计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(1): 119−131, 2025

生物序列比对动态规划算法的统一形式化构造与 Isabelle 验证

石海鹤[1] 蓝孙文[1] 刘日明[1] 石海鹏[2] 王 岚[1] 钟林辉[1]
[1]（江西师范大学计算机信息工程学院 南昌 330022）
[2]（江西师范大学软件学院 南昌 330022）
（haiheshi@jxnu.edu.cn）

Unified Formal Construction and Isabelle Verification of the Dynamic Programming Algorithms for Biological Sequence Alignment

Shi Haihe[1], Lan Sunwen[1], Liu Riming[1], Shi Haipeng[2], Wang Lan[1], and Zhong Linhui[1]
[1]（*School of Computer Information Engineering*, *Jiangxi Normal University*, *Nanchang* 330022）
[2]（*School of Software*, *Jiangxi Normal University*, *Nanchang* 330022）

Abstract Sequence alignment is a classical problem in biological sequence analysis, aiming to find out the similarity between sequences, which is of great significance for discovering functional, structural and evolutionary information in biological sequences. The problem can be divided into two categories: pairwise sequence alignment and multiple sequence alignment. The existing work is focused on specific algorithms, and no general solution is designed. In addition, there are few researches on the trustworthy of algorithms. By deeply analyzing the properties of sequence alignment problem and describing the essential characteristics of problem solving, a unified construction framework of sequence alignment dynamic programming algorithm seqAlign is designed based on the problem formal specification and formal method PAR. The process of constructing a multiple sequence alignment algorithm with three sequences by using the framework is further demonstrated, and the constructed results are formally verified by Isabelle theorem prover. Finally, the C++ executable program of the algorithm is generated by the code generation system of PAR platform. The process of mechanized construction of other sequence alignment algorithms using seqAlign framework is analyzed. Through strict specification refinement and formal verification, the reliability of the generated algorithm is effectively guaranteed. The developed seqAlign framework provides a general solution for the class of sequence alignment problems, which significantly improves the efficiency of generating sequence alignment algorithm families. The successful application of the designed seqAlign framework to sequence alignment problem in biological sequence analysis can provide a reference for the construction of highly reliable algorithms in complex bioinformatics field from the perspective of methodology and practice.

Key words sequence alignment; PAR method; formal construction; Isabelle theorem prover

摘 要 序列比对是生物序列分析中的一个经典问题，旨在找出序列之间的相似性，它对于发现生物序列中的功能、结构和进化信息都具有重要的意义. 该问题可分为双序列比对和多序列比对 2 类，现有工作多针对特定算法展开，没有设计通用的求解方法；此外，甚少涉及算法可信性的研究. 从生物序列比对问题的形式化规约出发，通过深入分析问题的性质，刻画问题求解的本质特征，借助形式化方法 PAR（partition and recursion）设计了序列比对动态规划算法的统一构造框架 seqAlign；展示了应用该框架构造序列数为

收稿日期：2023-08-23；**修回日期**：2024-01-22
基金项目：国家自然科学基金项目（62062039）
This work was supported by the National Natural Science Foundation of China (62062039).
通信作者：石海鹏 (option2001@jxnu.edu.cn)

3 的多序列比对算法的过程,并使用 Isabelle 定理证明器对构造结果进行形式化验证;利用 PAR 平台生成了该算法的 C++可执行程序,进一步分析了由 seqAlign 框架机械化构造其他类型序列比对算法的过程.通过严密的规约精化和形式验证,有效地保证了生成算法的可信性;开发的 seqAlign 框架提供了序列比对问题类的通用求解方案,显著提高了序列比对算法族生成的效率.研究结果在生物序列分析中序列比对问题上的成功应用,从方法学和实践上可为复杂生物信息学领域高可靠算法的构造提供参考.

关键词 序列比对;PAR 方法;形式构造;Isabelle 定理证明器

中图法分类号 TP301.6

DOI: 10.7544/issn1000-1239.202330698 **CSTR:** 32373.14.issn1000-1239.202330698

软件发展的核心问题之一是如何保证软件系统安全可靠.随着社会基础设施中的软件系统越来越重要,如何保证复杂软件正确性和可靠性的需求日益突出[1],开发可靠的软件系统已经成为一项巨大的挑战[2].算法作为软件系统的核心,其可靠性显得尤为重要.

序列比对是计算机在解决生物学问题应用中的一个经典问题,旨在找出序列之间的相似性关系,以此来判定序列的同源性[3].按照比对的序列个数,可以将序列比对问题分为双序列比对和多序列比对 2 类.现有工作[4-16]主要关注序列比对特定问题的求解,没有为其提出一个统一的求解方法,算法开发效率较低.另外,算法可信性研究较少,导致算法的正确性无法有效保证.

形式化方法以严格的数学化和机械化方法为基础来规约、设计、构建、验证、演化软件系统,可以有效地保障和提高系统的可信性.软件形式化开发主要是构造、证明形式规约之间的等价转换和精化关系,以形式模型为指导,逐步求精,开发出满足需求的软件系统,也称为构造即正确(correctness-by-construction)的开发[17].许多著名的研究机构都投入了大量人力物力从事这方面的研究.美国宇航局于 2011 年 11 月发射的“好奇号”火星探测器研制过程中,就广泛使用了形式化方法以提高控制软件的可信性.在国内,2023 年 4 月华为联合 CCF 发布的第 3 期胡杨林基金形式化专项中包含了 5 个产业课题方向以及若干开放课题,力求实现关键基础技术底座自主、领先.此外,华为自主研发的鸿蒙操作系统 HarmonyOS 也使用了形式化方法.

薛锦云[18]提出的形式化方法 PAR(partition and recursion)提供了 Radl 语言来书写规约和描述算法,设计了规约精化规则,提出了循环不变式开发新策略,支持构造即正确的开发.使用 PAR 方法对算法程序进行形式化推导,不但能从逻辑上保证算法程序的正确性,而且能够揭示算法程序设计的本质特征.

本文借助 PAR 方法,对生物序列比对问题的特点进行深入分析,描述序列比对问题的通用性质,得到序列比对问题的通用形式化 Radl 规约,开发了序列比对动态规划(dynamic programming, DP)算法的统一构造框架 seqAlign,并通过严格的形式化推导和形式化证明来确保生成算法的可靠性.进一步展示了应用该框架构造序列数为 3 的多序列比对算法的过程,并使用 Isabelle 定理证明器对构造结果进行形式化验证,利用 PAR 平台的代码生成系统生成了该算法的 C++可执行程序.以 GenBank 数据库中 3 段关于光敏色素 B 的植物 DNA 序列作为输入,呈现了所生成序列比对算法的运行效果.总结分析了由 seqAlign 框架机械化构造其他类型序列比对算法的过程.应用 seqAlign 框架构造序列比对算法,不仅能有效保证生成的序列比对算法的可信性,而且可用于序列比对这一类问题的求解,避免了从问题出发逐步构造可靠算法的繁琐,只需简单地代入序列数和分析个性化性质即可快速生成特定序列比对算法,极大提高了序列比对算法生成的效率.本文开展的工作可为领域复杂算法的形式化构造提供一个有价值的参考框架.

1 相关工作

对于双序列比对问题,最为经典的确定性算法有基于全局比对的 NW(Needleman-Wunsch)算法[4]和基于局部比对的 SW(Smith-Waterman)算法[5].此外,Chowdhury 等人[6]提出了一种基于遗传算法的新的序列比对技术,可达到全局或接近全局最优解的能力;Alawneh 等人[7]提出了一种混合并行方法,结合了多核 CPU 的能力和当代 GPU 的能力,显著加快了目标算法的执行;Rashed 等人[8]使用 FPGA 和定制卷积神经网络来加速 DNA 的双序列比对,可达到

98.3% 的精度；Song 等人[9]定义了在序列比对系统中实现强化学习的环境和代理，提出一种新的双序列比对方法，使用深度强化学习来改进旧的双序列比对算法. 多序列比对算法主要包括动态规划算法和启发式比对算法. 前者是精准比对，可得到正确的结果，但是对计算机资源的消耗过高；后者是近似比对，占用资源较少，但只能得到近似解. 启发式比对算法可分为渐进式比对，如 Clustal 算法[10]，以及迭代式比对，如 MAFFT 算法[11]、MUSCLE 算法[12] 等. 后续研究中，Abuín 等人[13] 和 Wan 等人[14] 应用云计算实现 MSA 的并行化，分别成功地比对了 3.4 GB 和 15 GB 的序列文件，并在一定程度上保持了算法的准确性；Smirnov 等人[15] 提出了 MAGUS，使用与 PASTA[16] 相似的初始步骤，随后使用图聚类合并来合并子集比对，在大型数据集上产生了更高的准确性，比 PASTA 更快. 以上工作都是针对两类不同的问题分别设计解决方案，并未将其视为一类统一的问题寻找通用求解算法.

