \wedge m!u!v = G\ u\ v\ \overline{\lambda}$. 动态规划类算法循环不变式的构造依据以下两点: 第一, 最优子结构性质, 即求解问题 P 的最优解 $f(P)$ 由其子问题 Pi 的最优解 $f(Pi)$ 组合而成; 第二, 循环不变式是隐含后置条件. 综上可以得出循环不变式满足在循环遍历求解二维最优决策表 m 的每一个子问题时都是最优解, 即可得求解的每个子问题解与函数 G 对应求解结果相等, 从而可得在求解的过程中命令式动态规划类算法和函数 G 是等价的, 最终可以证明后置断言满足最优决策表 m 最终目标求解的结果 $m!u!v = G\ u\ v\ \overline{\lambda}$. 本文给出了动态规划类算法的循环不变式构造过程如下.

步骤 INV_1. 给出循环变量范围. 对二维最优决策表 m 进行更新需要使用更新函数, 更新函数的证明依赖循环变量 (设 x 为外循环变量, y 为内循环变量) 的范围, 因此需要给出循环变量的范围. 其中循环变量的上界确定, 依赖于求解 m 时遍历的方式. 例如在前文图 2 中自顶向下遍历求解 m, 因此上界确定依据的是二维决策表 m 的大小. 以图 2 遍历方式为例, 在双重循环中, 外循环变量 x 即为阶段 i 对应行的长度, 即 $i<length\ m$; 内循环变量 y 即为状态 j 对应列的长度, 即 $j<length\ (m!(i–1))$. 但在循环完毕时, 循环变量会进入循环不变式中进行判断, 所以循环不变式要满足跳出循环的条件, 即外循环不变式 i 的范围要包含 $i=length\ m$, 同理内循环不变式 j 的范围要包含 $j=length\ (m!(i–1))$. 此外 j 跳出循环会进入到外循环不变式进行判断, 因此在外循环不变式中 j 的范围要包含 $j=length\ (m!(i–1))$; 循环变量的下界确定, 依据递推关系式等式两边所要求解的阶段和状态大于等于 0, 例如递推关系式是关于 $m(i,j)$ 与 $m(i–1,j)$ 的关系, 则要满足 i 必须大于等于 1, j 大于等于 0. 可得图 2 遍历方式下外循环不变式的循环变量的范围为 $i\geqslant a \wedge i\leqslant length\ m \wedge j\leqslant length\ (m!(i–1)) \wedge j\geqslant b$; 内循环不变式的循环变量的范围为 $i\geqslant a \wedge i<length\ m \wedge j\leqslant length\ (m!(i–1)) \wedge j\geqslant b$. 其中 a, b 分别表示 i, j 的下界, 根据具体问题的取值为 0 或 1.

综上所述, 给出双重循环外循环不变式的循环变量的范围为 $x\geqslant a_{下} \wedge x\leqslant a_{上} \wedge y\leqslant b_{上} \wedge y\geqslant b_{下}$; 双重循环内循环不变式的循环变量的范围为 $x\geqslant a_{下} \wedge x<a_{上} \wedge y\leqslant b_{上} \wedge y\geqslant b_{下}$. 其中 $a_{上}$, $a_{下}$分别表示外循环变量 x 的上界和下界; $b_{上}$, $b_{下}$分别表示内循环变量 y 的上界和下界, $a_{下}$和 $b_{下}$根据具体问题的取值为 0 或 1.

步骤 INV_2. 确定求解问题变化规律. 循环遍历 (设 x 为外循环变量, y 为内循环变量) 求解二维最优决策表 m 的每一个子问题都要满足最优解, 则所求结果应满足与函数式求解结果相等. 以图 2 遍历方式为例, 阶段 i 即外循环变量, 状态 j 即内循环变量, 对二维最优决策表 m 进行外循环遍历到第 i 行时, 则有已经求解的 $k\ (\forall k.\ k < i)$ 行元素的值都等于 $G\ k\ j\ \overline{\lambda}$; 此时对二维最优决策表 m 进行内循环遍历到 j 列, 则有 i 行已经遍历的 $k\ (\forall k.\ k < j)$ 列元素的值都等于 $G\ i\ k\ \overline{\lambda}$. 此外在进入内循环时, 是已知外循环任意 k 小于 i, k 行元素都满足求解的值是最优解, 通过一次内循环完成 i 行所有元素的求解, 当跳出内循环时, 就可以得到已经遍历的 $k\ (\forall k.\ k \leqslant i)$ 行元素的值都等于 $G\ k\ j\ \overline{\lambda}$. 对步骤 INV_1 求解时, 存在一个特殊情况 b=1. 在这种情况下对于第 0 列的初始数据 $(\forall k.\ k < length\ m \to m!k!0 = 0)$

进入不到内循环进行证明 $(\forall k.\ k<j\rightarrow m!i!k=G\ i\ k\ \overline{\lambda})$, 所以循环不变式需加入第 0 列数据. 可得, 图 2 遍历方式下根据 b 的取值将确定求解问题变化规律分为两种情况. 第 1 种, b=0 时给出外循环不变式的求解问题变化规律为 $(\forall k.\ k<i\rightarrow\ (\forall j.\ j<length(m!(k-1))\rightarrow m!k!j=G\ k\ j\ \overline{\lambda})$, 内循环不变式的变化规律为 $(\forall k.\ k<i\rightarrow\ (\forall j.\ j<length(m!(k-1))\rightarrow m!k!j=G\ k\ j\ \overline{\lambda})\ )\wedge\ (\forall k.\ k<j\rightarrow m!i!k=G\ i\ k\ \overline{\lambda})$; 第 2 种, b=1 时只需要在第 1 种情况 b=0 的内外循环不变式分别加上 $(\forall k.\ k<length\ m\rightarrow m!k!0=0)$.

根据图 2 确定求解问题的变化规律, 归纳双重循环外循环不变式求解问题的变化规律. 图 2 中满足阶段 i 即外循环变量, 状态 j 即内循环变量, 但并非所有动态规划都满足该性质, 因此需设通过当前外循环变量 x 和内循环变量 y 的值表示此时的阶段 i 和状态 j, 即设 i=$ii(x,y)$, j=$jj(x,y)$. 综上所述, 根据 $b_{下}$的取值将确定求解问题变化规律分为两种情况. 第 1 种, $b_{下}$=0 时给出双重循环外循环不变式的求解问题变化规律为 $(\forall k.\ k<x\rightarrow\ (\forall y.\ y<b_{上}\rightarrow m!ii(k,y)!jj(k,y)=G\ ii(k,y)\ jj(k,y)\ \overline{\lambda}))$, 双重循环内循环不变式的变化规律为 $(\forall k.\ k<x\rightarrow(\forall y.\ y<b_{上}\rightarrow m!ii(k,y)!jj(k,y)=G\ ii(k,y)\ jj(k,y)\ \overline{\lambda}))\wedge\ (\forall k.\ k<y\rightarrow m!ii(x,k)!jj(x,k)=G\ ii(x,k)\ jj(x,k)\ \overline{\lambda})$; 第 2 种, $b_{下}$=1 时只需要在第 1 种情况 $b_{下}$=0 的内外循环不变式分别加上根据遍历方式内循环变量 y=0 的初始元素值.

步骤 INV_3. 满足二维最优决策表性质. 动态规划类算法的求解结果存放在二维最优决策表 m 中, 因此循环不变式依赖于二维最优决策表 m 的性质, 其中主要包括二维最优决策表 m 的高度和宽度性质. 高度要满足大于等于 1, 即 $length\ m\geqslant 1$; 宽度要满足 m 每一列的长度必须是相等的, 并且 m 的宽度必须大于等于 1. 即 $(\forall k.\ k<length\ m\rightarrow length\ (m!k)\ =length\ (m!(k-1))\wedge length\ (m!k)\geqslant 1)$; 还存在特殊情况二维最优决策表 m 的高度和宽度相等, 即 $length(m!1)\ =length\ m$.

综上所述, 给出双重循环内外循环不变式满足二维最优决策表性质为 $length\ m\geqslant 1\wedge(\forall k.\ k<length\ m\rightarrow length(m!k)\ =length(m!(k-1))\wedge length(m!k)\geqslant 1)$, 当二维最优决策表 m 的高度和宽度相等时, 只需要再加上 $length(m!1)\ =length\ m$.

步骤 INV_4. 构造动态规划类算法的循环不变式, $\mathrm{INV}\{\mathrm{INV}_1\wedge\mathrm{INV}_2\wedge\mathrm{INV}_3\}$. 如表 1 所示.

表 1 动态规划类算法的循环不变式

循环不变式构造步骤	外循环不变式构造	内循环不变式构造
步骤INV_1. 给出循环变量范围	$x\geqslant a_{下}\wedge x\leqslant a_{上}\wedge y\leqslant b_{上}\wedge y\geqslant b_{下}$	$x\geqslant a_{下}\wedge x<a_{上}\wedge y\leqslant b_{上}\wedge y\geqslant b_{下}$
步骤INV_2. 确定求解问题变化规律	$(\forall k.\ k<x\rightarrow$ $(\forall y.\ y<b_{上}\rightarrow$ $m!ii(k,y)!jj(k,y)=G\ ii(k,y)\ jj(k,y)\ \overline{\lambda}))$ $\wedge y=0$情况下的初始值 //$b_{下}=1$ 时加入	$(\forall k.\ k<x\rightarrow$ $(\forall y.\ y<b_{上}\rightarrow$ $m!ii(k,y)!jj(k,y)=G\ ii(k,y)\ jj(k,y)\ \overline{\lambda})\ )$ $\wedge\ (\forall k.\ k<y\rightarrow$ $m!ii(x,k)!jj(x,k)=G\ ii(x,k)\ jj(x,k)\ \overline{\lambda})$ $\wedge y=0$情况下的初始值// $b_{下}=1$ 时加入
步骤INV_3. 满足二维最优决策表性质	$length\ m\geqslant 1$ $\wedge(\forall k.\ k<length\ m\rightarrow$ $length\ (m!k)\ =length\ (m!(k-1))$ $\wedge length\ (m!k)\geqslant 1)$ $\wedge length(m!1)\ =length\ m$ //m的高度和宽度相等时加入	$length\ m\geqslant 1$ $\wedge(\forall k.\ k<length\ m\rightarrow$ $length\ (m!k)\ =length\ (m!(k-1))$ $\wedge length\ (m!k)\geqslant 1)$ $\wedge length(m!1)\ =length\ m$ //m的高度和宽度相等时加入
步骤INV_4. 构造动态规划类算法的循环不变式	$\mathrm{INV}\{\mathrm{INV}_1\wedge\mathrm{INV}_2\wedge\mathrm{INV}_3\}$	$\mathrm{INV}\{\mathrm{INV}_1\wedge\mathrm{INV}_2\wedge\mathrm{INV}_3\}$

基于表 1 动态规划类算法的循环不变式, 可以设 $INV\_DP\_E(a_{上}, a_{下}, b_{上}, b_{下}, G, \overline{\lambda})$ 和 $INV\_DP\_I(a_{上}, a_{下}, b_{上}, b_{下}, G, \overline{\lambda})$ 分别为双重循环的外循环不变式和内循环不变式, 输入参数为 $a_{上}$、$a_{下}$、$b_{上}$、$b_{下}$, 、G 和 $\overline{\lambda}$. $a_{上}$、$a_{下}$分别表示外循环变量 x 的上界和下界; $b_{上}$、$b_{下}$分别表示内循环变量 y 的上界和下界, $a_{下}$和 $b_{下}$, 取值是 0 或 1; G 表示动态规划类算法的函数式定义; $\overline{\lambda}$ 表示函数式动态规划类算法除去递归变量的其他变量. 例如 $INV\_DP\_E(a_{上}, a_{下}, b_{上}, b_{下}, G, \overline{\lambda})$ 的定义如下.

