ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20221025164232
Original scientific paper

Correctness by Construction for Pairwise Sequence Alignment Algorithm in Bio-Sequence

Haihe SHI, Sunwen LAN, Riming LIU, Haipeng SHI*

Abstract: Pairwise sequence alignment is a classical problem in bioinformatics, aiming at finding the similarity between two sequences, which is important for discovering functional, structural and evolutionary information in biological sequences. More algorithms have been developed for the sequence alignment problem. There is no formal development process for the existing pairwise sequence algorithms and leads to the low trustworthiness of those algorithms. In addition, the application of formal methods in the field of bioinformatics algorithm development is rarely seen. In this paper, we use a formal method PAR to construct a pairwise sequence algorithm, analyze the essence of the pairwise sequence alignment problem, construct the Apla algorithm program by stepwise refinement, and further verify its correctness. Finally a highly reliable and executable pairwise sequence alignment algorithm program is generated from Apla program via PAR platform. The formal construction process ensures the reliability of algorithm, and also demonstrates the algorithm design idea clearly, which makes the originally difficult algorithm design process easier. The successful practice of this method on the pairwise sequence alignment problem in biological sequence analysis can provide a reference for the construction of highly reliable algorithms in complex bioinformatics from both methodological and practical aspects.

Keywords: automatic verification; correctness by construction; pairwise sequence alignment; PAR method

1 INTRODUCTION

The biological macromolecular sequences studied in current biological analysis usually evolved from ancient ancestral sequences. If two related sequences evolved from the same ancestral sequence, then they are said to have homology, and one of the main bases for judging the homology between the two is similarity [1]. Sequence similarity is usually assessed by the character differences between sequences, where an effective measure to analyze whether sequences have sufficient similarity to each other is sequence alignment. Pairwise sequence alignment aims to find the similarity relationship between two sequences and is a classical problem in the application of computers to solve biological problems. The most classical deterministic pairwise sequence alignment algorithms are the NW (Needleman-Wunsch) algorithm [2] based on global matching and the SW (Smith-Waterman) algorithm [3] based on local matching. Subsequently, a series of algorithms based on the above algorithms for parallel optimization [4-7], space complexity optimization [8-9], and backtracking strategy improvement [10] have been developed in domestic and international research. In addition, Garai et al. [11] proposed a new sequence alignment technique based on genetic algorithms that can achieve global or near-global optimal solution capability and obtained good performance; Rashed et al. [12] used FPGAs and custom convolutional neural networks to accelerate pairwise sequence alignment of DNA, which can achieve 98.3% accuracy; Song et al. [13] defined an environment and agent for implementing reinforcement learning in sequence alignment systems, and proposed a new method for pairwise sequence alignment that uses deep reinforcement learning to improve older pairwise sequence alignment algorithms.

The main focus of the above work is on the optimization of the pairwise sequence alignment algorithm, but the research on the reliability of the algorithm is missing, and the correctness of the algorithm cannot be guaranteed. The purpose of this paper is to propose an algorithm design method that can be applied to biological sequence analysis, using formal techniques to ensure the correctness of the algorithm, so as to solve the above-mentioned problems and finally generate an executable and highly reliable pairwise sequence alignment algorithm program. The formal method is based on a rigorous mathematical and mechanistic approach to statute, design, construct, verify, and evolve computational systems, and is an important method for improving and ensuring the quality of computational systems, which can effectively improve and guarantee the credibility of the systems. In recent years, many famous research institutions have invested a lot of human and material resources in the research of this area. For example, SpaceOS, an aerospace operating system independently developed by the Fifth Academy of China Aerospace Science and Technology Corporation, and HarmonyOS, Hongmeng operating system independently developed by Huawei, both use formal methods. Formal development is mainly to construct and prove the equivalence transformation and refinement relationship between formal statutes, and to develop a system that meets the needs by progressively refining the formal model of the system as a guide, which is also called correctness by construction [14]. The PAR method [15] provides the Radl language to write the statute and describe the algorithm, designs the rules for the statute transformation, and supports formal development. The formal derivation of algorithmic programs using the PAR method not only ensures the correctness of algorithmic programs logically, but also reveals the essential features of algorithmic program design.

In this paper, we use the PAR method to first analyze the dual sequence matching problem in depth, and then develop the recurrence relations and loop invariants for the solution of the problem by dividing the subproblem and formalizing the transformation of the statute. In order to ensure the reliability of the constructed pairwise sequence alignment algorithm, this paper formally proves the correctness of the Apla program, and then generates the executable program using the code generation system of PAR platform. In the verification process, it is tedious and error-prone to verify the correctness of the program manually using Dijkstra's weakest pre-predicate method. Therefore, in this paper, the Isabelle theorem provers are

used instead of manual verification to overcome the drawbacks of manual verification and effectively improve the reliability of the constructed algorithm. In addition, the Apla program is easy to prove formally, and the code generation system of the executable program in PAR platform can guarantee the semantic consistency from the Apla program to the executable program. By verifying the correctness of the Apla program first and then generating the executable program from the Apla program according to the code generation system of the executable program in PAR platform, the verification work can be easier and the reliability of the algorithm can be guaranteed. In this paper, a classical algorithm in bioinformatics is used as an example to show the process of constructing the algorithm, and finally a highly reliable pairwise sequence alignment algorithm is generated. The work carried out in this paper can provide a valuable reference framework for the formal construction of complex algorithms in the field of bioinformatics.