在算法形式构造方面，Durán[19] 提出了一种基于 Möller 代数的高效命令式网络算法的转换开发方法，并使用此方法推导了树的最短路径算法；Abrial 等人[20] 使用 Event-B 方法设计了最小生成树问题模型，并将其转化为相应的 Prim 算法；Almeida 等人[21] 从一个通用规范出发，由相似的转换步骤序列推导证明了 3 种已知的排序算法；Nedunuri 等人[22] 通过应用适当地保持语义一致性转换，将一个朴素的算法变成一个高效的程序，推导了 3 个最大子段和问题的程序；Mu[23] 使用等式推理构造了 n 皇后问题的回溯算法；Lammich[24] 提出了一个基于 Isabelle/HOL 生成经过验证的 LLVM 程序的框架，生成了经过验证的二分查找和 Knuth-Morris-Pratt 字符串搜索算法的 LLVM 实现；张健等人[25] 提出了一种正反例归纳合成方法，能够根据用户编辑的含有少量元组的示例表自动合成满足用户期望的 SQL 查询程序等. 已有工作使用方法众多，涉及领域很广，但未见有将形式化方法应用于同一类生物信息学领域问题的通用算法构造.

基于分划递推技术的 PAR 方法是一种简单实用的形式化方法，涵盖了算法开发的全阶段，可以清晰地展示算法的设计过程，快速开发同一类相似问题的求解算法，适用于构造即正确的算法开发[26]. 文献 [27−28] 使用 PAR 方法形式化开发了一系列组合数学问题的算法；文献 [29−31] 结合分划递推、领域工程和泛型编程，设计了高可信的可重用领域算法构件，装配了 3 类生物信息学的算法，说明了使用 PAR 方法开发相关算法的可行性. 生物序列包括 DNA、RNA 和生物大分子序列，可看成是组合数学中的序列结构，将 PAR 方法应用在生物信息学领域算法的形式化开发中具有充足的优势.

2 算法的统一形式构造与自动化验证框架

这里，我们首先简要介绍 PAR 方法开发算法的过程；然后通过分析生物序列比对问题的特征，进行 seqAlign 算法统一求解框架的形式构造.

2.1 算法的统一构造与生成流程

由 PAR 方法设计算法的统一构造框架，并进一步生成高可靠算法的流程分为 3 个步骤，如图 1 所示.

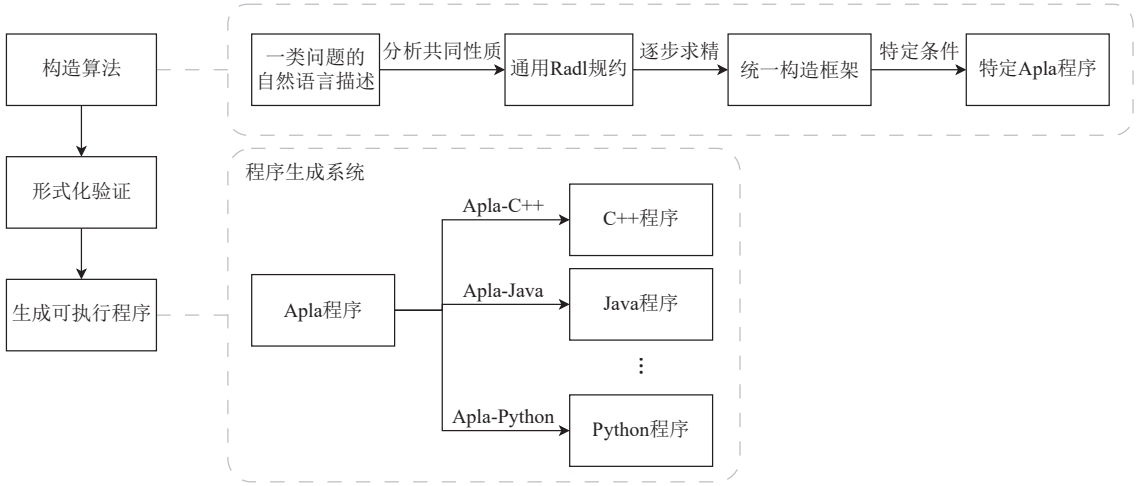

Fig. 1 Development flowchart of highly reliable algorithm based on PAR

图 1 基于 PAR 的高可靠算法开发流程图

2.1.1 构造算法

构造特定算法的程序分为3个步骤:第1步对一类问题深入分析,得到问题的通用规约;第2步使用规约精化规则对Radl规约进行精化推导,得到问题求解序列递推关系和Radl算法,从而设计该类问题的统一构造框架;第3步根据统一构造框架和特定条件,生成相应的Apla算法程序.

本文中的量词采用一种统一的表示形式 $(Q\ i : r(i) : f(i))$,它的含义是:在范围 $r(i)$ 上对函数 $f(i)$ 施行量词 Q 对应的 q 运算所得的值.主要用到的量词为MAX和Σ,对应的运算为max和+,分别表示求最大值和求和.利用量词的性质可以对规约作等价变换,以揭露问题求解的思想[26].下面给出了规约精化所使用的性质.

1)单一范围:$(Q\ i : i=k : f(i)) = f(k)$.

2)范围分裂:$(Q\ i : r(i) : f(i)) = (Q\ i : r(i) \wedge b(i) : f(i))\ q\ (Q\ i : r(i) \wedge \neg b(i) : f(i))$,其中 $b(i)$ 是布尔表达式.

3)函数结合律:$(Q\ i : r(i) : f(i)\ q\ g(i)) = (Q\ i : r(i) : f(i))\ q\ (Q\ i : r(i) : g(i))$.

4)范围析取:若 $a\ q\ a = a$,则称运算 q 是幂等的.对于幂等的运算 q,有 $(Q\ i : r(i) \vee s(i) : f(i)) = (Q\ i : r(i) : f(i))\ q\ (Q\ i : s(i) : f(i))$.

2.1.2 算法的形式化验证

构造正确的程序需要数学证明[32],因为传统的软件测试等方法只能证明系统软件中存在错误,而不能彻底消除错误.故要确保构造的程序是完全正确的,还需对其进行形式化验证.此外,通过程序正确性证明,还可加深对程序的理解.

由于Floyd归纳断言方法[33]和Hoare公理系统方法[34]书写恰当的中间断言十分困难,而Dijkstra最弱前置谓词法以定义的最弱前置谓词来书写中间断言[32,35],更加具有实用性.因此,本文选取Dijkstra最弱前置谓词法来证明Apla程序的正确性.该方法通过验证5个条件来证明循环语句的正确性:

1) $Q_{\text{pro}} => \rho$;

2) $\rho \wedge C_i => WP\left(\text{“}S_i\text{”}, \rho\right), 1 \leqslant i \leqslant n$;

3) $\rho \wedge \neg Guard => R_{\text{pro}}$;

4) $\rho \wedge Guard => \tau > 0$;

5) $\rho \wedge C_i => WP\left(\text{“}\tau_1 := \tau; S_i\text{”}, \tau_1 < \tau\right), 1 \leqslant i \leqslant n$.

其中,Q_{pro} 为前置断言,R_{pro} 为后置断言,ρ 为循环不变式,τ 为界函数,C_i 和 S_i 分别为循环条件分支语句和与其对应的循环体,$Guard$ 为所有分支 C_i 的析取.

Isabelle是当前被广泛使用的交互式定理证明器,相较于手工验证,自动化程度和可靠性更高.本文使用Dijkstra最弱前置谓词法,并以Isabelle定理证明器辅助证明所得Apla程序的正确性.将程序的验证条件输入到Isabelle中,使用其提供的自动证明策略证明.

2.1.3 生成可执行程序

PAR平台中包含系列程序生成系统[36-37],可以由抽象Apla程序自动生成C++/C#、Java等可执行程序.使用程序生成系统,可将形式化验证后的Apla程序进一步转换成对应的高可靠可执行算法程序.

2.2 序列比对DP算法的统一构造框架seqAlign

这里,分析序列比对问题的性质,抽象出共同规约,逐步精化得到递推关系,并开发序列比对DP算法的统一构造框架seqAlign.由于序列比对的启发式算法只能得到近似解,故seqAlign框架仅考虑可得到精确解的DP算法.