$INV\_DP\_E(a_{上},a_{下},b_{上},b_{下},G,\overline{\lambda}) = x \geqslant a_{下} \wedge x \leqslant a_{上} \wedge y \leqslant b_{上} \wedge y \geqslant b_{下}$ //INV1

$\wedge(\forall k.k < x \rightarrow (\forall y.y < b_{上} \rightarrow m!ii(k,y)!jj(k,y) = G\ ii(k,y)\ jj(k,y)\ \overline{\lambda}))$ //INV2

$\wedge y = 0$情况下的初始值 //$b_{下} = 1$时, INV2包含这条语句

$\wedge length\ m \geqslant 1$ //INV3

$\wedge(\forall k\ .k < length\ m \rightarrow length\ (m!k) = length\ (m!(k-1)) \wedge length\ (m!k) \geqslant 1)$ //INV3

$\wedge length\ (m!1) = length\ m$ //m的高度和宽度相等时, INV3 加入

2.4 代码生成

本节首先介绍文献 [10] 给出的 IMP 语言的基本结构和语法, 进而提出动态规划问题 IMP 代码的设计步骤. 然后通过 C++代码自动生成系统将抽象的 IMP 代码转换为能够在计算机上直接运行的 C++代码[22].

2.4.1 极简命令式编程语言 (IMP)

IMP[10]是一种极简命令式编程语言, 力求简洁, 只保留最少的命令 (command), 包括空语句、赋值语句、顺序语句、条件语句和循环语句, 其中空语句 **SKIP** 可以用于构造没有 **ELSE** 的条件语句等. 在 Isabelle/HOL 中, datatype 归纳定义的 com 数据类型的 IMP 抽象语法如表 2 的抽象语法列所示, 其 4 个复合构造函数的具体中缀语法如表 2 的具体语法列所示. 其中, **Assign** *vname aexp* 表示赋值语句, 将一个算术表达式的值赋给一个变量, 即对应 *vname*::=*aexp*; **Seq** *com com* 表示顺序语句, 将两个语句按照顺序执行, 即对应 *com*;;*com*; **If** *bexp com com* 表示条件语句, 根据一个布尔表达式的值选择性地执行两个语句中的一个, 即对应 **IF** *bexp* **THEN** *com* **ELSE** *com*; **While** *bexp com* 表示循环语句, 当一个布尔表达式的值为真时, 反复执行一个语句, 即对应 **WHILE** *bexp* **DO** *com*. 编写 IMP 代码时, 需要根据这些语句的含义来正确地组合和使用它们从而实现程序的功能.

表 2 IMP 语言的命令式定义

抽象语法	具体语法
datatype *com*=**SKIP**	*com*::=**SKIP**
| **Assign** *vname aexp*	| *vname*::=*aexp*
| **Sep** *com com*	| *com*;;*com*
| **If** *bexp com com*	| **IF** *bexp* **THEN** *com* **ELSE** *com*
| **While** *bexp com*	| **WHILE** *bexp* **DO** *com*

2.4.2 IMP 代码设计

基于文献 [10] 给出的 IMP 语言的基本结构和语法, 本节设计了动态规划 IMP 代码. 其最主要的循环体部分的编写依赖于第 2.2 节归纳的动态规划递推关系式, 设计步骤如下.

步骤 IMP_1. 确定初始变量和参数以及赋予初值. 根据递推关系式定义初始变量和参数, 这些变量包括: 循环变量 (双重循环对应 x, y)、最优二维决策表 m.

步骤 IMP_2. 确定循环的起始条件和终止条件. 根据具体问题的定义确定循环变量的起始条件和终止条件, 以确保循环能够在正确的范围内执行; 在双重循环中循环变量的终止条件对应二维最优决策表 m 的长度和宽度, 初始条件的确定和循环变量的下界确定相同, 要满足递推关系式两边的阶段和状态大于等于 0, 设置 x 和 y 的初值为 $a_{下}$和 $b_{下}$.

步骤 IMP_3. 确定循环体中当前状态的转移. 根据动态规划类算法递推关系式中的转移关系可得, 对于具体问题需要分类讨论阶段 i 和状态 j 的关系 $Judge(Ji(i), Jj(j))$ 以及对应的递推关系式, 例如一个递推关系式包含两种情况 $Judge_1(Ji(i), Jj(j))$ 和 $Judge_2(Ji(i), Jj(j))$, 则需要使用对应的递推关系式从已知状态 $m[i', j']$ 的值计算得到当前状态 $m[i, j]$ 的值. 具体语法是:

$$\begin{aligned}&(\textbf{IF}\ Judge_1(Ji(i), Jj(j))\ \textbf{THEN}\ m[i,j] ::= \underset{(i',j')\in D(i,j)}{\text{opt}}\ t(m[i',j'], X(i,j));;\\ &\textbf{ELSE}(\textbf{IF}\ Judge_2(Ji(i), Jj(j))\ \textbf{THEN}\ m[i,j] ::= \underset{(i',j')\in D(i,j)}{\text{opt}}\ t(m[i',j'], X(i,j));;\\ &\quad \textbf{ELSE SKIP});;);;\end{aligned}$$

并非所有动态规划都满足阶段 i 即对应外循环变量, 状态 j 即对应外循环变量, 因此需设通过当前外循环变量 x 和内循环变量 y 的值表示此时的阶段 i 和状态 j, 即设 $i=ii(x,y), j=jj(x,y)$.

步骤 IMP_4. 确定更新循环变量的值. 根据循环的定义和问题的要求, 更新循环变量的值, 以便进行下一次循环迭代或终止循环. 在双重循环中遍历二维最优决策表 m 的循环变量 x 和 y 都是加 1 操作进行更新.

步骤 IMP_5. 设计动态规划 IMP 代码.

```
x ::= a下, y ::= b下, init m;;
WHILE x ⩽ a上 − 1
DO
    (y ::= b下;;   //程序片段S1, 根据具体问题要求可进行调整和扩展
    WHILE y ⩽ b上 − 1
    DO
      ((IF Judge1(Ji(i), Jj(j)) THEN m[i, j] ::= opt_{(i',j')∈D(i,j)} t(m[i', j'], X(i, j));;
      ELSE(IF Judge2(Ji(i), Jj(j)) THEN m[i, j] ::= opt_{(i',j')∈D(i,j)} t(m[i', j'], X(i, j));;
          ELSE SKIP);;);;   //i = ii(x, y), j = jj(x, y)
      y ::= y + 1;;
      );;   //内循环语句S2
  x ::= x+1;;   //程序片段S3, 根据具体问题要求可进行调整和扩展
);;
```

2.4.3 C++代码生成

通过以上步骤得到的 IMP 代码仍然是抽象程序, 无法在计算机上直接编译运行. 需要借助 C++程序自动生成系统, 将 IMP 代码转换成 C++可执行代码, 并将其交付给第三方编译器进行编译, 从而保证了转换系统的可靠性, 实现了从问题描述到正确开发可执行程序的全过程.

该自动程序转换系统的总体结构如图 3 所示, 其主要模块如下.

1) IMP 代码输入: 用户通过文件输入或手工输入 IMP 代码, 作为问题描述的初始输入.

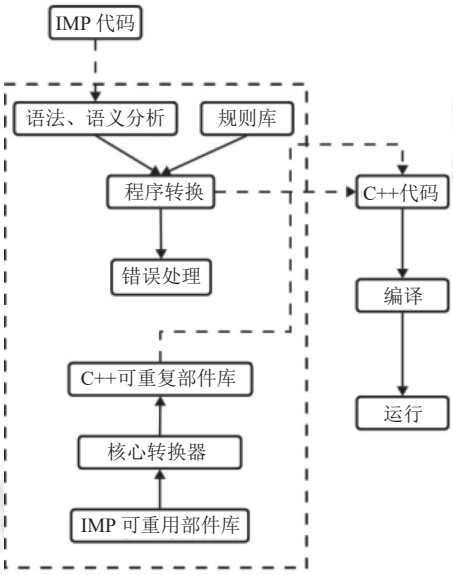

图 3 C++生成系统总体结构图

2) 语法和语义分析: 自动生成系统对 IMP 代码进行严格的语法和语义分析, 并提供详细的错误信息, 包括语法和语义错误等问题.

3) IMP 到 C++转换: 利用系统内部的 IMP 到 C++的转换规则, 将通过语法和语义分析的正确 IMP 代码转换为相应的 C++代码.

4) 第三方编译器调用: 自动生成系统调用第三方 C++编译器, 直接对生成的 C++代码进行编译; 编译后的可执行程序被运行.

通过这个系统, 本文在第 4.1.4 节给出了对 0-1 背包问题和最长公共子序列问题从 IMP 代码生成可在计算机上直接运行的 C++代码的过程. 保证了转换的可靠性和问题求解的有效性.

3 动态规划类算法的机械化验证

本节首先介绍文献 [10] 给出的 VCG 的基本原理、工作方式和优势, 进而基于第 2 节命令式动态规划类算法程序的设计, 得出了前后置断言、IMP 代码和循环不变式, 将其输入 VCG, 可自动生成程序正确性的验证条件. 然后通过 Isabelle/HOL 定理证明器进行机械验证, 从而保证命令式动态规划类算法的正确性.

3.1 验证条件自动生成器 (VCG)

VCG (verification condition generator)[10]主要是将 Hoare 逻辑的可证明性简化为断言的可证明性, 即给定一个欲证明的霍尔三元组{P} c {Q}, 从中计算一个断言 A, 使得{P} c {Q}是可证的当且仅当 A 是可证的.

