软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(2):203−215 [doi: 10.13328/j.cnki.jos.005098]

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

文件比较算法 fcomp 在 Isabelle/HOL 中的验证*

宋丽华[1], 王海涛[2], 季晓君[1], 张兴元[1]

[1](解放军理工大学 指挥信息系统学院,江苏 南京 210007)

[2](解放军理工大学 信息管理中心,江苏 南京 210014)

通讯作者: 宋丽华, E-mail: songlihua_mail@163.com

摘 要: 基于机器定理证明的形式验证技术不受状态空间限制,是保证软件正确性、避免因潜在软件缺陷带来严重损失的重要方法.文件比较算法(file comparison algorithm)是一类成员众多,应用极为广泛,跨越生物信息学、情报检索、网络安全等多个应用领域的基础算法.在交互式定理证明器 Isabelle/HOL 中对 Miller 和 Myers 在 1985 年提出的基于行的文件比较算法 fcomp 做了形式化,改正了算法关于边界变量迭代的一个小错误,证明了改正后算法的可终止性和正确性;对算法时间复杂性做了完全形式化的分析,印证了算法的非形式化分析结论,为今后更多文件比较算法的形式验证提供了可供借鉴的经验.

关键词: 文件比较算法;fcomp;交互式定理证明;Isabelle/HOL

中图法分类号: TP301

中文引用格式: 宋丽华,王海涛,季晓君,张兴元.文件比较算法 fcomp 在 Isabelle/HOL 中的验证.软件学报,2017,28(2):203−215. http://www.jos.org.cn/1000-9825/5098.htm

英文引用格式: Song LH, Wang HT, Ji XJ, Zhang XY. Verification of file comparison algorithm fcomp in Isabelle/HOL. Ruan Jian Xue Bao/Journal of Software, 2017,28(2):203−215 (in Chinese). http://www.jos.org.cn/1000-9825/5098.htm

Verification of File Comparison Algorithm fcomp in Isabelle/HOL

SONG Li-Hua[1], WANG Hai-Tao[2], JI Xiao-Jun[1], ZHANG Xing-Yuan[1]

[1](College of Command Information Systems, PLA University of Science and Technology, Nanjing 210007, China)

[2](Information Management Center, PLA University of Science and Technology, Nanjing 210014, China)

Abstract: Being unbound to the state space size, mechanical theorem proving is an important method in ensuring software's correctness and avoiding serious damage from program bugs. File comparison algorithms constitute a large family of algorithms which find wide range of application domains including bio-informatics, information retrieval and network security. This paper presents a work on formalization of fcomp, an efficient line oriented file comparison algorithm suggested by Miller and Myers in 1985, in the interactive theorem prover Isabelle/HOL. A small bug in fcomp's bound variable iteration is identified, and the termination and correctness of the modified algorithm is established. Formal analysis of time complexity is also performed which coincides with the algorithm designers' own results. The presented work lays a valuable foundation for subsequent formal checking of other file comparison algorithms.

Key words: file comparison algorithm; fcomp; interactive theorem proving; Isabelle/HOL

信息化社会软件缺陷和错误带来的危害日益突出,以严格数学方法为基础的形式验证技术(formal verification)是提高软件质量的重要途径.基于模型检测(model checking)的形式验证技术虽然具有简单、可自动执行等优点,但受到状态空间规模的限制.对于操作数组、图、树等复杂数据结构的顺序程序(算法),机器定理证明(mechanical theorem proving)技术可以避免逐一检查无限的状态空间,因此更为适宜[1].

* 基金项目: 江苏省自然科学基金(BK20130070)
Foundation item: Natural Science Foundation of Jiangsu Province of China (BK20130070)
收稿时间: 2015-11-21; 修改时间: 2016-01-04, 2016-03-22; 采用时间: 2016-04-15; jos 在线出版时间: 2016-07-30
CNKI 网络优先出版: 2016-08-01 09:38:51, http://www.cnki.net/kcms/detail/11.2560.TP.20160801.0938.001.html

使用交互式定理证明(interactive theorem proving)工具对算法性质进行机器证明已成为一种发展趋势.交互式定理证明器Isabelle支持多种对象逻辑,其高阶逻辑HOL具有丰富的类型系统包含多种自动证明工具,其基于Isar的结构化证明可读性很高.此外,Isabelle/HOL中的代码生成器工具允许用户把高阶逻辑表示的算法规约直接转化为Ocaml,Scala等语言书写的可执行程序,最大化地降低从算法描述到代码实现过程中的不可知因素,确保最终程序的正确性.