2.2.1 序列比对问题

双序列比对和多序列比对问题存在共性,都是参与比对的序列中的元素逐一连续地比对,最终结果是要得到最佳比对得分,以此来评定序列之间的相似性.双序列比对问题中参与比对的序列数目为2,多序列比对问题中参与比对的序列数目大于2.两者之间的区别只有参与比对的序列数目,故可将这两类问题刻画成一类问题.

给定 k 个参与比对的序列,其中 k 不小于2.序列中的元素可以有4种匹配方式:匹配 *match*、全空配 *smatch*、错配 *nmacth* 和空配 *space*,分别表示2个相同的元素匹配、2个空位匹配、2个不相同的元素匹配和其中1个序列中元素与空位匹配.每种匹配方式对应一次罚分,最常用的罚分规则有2种:权值恒定模型和仿射罚分模型,不同的罚分规则对应不同的序列相似性评判标准.为了更好地展示构造过程,本文使用固定罚分规则,如下所示:

$$sc\_rule \begin{cases} match = 1, \\ smatch = 0, \\ nmatch = -1, \\ space = -2. \end{cases}$$

从每个序列的第1个元素开始从前往后匹配,直到每个序列的元素全部匹配完,这为1次比对,比对得分为每次元素匹配对应的罚分之和.序列比对的最终目的则是在所有可能的比对情况中得到最大的比对得分.

2.2.2 通用形式规约

序列比对问题也可以描述成输入 k 条($k \geqslant 2$)序列,往序列中插入若干以“#”表示的空位,使每条序列对齐,即长度相等,从所有对齐的方式中找出最大的分

值. 这里用 *seqList* 表示输入的序列集合，用 $n_1, n_2, \cdots, n_k$ 分别表示参与比对的 k 条序列的长度. 每条序列对齐可以看成是一系列的单列对齐，即每条序列要么提供序列中的元素，要么提供空位，但是不能所有的序列都提供空位. 图 2 展示了输入 [A, C], [A, T, G, C], [A, A, C] 这 3 条序列的其中一种对齐过程.

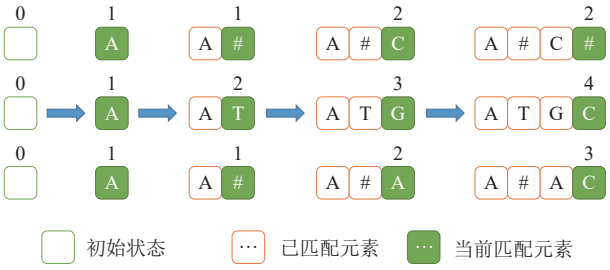

Fig. 2 Sample sequence alignment procedure

图 2 序列对齐过程样例

用 1 个 k 维向量 $(i_1, i_2, \cdots, i_k)$ 来描述当前的对齐进度，$i_1, i_2, \cdots, i_k$ 分别表示 k 个序列已提供元素的长度. 序列的 1 次全局比对可以看作是从初始对齐进度 (0, 0, ⋯, 0) 到终止对齐进度 $(n_1, n_2, \cdots, n_k)$ 的过程，这个过程包括了一系列的单列对齐. 由于每一次匹配中序列的两两匹配只可能是匹配、全空配、错配和空配 4 种方式，序列在这次单列对齐中要么提供空位，要么提供第 1 个未比对元素，而且 1 次匹配中不能要求全部序列都提供空位. 将当前的对齐进度 $(i_1, i_2, \cdots, i_k)$ 推进到下一个对齐进度 $(j_1, j_2, \cdots, j_k)$ 需满足的规则作如下描述：

规则 1. $\forall u \in \{1,2,\cdots,k\}, j_u = i_u + m_u \ (m_u = 1 \text{或} m_u = 0)$.

规则 2. $1 \leqslant \sum_{u=1}^{k} m_u \leqslant k$.

当前对齐序列最后一列中，当 $m_u = 1$时，表示第 u 个序列提供第 $i_u + 1$个元素，即 $seqList[u][i_u + 1]$；当 $m_u = 0$时，表示第 u 个序列提供空位. 规则 1 的一个例子可见图 3, 图 3(a) 为序列对齐进度，图 3(b) 为单列对齐. 很显然，当前参与匹配的非空位元素的个数不少于 1，且最多不能超过参与比对的序列个数.

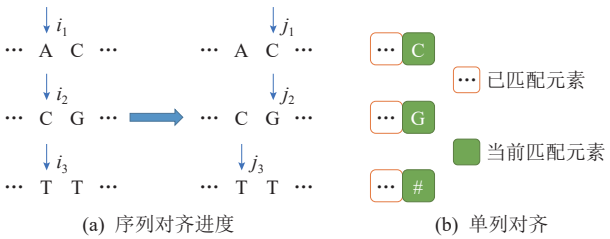

Fig. 3 Example of rule 1

图 3 规则 1 示例

为了简便记录对齐过程，这里用一个长度为 k 的序列 $[C_1, C_2, \cdots, C_k]$ 来存储从当前对齐进度到下一个对齐进度的单列对齐. 当$m_u = 1$时，$C_u = seqList[u][i_u + 1]$；当$m_u = 0$时，$C_u =$‘#’. 例如，图 2 中最后一步单列对齐，是从对齐进度 (2, 3, 2) 到 (2, 4, 3)，则这一步的单列对齐为 [#, *seqList*[2][4], *seqList*[3][3]]，即 [#, C, G]. 可以直观地看出，序列的对齐过程是若干单列对齐的集合，计算每步单列对齐的分值之和可得序列比对得分. 以 $align(seqList, i_1, i_2, \cdots, i_k)$ 表示从初始对齐进度 (0, 0, ⋯, 0) 到终止对齐进度 $(i_1, i_2, \cdots, i_k)$ 所有满足规则 1 和规则 2 的对齐过程集合，我们的目标则变成了从每个对齐过程的比对得分中找出最大值. 据此，我们可以得到序列比对问题的通用规约.

Specification: *SeqAlignment*

|[in: *k*:integer; *seqList*:*list*(*BioSeq*(*enum*), *k*); *len*:*list*(integer, *k*) out: *ms*:integer]|

AQ: given *seqList*, $k \geq 2$;

AR: *maxScore*(*seqList*, *len*) ≡ (MAX *cs*:*cs* ∈ *align*(*seqList*, *len*): (Σ*sgl*: *sgl* ∈ *cs*:*sglScore*(*sgl*))),

$sglScore(sgl) \equiv (\Sigma i, j{:}0 \leqslant i < j \leqslant k{:}rule(sgl[i], sgl[j]))$,

$$rule(ch1, ch2) = \begin{cases} match, ch1 = ch2 \neq \text{‘\#’}, \\ smatch, ch1 = ch2 = \text{‘\#’}, \\ nmatch, ch1 \neq ch2, \\ space, ch1 = \text{‘\#’} \vee ch2 = \text{‘\#’}. \end{cases}$$

其中 *BioSeq* 是我们自定义的类型构件，用来存储生物序列和操作生物序列. 构件中包含 3 个方法，函数 *rule* 和 *sglScore* 分别计算两两元素和单列对齐的匹配分值. 在算法的构造过程中，可以使用 *BioSeq* 构件中的方法，极大地增加了可重用性和简便性. 图 4 给出了 *BioSeq* 的定义和其中 3 个方法的抽象表达.

```
define ADT BioSeq(sometype elem, [size]);
  private BioSeq=list(elem);

  function rule(ch1, ch2:character):integer;
  {rule≡ch1='#' ∧ch2='#' ? smatch:ch1='#'
  ∨ch2='#' ? space:ch1=ch2 ? match:nmatch}

  function sglScore(baseList:list(character)):integer;
  {sglScore≡(Σ i, j:0≤i<j≤k:rule(baseList[i], baseList[j])}

  function calculate(flag:bool; score:integer):integer;
  {calculate≡flag ? score:INT_MIN}
enddef;
```

Fig. 4 *BioSeq* component

图 4 *BioSeq* 构件

图 4 中 sometype 修饰泛型参数，表示生物序列的元素类型可以改变，*size* 为可选参数，缺省表示序列长度没有限制，反之表示序列长度固定为 *size*. 建立一个枚举类型 *dnaEnum*=「A, C, G, T」，将泛型参

数设置为此枚举类型，就可以得到一个 DNA 序列类型 DNASeq. 如此可以很简便地得到 3 种生物序列类型：DNA 序列 DNASeq、RNA 序列 RNASeq 和蛋白质序列 ProSeq.

由规则 1 和规则 2，我们可以得到单列对齐可能的情况种数 $count_s$ 有 2 个性质：

性质 1. $count_s = \sum_{i=1}^{k} C_k^i$.