2 RELATED WORK

The formal development method based on the formal specification language has been developed so far, and many methods have been formed, such as VDM [18], Z [19], Event-B [23] and Designware [24]. The theoretical basis of VDM is first-order logic and abstract data types, support data specificity and operation decomposition, etc. Similar to VDM, Z is also a specification language with refinement as the core, but neither support object-oriented software specification and refinement. To solve this problem, extended from these two methods to VDM++[20] and Object-Z [21]. B [22] is a formal description language, which can also be used for formal proof. It is developed on the basis of VDM and Z methods. It is a relatively mature method. Event-B is a method that is refined and expanded based on method B. It has been used to develop many practical projects, and it has certain advantages in the development of large-scale projects. But Event-B uses abstract machine to describe the software development process, and it is too complicated to use it to develop algorithms. Designware is a system developed by professor DR Smith of the Kestrel Institute in the United States, which can realize the transformation development from the specification to the algorithm, but the requirements for users are too high.

In domestic and foreign research, there have been many works on formalized development of algorithms. For example, Bird et al. [25] used calculus of lists in the derivation process, and finally synthesized a very short functional program with a functional version of the Knuth-MorrisPratt algorithm for pattern matching; JE Durán [26] proposed a transformation development method for efficient imperative network algorithms based on the Mller algebra of formal languages and derived the shortest path tree algorithm based on this method; Abrial et al. [27] designed and implemented the minimum spanning tree and Prim problem models using the Event-B method and transformed them into the corresponding solution algorithms; Almeida et al. [28] demonstrate three known sorting algorithms derived from a similar sequence of transformation steps from a general specification; Nedunuri et al. [29] transformed a naive algorithm into an efficient program by applying appropriate semantic-preserving transformations, and deduced three maximum segment sum problems; Mu et al. [30] construct a problem specification using equality reasoning the backtracking algorithm of *n*-queens. These existing works are mainly applied to general areas in combinatorics, but formal methods have not been applied to develop algorithms in the field of bioinformatics.

The PAR method based on partition and recursion technology covers all stages of algorithm development, and can clearly show the design process of the algorithm, which is very suitable for the construction of the algorithm, that is, the correct development. Before this, the works of literature [31, 32] used the PAR method to formally develop a series of algorithms for combinatorial mathematics problems, illustrating the feasibility of using the PAR method to develop related algorithms. Biological sequences and combinatorial mathematics are inextricably linked. Biological sequences include DNA, RNA, and biological macromolecular sequences. These can be regarded as sequence structures in combinatorial mathematics, so this paper applies the PAR method to formally develop algorithms in the field of bioinformatics.

3 FORMAL CONSTRUCTION METHOD OF PAIRWISE SEQUENCE MATCHING ALGORITHM

In this section, we first introduce the formal development method PAR used in this paper, and then use the PAR method to derive general recurrence relations for various penalty rules from the problem statute. In addition, the Apla program for the pairwise sequence alignment algorithm is constructed using a specific penalty rule as an example.

3.1 Overview of the PAR Approach

The PAR method is a simple and practical formal method for designing algorithms, which covers a variety of known algorithm design techniques and supports the whole process of algorithm program development [16]. The strategy of the PAR method to develop an algorithm is shown in Fig. 1.

The specification in this paper adopts a unified quantifier representation $(Q\ i : r(i) : f(i))$, which means "the value obtained by performing the q operation corresponding to the quantifier Q on the function $f(i)$ over the range $r(i)$". The quantifiers used in this article are MAX and Σ, and the corresponding operations are max and +, which represent the maximum value and the sum, respectively. Using the properties of quantifiers, the specification can be equivalently transformed to expose the problem The idea of solving [15]. The properties used in this paper for specification refinement are given below:

(1) Single range: $(Q\ i : i = k : f(i)) = f(k)$

(2) Range splitting: $(Q\ i : r(i) : f(i)) = (Q\ i : r(i) \wedge \vee b(i) : f(i))\ q\ (Q\ i : r(i) \wedge \neg\ b(i) : f(i))$

(3) Associative law of function: $(Q\ i : r(i) : f(i)\ q\ g(i)) = (Q\ i : r(i) : f(i))\ q\ (Q\ i : r(i) : g(i))$

(4) Disjunctive range: if $aqa = a$, the operation q is said to be idempotent. For the idempotent operation q, there is $(Q\ i : r(i) \vee s(i) : f(i)) = (\ Q\ i: r(i) : f(i))\ q\ (Q\ i : s(i) : f(i))$