文件比较算法是一类应用极为广泛的基础算法.本文基于Isabelle/HOL对Miller和Myers在1985年提出的基于行的文件比较算法fcomp[2]作了形式化,发现了算法中关于边界变量迭代的一个小错误,证明了改正后算法的正确性,分析了算法的时间复杂性.

1 相关研究

目前能够找到的关于文件比较算法的形式验证工作非常少,没有见到针对其重要性质如正确性的机器证明.在一些较为相关的工作中,Khanna等人[3]手工分析了文本数据合并算法diff3的无冲突合并的修改局部性前提条件以及幂等性(idempotence)、稳定性(stability)等性质,得出了一些与人们之前对该算法的直觉不尽相同甚至截然相反的结论.但对于作为diff3工作基础的文件比较算法diff,只作为黑盒处理,假定其是正确的.Moore等人[4]使用定理证明器ACL2证明了Boyer-Moore快速字符串匹配算法[5]的正确性,方法是把Boyer-Moore算法和一种显然正确的简单算法同时表示为Lisp函数并证明两者等价.以使用线性时态逻辑描述的手工证明[6]为起点,Besta等人[7]使用PVS证明器检查了Boyer-Moore快速字符串匹配算法[5]的待匹配字符串预处理算法的安全性(safety)、活性(liveness)等性质证明.该预处理算法比较困难且容易出错,出现过数个更正版本,机器证明对算法最后更正版本的正确性做出了最终确认.

2 文件比较算法简介

任意给定两个文件(或字符串),通过执行一系列的插入、删除、复制、替换、追加等操作,可以将其中一个文件转换为另一个文件,这一由插入、删除等操作组成的编辑命令序列称为从第1个文件到第2个文件的编辑脚本(edit script).所有编辑脚本中长度最短者称为最短编辑脚本(shortest edit script),最短编辑脚本的长度,称为两个文件的编辑距离(edit distance).文件比较算法计算两个文件(字符串)的编辑距离和最短编辑脚本,从20世纪70年代被提出来[8,9],至今仍是一个活跃的研究方向[10−14],广泛用于生物信息学、情报检索、模式识别、版本控制、网络安全、网络测量等领域[15−18].

本文选作形式验证对象的文件比较算法fcomp是一种基于行的算法,把文件的每一行视作一个不可分割的字符,只考虑插入和删除两种编辑操作.在此假定下,文件被简化为字符串,此后,我们将fcomp作为字符串上的算法进行讨论.用 d 表示编辑距离,用 m 和 n 分别表示两个文件的长度,fcomp执行时间不超过$(2d+1)\min(m,n)$.在典型输入下,fcomp的速度是Unix diff的4倍[2].

2.1 算法设计思想

fcomp算法的设计核心是一个如图1所示的$(m+1)\times(n+1)$阶编辑距离矩阵,m 和 n 分别为输入字符串 A,B 的长度.元素 $D[i,j]$(行、列均从0开始编号)表示 A 的前 i 个字符($A[1{:}i]$)到 B 的前 j 个字符($B[1{:}j]$)的编辑距离.例如,图1中 $D[5,4]=3$,说明字符串 *abcab* 与 *cbab* 的编辑距离为3.事实上,执行编辑脚本“删除1号字符 *a*;在2号字符 *b* 前插入一个 *c*;删除3号字符 *c*”可将 *abcab* 变换为 *cbab*,并且再没有比这更短的脚本.容易看出,矩阵右下角元素 $D[m,n]$即为所求 A,B 的编辑距离.