VCG 的工作方式类似于上面概述的 Hoare 逻辑, 其实现基于两个函数[10]: *pre* 和 *vc*. *pre* 函数类似于 wp (weakest precondition), *vc* 函数的作用则是生成验证条件, 它模拟了 Hoare 逻辑规则的逆向应用, 并收集过程中出现的断言之间的蕴涵关系, 然后生成相应的验证条件. *pre* 和 *vc* 函数具体实现如表 3 所示, 其中, assn[10]表示一个程序状态映射到布尔值的函数: *type_synonym assn = state ⇒ bool*.

表 3 *pre* 和 *vc* 函数定义

*pre*函数	*vc*函数
fun pre :: *acom ⇒ assn ⇒ assn* where	***fun vc*** :: *acom ⇒ assn* ⇒ bool where
pre **SKIP** *Q* = *Q*	*vc* **SKIP** *Q* = True
pre (*x* ::= *a*) *Q* = (λ*s*. *Q* (*s*[*a*/*x*]))	*vc* (*x* ::= *a*) *Q* = True
pre (*C*1;; *C*2) *Q* = *pre C*1 (*pre C*2 *Q*)	*vc* (*C*1;; *C*2) *Q* = (*vc C*1 (*pre C*2 *Q*) ∧ *vc C*2 *Q*)
pre (**IF** *b* **THEN** *C*1 **ELSE** *C*2) Q	*vc* (**IF** *b* **THEN** *C*1 **ELSE** *C*2) *Q* = (*vc C*1 *Q* ∧ *vc C*2 *Q*)
= (λ*s*. **IF** *bval b s* **THEN** *pre C*1 *Q s* **ELSE** *pre C*2 *Q s*)	*vc* ({*I*} **WHILE** *b* DO *C*) *Q* =
pre ({*I*} **WHILE** *b* **DO** *C*) *Q* = *I*	((∀ *s*. (*I s* ∧ *bval b s* ⟶ *pre C I s*) ∧ (*I s* ∧ ¬ *bval b s* ⟶ *Q s*)) ∧ *vc C I*)

VCG 的优势是进行验证程序的开发者不需要了解任何有关 Hoare 的逻辑知识, 只需要提供 IMP 代码及其前后置断言和循环不变式, 即可生成程序正确性的验证条件. VCG 支持将 IMP 语言中的“::=”“;; (正式形式)”表示为“:=”“; (传统形式)”[10].

3.2 动态规划类 VCG 代码

运用 VCG 需要已知前后置条件、循环不变式以及 IMP 代码, 并将这三者作为输入. VCG 能够自动生成验证 IMP 代码正确性所需的验证条件[10], 将这三者的输入组合称为 VCG 代码[23]. 基于上述已有研究成果, 我们给出动态规划类算法的双重循环 VCG 代码结构.

首先, 给出在 Isabelle/HOL 中二维数组更新函数. 由于动态规划类算法的求解结果存放在二维最优决策表 *m* 中, 求解的过程需要更新最优决策表 *m*, 所以定义函数 *list_list_update* 对二维最优决策表 *m* 中的元素进行更新操作. *list_list_update* 的代码如下所示.

definition *list_list_update* :: “*nat list list ⇒ nat ⇒ nat ⇒ nat ⇒ nat list list*”
where “*list_list_update dlist i j value* = *list_update dlist i* (*list_update* (*dlist*!*i*) *j value*)”

然后, 给出动态规划类算法的双重循环 VCG 代码程序结构.

lemma < 算法名称 >: “VARS < 变量名 :: 类型 >
{$length\ m \geqslant 1$} //前置条件P
$x := a_{下}; y := b_{下}; init\ m; init\ \overline{\lambda}$; //初始化参数$S_init$, 函数式定义相关的参数$\overline{\lambda}$
WHILE $x \leqslant a_{上} - 1$ //进入外循环条件$Guard_e$
$INV\{INV\_DP\_E(a_{上}, a_{下}, b_{上}, b_{下}, G, \overline{\lambda})\}$ //外循环不变式ρ_e
DO
    $y := b_{下}$; //程序片段$S1$, 根据具体问题要求可进行调整和扩展
    WHILE $y \leqslant b_{上} - 1$ //进入内循环条件$Guard_i$
    $INV\{INV\_DP\_I(a_{上}, a_{下}, b_{上}, b_{下}, G, \overline{\lambda})\}$ //内循环不变式ρ_i
    DO
        IF $Judge_1(Ji(i), Jj(j))$ **THEN** $m := list\_list\_update\ m\ i\ j \underset{(i',j')\in D(i,j)}{\text{opt}}\ t(m!i'!j', X(i,j))$
        ELSE (**IF** $Judge_2(Ji(i), Jj(j))$ **THEN** $m := list\_list\_update\ m\ i\ j \underset{(i',j')\in D(i,j)}{\text{opt}}\ t(m!i'!j', X(i,j))$
            ELSE SKIP
            FI) //$i = ii(x,y), j = jj(x,y)$
        FI); //**IF**分支的多少由递推关系式确定
        $y := y + 1$
    OD //内循环语句$S2$
    $x := x + 1$ //程序片段$S3$, 根据具体问题要求可进行调整和扩展
OD
{$x = length\ l \wedge m!u!v = G\ u\ v\ \overline{\lambda}$}” //后置条件$Q$
< 证明过程 >

3.3 Isabelle/HOL 验证

在应用 VCG 验证动态规划类算法 IMP 程序的正确性时, 证明语句 apply vcg 将模拟 Hoare 逻辑规则的逆向应用, 并收集过程中出现的断言之间的蕴涵关系, 自动生成对应的验证条件. 然后, 这些验证条件将被传递给 Isabelle/HOL 定理证明器[24,25]. 根据第 3.2 节双重循环 VCG 代码, 可以归纳出 VCG 自动生成的单重循环 (SL) 和双重循环 (DL) 验证条件.

■ 如果是 SL, 则对应 3 个验证条件.

SL 生成对应的 3 个验证条件.

验证条件 SL_1. $P \Rightarrow wp(\text{‘}S\_init\text{’}, \rho)$
验证条件 SL_2. $\rho \wedge Guard \Rightarrow wp(\text{‘}S\text{’}, \rho)$
验证条件 SL_3. $\rho \wedge \neg Guard \Rightarrow Q$

■ 如果是 DL, 则对应 5 个验证条件.

DL 生成对应的 5 个验证条件

验证条件 DL_1. $P \Rightarrow wp(\text{‘}S\_init\text{’}, \rho_e)$
验证条件 DL_2. $\rho_e \wedge Guard_e \Rightarrow wp(\text{‘}S1\text{’}, \rho_i)$
验证条件 DL_3. $\rho_i \wedge Guard_i \Rightarrow wp(\text{‘}S2\text{’}, \rho_i)$
验证条件 DL_4. $\rho_i \wedge \neg Guard_i \Rightarrow wp(\text{‘}S3\text{’}, \rho_e)$
验证条件 DL_5. $\rho_e \wedge \neg Guard_e \Rightarrow Q$

4 典型实例: 0-1 背包问题 (KS) 和最长公共子序列问题 (LCS)

本节基于第 2 节命令式动态规划类算法程序设计和第 3 节机械化验证, 实例化了 KS 和 LCS 问题的命令式动

态规划算法的设计和验证, 从而完成了一般形式得到具体算法的推导. KS 和 LCS 问题的完整验证脚本可以参阅: https://github.com/hahahahaha-sh/IMP_DP/tree/IMP_DP_proof

4.1 算法设计

4.1.1 问题规约描述

■ KS 问题描述: 给定 n 种物品和一个背包, n 种物品的重量和价值分别使用一维数组 w 和 v 表示, $w=[w_1, w_2,\ldots, w_n]$, $v=[v_1, v_2,\ldots, v_n]$, 可得第 i 种物品的重量是 w_i, 价值是 v_i, 背包的容量为 C. 要从 n 种物品中选择若干物品放入背包中, 使得背包的重量不超过 C, 并且获得的价值最大. 在选择装物品时, 不能将同一种物品装入背包多次, 也不能将物品分割装入[1]. 引入二维最优决策表 $m_{ks}[i, j]$ 记录 i 种物品和所剩容量 j 能获得的最大价值.

■ LCS 问题描述: 给定两个序列 A, B, 分别使用一维数组 $A=[a_1, a_2,\ldots, a_m]$ 和 $B=[b_1, b_2,\ldots, b_n]$ 来表示. 求解在序列 A 和 B 的公共子序列中序列长度最长的公共子序列 Z, 最长公共子序列往往不止一个. 引入二维最优决策表 $m_{lcs}[i, j]$ 记录 $A_i=[a_1, a_2,\ldots, a_i]$ 和 $B_j=[b_1, b_2,\ldots, b_j]$ 的最长公共子序列的长度[26].

从第 2.1 节中描述的问题规约可以得知, 动态规划类算法的后置条件 Q 依赖于文献 [12] 中给出的动态规划类算法函数式定义. 因此, 在对 KS 和 LCS 问题进行后置断言的描述时, 需要运用相应的函数式定义. 求解 KS 问题最大价值函数式定义如下.

```
fun knapsack :: "nat ⇒ nat ⇒ nat list ⇒ nat list ⇒ nat"
WHERE "knapsack 0 W w v = 0" |
"knapsack (Suc i) W w v = (IF W < w!(Suc i) THEN knapsack i W w v
                           ELSE max (knapsack i W w v) (v!(Suc i) + knapsack i (W − w!(Suc i)) w v))"
```

求解 LCS 问题最长公共子序列长度的函数式定义如下.

```
fun lcs :: "nat ⇒ nat ⇒ char list ⇒ char list ⇒ nat"
WHERE "lcs 0 _ A B = 0" |
"lcs _ 0 A B = 0" |
"lcs (Suc i) (Suc j) A B = (IF A !(Suc i) = B ! (Suc j) THEN 1 + lcs i j A B
                            ELSE max (lcs i (j + 1) A B) (lcs (i + 1) j A B))"
```

根据第 2.1 节动态规划类算法程序规约的描述, 可以很快得出 KS 和 LCS 问题的程序规约, 遍历方式如前文图 2 所示, i 是遍历 m 的外循环变量, 即 x 为 i; 最终结束跳出循环时 i 的取值为 $length\ m$, 即 $l=m$; u 和 v 分别表示最后目标问题的阶段和状态, $u=length\ m-1$, $v = length\ (m[i-1])-1$. 如表 4 所示.

表 4 程序规约

动态规划算法	前置条件P	后置条件Q
KS问题	$length\ m_{ks} \geqslant 1$	$i = length\ m_{ks} \wedge m_{ks}!u!v = knapsack\ u\ v\ w\ v$
LCS问题	$length\ m_{lcs} \geqslant 1$	$i = length\ m_{lcs} \wedge m_{lcs}!u!v = lcs\ u\ v\ A\ B$

4.1.2 递推关系式归纳

根据第 2.2 节动态规划类算法递推关系式的归纳, 可得 KS 和 LCS 问题的递推关系式归纳步骤如下.