证明. 单列对齐中的前后 2 个对齐进度 $(i_1, i_2, \cdots, i_k)$ 和 $(j_1, j_2, \cdots, j_k)$，根据规则 1，单列对齐中每条序列要么提供空位，要么提供序列中的元素，再联合规则 2，可以得出单列对齐中非空位元素的个数范围是 $[1, k]$. 换句话说，每一种情况确定非空位元素的位置是组合问题，将每种情况求和即可得到 $count_s$. 证毕.

性质 2. $align(seqList, i_1, i_2, \cdots, i_k) =$

$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) \cup$
$\{[i_1, \cdots, i_{k-1}, i_k]\} \vee align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k) \cup$
$\{[i_1, \cdots, i_{k-1}, \#]\} \vee \cdots \vee align(seqList, i_1, \cdots, i_{k-1}-1, i_k) \cup$
$\{[\#, \cdots, i_{k-1}, i_k]\} \vee \cdots \vee align(seqList, i_1-1, i_2, \cdots, i_k) \cup$
$\{[i_1, \#, \cdots, \#]\} \vee \cdots \vee align(seqList, i_1, \cdots, i_{k-1}, i_k-1) \cup$
$\{[\#, \cdots, \#, i_k]\}$.

证明. 对于某个对齐进度，它的来源即前一个对齐进度可能有 $\sum_{i=1}^{k} C_k^i$ 种. 根据规则 1 和性质 1，将 $align(seqList, i_1, i_2, \cdots, i_k)$ 的最后一步单列对齐排列组合的情况一一列举出来，$[i_1, \cdots, i_{k-1}, i_k]$，$[i_1, \cdots, i_{k-1}, \#]$，…，$[\#, \cdots, i_{k-1}, i_k]$，…，$[i_1, \#, \cdots, \#]$，…，$[\#, \cdots, \#, i_k]$，即得上式. 证毕.

2.2.3 构造递推关系

将前面得到的规约一般化，用 $score(cs)$ 代替 $(\Sigma sgl : sgl \in cs : sglScore(sgl))$，进行以下推导：

$maxScore(seqList, i_1, i_2, \cdots, i_k) \equiv$

[*maxScore* 定义展开]

$(\text{MAX } cs : cs \in align(seqList, i_1, i_2, \cdots, i_k) : score(cs)) =$

[性质 2]

$(\text{MAX } cs : cs \in (align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) \cup$
$\{[i_1, \cdots, i_{k-1}, i_k]\} \vee align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k) \cup$
$\{[i_1, \cdots, i_{k-1}, \#]\} \vee \cdots \vee align(seqList, i_1, \cdots, i_{k-1}-1, i_k) \cup$
$\{[\#, \cdots, i_{k-1}, i_k]\} \vee \cdots \vee align(seqList, i_1-1, i_2, \cdots, i_k) \cup$
$\{[i_1, \#, \cdots, \#]\} \vee \cdots \vee align(seqList, i_1, \cdots, i_{k-1}, i_k-1) \cup$
$\{[\#, \cdots, \#, i_k]\}) : score(cs)) =$

[范围析取]

$\max((\text{MAX } cs : cs \in$
$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) \cup$
$\{[i_1, \cdots, i_{k-1}, i_k]\} : score(cs)), \cdots, \max((\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-2}, i_{k-1}-1, i_k) \cup$
$\{[\#, \cdots, \#, i_{k-1}, \#]\} : score(cs)), (\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-1}, i_k-1)) \cup$
$\{[\#, \cdots, \#, i_k]\} : score(cs)))) \equiv$

[变量替换]

$\max((\text{MAX } cs : cs \in$
$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) :$
$score(cs \cup \{[i_1, \cdots, i_{k-1}, i_k]\})), \cdots, \max((\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-2}, i_{k-1}-1, i_k) :$
$score(cs \cup \{[\#, \cdots, \#, i_{k-1}, \#]\})), (\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-1}, i_k-1)) : score(cs \cup$
$\{[\#, \cdots, \#, i_k]\})))) \equiv$

[*sglScore* 定义收缩]

$\max((\text{MAX } cs : cs \in$
$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) : (\Sigma sgl : sgl \in cs \cup$
$\{[i_1, \cdots, i_{k-1}, i_k]\} : sglScore(sgl))), \cdots, (\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-1}, i_k-1)) : (\Sigma sgl : sgl \in cs \cup$
$\{[\#, \cdots, \#, i_k]\} : sglScore(sgl)))) \equiv$

[范围分裂]

$\max((\text{MAX } cs : cs \in$
$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) : (\Sigma sgl : sgl \in$
$cs : sglScore(sgl)) + (\Sigma sgl : sgl = [i_1, \cdots, i_{k-1}, i_k] :$
$sglScore(sgl))), \cdots, (\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-1}, i_k-1)) : (\Sigma sgl : sgl \in$
$cs : sglScore(sgl)) + (\Sigma sgl : sgl = [\#, \cdots, \#, i_k] :$
$sglScore(sgl)))) \equiv$

[单一范围]

$\max((\text{MAX } cs : cs \in$
$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) : (\Sigma sgl : sgl \in$
$cs : sglScore(sgl)) + sglScore([i_1, \cdots, i_{k-1}, i_k])), \cdots,$
$(\text{MAX } cs : cs \in align(seqList, i_1, \cdots, i_{k-1}, i_k-1)) :$
$(\Sigma sgl : sgl \in cs : sglScore(sgl)) +$
$sglScore([\#, \cdots, \#, i_k]))) \equiv$

[函数结合律]

$\max((\text{MAX } cs : cs \in$
$align(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) : score(cs)) +$
$sglScore([i_1, \cdots, i_{k-1}, i_k]), \cdots, (\text{MAX } cs : cs \in$
$align(seqList, i_1, \cdots, i_{k-1}, i_k-1)) : score(cs)) +$
$sglScore([\#, \cdots, \#, i_k])) \equiv$
$\max(maxScore(seqList, i_1-1, \cdots, i_{k-1}-1, i_k-1) +$
$sglScore([i_1, \cdots, i_{k-1}, i_k]), \cdots,$
$maxScore(seqList, i_1, \cdots, i_{k-1}, i_k-1) +$
$sglScore([\#, \cdots, \#, i_k]))$.

不难看出，原问题被划分为$\sum_{i=1}^{k}C_k^i$个子问题. 由此，可以得到序列比对问题一般情况的通用递推关系：

$$maxScore(seqList,i_1,i_2,\cdots,i_k)=$$
$$\max\begin{cases} maxScore(seqList,i_1-1,\cdots,i_{k-1}-1,i_k-1)+ \\ \quad sglScore([i_1,\cdots,i_{k-1},i_k]), \\ maxScore(seqList,i_1-1,\cdots,i_{k-1}-1,i_k)+ \\ \quad sglScore([i_1,\cdots,i_{k-1},\#]),\cdots, \\ maxScore(seqList,i_1,i_2-1,\cdots,i_k-1)+ \\ \quad sglScore([\#,i_2,\cdots,i_k]),\cdots, \\ maxScore(seqList,i_1-1,i_2,\cdots,i_k)+ \\ \quad sglScore([i_1,\#,\cdots,\#]),\cdots, \\ maxScore(seqList,i_1,\cdots,i_{k-1},i_k-1)+ \\ \quad sglScore([\#,\cdots,\#,i_k]). \end{cases} \tag{1}$$

原问题被分划成了若干子问题，而对齐进度中每条序列已提供元素的长度不能小于 0. 所以，如果子问题的对齐进度中有序列提供的长度小于 0，那么这个子问题将不存在. 本文将不存在的子问题的分值设置成 integer 类型的最小值，并且把实现这个功能的 *calculate* 函数集成到 *BioSeq* 构件中. 根据指定特定罚分规则的递推关系，即可构造出如下序列比对问题的统一 Radl 求解算法：

算法 1. seqAlign.

|[var k, i_1, i_2, ⋯, i_k:integer;

score:*array*(0.. n_1, *array*(0.. n_2, ⋯, *array*(0.. n_k, integer))); *seqList*:*list*(*BioSeq*(), k);]|

$\{AQ\wedge AR\}$

begin : $i_1=0++1\wedge i_2=0++1\wedge\cdots\wedge i_k=0++1$;

termination : $i_1=n_1\wedge i_2=n_2\wedge\cdots\wedge i_k=n_k$;

recur: 式(1)；

end

2.2.4 开发循环不变式

文献 [38] 给出了循环不变式的新定义，即反映循环体中所有循环变量的变化规律并在循环体执行前后都为真的谓词称为该循环体的循环不变式. 其中，循环变量定义为在循环体中，其值随着循环体的执行不断发生变化的变量.