从图1可以观察到编辑距离矩阵上元素的几条分布规律,这也是fcomp算法的基本设计依据:

- 同一行或同一列的相邻元素相差1;
- 对角线 k 上的元素从$|k|$开始,逐渐以2为单位跃增;
- 若 $A[i+1]=B[j+1]$(A 的第 $i+1$ 个字符与 B 的第 $j+1$ 个字符相同),则 $D[i+1,j+1]=D[i,j]$;
- $d(d\geqslant 0)$只出现在对角线$-d,-d+2,\ldots,d-2,d$ 上.

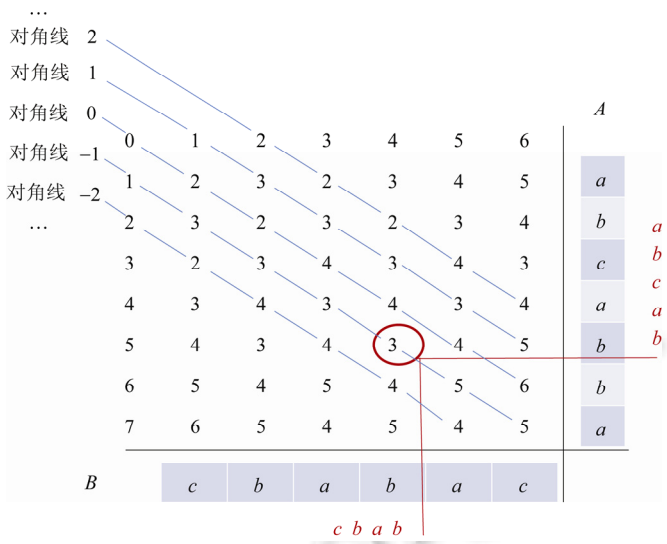


Fig.1 An illustrating matrix of edit distance where *A*=*abcabba*, *B*=*cbabac*

图 1 一个示例编辑距离矩阵(*A*=*abcabba*,*B*=*cbabac*)

算法使用一个迭代过程由小到大依次填充矩阵中的元素值.首先,找出矩阵中所有的 0.根据前面的观察,0 只可能出现在对角线 0 上.实际上,找出 0 意味着找出 *A* 和 *B* 的所有公共字头.之后,按照下面将要叙述的 3 条规则,依次填充矩阵上所有的 1,所有的 2,…,直到元素 *D*[*m*,*n*]被填充为止.在这个过程中,算法会一步步地累积编辑脚本,每填充一个 *D*[*i*,*j*],就得到 *A*[1:*i*]到 *B*[1:*j*]的一个最短编辑脚本(长为 *D*[*i*,*j*]),直至填充 *D*[*m*,*n*],得到 *A* 与 *B* 的最短编辑脚本.在最终填充 *D*[*m*,*n*]之前,究竟需要填充哪些元素是不可预知的.

规则 1(下移规则). 假设 *D*[*i*–1,*j*]已填充值 *d*,长为 *d* 的编辑脚本 *s* 将 *A*[1:*i*–1]转换为 *B*[1:*j*],而 *D*[*i*,*j*]尚未填充,则 *D*[*i*,*j*]=*d*+1(*D*[*i*,*j*]尚未填充这一事实说明其值大于或等于已填充的 *D*[*i*–1,*j*]值,由于同一列相邻元素相差 1,故 *D*[*i*,*j*]=*D*[*i*–1,*j*]+1),且在 *s* 末尾追加一条编辑命令"删除第 *i* 号字符"即为 *A*[1:*i*]与 *B*[1:*j*]的最短编辑脚本.

规则 2(右移规则). 假设 *D*[*i*,*j*–1]已填充值 *d*,长为 *D*[*i*,*j*–1]的编辑脚本 *s* 将 *A*[1:*i*]转换为 *B*[1:*j*–1],而 *D*[*i*,*j*]尚未填充,则 *D*[*i*,*j*]=*d*+1(原因与规则 1 类似),且在 *s* 末尾追加一条编辑命令"在第 *i* 号字符之后插入一个字符 *B*[*j*](*B* 的第 *j* 个字符)"即为 *A*[1:*i*]与 *B*[1:*j*]的最短编辑脚本.