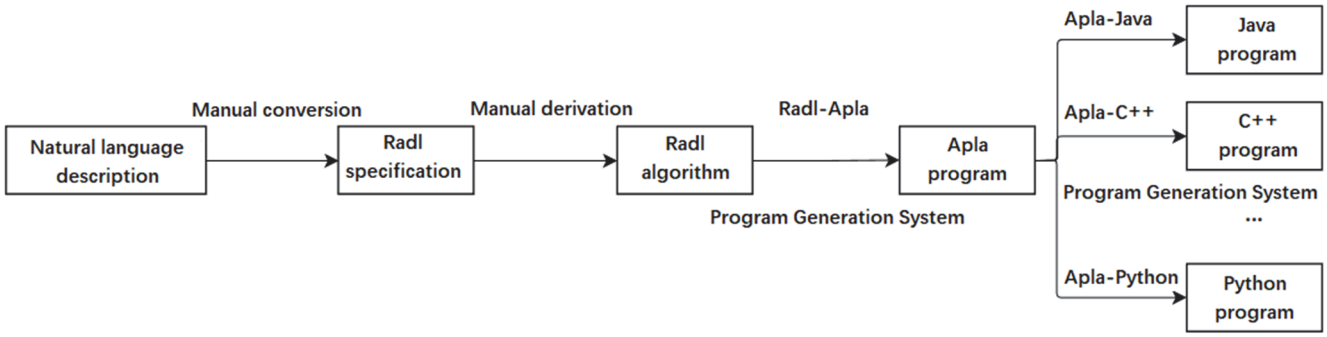

Figure 1 Strategies for constructing algorithms using the PAR method

3.2 Formal Construction of Pairwise Sequence Alignment Algorithm

The natural language description of the pairwise sequence alignment problem is as follows: Given two sequences seq1 and seq2 of length m and n, respectively, elements in the sequence can be matched in three ways - match, nmatch, and space, respectively, indicates that two identical elements match, two different elements match, and one of the elements in the sequence matches a gap. Each matching method corresponds to a penalty, and there are many kinds of penalty rules; the most commonly used are two - for constant weight model and affine penalty model, different penalty rules correspond to different sequence similarity evaluation criteria. To better demonstrate the construction process, this paper uses the simplest penalty rules, as shown below:

3.2.1 Problem Specification

The natural language description of the pairwise sequence alignment problem is as follows: Given two sequences seq1 and seq2 of length m and n, respectively, elements in the sequence can be matched in three ways - match, nmatch, and space, respectively, indicates that two identical elements match, two different elements match, and one of the elements in the sequence matches a gap. Each matching method corresponds to a penalty, and there are many kinds of penalty rules, the most commonly used are two - for constant weight model and affine penalty model, different penalty rules correspond to different sequence similarity evaluation criteria. To better demonstrate the construction process, this paper uses the simplest penalty rules, as shown below:

$$\text{sc\_rule}\begin{cases}\text{match}=1\\ \text{nmatch}=-1\\ \text{space}=-2\end{cases} \tag{1}$$

Starting from the first element of the two sequences and matching from front to back, until all the elements of the two sequences are matched, it is an alignment, and the alignment score is the sum of the corresponding penalty points for each element matching. The goal is to obtain the largest alignment score from all possible alignments, thereby assessing the similarity between sequences.

A coordinate in the form of (i, j) is used to represent the current matching position, i represents the matched length of seq1, and j represents the matched length of seq2. A quaternion sequence is used to represent an element match, $[i, j, i+1, j+1]$ indicates that the $(i+1)$-th element of seq1 matches or mismatches the $(j+1)$-th element of seq2, $[i, j, i+1, j]$ and $[i, j, i, j+1]$ means that the $(i+1)$-th element of seq1 and the $(j+1)$-th element of seq2 match the gap respectively. Then an alignment of two sequences of length i and j can be regarded as starting from the coordinate (0, 0) to coordinate (i, j) continuous and non-repeated set of each element match, where two matches are consecutive means that the coordinates after one match are the coordinates before the next match, and non-repetition refers to the coordinates before each match no repetition, use align(i, j) to represent an alignment that satisfies this condition. It is not difficult to draw from the above definition, align(i, j) is in the form of $\{[0, 0, i_1, j_1], [i_1, j_1, i_2, j_2], [i_2, j_2, i_3, j_3], \ldots, [i_{k-1}, j_{k-1}, i_k, j_k], [i_k, j_k, i, j]\}$ set of quaternary sequences, each element in the set is a match. Finally, the Radl specification of the pairwise sequence alignment problem can be obtained:

|[in: m, n: integer; seq1$[0 : m-1]$, seq2$[0 : n-1]$:array of char out: ms: integer]|

AQ: Given two sequences seq1$[0 : m-1]$, seq2$[0: n-1]$, $m > 1$, $n > 1$;

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

where rule$([i1, j1, i2, j2])$ can be defined as follows according to the penalty rules used in this paper:

$$\text{rule}\left([i1, j1, i2, j2]\right) = \begin{cases}\text{match}, & i2 = i1+1 \wedge j2 = j1+1 \wedge \text{seq1}[i2] = \text{seq2}[j2]\\ \text{nmatch}, & i2 = i1+1 \wedge j2 = j1+1 \wedge \text{seq1}[i2] \neq \text{seq2}[j2]\\ \text{space}, & (i2 = i1+1 \wedge j2 = j1) \vee (j2 = j1+1 \wedge i2 = i1)\end{cases} \tag{2}$$

3.2.2 Search for Recursion Relation

According to the definition of align(i, j), it can be concluded that the last element of align(i, j) is $[i-1, j, i, j]$ or $[i, j-1, i, j]$ or $[i-1, j-1, i, j]$, which means align(i, j) = align$(i-1, j) \cup \{[i-1, j, i, j]\} \vee$ align$(i, j-1) \cup \{[i, j-1, i, j]\} \vee$ align$(i-1, j-1) \cup \{[i-1, j-1, i, j]\}$. Use score$(cs)$ instead of $(\Sigma sgl : sgl \in cs : \text{rule}(sgl))$, consider the general case:

maxScore(i, j)

= (MAX cs : cs = align(i, j) : score(cs))

= (MAX cs : (cs = align$(i-1, j) \cup \{[i-1, j, i, j]\}) \vee$ (cs = align$(i, j-1) \cup \{[i, j-1, i, j]\}) \vee (cs$ = align$(i-1, j-1) \cup \{[i-1, j-1, i, j]\}$): score$(cs)$)

[disjunctive range]

= max((MAX cs : cs = align($i-1, j$) ∪ {[$i-1, j, i, j$]} : score(cs)), max((MAX cs : cs = align($i, j-1$) ∪ {[$i, j-1, i, j$]} : score(cs)), (MAX cs : cs = align($i-1, j-1$) ∪ {[$i-1, j-1, i, j$]} : score(cs))))