由得到的 Radl 算法可以看出，该算法有 k 层循环，变量 i_u 控制第 u 层循环，每层循环分别对应不同的循环不变式. 用变量 *ms* 存放 *maxScore*(*seqList*, i_1, i_2, ⋯, i_k) 的值，那么循环内的所有变量为 i_1, i_2, ⋯, i_k 和 *ms*. 首先分析最外层循环，跟随循环体执行发生变化的变量为 i_1 和 *ms*，其他变量与外层循环无关，因为无论最外层循环执行到哪一步，最外层循环体执行前其他变量的值都为 0. 循环变量 *ms* 的值始终为 (MAX cs : cs = *align*(*seqList*, i_1, i_2, ⋯, i_k) : *score*(cs))，循环变量 i_1 的变化范围为 [0, n_1]. 据此循环变量的变化规律，用谓词精确表示，即得出外层循环的循环不变式：

$$ms=maxScore(seqList,i_1,i_2,\cdots,i_k)=(\text{MAX } cs: cs=align(seqList,i_1,i_2,\cdots,i_k))\wedge(0\leqslant i_1\leqslant n_1).$$

同理，可构造其他层的循环不变式.

3 特例研究

在通用框架的基础上，以输入 3 条序列的多序列比对为例，构造序列比对算法的 Apla 程序. 通过算法程序的形式化验证，生成高可靠的 C++可执行程序.

3.1 三序列比对算法程序生成

2.2 节分析了序列比对问题的性质，得到了序列比对算法的统一构造框架 seqAlign. 要得到序列数为 3 的序列比对算法的递推关系，只需将 k=3 代入 seqAlign 框架的递推关系中：

$$maxScore(seqList,i_1,i_2,i_3)=$$
$$\max\begin{cases} maxScore(seqList,i_1-1,i_2-1,i_3-1)+ \\ \quad sglScore([i_1,i_2,i_3]),i_1,i_2,i_3>0, \\ maxScore(seqList,i_1-1,i_2-1,i_3)+ \\ \quad sglScore([i_1,i_2,\#]),i_1,i_2>0, \\ maxScore(seqList,i_1-1,i_2,i_3-1)+ \\ \quad sglScore([i_1,\#,i_3]),i_1,i_3>0, \\ maxScore(seqList,i_1,i_2-1,i_3-1)+ \\ \quad sglScore([\#,i_2,i_3]),i_2,i_3>0, \\ maxScore(seqList,i_1-1,i_2,i_3)+ \\ \quad sglScore([i_1,\#,\#]),i_1>0, \\ maxScore(seqList,i_1,i_2-1,i_3)+ \\ \quad sglScore([\#,i_2,\#]),i_2>0, \\ maxScore(seqList,i_1,i_2,i_3-1)+ \\ \quad sglScore([\#,\#,i_3]),i_3>0. \end{cases} \tag{2}$$

考虑特殊情况，当 i_1, i_2, i_3 都为 0 时，即当前处于序列比对的初始状态，对齐进度为 (0, 0, 0). 此时，*maxScore*(*seqList*, i_1, i_2, i_3) = 0. 将一般情况和特殊情况结合，可得到如下的 Radl 算法：

算法 2. mulSeqAlign.

|[var k, i_1, i_2, i_3:integer; *score*: *array*(0.. n_1, *array*(0.. n_2, *array*(0.. n_3, integer))); *seqList*: *list*(*BioSeq*(), 3);]|

$\{AQ\wedge AR\}$

begin : $i_1=0++1\wedge i_2=0++1\wedge i_3=0++1$;

termination : $i_1=n_1\wedge i_2=n_2\wedge i_3=n_3$;

recur: 式(2)；

end

循环不变式如下：

$ms = maxScore(seqList, i_1, i_2, i_3) = (\text{MAX } cs : cs = align(seqList, i_1, i_2, i_3)) \wedge (0 \leqslant i_1 \leqslant n_1)$，

$ms = maxScore(seqList, i_1, i_2, i_3) = (\text{MAX } cs : cs = align(seqList, i_1, i_2, i_3)) \wedge (0 \leqslant i_2 \leqslant n_2)$，

$ms = maxScore(seqList, i_1, i_2, i_3) = (\text{MAX } cs : cs = align(seqList, i_1, i_2, i_3)) \wedge (0 \leqslant i_3 \leqslant n_3)$.

根据得到的递推关系和 Radl 算法以及循环不变式，经过简单的推导，可以简捷地得到基于 *BioSeq* 构件编写的 Apla 抽象算法程序. 通过严密的规约求精，保证了序列比对 Apla 算法程序的正确性. 限于篇幅，这里只给出比对算法的主体函数 *matching*，如图 5 所示.

3.2 基于 Isabelle 的自动化验证

创建一个新的理论 *multSeqAlignment* 用于本文中 Apla 程序的形式化证明，理论名需要与文件名一致. 由于 Isabelle 自带的 Main 理论中包含了本文所用到的 list 类型，故这里只需要用 import 引入 Main 理论即可. 使用 definition 以及 fun 命令定义递推关系 *maxScore*、分值计算规则 *rule* 和 *sglScore*，如图 6 所示.

本文构造的序列比对算法程序的主体是 3 层循环，循环中的递推关系和罚分规则是经过严格推导得到的，故只需证明双层循环的正确性即可. 使用 Dijkstra 最弱前置谓词法证明循环语句的正确性，找到循环语句的循环不变式 ρ 和界函数 τ 是关键. 其中，Apla 程序中的循环不变式已通过求解并且在上文给出. 另外，3 层循环对应的界函数显然可以为 $n_1 - i_1 + 1$，$n_2 - i_2 + 1$，$n_3 - i_3 + 1$. 接下来，将验证循环程序需要的 5 个条件形式化表示.

1) $i = 0 \wedge j = 0 \wedge k = 0 ==> ms = (maxScores_1 s_2 s_3 ijk) \wedge (0 \leqslant i \wedge i \leqslant n_1)$;

2) $ms = (maxScores_1 s_2 s_3 ijk) \wedge (0 \leqslant i \wedge i \leqslant n_1) \wedge i < n_1 ==> (maxScores_1 s_2 s_3 (\text{Suc } i)\ jk) = (maxScores_1 s_2 s_3 (\text{Suc } i)\ jk) \wedge (0 \leqslant \text{Suc } i \wedge \text{Suc } i \leqslant n_1)$;

3) $ms = (maxScore\ s_1 s_2 s_3 ijk) \wedge (0 \leqslant i \wedge i \leqslant n_1) \wedge \neg i < n_1 ==> ms = (maxScore\ s_1 s_2 s_3 n_1 jk)$;

4) $ms = (maxScore\ s_1 s_2 s_3 ijk) \wedge (0 \leqslant i \wedge i \leqslant n_1) \wedge i < n_1 ==> 0 \leqslant n_1 - i + 1$;

5) $ms = (maxScore\ s_1 s_2 s_3 ijk) \wedge (0 \leqslant i \wedge i \leqslant n_1) \wedge i < n_1 ==> n_1 - i < n_1 - i + 1$.

上述 5 条引理可以用自动证明工具 auto 自动消解证明目标，只需使用“by auto”命令即可完成证明，此时证明状态全部变为了“No subgoals！”，表示 5 条引理皆被证明是正确的. 由于 3 层循环的证明并无二致，这里不再给出其他循环层的证明过程. 经上述证明，说明循环不变式在循环执行前后以及循环执行过程中皆为真，验证了生成的 Apla 程序的正确性.

3.3 可执行程序的生成

在使用 Isabelle 定理证明器验证 Apla 算法程序的正确性之后，通过 PAR 平台中的 Apla—>C++程序生成系统，生成了对应的 C++程序.