规则 3(对角线下滑规则). 假设 *D*[*i*–1,*j*–1]已填充值 *d*,长为 *d* 的编辑脚本 *s* 将 *A*[1:*i*–1]转换为 *B*[1:*j*–1],且 *A*[*i*]=*B*[*j*],则 *D*[*i*,*j*]=*d*,*s* 即为 *A*[1:*i*]与 *B*[1:*j*]的最短编辑脚本.

假设算法已经填充了矩阵中所有的 0,1,…,*d*–1,此时根据规则 1 从某个 *d*–1 元素下移一位,或者根据规则 2 从某个 *d*–1 元素右移一位,再根据规则 3 沿对角线连续下滑若干位即可填充所有的 *d*.

为了提高效率,fcomp 算法又对上述的填充过程做了两点改进.

改进 1. 算法并不计算、保存整个矩阵,而是在每个迭代步 *d*,用一维数组 *last_d* 记录相关对角线(对角线 –*d*,–*d*+2,…,*d*–2,*d*)上距离矩阵右下角最近的那些 *d* 元素.*last_d*[*k*](*k*=–*d*,–*d*+2,…,*d*–2,*d*)表示对角线 *k* 上值为 *d* 且距离右下角最近的元素纵坐标值(横坐标=纵坐标+对角线编号).另一个一维数组 *script* 存储了与 *last_d* 对应的 *A*,*B* 子串的最短编辑脚本.由于含有 *d*–1 和含有 *d* 的对角线是交错的,所以算法可以在同一个 *last_d*,*script* 数组上迭代计算.

怎样利用 *last_d*[*k*–1],*last_d*[*k*+1]得到 *last_d*[*k*]?既可以对 *k*+1 对角线上记录的 *d*–1 元素下移,也可以对 *k*–1 对角线上记录的 *d*–1 元素右移.算法选择两者中最有利(更靠近右下角)的那个,即当 *last_d*[*k*+1]≥*last_d*[*k*–1]时下移,否则右移.移到对角线 *k* 上以后,再沿对角线尽可能地下滑.此处的例外是,当 *k*=±*d* 时,*last_d* 数组只在其一侧有效,故在下移或右移中只有 1 种选择.

改进 2. 理论上迭代步 d 需要更新 $last\_d$ 在对角线$-d,-d+2,\ldots,d-2,d$ 的值.但如果之前某步计算得到 $last\_d[k]=m$,说明对角线 k 已触到矩阵下边界,此后就没必要再填充 k 左侧的那些对角线,因为它们对 $D[m,n]$的取值没有影响.同样的情况也发生在触到矩阵右边界时,若某步 $last\_d[k]=n-k$,则 k 右侧的那些对角线此后也无需再填充.故此,算法用一对边界变量 $lower$,$upper$ 限制需要更新的对角线范围.在迭代步 d,只有满足 $lower\leqslant k\leqslant upper$ 且 $k\in\{-d,-d+2,\ldots,d-2,d\}$的 k 才被更新.正常情况下,$lower$ 每步减 1,$upper$ 每步加 1.当出现某对角线 k 触下边界时,$lower$ 被设置为 $k+1$;出现 k 触右边界时,$upper$ 被设置为 $k-1$.

2.2 算法分析

Miller 等人[2]用基本数学归纳法证明了算法是正确的.但和所有的非形式化证明一样,这一证明只是验证了算法设计思路的正确性,并没有做到对算法每一个步骤的严格检查,而错误往往就隐藏在繁冗的细节之中.事实上,无论是文献[2]正文的算法描述,还是其附录的 C 程序实现,对上边界变量 $upper$ 的迭代求值都是错误的.

如上所述,为了提高效率,某对角线 k 触到矩阵右边界后,其右侧的对角线就无需再参与今后的迭代.如果同时有多条对角线触右边界,而其中的最小(最下方)值为 k,显然上边界变量 $upper$ 应该在下一个迭代步更新为 $k-1$.为了清楚地说明这一点,以图 2 为例,当前 $lower=-4$,$upper=4$.带箭头的实线表示应用下移或右移规则、对角线下滑规则对 $last\_d$ 数组作更新.假设 $last\_d$ 在对角线$-4,-2,0,2,4$ 上更新后,2,4 两条对角线同时触右边界.可以看出,此时 $upper$ 应更新为 1,下一步开始不必再关心 1 右侧的那些对角线.但是,按照文献[2]给出的 fcomp 算法,$upper$ 将被更新为触右边界对角线中的最大值减 1,应用于图 2 将得到 3.