= max((MAX cs : cs = align($i-1, j$) : score(cs ∪ {[$i-1, j, i, j$]})), max((MAX cs : cs = align($i, j-1$) : score(cs ∪ {[$i, j-1, i, j$]})), (MAX cs : cs = align($i-1, j-1$) : score(cs ∪ {[$i-1, j-1, i, j$]}))))

Analyze the following cases, where end is the last element match of the sequence alignment:

score(case ∪ {end})

= (Σsgl : sgl ∈ (case ∪ {end}) : rule(sgl))

[range splitting]

= (Σsg : sgl ∈ (case ∪ {end}) ∧ sgl ≠ end : rule(sgl)) + (Σsg : sgl ∈ (case ∪ {end}) ∧ sgl = end : rule(sgl))

= (Σsg : sgl ∈ case : rule(sgl)) + (Σsg : sgl = end : rule(sgl))

[single range]

= score(case) + rule(end)

On this basis, the following derivations can be made:

maxScore(i, j)

= max((MAX cs : cs = align($i-1, j$) : score(cs) + rule([$i-1, j, i, j$])), max((MAX cs : cs = align($i, j-1$) : score(cs) + rule([$i, j-1, i, j$])), (MAX cs : cs = align($i-1, j-1$) : score(cs) + rule([$i-1, j-1, i, j$]))))

[associative law of function]

= max((MAX cs : cs = align($i-1, j$) : score(cs)) + rule([$i-1, j, i, j$]), max((MAX cs : cs = align($i, j-1$) : score(cs)) + rule([$i, j-1, i, j$]), (MAX cs : cs = align($i-1, j-1$) : score(cs)) + rule([$i-1, j-1, i, j$])))

= max(maxScore($i-1, j$) + rule([$i-1, j, i, j$]), max(maxScore($i, j-1$) + rule([$i, j-1, i, j$]), maxScore($i-1, j-1$) + rule([$i-1, j-1, i, j$])))

It is not difficult to see that the original problem is divided into three sub-problems. From this, the general recurrence relation of the general case of the pairwise sequence alignment problem is obtained:

$$\max \text{Score}(i,j) = \\ = \max \begin{cases} \text{maxScore}(i-1,j) + \text{rule}([i-1,j,i,j]) \\ \text{maxScore}(i,j-1) + \text{rule}([i,j-1,i,j]) \\ \text{maxScore}(i-1,j-1) + \text{rule}([i-1,j-1,i,j]) \end{cases} \quad (3)$$

Different recursion relationships can be obtained by designing different rule penalty rules, so as to obtain pairwise sequence alignment algorithms with different sequence similarity evaluation criteria. In the penalty rules used in this paper, rule([$i-1, j, i, j$]) and rule([$i, j-1, i, j$]) are always equal to space, and the value of rule([$i-1, j-1, i, j$]) is only related to the value of seq1[i] and seq2[j], so the definition of rule can be simplified as follows:

$$\text{rule}(i,j) = \begin{cases} \text{match}, \text{seq1}[i] = \text{seq2}[j] \\ \text{nmatch}, \text{seq1}[i] \neq \text{seq2}[j] \end{cases} \quad (4)$$

Corresponding to the penalty rules, the general recurrence relation can be expressed as follows:

$$\text{maxScore}(i,j) = \\ = \max \begin{cases} \text{maxScore}(i-1,j) + \text{space} \\ \text{maxScore}(i,j-1) + \text{space} \\ \text{maxScore}(i-1,j-1) + \text{rule}(i,j) \end{cases} \quad (5)$$

Considering the special value, when i and j are both 0, it means that two empty sequences are aligned, obviously maxScore(0, 0) = 0; when one of i and j is 0 and the other is not 0, it means a sequence that is not empty is aligned with another empty sequence, there is only one case, that is, every element in the sequence which is not empty matches the gap, maxScore(i, 0) = space × i, maxScore(0, j) = space × j. From this, the recurrence relation can be obtained:

$$\max \text{Score}(i,j) = \begin{cases} 0, i = 0 \wedge j = 0 \\ \text{space} \times i, i \neq 0 \wedge j = 0 \\ \text{space} \times j, i = 0 \wedge j \neq 0 \\ \text{Eq.}(5), i \neq 0 \wedge j \neq 0 \end{cases} \quad (6)$$

According to the recurrence relation of the specified specific penalty rules, the following Radl algorithm is constructed to solve the pairwise sequence alignment problem:

ALGORITHM: pairwiseAlignment
|[Var score: array(0..m, array(0..n, integer)); seq1: array(0..m, char); seq2: array(0..n, char); i, j: integer;]|
{$AQ \wedge AR$}
BEGIN: i = 0 ++1 ∧ j = 0 ++1;
TERMINATION: $i = m \wedge j = n$;
RECUR: Eq. (6);
END.

3.2.3 Develop Cycle Invariants

Reference [33] gives a new definition of loop invariant, that is, the predicate that reflects the variation law of all loop variables in the loop body and is true before and after the loop body is executed is called the loop invariant of the loop body. Among them, the loop variable is defined as the variable in the loop body whose value changes continuously as the loop body executes.