步骤 RC_1. 定义阶段和状态.

■ KS 问题: 以物品的种类 n 来划分阶段, 则阶段变量 i 表示的是可选择的物品种类, 状态变量 j 表示容量.

■ LCS 问题: 以序列 A 的字符来划分阶段, 则阶段变量 i 表示 A 的第 i 个字符, 状态变量 j 表示序列 B 的第 j 个字符.

步骤 RC_2. 确定决策.

■ KS 问题: 能够进入到判断前 i 种物品放进 j 容量的背包中的决策, 用阶段 i', 状态 j'表示.

■ LCS 问题: 能够进入到判断序列 A 的第 i 个字符 a_i 与序列 B 的第 j 个字符 b_j 是否相等的决策, 用阶段 i', 状态 j'表示.

步骤 RC_3. 给出允许决策集合.

■ KS 问题: 当前阶段和状态是把前 i 种物品放进 j 容量的背包中, 仅考虑第 i 件物品, 则把问题转换成 i–1 种物品的问题. ① 如果第 i 个物品放置, 则状态 j'的取值是 $j–w_i$; ② 如果第 i 个物品不放置, 则状态 j'的取值是 j. 两种情况对应的允许决策集合 $D(i,j)=\{(i-1,j),(i-1,j-w_i)\}$.

■ LCS 问题: 当前阶段和状态是判断序列 A 的第 i 个字符 a_i 与序列 B 的第 j 个字符 b_j 是否相等. ① 如果 a_i 与 b_j 相等, A_{i-1} 和 B_{j-1} 的最长公共子序列加上 $a_i(=b_j)$ 即为 A_i 和 B_j 的一个最长公共子序列, 所以阶段 i'的取值是 i–1, 状态 j'的取值是 j–1, 所以阶段 i'和状态 j'的取值在情况①中, 对应的允许决策集合 $D(i,j)=\{(i-1,j-1)\}$; ②如果 a_i 与 b_j 不相等, 必须解两个子问题, 即找出 A_i 和 B_{j-1} 的一个最长公共子序列以及 A_{i-1} 和 B_j 的一个最长公共子序列. 这两个公共子序列中较长者即为 A_i 和 B_j 的一个最长公共子序列, 所以阶段 i'和状态 j'的取值在情况②中, 对应的允许决策集合 $D(i,j)=\{(i,j-1),(i-1,j)\}$. 两种情况对应总的允许决策集合 $D(i,j)=\{(i-1,j-1),(i,j-1),(i-1,j)\}$.

步骤 RC_4. 定义函数.

■ KS 问题:

● 辅助函数 $X(i,j)$ 表示第 i 个物品装入时, 即 $i'=i–1, j'=j–w_i$, 求解 $m_{ks}[i–1,j–w_i]$ 与价值 v_i 之和.

● 在考虑第 i 件物品装入情况时, 分为可以装入和不可以装入; 在可以装入情况下, 依赖函数 t 需要进行装入的价值 $(m_{ks}[i–1,j–w_i]+v_i)$ 和未装入的价值 $(m_{ks}[i–1,j])$ 之间的比较. 在不可以装入情况下, 依赖函数 t 不需要进行判断, 取未装入的价值 $(m_{ks}[i–1,j])$.

● KS 问题求解的是存入背包最大的价值, 所以最优函数 opt 表示的是 max.

● $F[i,j]$ 表示求解最优决策表 $m_{ks}[i,j]$, $m_{ks}[i,j]$ 表示前 i 种物品放进背包容量为 j 取得的最优解.

● 对于 KS 问题分情况讨论递推关系式, 判断的是第 i 种物品重量是否能存入容量为 j 的背包, 因此关于 i 的函数 Ji 是取第 i 种物品的重量 w_i, 关于 j 的函数 Jj 是取 j 的值, 关于 i,j 的判断函数 $Judge$ 是判断 w_i 和 j 的大小.

■ LCS 问题:

● 辅助函数 $X(i,j)$ 表示 $a_i=b_j$ 时, 即 $i'=i–1, j'=j–1$, 求解 $m_{lcs}[i–1,j–1]$ 与 1 之和.

● 在考虑 a_i 与 b_j 是否相等的时候, 分为相等和不相等. 当 $a_i\neq b_j$, 依赖函数 t 需要进行 A_i 和 B_{j-1} 的最长公共子序列长度 $m_{lcs}[i,j–1]$ 与 A_{i-1} 和 B_j 的最长公共子序列长度 $m_{lcs}[i–1,j]$ 之间的比较; 当 $a_i=b_j$, 依赖函数 t 不需要进行判断, 取 $m_{lcs}[i–1,j–1]$ 与 1 之和.

● LCS 问题求解的是序列 A 和 B 的公共子序列中长度最长的公共子序列 Z, 所以最优函数 opt 表示的是 max.

● $F[i,j]$ 表示求解最优决策表 $m_{lcs}[i,j]$, $m_{lcs}[i,j]$ 表示序列 A_i 和 B_j 的公共子序列长度取得的最优解.

● 对于 LCS 问题分情况讨论递推关系式, 判断的是 A 字符串的第 i 个字符是否等于 B 字符串的第 j 个字符, 因此关于 i 的函数 Ji 是取 A 字符串的第 i 个字符 $A[i]$, 关于 j 的函数 Jj 是取 B 字符串的第 j 个字符 $B[j]$, 关于 i,j 的判断函数 $Judge$ 是判断 $A[i]$ 和 $B[j]$ 是否相等.

通过上述函数的定义, 能够更清晰地理解 KS 和 LCS 问题的求解过程和关键步骤. 如表 5 所示总结了函数在 KS 和 LCS 问题中的具体实例化.

表 5 定义函数实例化

定义函数	KS问题	LCS问题
辅助函数$X(i,j)$	$m_{ks}[i–1,j–w_i]+v_i$	$m_{lcs}[i–1,j–1]+1$
依赖函数t	$m_{ks}[i,j]$表示$m_{ks}[i–1,j–w_i]+v_i$和$m_{ks}[i–1,j]$之间的比较 or $m_{ks}[i,j]=m_{ks}[i–1,j]$	$m_{lcs}[i,j]=m_{lcs}[i–1,j–1]+1$ or $m_{lcs}[i,j]$表示$m_{lcs}[i,j–1]$和$m_{lcs}[i–1,j]$之间比较
最优函数opt	max	max
最优指标函数$F[i,j]$	$m_{ks}[i,j]$	$m_{lcs}[i,j]$
判断函数$Judge$	$j\geqslant w_i$ or $0\leqslant j<w_i$	$A[i]=B[j]$ or $A[i]\neq B[j]$

步骤 RC_5. 归纳 KS 和 LCS 问题的递推关系式. 依据第 2.2 节归纳的递推关系式 $Judge(Ji(i),Jj(j)) \to F(i,j) = \underset{(i',j')\in D(i,j)}{\text{opt}}\ t(F(i',j'),X(i,j))$ 和表 5 对函数的实例化, 可得 KS 和 LCS 问题的递推关系式如表 6 所示.

表 6 递推关系式

动态规划算法	递推关系式
KS问题	$m_{ks}[i,j]=\begin{cases}\max(m_{ks}[i-1,j-w_i]+v_i, m_{ks}[i-1,j]), & j\geqslant w[i]\\ m_{ks}[i-1,j], & 0\leqslant j<w[i]\end{cases}$
LCS问题	$m_{lcs}[i,j]=\begin{cases}m_{lcs}[i-1,j-1]+1, & A[i]=B[j]\\ \max(m_{lcs}[i,j-1], m_{lcs}[i-1,j]), & A[i]\neq B[j]\end{cases}$

表 6 中:

- $m_{ks}[i–1, j–w_i]$ 表示前 i–1 种物品装入容量为 $j–w_i$ 的背包所得价值.
- $m_{ks}[i–1, j]$ 表示前 i–1 种物品装入容量为 j 的背包所得价值.
- $\max(m_{ks}[i–1, j–w_i]+v_i, m_{ks}[i–1, j])$ 表示第 i 种物品两种不同的装入方式, 所取的价值也不同, 所以取两者中的最大值.
- $m_{lcs}[i–1, j–1]$、$m_{lcs}[i–1, j]$ 和 $m_{lcs}[i, j–1]$ 分别求解各自长度对应的 A 和 B 的最长公共子序列的长度.

4.1.3 循环不变式构造

根据第 2.3 节动态规划类算法循环不变式的构造和第 4.1.1 节给出的 KS 和 LCS 问题函数式定义, 可得 KS 和 LCS 问题的内外循环不变式, 如表 7、表 8 所示.

表 7 KS 问题内外循环不变式构造

循环不变式构造步骤	KS问题外循环不变式构造	KS问题内循环不变式构造
步骤INV_1. 给出循环变量范围	$i\geqslant 1\wedge i\leqslant length\ m$ $\wedge j\leqslant length(m!(i-1))\wedge j\geqslant 0$	$i\geqslant 1\wedge i<length\ m$ $\wedge j\leqslant length(m!(i-1))\wedge j\geqslant 0$
步骤INV_2. 确定求解问题变化规律	$(\forall k.\ k<i\to$ $(\forall j.\ j<length(m!(k-1))\to$ $m!k!j=knapsack\ k\ j\ w\ v))$	$(\forall k.\ k<i\to$ $(\forall j.\ j<length(m!(k-1))\to$ $m!k!j=knapsack\ k\ j\ w\ v))$ $\wedge(\forall k.\ k<j\to m!i!k=knapsack\ i\ k\ w\ v)$
步骤INV_3. 满足二维最优决策表性质	$length\ m\geqslant 1$ $\wedge(\forall k.\ k<length\ m\to length(m!k)$ $=length(m!(k-1))\wedge length(m!k)\geqslant 1)$	$length\ m\geqslant 1$ $\wedge(\forall k.\ k<length\ m\to length(m!k)$ $=length(m!(k-1))\wedge length(m!k)\geqslant 1)$
步骤INV_4. 构造动态规划类算法的循环不变式	$INV\{INV_1\wedge INV_2\wedge INV_3\}$	$INV\{INV_1\wedge INV_2\wedge INV_3\}$