为了方便查看最佳比对方式，我们在前述三序列比对算法的 Apla 程序的基础上，存储最佳比对方式的每一步匹配，并加入了输出最佳比对方式的过程. 在计算当前对齐进度 (i, j, k) 的最大比对得分时，得到最大比分的来源是可以确定的，也就是前一次的对齐进度. 因此，如需得到最佳比对方式，只需要在计算每一个对齐进度的最大比分时，使用一个 3 维数组 *path* 来保存最大比分的来源. 由性质 1 可知，三序列比对最大比分的来源可能有 7 种情况，故将 *path* 中元素取值范围设为 {0,1,2,3,4,5,6}. 将最大比分来源 $(i-1, j-1, k-1)$, $(i-1, j-1, k)$, $(i-1, j, k-1)$, $(i, j-1,$

```
function matching (var i, j, k:integer):integer;
begin
  if (i=0) ∧ (j=0) ∧ (k=0) → matching :=0;
  [] (i≠0) ∨ (j≠0) ∨ (k≠0) → matching :=
  max(bioSeq.calculate(i > 0 ∧ j > 0 ∧ k > 0, score[i−1, j−1, k−1] + bioSeq.sglScore([seq1[i−1], seq2[j−1], seq3[k−1]])),
  max(bioSeq.calculate(i > 0 ∧ j > 0, score[i−1, j−1, k] + bioSeq.sglScore([seq1[i−1], seq2[j−1], '#'])),
  max(bioSeq.calculate(i > 0 ∧ k > 0, score[i−1, j, k−1] + bioSeq.sglScore([seq1[i−1], '#', seq3[k−1]])),
  max(bioSeq.calculate(j > 0 ∧ k > 0, score[i, j−1, k−1] + bioSeq.sglScore(['#', seq2[j−1], seq3[k−1]])),
  max(bioSeq.calculate(i > 0, score[i−1, j, k] + bioSeq.sglScore([seq1[i−1], '#', '#'])),
  max(bioSeq.calculate(j > 0, score[i, j−1, k] + bioSeq.sglScore(['#', seq2[j−1], '#'])),
  bioSeq.calculate(k > 0, score[i, j, k−1] + bioSeq.sglScore(['#', '#', seq3[k−1]]))))))));
  fi;
end;
```

Fig. 5 *matching* function

图 5 *matching* 函数

```
datatype DNABase = A | G | C | T
type_synonym DNASeq = "DNABase list"

definition rule :: "DNASeq => DNASeq => int => int => int" where
"rule s1 s2 i j = (if i = −1 ∧ j = −1 then 0 else if i = −1 ∨ j = −1 then −2 else if (s1! nat i) = (s2! nat j) then 1 else −1)"

definition sglScore :: "DNASeq => DNASeq => DNASeq => int => int => int =>  int"  where
"sglScore s1 s2 s3 i j k = ((rule s1 s2 i j) + (rule s1 s3 i k) + (rule s2 s3 j k))"

fun maxScore :: "DNASeq => DNASeq => DNASeq => nat => nat => nat => int" where
"maxScore s1 s2 s3 0 0 0 = 0" |
"maxScore s1 s2 s3 (Suc i) 0 0 = ((maxScore s1 s2 s3 i 0 0) + (sglScore s1 s2 s3 (int i) (−1) (−1)))" |
"maxScore s1 s2 s3 0 (Suc j) 0 = ((maxScore s1 s2 s3 0 j 0) + (sglScore s1 s2 s3 (−1) (int j) (−1)))" |
"maxScore s1 s2 s3 0 0 (Suc k) = ((maxScore s1 s2 s3 0 0 k) + (sglScore s1 s2 s3 (−1) (−1) (int k)))" |
"maxScore s1 s2 s3 (Suc i) (Suc j) 0 = (max ((maxScore s1 s2 s3 i j 0) + (sglScore s1 s2 s3 (int i) (int j) (−1)))
(max ((maxScore s1 s2 s3 (Suc i)j 0) + (sglScore s1 s2 s3 (−1) (int j) (−1))) ((maxScore s1 s2 s3 i (Suc j) 0) + (sglScore s1 s2
s3 (int i) (−1) (−1)))))" |
"maxScore s1 s2 s3 (Suc i) 0 (Suc k) = (max ((maxScore s1 s2 s3 i 0 k) + (sglScore s1 s2 s3 (int i) (−1) (int k)))
(max ((maxScore s1 s2 s3 (Suc i) 0 k) + (sglScore s1 s2 s3 (−1) (−1) (int k))) ((maxScore s1 s2 s3 i 0 (Suc k)) + (sglScore s1 s2
s3 (int i) (−1) (−1)))))" |
"maxScore s1 s2 s3 0 (Suc j) (Suc k) = (max ((maxScore s1 s2 s3 0 j k) + (sglScore s1 s2 s3 (−1) (int j) (int k)))
(max ((maxScore s1 s2 s3 0 (Suc j)k) + (sglScore s1 s2 s3 (−1) (−1) (int k))) ((maxScore s1 s2 s3 0 j (Suc k)) + (sglScore s1 s2
s3 (−1) (int i) (−1)))))" |
"maxScore s1 s2 s3 (Suc i) (Suc j) (Suc k) = (max ((maxScore s1 s2 s3 i j k) + (sglScore s1 s2 s3 (int i) (int j) (int k)))
(max ((maxScore s1 s2 s3 i j (Suc k)) + (sglScore s1 s2 s3 (int i) (int j) (−1))) (max ((maxScore s1 s2 s3 i (Suc j) k) + (sglScore
s1 s2 s3 (int i) (−1) (int k)))
(max ((maxScore s1 s2 s3 (Suc i) j k) + (sglScore s1 s2 s3 (−1) (int j) (int k))) (max ((maxScore s1 s2 s3 i (Suc j) (Suc k)) +
(sglScore s1 s2 s3 (int i) (−1) (−1)))
(max ((maxScore s1 s2 s3 (Suc i) j (Suc k)) + (sglScore s1 s2 s3 (−1) (int j) (−1))) ((maxScore s1 s2 s3 (Suc i) (Suc j) k) +
(sglScore s1 s2 s3 (−1) (−1) (int k)))))))))"
```

Fig. 6　Recursive relations and penalty rules

图 6　递推关系和罚分规则

k−1), (i−1, j, k), (i, j−1, k), (i, j, k−1) 分别标记为以上 7 种情况的 7 个值, 记录在 *path*[i, j, k] 中.

确定了数组 *path* 的值后, 从终止对齐进度 (n_1, n_2, n_3) 开始往前遍历, 直到初始对齐进度, 即 (0, 0, 0), 记录比分来源的函数 *matching* 见图 7. 用 3 个栈存放三序列对齐后的结果, 将每一次参与匹配的序列元素

```
function matching (var i, j, k:integer):integer;
var
    m : integer;
    temp : array[0..6, integer];
begin
    if (i=0) ∧ (j=0) ∧ (k=0) → matching := 0;
    [] (i≠0) ∨ (j≠0) ∨ (k≠0) →
       temp[0] = bioSeq.calculate(i > 0 ∧ j > 0 ∧ k > 0, score[i−1, j−1, k−1] + bioSeq.sglScore(seq1[i−1], seq2[j−1], seq3[k−1])),
       temp[1] = bioSeq.calculate(i > 0 ∧ j > 0, score[i−1, j−1, k] + bioSeq.sglScore(seq1[i−1], seq2[j−1], '#'));
       temp[2] = bioSeq.calculate(i > 0 ∧ k > 0, score[i−1, j, k−1] + bioSeq.sglScore(seq1[i−1], '#', seq3[k−1]));
       temp[3] = bioSeq.calculate(j > 0 ∧ k > 0,  score[i, j−1, k−1] + bioSeq.sglScore('#', seq2[j−1], seq3[k−1]));
       temp[4] = bioSeq.calculate(i > 0, score[i−1, j, k] + bioSeq.sglScore(seq1[i−1], '#', '#'));
       temp[5] = bioSeq.calculate(j > 0, score[i, j−1, k] + bioSeq.sglScore('#', seq2[j−1], '#'));
       temp[6] = bioSeq.calculate(k > 0, score[i, j, k−1] + bioSeq.sglScore('#', '#', seq3[k−1]));
       matching := max(temp[0], max(temp[1], max(temp[2], max(temp[3], max(temp[4], max(temp[5], temp[6])))));
       m = 0;
       do m < 7 →
          if matching = temp[m] → path[i, j, k] = m + 1;
          fi;
          m = m + 1;
    od;
   fi;
end;
```

Fig. 7　The function *matching* that records the source of the score

图 7　记录比分来源的函数 *matching*

存入三序列对应的栈中, 最后将栈顶元素依次出栈即可得到最佳比对方式.

确保三序列比对算法 Apla 程序的正确性后, 加上文件输入、输出语句, 以“#”表示空位, 使用 PAR 平台中的程序生成系统, 可以由抽象 Apla 程序生成 Java, C++, Python 等可执行程序. 这里仅给出 C++程序的一个实例运行结果, 如图 8 所示. 其中, 输入序列来源于 Genbank 数据库中 3 段关于光敏色素 B 的植物 DNA 序列.

4 其他序列比对算法形式构造的分析

根据实际应用情况, 这里对序列比对高可靠算法的通用构造过程总结如下:

1)分析特定输入序列数 $k(k\geq 2)$的序列比对问题, 根据独有特征简化通用规约和递推关系, 继而将 k 代入到本文提出序列比对算法统一构造框架 seqAlign 中得到 Radl 算法.

2)根据 Radl 算法开发出特定序列比对算法的循环不变式, 并基于此生成 Apla 抽象程序.