From the obtained Radl algorithm, it is not difficult to see that the algorithm has two layers of loops, the variable i controls the outer loop, the variable j controls the inner loop, and each loop corresponds to a different loop invariant. Use ms to store maxScore(i, j) value, then all the variables in the loop are i, j and ms. First analyze the outer loop, the variables that change with the execution of the loop body are i and ms, and the variable j has nothing to do with the outer loop, because no matter the outer loop executes at which step, the value of the variable j before the outer loop body is executed is 0. The value of the loop variable ms is always (MAX cs : cs = align(i,j) : score(cs)), and the range of the loop variable i is [0, m]. According to the change rule of the loop variable, it can be accurately

expressed by the predicate, that is, the loop invariant of the outer loop is obtained:

ms = maxScore(i, j) = (MAX cs : cs = align(i, j) : score(cs)) $\wedge$ ($0 \le i \le m$)

Then again, by analogy with the outer loop, the loop invariant of the inner loop can be obtained:

ms = maxScore(i, j) = (MAX cs : cs = align(i, j) : score(cs)) $\wedge$ ($0 \le j \le n$).

3.2.4 Synthesis Algorithm Program

According to the obtained recursion relationship, Radl algorithm, and loop invariant, the Apla abstract algorithm program can be easily obtained. Due to space limitations, only the Apla codes of the three main functions are given here. The main codes of the comparison algorithm are as follows:

```
function rule(var i, j : integer) : integer;
begin
   if seq1[i] = seq2[j] → rule := match;
   [] seq1[i] ≠ seq2[j] → rule := nmatch;
   fi;
end;
function matching(var i, j : integer) : integer;
begin
   if (i = 0) ∧ (j = 0) → matching := 0;
   [] (i = 0) ∧ (j ≠ 0) → matching := space * j;
   [] (j = 0) ∧ (i ≠ 0) → matching := space * i;
   [] (i ≠ 0) ∧ (j ≠ 0) →
      matching := max(score[i − 1, j] + space,
max(score[i, j − 1]+space, score[i − 1, j − 1] + rule(i −
1, j − 1)));
fi;
end;
procedure maxScore();
var
   i, j : integer;
begin
   i := 0;
   do i ≤ m →
      j := 0;
      do j ≤ n →
         score[i, j] := matching(i, j);
         j := j + 1;
      od;
      i := i + 1;
   od;
end;
```

The maximum alignment score can be calculated by calling the maxScore procedure, and the maximum alignment score is score [m, n].

4 AUTOMATIC VERIFICATION AND GENERATION OF PAIRWISE SEQUENCE ALIGNMENT ALGORITHM

Based on the previously obtained Apla algorithm for pairwise sequence alignment, this section verifies it formally. After verifying its correctness, Apla is then generated into an executable program using the program generation system of PAR platform.

4.1 Automated Verification of Pairwise Sequence Alignment Algorithms

Constructing a correct program requires mathematical proof [34], because program testing can effectively find the existence of bugs, but it cannot show that there are no bugs in the program. Therefore, to ensure that the above program is completely correct, it needs to be formally verified. In addition, through the program's correctness proof, the understanding of the program can also be deepened. In this paper, Dijkstra's weakest pre-predicate method is used, and the Isabelle theorem prover is used to prove the correctness of the obtained Apla program.

4.1.1 Automated Verification Strategy

Based on the obtained recurrence relation and loop invariant, the correctness of the Apla abstract algorithm program can be verified. There are many ways to prove the correctness of the program, because the Floyd inductive assertion method [36] and the Hoare axiom system method [37] are very difficult to write appropriate intermediate assertions, and Dijkstra's weakest pre-predicate method is defined by the weakest pre-predicate WP to write intermediate assertions [34, 35], which is more practical. Use S for statements and R for post-assertion of S. Then the execution of S starts from any state and must terminate within a finite time, and the termination state satisfies R. The predicate WP("S", R) represents the set of these states and is called the weakest pre-predicate. Therefore, this paper chooses Dijkstra's weakest pre-predicate method to prove the correctness of the Apla program. Dijkstra's weakest pre-predicate method to prove the correctness of the loop statement needs to meet five conditions:

(1) Q => ρ;
(2) $\rho \wedge C_i$ => WP("S_i", ρ), $1 \le i \le n$;
(3) $\rho \wedge \neg$ Guard => R;
(4) $\rho \wedge$ Guard => $\tau > 0$;
(5) $\rho \wedge C_i$ => WP("$\tau_1 := \tau$; S_i", $\tau_1 < \tau$), $1 \le i \le n$.

Among them, Q is the pre-predicate, R is the post-predicate, ρ is the loop invariant, τ is the bound function, C_i and S_i are the loop conditional branch statement and its corresponding loop body respectively, Guard is the disjunction of all branches C_i.

Input the conditions of the proving program into Isabelle in Isabelle's grammar and prove it using the automatic proof strategy it provides.

4.1.2 Verification Process

Create a new theory seqalign for the formal proof of the Apla program in this article. The theory name needs to be consistent with the file name. Since the Main theory contains the list type used in this article, it is only necessary to use import to import the Main parent theory. The recurrence relation for maxScore and the rule function used are translated into the language used in Isabelle, as shown in Fig. 2.