表 8 LCS 问题内外循环不变式构造

循环不变式构造步骤	LCS问题外循环不变式构造	LCS问题内循环不变式构造
步骤INV_1. 给出循环变量范围	$i\geqslant 1\wedge i\leqslant length\ m$ $\wedge j\leqslant length(m!(i-1))\wedge j\geqslant 1$	$i\geqslant 1\wedge i<length\ m$ $\wedge j\leqslant length(m!(i-1))\wedge j\geqslant 1$
步骤INV_2. 确定求解问题变化规律	$(\forall k.\ k<i\to$ $(\forall j.\ j<length(m!(k-1))\to$ $m!k!j=lcs\ k\ j\ A\ B))$ $\wedge(\forall k.\ k<length\ m\to m!k!0=0)$	$(\forall k.\ k<i\to$ $(\forall j.\ j<length(m!(k-1))\to$ $m!k!j=lcs\ k\ j\ A\ B))$ $\wedge(\forall k.\ k<j\to m!i!k=lcs\ i\ k\ A\ B)$ $\wedge(\forall k.\ k<length\ m\to m!k!0=0)$
步骤INV_3. 满足二维最优决策表性质	$length\ m\geqslant 1$ $\wedge(\forall k.k<length\ m\to length(m!k)$ $=length(m!(k-1))\wedge length(m!k)\geqslant 1)$	$length\ m\geqslant 1$ $\wedge(\forall k.k<length\ m\to length(m!k)$ $=length(m!(k-1))\wedge length(m!k)\geqslant 1)$
步骤INV_4. 构造动态规划类算法的循环不变式	$INV\{INV_1\wedge INV_2\wedge INV_3\}$	$INV\{INV_1\wedge INV_2\wedge INV_3\}$

KS 问题的遍历方式如图 2 所示, 则阶段 i 即外循环变量, 状态 j 即内循环变量; 在步骤 INV_1 中给出循环变量范围时, 循环变量的下界确定依据的是第 4.2 节归纳的 KS 问题递推关系式, 要求递推关系式等式两边满足所要求解的阶段和状态大于等于 0, 可得 $i \geqslant 1$ 和 $j \geqslant 0$, 即 $a_{下}=1$, $b_{下}=0$; 循环变量的下界依据的是二维决策表 m 的大小, 即 $i \leqslant length\ m$; $j \leqslant length\ (m!(i-1))$, 即 $a_{上}=length\ m$, $b_{上}=length\ (m!(i-1))$; 在步骤 INV_2 中确定求解问题变化规律时, 需要确定 $G\ i\ j\ \overline{\lambda}$, 即对应 KS 问题的函数式定义 $knapsack\ i\ j\ w\ v$; 在步骤 INV_3 中只有满足初始化的背包容量和物品种类相等时才加入 $length\ (m!1)=length\ m$. 综上可得, KS 问题的外循环不变式和内循环不变式分别是 $INV\_DP\_E(length\ m, 1, length\ (m!(i-1)), 0, knapsack, w, v)$ 和 $INV\_DP\_I(length\ m, 1, length\ (m!(i-1)), 0, knapsack, w, v)$.

LCS 问题的遍历方式如图 2 所示, 则阶段 i 即外循环变量, 状态 j 即内循环变量; 在步骤 INV_1 中给出循环变量范围时, 循环变量的下界确定依据的是第 4.2 节归纳的 LCS 问题递推关系式, 要求递推关系式等式两边满足所要求解的阶段和状态大于等于 0, 可得 $i \geqslant 1$ 和 $j \geqslant 1$, 即 $a_{下}=1$, $b_{下}=1$; 循环变量的下界依据的是二维决策表 m 的大小, 即 $i \leqslant length\ m; j \leqslant length\ (m!(i-1))$, 即 $a_{上}=length\ m$, $b_{上}=length\ (m!(i-1))$. 在步骤 INV_2 中确定求解问题变化规律时, 需要确定 $G\ i\ j\ \overline{\lambda}$, 即对应 LCS 问题的函数式定义 $lcs\ i\ j\ A\ B$; 在步骤 INV_3 中满足两个序列 A、B 长度相等则加入 $length\ (m!1)=length\ m$. 综上可得, LCS 问题的外循环不变式和内循环不变式分别是 $INV\_DP\_E(length\ m, 1, length\ (m!(i-1)), 1, lcs, A, B)$ 和 $INV\_DP\_I(length\ m, 1, length\ (m!(i-1)), 1, lcs, A, B)$.

4.1.4 代码生成

根据第 2.4 节编写 IMP 代码的过程和第 4.1.2 节归纳的 KS 和 LCS 问题递推关系式, 可以得到 KS 和后文 LCS 问题相应的 IMP 代码, 并且通过 C++程序自动生成系统, 将 IMP 抽象程序自动生成对应的 C++代码. 如图 4 和后文图 5 分别是 KS 和 LCS 问题 IMP 代码转换成 C++代码, 其左边为 IMP 代码, 右边为转换的 C++代码. 对生成的 C++代码运行, 结果正确, 如后文图 6 和图 7 所示.

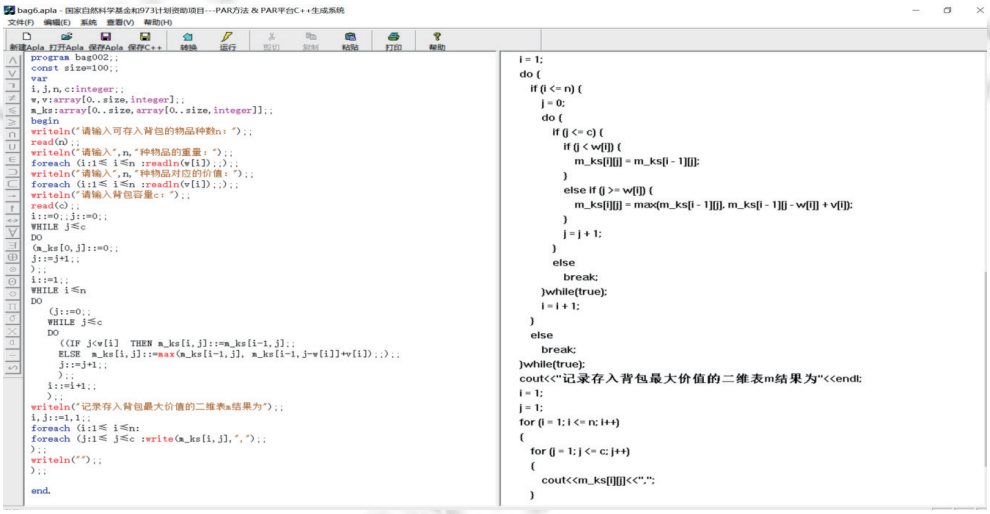

图 4 自动生成 KS 问题的 C++代码

4.2 算法验证

4.2.1 VCG 代码

本节通过以上动态规划类算法的问题规约、递推关系式、循环不变式, 在 Isabelle/HOL 中利用 VCG 对 KS 和 LCS 问题生成验证条件的关键代码, 分别对应算法 1 和算法 2.

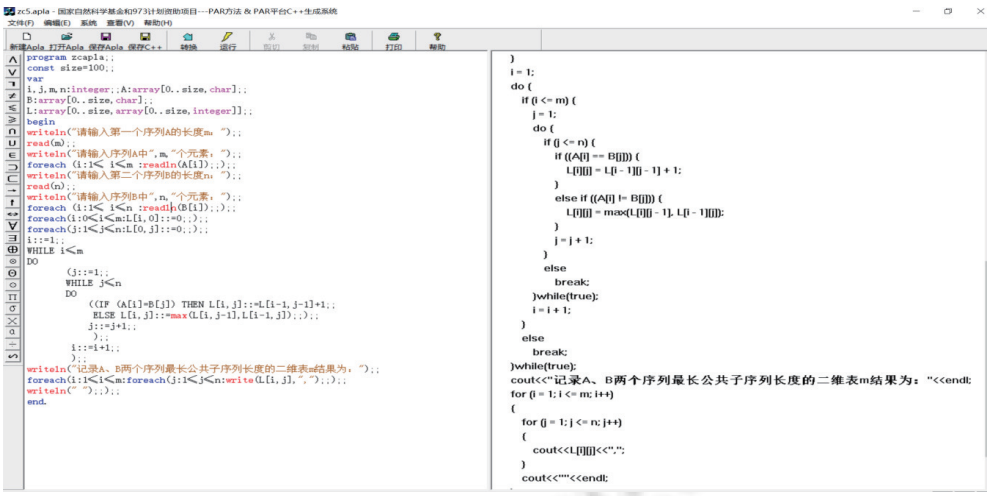

图 5 自动生成 LCS 问题的 C++代码

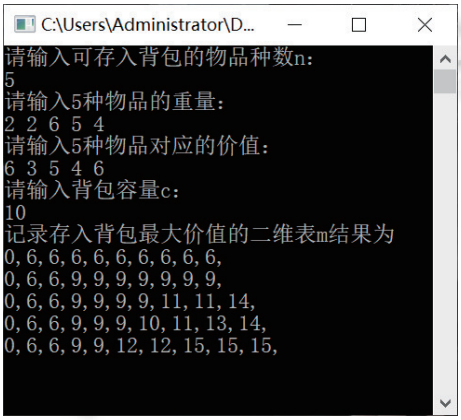

图 6 KS 问题的 C++代码运行结果

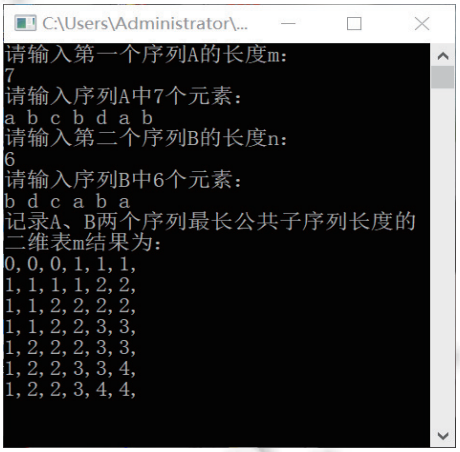

图 7 LCS 问题的 C++代码运行结果

算法 1. 0-1 背包问题 KS.