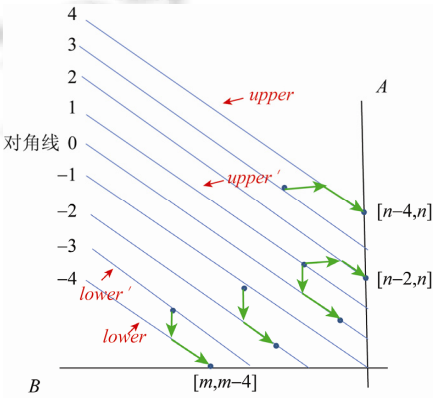


Fig.2 Iteration of bounds *lower* and *upper*

图 2 边界变量 $lower$,$upper$ 的迭代更新

诚然,这并非多么严重的错误,虽然牺牲了部分效率,并可能带来数组越界问题,但只要仔细检查,这个错误并不难以纠正.只是,人的“细致”并非 100%可靠.而类似这样的错误在我们各行业软件中大量存在,其中一些甚至可能带来更为严重的后果.这些细节性错误是非形式化证明的盲点,只有在完全形式化的证明中才会无所遁形.形式验证是确保算法程序正确、可靠的唯一手段.

3 算法在 Isabelle/HOL 中的形式化

本文首先在 Isabelle/HOL 中建立了编辑距离的形式模型;然后逐条验证了第 3.1 节列出的编辑距离矩阵元素分布规律,在其基础上确认了左移、右移、对角线下滑这 3 条移动规则的正确性;接下来,利用 Locale[19]对上边界变量求值修正后的 fcomp 算法做了形式化描述,将算法实现为一个尾递归函数,证明了算法的可终止性和正确性;最后,用一个步数计数器分析了算法的时间复杂性,证明了其上界.工作量统计见表 1.

Table 1 Amount of work

表 1 工作量统计

内容	代码量(行)
编辑距离形式化	500
矩阵元素分布规律验证	715
算法形式化	200
可终止性和正确性证明	1 630
时间复杂性分析	865
总计	3 910

3.1 编辑距离形式化

为了验证算法,首先需要形式化编辑距离的概念.按照文献[2],编辑距离为最短编辑脚本的长度.这里将编辑脚本定义为编辑命令序列,编辑距离定义为编辑脚本长度集合中的最小值,如图 3 所示(图中编辑函数 convert 按照编辑脚本作字符串变换).显然,长度等于编辑距离的脚本即为最短编辑脚本,反之亦然.

```
typedecl symbol   //字符集
type_synonym String = "symbol list" //字符串
datatype command =   //编辑命令
            Del nat   //删除命令,删除 nat 位置的字符
          | Ins nat symbol   //插入命令,在 nat 位置插入字符 symbol
type_synonym script = "command list"   //编辑脚本即为编辑命令的序列
//cs 是 A 到 B 的编辑脚本,当且仅当编辑函数作用于 A 和 cs 得到 B
definition script_of :: "script ⇒ String ⇒ String ⇒ bool"
    where "script_of cs A B ≡ convert A cs = B"
definition set_length :: "String ⇒ String ⇒ nat set"   //所有 A 到 B 编辑脚本的长度集合
    where "set_length A B ≡ {length s | s. script_of s A B}"
definition distance :: "String ⇒ String ⇒ nat"   //定义长度集合中的最小值为编辑距离
    where "distance A B ≡ (LEAST n. n ∈ set_length A B)"
definition shortest_s_of :: "script ⇒ String ⇒ String ⇒ bool"   //最短编辑脚本
    where "shortest_s_of s A B ≡ script_of s A B ∧ (∀s'. script_of s' A B → length s ≤ length s')"
```

Fig.3 Formalization of edit script and edit distance

图 3 编辑脚本和编辑距离形式化

从编辑命令角度建立的直观模型虽然方便人的观察和操作,却不方便证明,因为其中只包含输入字符串的部分信息.为了证明编辑距离有关性质,我们又定义了另一个基于编辑策略的模型,并证明了两个模型的等价性,限于篇幅,此处不详细介绍.表 1 的工作量统计包含了这一部分内容.