The main body of the pairwise sequence alignment algorithm program constructed in this paper is a double-layer loop. The recurrence relation and penalty rules in the loop are strictly derived, so it is only necessary to prove the correctness of the double-layer loop. To prove the five

theorems of the loop statement in Dijkstra's weakest pre-predicate method, finding the loop invariant ρ and the bound function τ of the loop statement is the key. The loop invariant in the Apla program has been solved in Section 2. In addition, it is obvious that the bound functions corresponding to the double-layer loop can be $m-i+1$ and $n-j+1$ respectively. The values of the parts of the theorem are substituted into them, expressed in the syntax of Isabelle, and entered into the created seqalign theory, and the form of the five theorems proving the outer loop in Isabelle is given below, as in Fig. 3.

```
datatype base = A | C | T | G | Gap

definition rule :: "base list ⇒ base list ⇒ nat ⇒ nat ⇒ int" where
"rule seq1 seq2 i j = (if (seq1! i) = (seq2! j) then 1 else -1)"

fun maxScore :: "base list ⇒ base list ⇒ nat ⇒ nat ⇒ int" where
"maxScore seq1 seq2 0 n = (-2) * int n" |
"maxScore seq1 seq2 m 0 = (-2) * int m" |
"maxScore seq1 seq2 (Suc m) (Suc n) = max (max ((maxScore seq1 seq2 m (Suc n))-2)
                                    ((maxScore seq1 seq2 (Suc m) n)-2))
                                   ((maxScore seq1 seq2 m n)+(rule seq1 seq2 m n))"
```

Figure 2 Recursive relationships and penalty rules proof

```
lemma wp1_1 : "i = 0 ∧ j = 0 ∧ ms = 0 ⟹ ms = (maxScore seq1 seq2 i j) ∧ (0≤i ∧ i≤m)"
  apply auto
  done

lemma wp1_2 : "ms = (maxScore seq1 seq2 i j) ∧ (0≤i ∧ i≤m) ∧ i<m ⟹
(maxScore seq1 seq2 (Suc i) j) = (maxScore seq1 seq2 (Suc i) j) ∧ (0≤(Suc i) ∧ (Suc i)≤Suc m)"
  apply auto
  done

lemma wp1_3 : "ms = (maxScore seq1 seq2 i j) ∧ (0≤i ∧ i≤m) ∧ ¬i<m ⟹
ms = (maxScore seq1 seq2 m j)"
  apply auto
  done

lemma wp1_4 : "ms = (maxScore seq1 seq2 i j) ∧ (0≤i ∧ i≤m) ∧ i<m ⟹ 0<m-i+1"
  apply auto
  done

lemma wp1_5 : "ms = (maxScore seq1 seq2 i j) ∧ (0≤i ∧ i≤m) ∧ i<m ⟹ m-i<m-i+1"
  apply auto
  done
```

Figure 3 Proof theorems for outer loops

The above five lemmas can be automatically verified directly by the auto strategy. Add *apply auto* after each lemma, and the proof state as shown in Fig. 4 has all changed to "No subgoals!", indicating that the theorem is proved to be correct. Finally, end with done the proof of each lemma. Since the proof of the inner loop is the same as the proof of the outer loop, there is no proof statement for the inner loop. The above proof shows that the loop invariant is before and after the loop execution and the loop body. All are true, and the Apla program finally generated in this article is indeed correct.

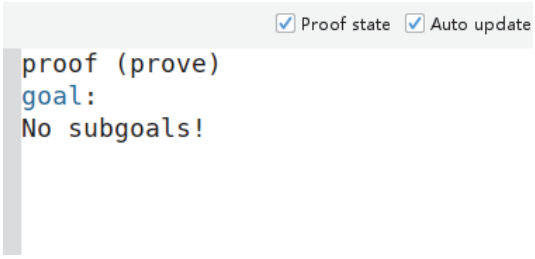

Figure 4 Proof state

4.2 Executable Program Generation

The pairwise sequence alignment algorithm is formally constructed and its correctness is verified. Based on the previous algorithm, we show the generation effect of the algorithm through the Apla -> C++ program generation system on the PAR platform.

To facilitate the viewing of the best alignment method, based on the Apla program of the aforementioned pairwise sequence alignment algorithm, we store the matching of each step of the best alignment method and add the process of outputting the best alignment method. When calculating the maximum alignment score for coordinates (i, j), the coordinates before the last match are determined. Therefore, if you want to get the best alignment, you only need to save the last match when calculating the maximum score for each coordinate. Here, a two-dimensional array path is used to save it. The value range of the elements in the array is {0, 1, 2, 3}. The above four values for path [i, j] represent the initial state of the comparison, seq2 [j] and Gap matching, seq1[i] and gap matching, and seq1[i] and seq2[j] match or mismatch. Therefore, the matching function can be changed as follows:

```
function matching(var i, j : integer) : integer;
var
   s1, s2, s3 : integer;
begin
   if (i = 0) ∧ (j = 0) → path[i, j] := 0; matching := 0;
   [] (i ≠ 0) ∧ (j = 0) → path[i, j] := 1; matching :=
space * i;
   [] (i = 0) ∧ (j ≠ 0) → path[i, j] := 2; matching :=
space * j;
   [] (i ≠ 0) ∧ (j ≠ 0) →
      s1, s2, s3 := score[i − 1, j] +s pace, score[i, j − 1]
+ space, score[i − 1, j − 1] + rule(i − 1, j − 1);
      if s1 ≤ s2 →
         if s2 ≤ s3 → path[i, j] := 3; matching := s3;
         [] s3 < s2 → path[i, j] := 2; matching := s2;
         fi;
      [] s2 < s1→
         if s1 ≤ s3 → path[i, j] := 3; matching := s3;
         [] s3 < s1 → path[i, j] := 1; matching := s1;
         fi;
      fi;
   fi;
end;
```

After determining the value of the path array, traverse forward from the coordinates (m, n) until the initial state of the comparison, and store each matching method into the stack, and finally the best alignment method can be obtained by outputting the top elements of the stack in turn. Due to space limitations, the code of the output process printPath is omitted here.