1. **lemma** *ks*: "*VARS m w v i j*
2. {*length m* ⩾1} //前置条件
3. *i*:=1; *j*:=0; *m*:=[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
4. [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
5. [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]];
6. *v*:= [(0::*nat*), 6, 3, 5, 4, 6]; *w*:= [(0::*nat*), 2, 2, 6, 5, 4]; //*i*, *j* 分别是对二维最优决策表 *m* 的行列进行遍历, *v* 表示物品价值的列表, *w* 表示物品重量的列表
7. **WHILE** *i* ⩽ *length m* – 1
8. *INV*{*INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*)} //外循环不变式
9. **DO**
10. *j*:=0;
11. **WHILE** *j* ⩽ *length* (*m*!*i*)–1

```
12.      INV{INV_DP_I(length m, 1, length (m!(i–1)), 0, knapsack, w, v)} //内循环不变式
13.      DO
14.           ( IF j<(w!(i)) THEN m:=list_list_update m (i) j (m!(i–1)!j)
15.           ELSE m:=list_list_update m (i) j (max (m!(i–1)!j) ((m!(i–1)!(j–w!(i)))+v!(i)))FI);
16.         j:=j+1
17.      OD;
18.      i:=i+1
19. OD
20. {(m!(i–1)!((length (m!(i–1)))–1)) =knapsack (i–1) ((length (m!(i–1)))–1) w v ∧ i=length m}"//后置条件
```

算法 2. 最长公共子序列问题 LCS.

```
1. lemma lcs: "VARS m A B i j
2. {length m ⩾1}//前置条件
3. j:=1; i:=1; m:=[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
4.          [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
5.          [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
6.          [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]; //i, j 分别是对二维最优决策表 m 的行列进行遍历
7. A:=[CHR "#", CHR "a", CHR "b", CHR "c", CHR "b", CHR "d", CHR "a", CHR "b"];
8. B:=[CHR "#", CHR "b", CHR "d", CHR "c", CHR "a", CHR "b", CHR "a"]; //A 和 B 表示字符序列
9. WHILE i ⩽ length m – 1
10. INV{INV_DP_E(length m, 1, length (m!(i–1)), 1, lcs, A, B)}//外循环不变式
11. DO
12.      j:=1;
13.      WHILE j ⩽ length (m!i)–1
14.      INV{INV_DP_I(length m, 1, length (m!(i–1)), 1, lcs, A, B)} //内循环不变式
15.      DO
16.           IF (A!(i))=(B!j) THEN m:=list_list_update m (i) j (m!(i–1)!(j–1)+1)
17.           ELSE m:=list_list_update m (i) j (max (m!(i)!(j–1)) (m!(i–1)!j)) FI;
18.           j:=j+1
19.      OD;
20.      i:=i+1
21. OD
22. { (m!(i–1)!((length (m!(i–1)))–1)) =lcs (i–1) ((length (m!(i–1)))–1) A B ∧ i=length m }"//后置条件
```

4.2.2 Isabelle/HOL 验证

根据第 3.3 节可知 KS 和 LCS 问题使用 apply vcg 分别自动生成 5 个验证条件, 通过 Isabelle/HOL 定理证明器验证其正确性.

4.2.2.1 KS 问题的命令式动态规划算法验证

对于 KS 问题的命令式动态规划算法验证, 需要完成 5 个验证条件的验证, 才能完成第 4.2.1 节中 KS 定理的验证. 如图 8 给出了证明定理 KS 相关的辅助引理以及它们之间的依赖关系.

由图 8 可知, 定理 KS 的证明主要由验证条件 1–5 来完成, 其定义如下.

验证条件 1. lemma ks_DL1: "*length m* ⩾1

⇒ *wp*('*i*:=1; *j*:=0; *m*:=[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]];
v:=[(0::*nat*), 6, 3, 5, 4, 6];[(0::*nat*), 2, 2, 6, 5, 4]', *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*))"

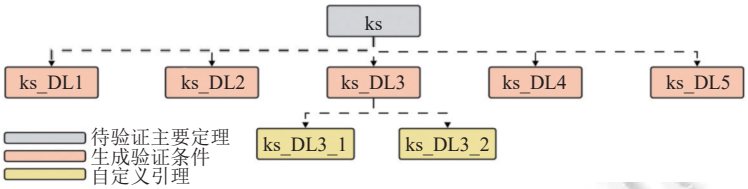

图 8 验证定理 ks 相关的辅助引理以及它们之间的依赖关系

验证条件 1 表示程序开始满足前置条件 P, 在执行初始化程序后得到的结果满足外循环不变式 *INV_DP_E* (*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*) 为真.

验证条件 2. *lemma* ks_DL2: "*INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*)∧*i* ⩽ *length m*–1

⇒ *wp*('*j*:=0', *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*))"

验证条件 2 表示外循环不变式 *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*) 为真时, 满足进入外循环的条件 *i* ⩽ *length m*–1, 进入外循环执行 *j*:=0 语句 (即外循环进入内循环之前的语句) 之后满足内循环不变式.

验证条件 3. lemma ks_DL3: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*) ∧ *j* ⩽ *length* (*m*!*i*)–1

⇒ *wp*('(IF *j*<(*w*!(*i*)) THEN *m*:=*list_list_update m* (*i*) *j* (*m*!(*i*–1)!*j*)

ELSE *m*:=*list_list_update m* (*i*) *j* (max (*m*!(*i*–1)!*j*) ((*m*!(*i*–1)!(*j*–*w*!(*i*)))+*v*!(*i*)))FI);

j:=*j*+1', *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*))"

验证条件 3 表示满足进入内循环的条件 *j* ⩽ *length* (*m*!*i*)–1, 进入内循环运行循环体之后依旧满足内循环不变式. 由于验证条件 3 的结论部分根据 *j* 与 *w*!*i* 的大小关系主要分成两部分, 需要对这两部分分别进行证明所求解的结果与 *knapsack* 函数求解的结果相等, 从而构造辅助引理 ks_DL3_1 和 ks_DL3_2, 其定义如下.

引理 1. lemma ks_DL3_1: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*) ∧ *j* ⩽ *length* (*m*!*i*)–1

⇒ *j* < *w*!*i*

⇒ (*m*!*i*)[*j*:=*m*!(*i*–*Suc* 0)!*j*]!*j* = *knapsack i j w v*"

引理 1 表示在 *j* < *w* ! *i* 的情况下, 二维最优决策表 *m* 中 *m*[*i*, *j*] 更新为 *m*[*i*–1, *j*], 证明其值是否等于函数 *knapsack* 对应位置所求解的值.

引理 2. lemma ks_DL3_2: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*) ∧ *j* ⩽ *length* (*m*!*i*)–1

⇒ ¬ *j* < *w* ! *i*

⇒ (*m* ! *i*)[*j* := max (*m* ! (*i* – *Suc* 0) ! *j*) (*m* ! (*i* – *Suc* 0) ! (*j*–*w*!*i*) + *v*!*i*)]!*j* = *knapsack i j w v*"

引理 2 表示在 ¬ *j* < *w* ! *i* 的情况下, 二维最优决策表 *m* 中 *m*[*i*, *j*] 更新为 *max* (*m* [*i*–1, *j*], *m* [*i*–1, *j*–*w*[*i*]]+*v*[*i*]), 证明其值是否等于函数 *knapsack* 对应位置所求解的值.

验证条件 4. lemma ks_DL4: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*)

∧ ¬ *j* ⩽ *length* (*m*!*i*)–1

⇒ *wp*('*i*:=*i*+1', *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*))"

验证条件 4 表示内循环执行结束¬ *j* ⩽ *length* (*m* ! *i*)–1, 内循环不变式 *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*) 为真证明执行 *i*:=*i*+1 语句 (即内循环结束转入外循环之间的语句) 之后外循环不变式依旧为真.

验证条件 5. lemma ks_DL5: "*INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0, *knapsack*, *w*, *v*)∧¬ *i* ⩽ *length m*–1

⇒ (*m*!(*i*–1)!((*length* (*m*!(*i*–1)))–1)) =*knapsack* (*i*–1) ((*length* (*m*!(*i*–1)))–1) *w v* ∧*i*=*length m*"

验证条件 5 表示外循环执行结束¬ *i* ⩽ *length m*–1, 外循环不变式 *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 0,

knapsack, *w*, *v*) 为真证明后置断言 Q 为真.

上述验证条件 1–5 的验证除了基于引理 1 和引理 2, 还依赖于 Sledgehammer 工具可调用另外的定理证明器寻找定理来辅助证明. 验证条件 1–5 在 Isabelle/HOL 中被机械证明之后, 定理 KS 可被验证. 如图 9 给出了验证条件成功验证后得到的 KS 定理.

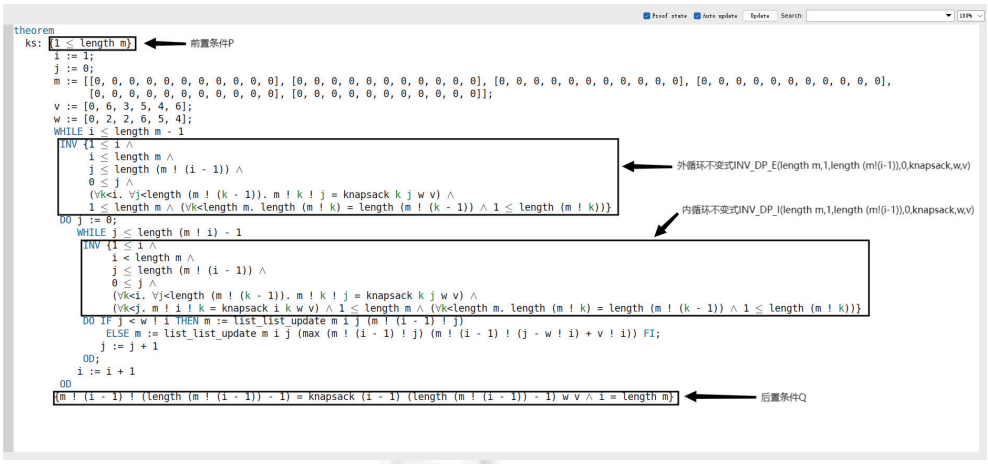

图 9 KS 定理

4.2.2.2 LCS 问题的命令式动态规划算法验证

对于 LCS 问题的命令式动态规划算法验证, 需要完成 5 个验证条件的验证, 才能完成第 4.2.1 节中 LCS 定理的验证. 如图 10 给出了证明定理 LCS 相关的辅助引理以及它们之间的依赖关系.

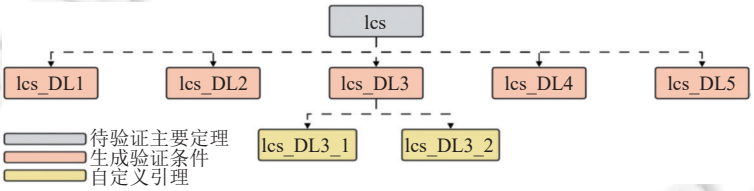

图 10 验证定理 LCS 相关的辅助引理以及它们之间的依赖关系

由图 10 可知, 定理 LCS 的证明主要由验证条件 6–10 来完成, 其定义如下.

验证条件 6. lemma lcs_DL1: "*length m* ⩾1

⇒ *wp*('*i*:=1; *j*:=1; *m*:=[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]];

A:=[CHR "#", CHR "a", CHR "b", CHR "c", CHR "b", CHR "d", CHR "a", CHR "b"];

B:=[CHR "#", CHR "b", CHR "d", CHR "c", CHR "a", CHR "b", CHR "a"]',

INV_DP_E(*length m*, 1, *length* (*m*!(*i*–1)), 0, *lcs*, *A*, *B*))"

验证条件 6 表示程序开始满足前置条件 P, 在执行程序后得到的结果满足外循环不变式 *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) 为真.