3)使用 Isabelle 语言, 将验证 Apla 程序正确性需满足的条件描述成定理, 然后使用 Isabelle 定理证明器加以验证.

4)在 PAR 平台的代码生成系统支撑下, 可由形式验证后的 Apla 程序自动生成得到对应的高级语言程序, 从而得到一个高可靠的序列比对算法程序.

比对序列条数大于等于 4 的情况与三序列比对问题本质上是一致的, 而与双序列比对问题有略微差别. 区别在于双序列比对的序列数为 2, 计算单次匹配的分值不需要序列之间两两匹配的分值相加, 只需要计算 1 次即可. 此时, 不再需要 *sglScore*, 只需留下 *rule* 规则. 针对双序列比对算法的特性, 将规约作如下简化:

Specification: *pairSequenceAlignment*

|[in: *m*, *n*:integer; *seq*1, *seq*2:*BioSeq*(*enum*); out: *ms*: integer]|

AQ: given *seq*1[0:*m*−1], *seq*2[0:*n*−1], *m*>1, *n*>1;

AR: $maxScore(seq_1,seq_2,m,n)=(\text{MAX } cs: cs=align(seq_1,seq_2,m,n):(\Sigma sgl: sgl\in cs: rule(sgl)))$.

其中 *rule* 的定义如下:

$$rule(ch_1,ch_2)=\begin{cases} space, ch_1=\text{‘\#’}\vee ch_2=\text{‘\#’},\\ match, ch_1\neq\text{‘\#’}\vee ch_2\neq\text{‘\#’}\wedge ch_1=ch_2,\\ nmatch, ch_1\neq\text{‘\#’}\vee ch_2\neq\text{‘\#’}\wedge ch_1\neq ch_2. \end{cases}$$

通过性质 1 和性质 2, 可以得出对齐进度 (i,j) 可能有 3 种来源, 即 $(i-1,j)$, $(i,j-1)$, $(i-1,j-1)$. 据此, 可以得到递推关系:

$$maxScore(seq_1,seq_2,i,j)=\max\begin{cases} maxScore(seq_1,seq_2,i-1,j)+\\ \quad rule(seq_1[i-1],\#),\\ maxScore(seq_1,seq_2,i,j-1)+\\ \quad rule(\#,seq_2[j-1]),\\ maxScore(seq_1,seq_2,i-1,j-1)+\\ \quad rule(seq_1[i-1],seq_2[j-1]). \end{cases}\tag{3}$$

进一步得到 Radl 算法:

算法 3. pairSeqAlign.

|[var *i*,*j*:integer; *score*:*array*(0..*m*,*array*(0..*n*, integer)); seq_1, seq_2:*BioSeq*();]|

$\{AQ\wedge AR\}$

begin : $i=0++1\wedge j=0++1$;

termination : $i=m\wedge j=n$;

recur: 式(3);

end

以及循环不变式:

$ms=maxScore(seq_1,seq_2,m,n)=(\text{MAX } cs: cs=align(seq_1,seq_2,m,n): score(cs))\wedge(0\leqslant i\leqslant m)$,

$ms=maxScore(seq_1,seq_2,m,n)=(\text{MAX } cs: cs=align(seq_1,seq_2,m,n): score(cs))\wedge(0\leqslant j\leqslant n)$.

由得到的递推关系和 Radl 算法以及循环不变式, 可推导出 Apla 抽象算法程序. 进一步通过 Isabelle 定

```
For the following three sequences:
TAAAGAAAAAGGGAAAACTGTACTTGTGACGGAGAGCAAAGAGAGAGAGAGAGAGAGAATGAGTTGAGTG    [length:70]
AGAGACAGCACACTCTCTCTTCTATCTATCCCCTGAGTCCCTGACCTTCCCTTTCGAGCCATCGCATCGT    [length:70]
AAGATAAATGGCGCACCAATCACCTTGACTCAATTATGTTCACCACCTCACTCCTCAGCCACAAACATTT    [length:70]
The max score is : −66
The process of multiple sequence alignment is shown below:
#AAAGAAAA##AGG#GAAA#ACTGTACTT#GTGAC#GGAGAG#C#AAAGAGAGAGAGAGAGAGAGAATGAGTTGAGTG
##GAGACAG####CACACTCTCTCTTCTATCTATCCCCTGAG#TCCCTGACCTTCCC#TTTCGAGCCATCGCATCGT#
A#GATAAATGG##CGCACC#AATCACCTT#GACTC#AATTATGT#TCACCACCTCAC#TCCTCAGCCACAAACATTT#
```

Fig. 8 C++ program's running result of three sequence alignment algorithm

图 8 三序列比对算法的 C++程序运行结果

理证明器验证，继而使用 PAR 平台的程序生成系统即可生成一个高可靠的双序列比对算法，与 NW 算法思想一致，代码细节略有不同.

5 总　　结

本文针对生物序列比对问题，开展了形式化方法 PAR 制导的问题类通用求解框架研究. 从生物序列比对问题出发，抽象出通用的 Radl 形式化规约，逐步精化得到统一的序列比对 DP 算法的构造框架 seqAlign. 将输入序列数代入 seqAlign 框架中，生成了特定的序列比对动态规划算法，极大提高了算法设计效率，经 Isabelle 定理证明器的验证以及 PAR 平台 C++可执行代码的生成，有效保证了所生成算法的正确性，表明了 seqAlign 统一求解框架的有效性和实用性.

本文的主要贡献有 3 个方面：

1）形式化推导了一个统一的序列比对问题求解框架 seqAlign，解决了一类生物序列比对问题. 从高抽象的问题通用规约出发，经严格的形式变换得到了一般化的求解递推关系. 对于特定的序列比对问题，只需将比对序列数代入到通用求解框架中，稍作改变即可得到具体的算法程序，在重用统一求解框架的同时，也实现了对算法设计过程的重用，显著提高了算法开发效率和可信性.

2）提供了一个形式化构建复杂生物信息学领域典型算法的成功样例. 在生物信息学序列分析领域，许多问题具有相似性，本文的研究方法可扩展到生物信息学领域其他算法的形式构造研究中.

3）基于 PAR 方法的通用序列比对问题求解框架的构造，降低了一类复杂问题的验证代价，促进了形式化验证在相关领域的实施. 在展示了一类序列比对问题令人惊叹的解题思想之外，形式化推导的过程也描绘出得到这种精美思路的清晰逻辑.

本文算法构造的过程借助了 PAR 平台 C++程序生成系统和 Isabelle 工具辅助验证，将尽可能多的创造性劳动转化为非创造性劳动，让原本困难的算法设计过程变为按部就班的工作. 接下来将进一步研究递推关系和循环不变式的自动生成方法，高效衔接算法形式构造与形式验证 2 部分工作，将更多算法形式构造中的创造性工作转化为机械性工作，以期不断提升自动化水平.

作者贡献声明：石海鹤设计研究方法、实验设计，并进行论文修改；蓝孙文完成实验部分并撰写论文；刘日明协助数据采集、处理和分析；石海鹏把握论文的总体创新性，验证理论分析和实验分析的正确性；王岚和钟林辉检验算法的性能，提出指导意见，并参与论文修改.

参　考　文　献

[1] Zhan Naijun, Wang Ji. Challenges and trends for specification, analysis, and verification of complex systems[J]. Science and Technology Foresight, 2023, 2(1): 7−22 (in Chinese)
(詹乃军, 王戟. 复杂系统规约、分析与验证发展现状与展望[J]. 前瞻科技, 2023, 2(1): 7−22)

[2] Hoare T. The verifying compiler: A grand challenge for computing research[J]. Journal of the ACM, 2003, 50(1): 63−69

[3] Wang Yongxian. Introduction to Bioinformatics: Algorithms and Applications for High Performance Computing [M]. Beijing: Tsinghua University Press, 2011 (in Chinese)
(王勇献. 生物信息学导论: 面向高性能计算的算法与应用[M]. 北京: 清华大学出版社, 2011)

[4] Needleman S B, Wunsch C D. A general method applicable to the search for similarities in the amino acid sequence of two proteins[J]. Journal of Molecular Biology, 1970, 48(3): 443−453

[5] Smith T F, Waterman M S. Identification of common molecular subsequences.[J]. Journal of Molecular Biology, 1981, 147(1): 195−197

[6] Garai G, Chowdhury B. A cascaded pairwise biomolecular sequence alignment technique using evolutionary algorithm[J]. Information Sciences, 2015, 297(3): 118−139

[7] Alawneh L, Shehab M A, Al-Ayyoub M, et al. A scalable multiple pairwise protein sequence alignment acceleration using hybrid CPU–GPU approach[J]. Cluster Computing, 2020, 23(7): 2677−2688