After ensuring the correctness of the Apla program for the pairwise sequence alignment algorithm, file input and file output statements are added on top of it. Using the code generation system in PAR platform [38, 39], the abstract Apla program can be generated into Java, C++, Python and other executable programs. The C++ code of the generated pairwise sequence alignment algorithm for the rule and matching functions is shown in Fig. 6.

```
For the following two sequences:
(22)CGGACCTCAGGCCCCAGGTGTC
(20)CGGACCACTGGCTCAAGAGT

The max score is : 6

The pairwise sequence alignment process is as follows(seq1-seq2):
CGGACCTCAGGCCCCAGGTGTC
CGGACCACTGG#CTCAAGAGT#
```

Figure 5 Pairwise sequence alignment algorithm C++ program running result

The two sequences involved in the matching are read from the text, and then the results of the pairwise sequence alignment are output to the text file where, "#" means empty space. For the sake of space, only a simple example of the C++ program is given here, as shown in Fig. 5.

```
int rule(const int &i,const int &j) {
    int ReturnValue;

    if (seq1[i] == seq2[j]) {
        ReturnValue = match;
    }
    else if (seq1[i] != seq2[j]) {
        ReturnValue = nmatch;
    }
    return (ReturnValue);
}

int matching(int &i,int &j) {
    int s1, s2, s3;
    int ReturnValue;

    if ((i == 0) && (j == 0)) {
        path[i][j] = 0;
        ReturnValue = 0;
    }
    else if ((i == 0) && (j != 0)) {
        path[i][j] = 2;
        ReturnValue =  - 2 * j;
    }
    else if ((j == 0) && (i != 0)) {
        path[i][j] = 1;
        ReturnValue =  - 2 * i;
    }
    else if ((i != 0) && (j != 0)) {
        s1 = score[i - 1][j] + space;
        s2 = score[i][j - 1] + space;
        s3 = score[i - 1][j - 1] + rule(i - 1, j - 1);
        if (s1 <= s2) {
            if (s2 <= s3) {
                path[i][j] = 3;
                ReturnValue = s3;
            }
            else if (s3 < s2) {
                path[i][j] = 2;
                ReturnValue = s2;
            }
        }
        else if (s2 < s1) {
            if (s1 <= s3) {
                path[i][j] = 3;
                ReturnValue = s3;
            }
            else if (s3 < s1) {
                path[i][j] = 1;
                ReturnValue = s1;
            }
        }
    }
    return (ReturnValue);
}
```

Figure 6 Generated C++ code for the rule and matching functions

5 CONCLUSION

In this paper, based on the PAR method, we adopt the formal method to obtain the recurrence relation for the general case from the Radl formal specification of the pairwise sequence alignment problem in biological sequence analysis. The special case is analyzed, and the Apla algorithm is obtained by adding a recursive condition to the recursive formula. Further, the correctness of the pairwise sequence alignment algorithm generated in this paper is ensured by the verification of theorem prover Isabelle. Finally, the C++ program generation system of PAR platform is used to generate the executable code from the Apla program.

The main contributions of this paper are as follows:

(1) This paper uses the PAR method to start from the problem specification, constructs a pairwise sequence alignment algorithm in biological sequence analysis, formally proves the correctness of the constructed algorithm, and finally generates an executable program, which effectively guarantees its high trustworthiness.

(2) This construction method demonstrates the amazing idea of the algorithm, and also shows the clear logic behind this idea. The work in this paper not only shows the design idea of the pairwise sequence alignment algorithm, but also adds special case values to the recurrence relations derived in the general case, which brings great convenience to generate the algorithm program and makes the design of the algorithm easier.

(3) The formal method is applied in the field of bioinformatics. As an instance, a pairwise sequence alignment algorithm is successfully constructed, which illustrates the effectiveness and practicability of the method.

(4) In the field of sequence analysis in bioinformatics, many problems are similar and sequences have many common features, so the work in this paper can provide an example for the formal construction of typical algorithms in the field of complex bioinformatics.

An important idea of the PAR approach is to convert as much creative work as possible into non-creative work, so that the otherwise difficult process of algorithm design becomes a step-by-step task. In this paper, the algorithm is constructed with the help of the PAR platform C++ program generation system and theorem prover Isabelle, which minimizes manual work. The next step is to further investigate methods for generating Apla programs from recursive relations and loop invariants, and to transform more creative work in the construction of algorithms into mechanical work. What's more, we will work on establishing the specification library. By putting some common types and operations into the specification library, the establishment of the specification and the derivation of the recursive relation can be made more convenient.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No.62062039, and the Natural Science Foundation of Jiangxi Province under Grant No. 20212BAB202017.

6 REFERENCES

[1] Wang, Y. X. (2011). *Introduction to Bioinformatics: Algorithms and Applications for High Performance Computing*. Tsinghua University Press.

[2] Needleman, S. B. & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, *48*(3), 443-453.

[3] Smith, T. F. & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*(1), 195-197.

[4] Liu, Y., Bertil, S., & Maskell, D. L. (2010). CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT

and virtualized SIMD abstractions. *BMC Research Notes, 3*(1), 93-93.
[5] Rucci. E., García, C., Botella, G. et al. (2015). Smith-waterman protein search with OpenCL on an FPGA. *TrustCom/BigDataSE/ISPA*, *3*, 208-213.
[6] Li, Y. (2015). *Parallel optimization design and implementation of biological sequence alignment algorithm*. Harbin Institute of Technology
[7] Alawneh, L., Shehab, M. A., Al-Ayyoub, M. et al. (2020). A scalable multiple pairwise protein sequence alignment acceleration using hybrid CPU-GPU approach. *Cluster Computing*, *23*(7). https://doi.org/10.1007/s10586-019-03035-8
[8] Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the Acm*, *18*(6), 341-343.
[9] Ukkonen, E. (1985). *Algorithms for approximate string matching*. Academic Press Professional, Inc.
[10] Tang, Y. R. (2004). An optimized global pairwise sequence alignment algorithm in bioinformatics. *Journal of Computer Applications*, *(S1)*, 307-308.
[11] Garai, G. & Chowdhury, B. (2015). A cascaded pairwise biomolecular sequence alignment technique using evolutionary algorithm. *Information Sciences*, *297*, 118-139. https://doi.org/10.1016/j.ins.2014.11.009
[12] Rashed, E. D., Obaya, M., & Moustafa, H. E. (2021). Accelerating DNA pairwise sequence alignment using FPGA and a customized convolutional neural network. *Computers & Electrical Engineering*, *92*(6), 107112. https://doi.org/10.1016/j.compeleceng.2021.107112
[13] Song, Y. J., Ji, D. J., Seo, H. et al. (2021). Pairwise Heuristic Sequence Alignment Algorithm Based on Deep Reinforcement Learning. *IEEE Open Journal of Engineering in Medicine and Biology*, *2*, 36-43. https://doi.org/10.1109/OJEMB.2021.3055424
[14] Wang, J., Zhan, N. J., Feng, X. Y. et al. (2019). Overview formalization method. *Journal of software, 30*(1), 33-61.
[15] Xue, J. (1997). A unified approach for developing efficient algorithmic programs. *Journal of Computer Science and Technology*, *12*, 314-329. https://doi.org/10.1007/BF02943151
[16] Shi, H. H. & Xue, J. Y. (2009). Formal Development of PAR based Algorithm. *Chinese Journal of Computers*, *5*, 982-991
[17] Shi, H. H. & Zhou, W. X. (2019). Design and Implementation of Components for Pairwise Sequence Alignment Algorithm Based on Dynamic Programming. *Journal of Computer Research and Development*, *56*(9), 11.
[18] Jones C. B. (1990). *Systematic software development using VDM*. Prentice Hall International Series in Computer Science, Second edition.
[19] Woodcock, J. & Davies, J. (1996). *Using Z: Specification, Refinement, and Proof*. Prentice-hall International Series in Computer Science.
[20] Durr, E. & Van Katwijk, J. (1992). VDM++, a formal specification language for object-oriented designs, *CompEuro-Proceedings Computer Systems and Software Engineering*, 214-219.
[21] Smith, G. (2012). *The Object-Z specification language*. Springer Science & Business Media.
[22] Abrial, J. R. & Hoare, A. (1996). *The B-book: assigning programs to meanings*. Cambridge: Cambridge university press
[23] Abrial, J. R. (2010). *Modeling in Event-B: system and software engineering.* Cambridge University Press
[24] Smith, D. R. (2001). *Designware: Software development by refinement. High integrity software*. Springer, Boston, MA, 3-21.
[25] Bird, R. S., Gibbons, J., & Jones, G. (1989). Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, *12*(2), 93-104.
[26] Durán, J. E. (2002). Transformational Derivation of Greedy Network Algorithms from Descriptive Specifications. *International Conference on Mathematics of Program Construction.*
[27] Abrial, J. R., Cansell, D., & Méry, D. (2003). Formal derivation of spanning trees algorithms. *International Conference of B and Z Users*, 457-476.
[28] Almeida, J. B. &Pinto, J. S. (2006). *Deriving sorting algorithms. Technical Report, DI-PURe-06.04.01*, Department of Information, University of Minho.
[29] Nedunuri, S. & Cook, W. R. (2009). Synthesis of fast programs for maximum segment sum problems. *ACM Sigplan Notices*, *45*(2), 117-126.
[30] Mu, S. C. (2021). Calculating a backtracking algorithm: an exercise in monadic program derivation. https://doi.org/10.48550/arXiv.2101.09409
[31] Sun, L. Y. & Xue, J. Y. (2006). Formal Derivation of the Minimum Spanning Tree Algorithm with PAR Method. *Computer Engineering*, *32*(21), 85-87. https://doi.org/10.3969/j.issn.1000-3428.2006.21.030
[32] You, Z. & Xue, J. Y. (2009). Formal Verification based on Isabelle Theorem Prover Algorithm Program. *Computer Engineering and Science*, *31*(10).
[33] Xue, J. (1993). Two new strategies for developing loop invariants and their applications. *Journal of Computer Science and Technology*, *8*(2), 147-154.
[34] Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice Hall, New Jersey.
[35] Dijkstra, E. W. & Scholten, C. S. (1990). *Predicate Calculus and Program Semantics*. Springer-Verlag, New York.
[36] Floyd, R. W. (1967). Assigning meanings to programs. *Proceedings Symposium on Applied Mathematics, American Mathematical Society*, 31-37.
[37] Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications ACM*, *12*(10), 576-580.
[38] Shi, H. H. (2004). *Apla-java Automatic Program Transformation System supporting Generic Programming*. Jiangxi Normal University.
[39] Lai, Y. (2002). *Development of APLA to C++ automatic program conversion system*. Jiangxi Normal University

Contact information:

Haihe SHI
Jiangxi Normal University,
School of Computer and Information Engineering,
Nanchang, China
E-mail: haiheshi@163.com

Sunwen LAN
Jiangxi Normal University,
School of Computer and Information Engineering,
Nanchang, China
E-mail: swlan1999@163.com

Riming LIU
Jiangxi Normal University,
School of Computer and Information Engineering,
Nanchang, China
E-mail: 1697608238@qq.com

Haipeng SHI
(Corresponding author)
Jiangxi Normal University,
School of Software,
Nanchang, China
E-mail: option2001@163.com