验证条件 7. lemma lcs_DL2: "*INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*)∧*i* ⩽ *length m*–1

⇒ *wp*('*j*:=1', *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*))"

验证条件 7 表示外循环不变式 *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) 为真时, 满足进入外循环的条件 *i* ⩽ *length m*–1, 进入外循环执行 *j*:=1 语句 (即外循环进入内循环之前的语句) 之后满足内循环不变式 *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*).

验证条件 8. lemma lcs_DL3: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) ∧ *j* ≤ *length* (*m*!*i*)–1

⇒ *wp*('IF (*A*!(*i*))=(*B*!*j*) THEN *m*:=*list_list_update m* (*i*) *j* (*m*!(*i*–1)!(*j*–1)+1)

ELSE *m*:=*list_list_update m* (*i*) *j* (max (*m*!(*i*)!(*j*–1)) (*m*!(*i*–1)!*j*)) FI;

j:=*j*+1', *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*))"

验证条件 8 表示满足进入内循环的条件 *j* ≤ *length* (*m*!*i*)–1, 进入内循环运行循环体之后依旧满足内循环不变式. 由于验证条件 8 的结论部分根据 (*A*!(*i*)) 和 (*B*!*j*) 的大小关系主要分成两部分, 需要对这两部分分别进行证明所求解的结果与 LCS 函数求解的结果相等, 从而构造辅助引理 lcs_DL3_1 和 lcs_DL3_2, 其定义如下.

引理 3. lemma lcs_DL3_1: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) ∧ *j* ≤ *length* (*m*!*i*)–1

⇒ *A* ! *i* = *B* ! *j*

⇒ (*m* ! *i*)[*j* := *Suc* (*m* ! (*i* – *Suc* 0) ! (*j* – *Suc* 0))] ! *j* = *lcs i j A B*"

引理 3 表示在 *A* ! *i* = *B* ! *j* 的情况下, 二维最优决策表 *m* 中 *m*[*i*, *j*] 更新为 *m* [*i*–1, *j*–1]+1, 证明其值是否等于函数 LCS 对应位置所求解的值.

引理 4. lemma lcs_DL3_2: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) ∧ *j* ≤ *length* (*m*!*i*)–1

⇒ *A* ! *i* ≠ *B* ! *j*

⇒ (*m* ! *i*)[*j* := max (*lcs i* (*j*–*Suc* 0) *A B*) (*m* ! (*i*–*Suc* 0) ! *j*)] ! *j* = *lcs i j A B*"

引理 4 表示在 *A* ! *i* ≠ *B* ! *j* 的情况下, 二维最优决策表 *m* 中 *m*[*i*, *j*] 更新的值是否等于函数 *lcs* 对应位置所求解的值. 由前提条件可知 *m* [*i*, *j*–1]=*lcs i* (*j*–*Suc* 0) *A B*, 所以 *m*[*i*, *j*] 更新为 max (*lcs i* (*j*–*Suc* 0) *A B* , *m* [*i*–1, *j*]), 从而判断是否等于函数 LCS 对应位置所求解的值. 使用上述引理 3、引理 4 对进行验证, 最终验证成功.

验证条件 9. lemma lcs_DL4: "*INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) ∧ ¬ *j* ≤ *length* (*m* ! *i*)–1

⇒ *wp*('*i*:=*i*+1', *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*))"

验证条件 9 表示内循环执行结束¬ *j* ≤ *length* (*m* ! *i*)–1, 内循环不变式 *INV_DP_I*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) 为真证明执行 *i*:=*i*+1 语句 (即内循环结束转入外循环之间的语句) 之后外循环不变式依旧为真.

验证条件 10. lemma lcs_DL5: "*INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*)∧¬ *i* ≤ *length m*–1

⇒ (*m*!(*i*–1)!((*length* (*m*!(*i*–1)))–1)) =*lcs* (*i*–1) ((*length* (*m*!(*i*–1)))–1) *A B* ∧*i*=*length m*"

验证条件 10 表示外循环执行结束¬ *i* ≤ *length m* – 1, 外循环不变式 *INV_DP_E*(*length m*, 1, *length* (*m*!(*i*–1)), 1, *lcs*, *A*, *B*) 为真证明后置断言 Q 为真.

上述验证条件 6–10 的验证除了基于引理 3 和引理 4, 还依赖于 Sledgehammer 工具可调用另外的定理证明器寻找定理来辅助证明. 验证条件 6–10 在 Isabelle/HOL 中被机械证明之后, 定理 LCS 可被验证. 如图 11 给出了验证条件成功验证后得到的 LCS 定理.

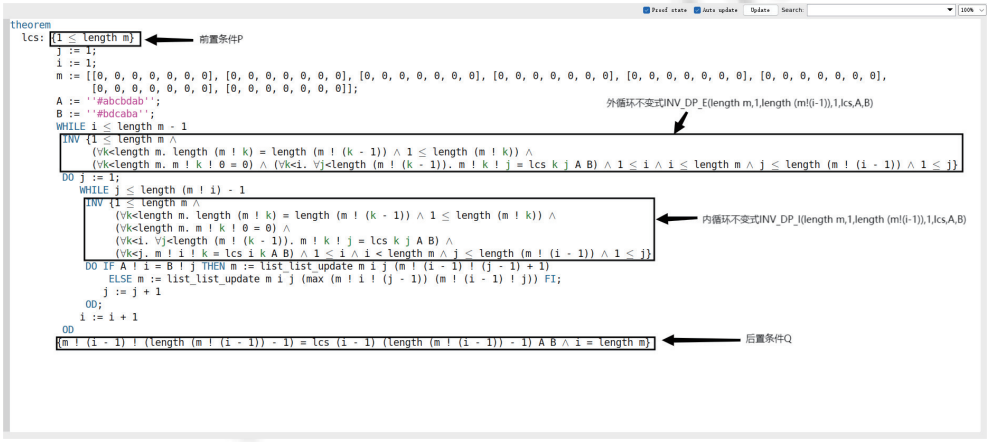

图 11 LCS 定理

4.3 框架的适用范围

本文依据提出的命令式动态规划类算法程序设计框架及机械化验证完成了 KS 和 LCS 问题的验证, 该框架基于 Wimmer 等人所实现的动态规划类算法的函数式建模[12], 通过证明命令式动态规划类算法程序与其等价性, 完成了命令式动态规划类算法的验证. 因此, 本文的框架主要适用于以下情况.

1) 嵌套循环结构. 大多数动态规划问题可以通过双重循环结构有效求解, 除了前文解决的 0-1 背包问题、最长公共子序列问题两个实例之外, 本文所提框架可用于设计与验证斐波那契数列、最短路径问题和最小编辑距离等其他算法. 例如最小编辑距离算法递推计算出两个字符串 A、B 间的最小编辑距离, 并且使用二维最优决策表 m_{ed} 记录, 基于本文所提框架可得其递推关系式是:

$$m_{ed}[i,j]=\begin{cases} m_{ed}[i-1,j-1] & A[i]=B[j] \\ \max(m_{ed}[i,j-1]+1, m_{ed}[i-1,j]+1, m_{ed}[i-1,j-1]+1) & A[i]\neq B[j] \end{cases}.$$

此外, 可拓展框架以应对多重循环结构 (矩阵连乘、最大子段和等) 问题, 将问题的状态空间扩展到了 3 个或更多维度, 例如除了本文第 2.1 节所提 i 和 j 两个维度, 还可能有步长等更多维度.

2) 非线性数据结构. 该框架除了能够处理数组结构问题以外, 在处理非线性数据结构问题时也具备广泛的适用性, 例如涉及到图结构或树结构时, 通常可以将问题的求解映射到备忘录, 进而将其转化为数组结构的问题. 该类问题包括邮局问题、最优二叉搜索树问题等.

例如证明树结构中最优二叉搜索树算法正确性, 该算法能将非线性数据结构 (二叉搜索树) 映射为备忘录 (二维表 m_{bst}), 限于篇幅, 详情可参阅 https://github.com/hahahahaha-sh/IMP_DP/tree/IMP_DP_proof/opt_bst. 以下是根据设计框架给出的前后置断言、递推关系式和循环不变式.

(a) 前后置断言

■ 前置条件 P: $length\ m_{\text{bst}} \geqslant 1$

■ 后置条件 Q: $r = length(m_{\text{bst}}[1]) \wedge m_{\text{bst}}!u!v = min\_wpl\ u\ v\ W$

(b) 递推关系式

依据第 2.2 节动态规划类算法递推关系式的归纳, 可将最优二叉搜索树的定义函数实例化如表 9 所示, 进一步可得递推关系式如表 10 所示.

表 9 定义函数实例化

定义函数	最优二叉搜索树
辅助函数$X(i,j)$	$m_{\text{bst}}[i,k-1]+m_{\text{bst}}[k+1,j]+W[i,j]$ $(i\leqslant k\leqslant j)$
依赖函数t	$m_{\text{bst}}[i,j]=W[i,j]$ or $m_{\text{bst}}[i,j]$表示k所有可能的情况下, $m_{\text{bst}}[i,k-1]+m_{\text{bst}}[k+1,j]+W[i,j]$ 之间的比较
最优函数opt	min
最优指标函数$F[i,j]$	$m_{\text{bst}}[i,j]$
判断函数$Judge$	$i=j$ or $i<j$

表 10 递推关系式

动态规划算法	递推关系式
最优二叉搜索树	$m_{\text{bst}}[i,j]=\begin{cases} W[i,j], & i=j \\ \min_{(i\leqslant k\leqslant j)}\{m_{\text{bst}}[i,k-1]+m_{\text{bst}}[k+1,j]+W[i,j]\}, & i<j \end{cases}$

(c) 循环不变式

根据第 2.3 节动态规划类算法循环不变式的构造, 可得最优二叉搜索树的内外循环不变式, 如表 11 所示.