[8] Rashed A E E D, Obaya M, El H, et al. Accelerating DNA pairwise sequence alignment using FPGA and a customized convolutional neural network[J]. Computers & Electrical Engineering, 2021, 92(6): 1−21

[9] Song Y J, Ji D J, Seo H, et al. Pairwise heuristic sequence alignment algorithm based on deep reinforcement learning[J]. IEEE Open Journal of Engineering in Medicine and Biology, 2021, 2: 36−43

[10] Clustal W, Thompson J D, Higgins D G, et al. Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice[J]. Nucleic Acids Research, 1994, 22(22): 4673−4680

[11] Katoh K, Misawa K, Kuma K, et al. MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform[J]. Nucleic Acids Research, 2002, 30(14): 3059−3066

[12] Edgar R C. MUSCLE: Multiple sequence alignment with high accuracy and high throughput[J]. Nucleic Acids Research, 2004, 32(5): 1792−1797

[13] Abuín J M, Pena T F, Pichel J C. PASTASpark: Multiple sequence alignment meets big data[J]. Bioinformatics, 2017, 33(18): 2948−2950

[14] Wan Shixiang, Zou Quan. HAlign-II: Efficient ultra-large multiple

sequence alignment and phylogenetic tree reconstruction with distributed and parallel computing[J]. Algorithms for Molecular Biology, 2017, 12(1): 1−10

[15] Smirnov V, Warnow T. MAGUS: Multiple sequence alignment using graph clustering[J]. Bioinformatics, 2021, 37(12): 1666−1672

[16] Mirarab S, Nguyen N, Guo Sheng, et al. PASTA: Ultra-large multiple sequence alignment for nucleotide and amino-acid sequences.[J]. Journal of Computational Biology: A Journal of Computational Molecular Cell Biology, 2015, 22(5): 377−386

[17] Wang Ji, Zhan Naijun, Feng Xinyu, et al. Overview formalization method[J]. Journal of Software, 2019, 30(1): 33−61 (in Chinese)
(王戟, 詹乃军, 冯新宇, 等. 形式化方法概貌[J]. 软件学报, 2019, 30(1): 33−61)

[18] Xue Jinyun. A unified approach for developing efficient algorithmic programs[J]. Journal of computer Science and Technology, 1997, 12(4): 314−329

[19] Durán J E. Transformational derivation of greedy network algorithms from descriptive specifications[C]// Proc of the 6th Int Conf on Mathematics of Program Construction. Beilin: Springer, 2002: 40−67

[20] Abrial J R, Cansell D, Méry D. Formal derivation of spanning trees algorithms[C]// Proc of the 3rd Int Conf of B and Z Users. Berlin: Springer, 2003: 457−476

[21] Almeida J B, Pinto J S. Deriving sorting algorithms[J]. arXiv preprint, arXiv: 0802.3881, 2008

[22] Nedunuri S, Cook W R. Synthesis of fast programs for maximum segment sum problems[J]. ACM SIGPLAN Notices, 2009, 45(2): 117−126

[23] Mu S C. Calculating a backtracking algorithm: An exercise in monadic program derivation[J]. arXiv preprint, arXiv: 2101.09409, 2021

[24] Lammich P. Generating verified LLVM from Isabelle/HOL[C]// Proc of the 10th Int Conf on Interactive Theorem Proving (ITP 2019). Berlin: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019, 22: 1−22: 19

[25] Zhang Jian, Li Yi, Peng Xin, et al. Inductive SQL synthesis with positive and negative tuples[J]. Journal of Software, 2023, 34(9): 4132−4152 (in Chinese)
(张健, 李弋, 彭鑫, 等. 正反例归纳合成 SQL 查询程序[J]. 软件学报, 2023, 34(9): 4132−4152)

[26] Shi Haihe, Xue Jinyun. PAR-based formal development of algorithms[J]. Chinese Journal of Computers, 2009, 32(5): 982−991 (in Chinese)
(石海鹤, 薛锦云. 基于 PAR 的算法形式化开发[J]. 计算机学报, 2009, 32(5): 982−991)

[27] Sun Lingyu, Xue Jinyun. Formal derivation of the minimum spanning tree algorithm with PAR method[J]. Computer Engineering, 2006, 32(21): 85−87 (in Chinese)
(孙凌宇, 薛锦云. 最小生成树算法的 PAR 方法形式化推导[J]. 计算机工程, 2006, 32(21): 85−87)

[28] You Zhen, Xue Jinyun. Formal verification of algorithmic programs based on the Isabelle theorem prover[J]. Computer Engineering and Science, 2009, 31(10): 85−89 (in Chinese)
(游珍, 薛锦云. 基于 Isabelle 定理证明器算法程序的形式化验证[J]. 计算机工程与科学, 2009, 31(10): 85−89)

[29] Shi Haihe, Zhou Weixing. Design and implementation of pairwise sequence alignment algorithm components based on dynamic programming[J]. Journal of Computer Research and Development, 2019, 56(9): 1907−1917 (in Chinese)
(石海鹤, 周卫星. 基于动态规划的双序列比对算法构件设计与实现[J]. 计算机研究与发展, 2019, 56(9): 1907−1917)

[30] Shi Haipeng, Chen Huan, Yang Qinghong, et al. A method for bio-sequence analysis algorithm development based on the PAR platform[J]. Big Data Mining and Analytics, 2023, 6(1): 11−20

[31] Xiao Cunwei, Shi Haihe, Wang Lan, et al. The construction of de novo sequence assembly algorithm based on hybrid strategy[J]. Journal of Jiangxi Normal University: Natural Science, 2022, 46(3): 300−307 (in Chinese)
(肖存威, 石海鹤, 王岚, 等. 基于混合策略的 de novo 序列拼接算法构造[J]. 江西师范大学学报: 自然科学版, 2022, 46(3): 300−307)

[32] Dijkstra E W. A Discipline of Programming[M]. Princeton, NJ: Prentice-hall, 1976

[33] Floyd R W. Assigning meaning to programs[J]. Communications of the ACM, 1967, 10(8): 365−371

[34] Hoare C A R. An axiomatic basis for computer programming[J]. Communications of the ACM, 1969, 12(10): 576−580

[35] Dijkstra E W, Scholten C S. Predicate Calculus and Program Semantics[M]. Berlin: Springer, 2012

[36] Shi Haihe. Apla-Java automatic program transformation system supporting generic programming [D]. Nanchang: Jiangxi Normal University, 2004 (in Chinese)
(石海鹤. 支持泛型程序设计的 Apla-Java 自动程序转换系统[D]. 南昌: 江西师范大学, 2004)

[37] Lai Yong. Development of APLA to C++ automatic program conversion system [D]. Nanchang: Jiangxi Normal University, 2002 (in Chinese)
(赖勇. APLA 到 C++自动程序转换系统的研制[D]. 南昌: 江西师范大学, 2002)

[38] Xue Jinyun. Two new strategies for developing loop invariants and their applications[J]. Journal of Computer Science and Technology, 1993, 8(2): 147−154

Shi Haihe, born in 1979. PhD, professor, PhD supervisor. Senior member of CCF. Her main research interests include formal method, bioinformatics, and software engineering.
石海鹤, 1979 年生. 博士, 教授, 博士生导师. CCF 高级会员. 主要研究方向为形式化方法、生物信息学、软件工程.

Lan Sunwen, born in 1999. Master. Student member of CCF. His main research interest includes formal method.
蓝孙文, 1999 年生. 硕士. CCF 学生会员. 主要研究方向为形式化方法.

Liu Riming, born in 1998. Master. Student member of CCF. His main research interest includes formal method.

刘日明, 1998 年生. 硕士. CCF 学生会员. 主要研究方向为形式化方法.

Shi Haipeng, born in 1977. PhD, associate professor, master supervisor. Member of CCF. Her main research interests include algorithm design and software engineering.

石海鹏, 1977 年生. 博士, 副教授, 硕士生导师. CCF 会员. 主要研究方向为算法设计、软件工程.

Wang Lan, born in 1974. Master, associate professor. Member of CCF. Her main research interests include algorithms and data structures, bioinformatics, and principles of compliers.

王　岚, 1974 年生. 硕士, 副教授. CCF 会员. 主要研究方向为算法和数据结构、生物信息学、编译原理.

Zhong Linhui, born in 1974. PhD, associate professor, master supervisor. Member of CCF. His main research interests include software architecture, component-based software development, and software evolution and maintenance.

钟林辉, 1974 年生. 博士, 副教授, 硕士生导师. CCF 会员. 主要研究方向为软件体系结构、构件化软件开发、软件演化与维护.