表 11 最优二叉搜索树内外循环不变式构造

循环不变式构造步骤	外循环不变式构造	内循环不变式构造
步骤INV_1. 给出循环变量范围	$r \leqslant length\ m!1 \wedge r \geqslant 1$ $\wedge i \geqslant 0 \wedge i \leqslant length\ m - r + 1$	$r < length\ m!1 \wedge r \geqslant 1$ $i \geqslant 0 \wedge i \leqslant length\ m - r + 1$
步骤INV_2. 确定求解问题变化规律	$(\forall k.\ k < r \rightarrow$ $(\forall i.\ i < length\ m - r + 1 \rightarrow$ $m!i!(i + k - 1) = min\_wpl\ i\ (i + k - 1)\ W))$	$(\forall k.\ k < r \rightarrow$ $(\forall i.\ i < length\ m - r + 1 \rightarrow$ $m!i!(i + k - 1) = min\_wpl\ i\ (i + k - 1)\ W))$ $\wedge(\forall k.\ k < i \rightarrow$ $m!k!(k + r - 1) = min\_wpl\ k\ (k + r - 1)\ W)$
步骤INV_3. 满足二维最优决策表性质	$length\ m \geqslant 1$ $\wedge(\forall k.k < length\ m \rightarrow length(m!k)$ $= length(m!(k-1)) \wedge length(m!k) \geqslant 1)$ $\wedge length(m!1) = length\ m$	$length\ m \geqslant 1$ $\wedge(\forall k.k < length\ m \rightarrow length(m!k)$ $= length(m!(k-1)) \wedge length(m!k) \geqslant 1)$ $\wedge length(m!1) = length\ m$
步骤INV_4. 构造动态规划类算法的循环不变式	$INV\{INV_1 \wedge INV_2 \wedge INV_3\}$	$INV\{INV_1 \wedge INV_2 \wedge INV_3\}$

5 总结和展望

高效的算法在实际应用中起到重要的作用, 动态规划算法的核心思想是将复杂问题分解为较小的子问题, 并利用子问题的解构建整体解, 由于子问题之间存在大量依赖关系和约束条件, 导致验证过程复杂, 对于验证命令式动态规划类算法程序正确性是一个难点.

本文基于 Wimmer 等人在 Isabelle/HOL 中已证明的动态规划算法函数式建模, 提出命令式动态规划类算法程序设计及验证的框架, 该框架通过证明命令式动态规划类算法与函数式代码等价, 完成命令式动态规划类算法程序的推导与机械验证, 避免考虑依赖关系和约束条件. 本文主要完成以下工作.

(1) 基于 Wimmer 等人实现的动态规划类算法函数式建模与验证, 提出命令式动态规划类算法程序设计框架. 该设计框架包含依据第 2.2 节归纳的动态规划类算法递推关系式生成第 2.4 节 IMP 代码以及基于 IMP 代码程序与函数式算法程序的等价性来描述第 2.1 节动态规划类算法问题规约和构造第 2.3 节循环不变式, 其中问题规约的后置条件和循环不变式分别表示运行结束和过程中的解与对应函数式解相等, 避免了在证明正确性时处理复杂的依赖关系和约束条件, 并且解决了命令式代码验证困难的问题.

(2) 提出命令式动态规划类算法程序机械化验证方法. 该验证方法基于工作 (1) 获得的问题规约、循环不变式和 IMP 代码输入 VCG, 自动生成正确性的验证条件, 并采用 Isabelle/HOL 定理证明器对其进行验证, 实现了命令式动态规划类算法程序验证的自动性和机械性, 从而有效地保证了算法程序的正确性和可靠性.

(3) 基于工作 (1)(2) 命令式动态规划类算法设计框架和机械化验证, 实例化了 0-1 背包问题和最长公共子序列问题的命令式动态规划算法的设计和验证, 并且对于嵌套循环结构 (斐波那契数列、最短路径问题和最小编辑距离等) 和非线性数据结构问题 (图结构或树结构) 分析推导框架也同样适用, 完成了从一般形式进而得到具体算法的推导, 从而确保了本文所提框架能够准确地验证命令式动态规划类算法程序.

本文对命令式动态规划类算法程序进行了设计和验证. 在验证部分, 验证条件能够自动生成且可进行机械化验证, 然而在设计部分仍然涉及较多手工推导. 在未来的研究工作中, 计划基于本文提出的设计框架, 进一步提高动态规划算法设计的自动化程度. 同时, 也考虑将本文提出的框架应用到其他算法中, 为它们的设计和验证提供有益的参考.

References:

[1] Bellman R. Dynamic programming. Science, 1966, 153(3731): 34–37. [doi: 10.1126/science.153.3731.34]

[2] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33–61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[3] Shi HH, Jie AQ, Xue JY. New algorithm of 0-1 knapsack problem. Computer Engineering, 2008, 34(17): 37–38, 49 (in Chinese with

English abstract). [doi: 10.3969/j.issn.1000-3428.2008.17.014]

[4] Wang CJ, Xue JY. Formal derivation of a kind of 0-1 knapsack problems algorithmic programs. Journal of Wuhan University (Natural Science Edition), 2009, 55(6): 674–680 (in Chinese with English abstract).

[5] You Y. The application research of algorithm formal method in three kinds of combinatorial mathematical problems [MS. Thesis]. Nanchang: Jiangxi Normal University, 2017 (in Chinese with English abstract).

[6] Dasgupta S, Papadimitriou CH, Vazirani UV. Algorithms. McGraw-Hill Education, 2006.

[7] Lew A, Mauch H. Dynamic Programming: A Computational Tool. Berlin: Springer, 2006. [doi: 10.1007/978-3-540-37014-7]

[8] Mofarah-Fathi L, Norvell TS. Formal derivation of dynamic programming algorithms. 2007. https://www.engr.mun.ca/~theo/Publications/submitted -NECEC2007.pdf

[9] Bortin M. A formalisation of the Cocke-Younger-Kasami algorithm. Archive of Formal Proofs, 2016. https://isa-afp.org/entries/CYK.html

[10] Nipkow T, Klein G. Concrete Semantics: With Isabelle/HOL. Springer, 2014. [doi: 10.1007/978-3-319-10542-0]

[11] Wimmer S, Hu SW, Nipkow T. Verified memoization and dynamic programming. In: Proc. of the 9th Int'l Conf. on Interactive Theorem Proving. Oxford: Springer, 2018. 579-596. [doi: 10.1007/978-3-319-94821-8_34]

[12] Wimmer S, Hu SW, Nipkow T. Monadification, memoization and dynamic programming. Archive of Formal Proofs, 2018. https://isa-afp.org/entries/Monad_Memo_DP.html

[13] Si XJ, Dai HJ, Raghothaman M, Naik M, Song L. Learning loop invariants for program verification. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 7762–7773.

[14] Wu T. Application of dynamic programming algorithm and its optimization on time efficiency [MS. Thesis]. Nanjing: Nanjing University of Science and Technology, 2008 (in Chinese with English abstract).

[15] Itzhaky S, Singh R, Solar-Lezama A, Yessenov K, Lu YQ, Leiserson C, Chowdhury R. Deriving divide-and-conquer dynamic programming algorithms using solver-aided transformations. ACM SIGPLAN Notices, 2016, 51(10): 145–164. [doi: 10.1145/3022671.2983993]

[16] Javanmard MM, Ahmad Z, Zola J, Pouchet LN, Chowdhury R, Harrison R. Efficient execution of dynamic programming algorithms on apache spark. In: Proc. of the 2020 IEEE Int'l Conf. on Cluster Computing (CLUSTER). Kobe: IEEE, 2020. 337–348. [doi: 10.1109/CLUSTER49012.2020.00044]

[17] Erwig M, Kumar P. Explainable dynamic programming. Journal of Functional Programming, 2021, 31: e10. [doi: 10.1017/S0956796821000083]

[18] Eddy SR. What is dynamic programming? Nature Biotechnology, 2004, 22(7): 909–910. [doi: 10.1038/nbt0704-909]

[19] Luus R. Iterative Dynamic Programming. Boca Raton: CRC Press, 2000.

[20] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1969, 12(10): 576–580. [doi: 10.1145/363235.363259]

[21] Ryan G, Wong J, Yao J, Gu RH, Jana S. CLN2INV: Learning loop invariants with continuous logic networks. arXiv:1909.11542, 2019.

[22] Zuo ZK, Huang ZP, Huang Q, Wang Y, Wang CJ. Research on new strategy of program refinement and automatic verification method. Journal of Zhengzhou University (Natural Science Edition), 2022, 54(5): 1–7 (in Chinese with English abstract). [doi: 10.13705/j.issn.1671-6841.2021338]

[23] Zuo ZK, Hu Y, Huang Q, Wang Y, Wang CJ. Automatic algorithm programming model based on the improved Morgan's refinement calculus. Wuhan University Journal of Natural Sciences, 2022, 27(5): 405–414. [doi: 10.1051/wujns/2022275405]

[24] Paulson LC, Nipkow T, Wenzel M. From LCF to Isabelle/HOL. Formal Aspects of Computing, 2019, 31(6): 675–698. [doi: 10.1007/s00165-019-00492-1]

[25] Jiang N, Li QA, Wang LM, Zhang XT, He YX. Overview on mechanized theorem proving. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 82–112 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5870.htm [doi: 10.13328/j.cnki.jos.005870]

[26] Ma BL, Zhang Z, Liu JX. Similarity calculation method applied to dynamic heterogeneous Web server system. Computer Engineering and Design, 2018, 39(1): 282–287 (in Chinese with English abstract). [doi: 10.16208/j.issn1000-7024.2018.01.049]

附中文参考文献:

[2] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33–61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[3] 石海鹤, 揭安全, 薛锦云. 0-1 背包问题的一种新解法. 计算机工程, 2008, 34(17): 37–38, 49. [doi: 10.3969/j.issn.1000-3428.

2008.17.014]
[4] 王昌晶, 薛锦云. 一类 0-1 背包问题算法程序的形式化推导. 武汉大学学报 (理学版), 2009, 55(6): 674–680.
[5] 游颖. 算法形式化方法在三类组合数学问题求解中的应用研究 [硕士学位论文]. 南昌: 江西师范大学, 2017.
[14] 吴涛. 动态规划算法应用及其在时间效率上的优化 [硕士学位论文]. 南京: 南京理工大学, 2008.
[22] 左正康, 黄志鹏, 黄箐, 王渊, 王昌晶. 程序求精新策略及自动验证方法研究. 郑州大学学报 (理学版), 2022, 54(5): 1–7. [doi: 10.13705/j.issn.1671-6841.2021338]
[25] 江南, 李清安, 汪吕蒙, 张晓瞳, 何炎祥. 机械化定理证明研究综述. 软件学报, 2020, 31(1): 82–112. http://www.jos.org.cn/1000-9825/5870.htm [doi: 10.13328/j.cnki.jos.005870]
[26] 马博林, 张铮, 刘健雄. 应用于动态异构 Web 服务器的相似度求解方法. 计算机工程与设计, 2018, 39(1): 282–287. [doi: 10.16208/j.issn1000-7024.2018.01.049]

左正康(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为形式化方法, 智能化软件.

游珍(1982－), 女, 博士, 副教授, CCF 高级会员, 主要研究领域为形式化方法, 分布式虚拟现实, 并发分布式计算.

孙欢(1997－), 女, 硕士生, CCF 学生会员, 主要研究领域为定理证明, 形式化方法.

黄箐(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为智能化软件.

王昌晶(1977－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为高可信软件, 智能化软件.

王唱唱(1999－), 女, 硕士生, 主要研究领域为定理证明, 形式化方法.