3.2 矩阵元素分布规律的形式验证

在编辑距离形式化的基础上,形式化并逐条验证了第 3.1 节观察到的矩阵元素分布规律.以此为基础,很容易验证 fcomp 算法的移动规则.下面给出 3 条规则中与最短编辑脚本有关的部分,编辑距离相关部分不再赘述.

定理 1(**下移规则**). "⟦legal (Suc i, j); S_of t (i, j); D (i, j) < D (Suc i, j)⟧⟹S_of (t @ [Del i]) (Suc i, j)".

定理 2(**右移规则**). "⟦legal (i, Suc j); S_of t (i, j); D (i, j) < D (i, Suc j)⟧⟹S_of (t @ [Ins i (B ! j)]) (i, Suc j)".

定理 3(**对角线下滑规则**). "⟦legal (Suc i, Suc j); S_of t (i, j); A ! i = B ! j ⟧⟹ S_of t (Suc i, Suc j)".

以上定理中用到的符号:谓词 legal 检查矩阵坐标是否在合法范围内;对合法坐标(*i*,*j*),*D* 返回 *A*[1:*i*]到 *B*[1:*j*]的最短编辑距离;“S_of t (i, j)”确定 *t* 是否为 *A*[1:*i*]到 *B*[1:*j*]的最短编辑脚本;“A ! i”取字符串 *A* 的第 *i*+1 个字符.

type_synonym coord = "nat * nat".   //(行,列),矩阵坐标类型

定义 1(**编辑距离矩阵**). D :: "coord ⇒ nat" where "D (i,j) = distance (take i A) (take j B)".

定义 2(**最短编辑脚本**). S_of :: "script ⇒ coord ⇒ bool" where "S_of t (i, j) = shortest_s_of t (take i A) (take j B)".

3.3 基于Locale的算法描述

将输入字符串 *A*,*B* 交换,则编辑距离矩阵将转置,原来的右(下)边界变成下(右)边界,原来的右(下)移规则变成下(右)移规则,原来的上边界变量 *upper*(*lower*)变成下边界变量 *lower*(*upper*).由于这个原因,fcomp 算法的设计里有很多以主对角线为中心对称的概念.为充分利用这一对称性,算法的形式描述和验证使用了 Locale.Locale 扩展是 Isabelle 中处理带参理论的方法,可以用来表示有复杂关联关系的模块化系统[19].基于以字符串 *A*,*B* 为参数的 Locale,对称的两个概念可以只定义一个,性质证明只做一次,其对称概念和性质通过 Locale 在 *B*,*A* 下的对称解释以及编辑距离的交换律直接获得,节约一半工作量.

图 4 在以 *A*,*B* 为参数,名为 fcomp 的 Locale 中定义了触下边界(hit_last_row)谓词;然后用 *B*,*A* 解释 fcomp,得到一个与之对称的结构 symm,并将触右边界(hit_last_col)谓词定义为 symm 中的触下边界;最后对两个谓词取或,定义了触边界(hit_last)谓词.这里,*converse* (*x*,*y*)=(*y*,*x*).fcomp 算法相关组件全部在 Locale fcomp 中定义,以后不再赘述.

```
locale fcomp =    //定义一个名为 fcomp 的 Locale
   fixes A :: "String"    //Locale 参数,字符串 A
   fixes B :: "String"    //Locale 参数,字符串 B
begin
primrec hit_last_row :: "coord ⇒ bool"    //触下边界谓词 hit_last_row
      where "hit_last_row (row,col) = (row ≥ length A)"
interpretation symm : fcomp B A    //用 B,A 作实参代入形参 A,B,得到 fcomp 的一个对称解释
done
definition hit_last_col :: "coord ⇒ bool"    //触右边界谓词 hit_last_col
      where "hit_last_col coord = symm.hit_last_row (converse coord)"
definition hit_last :: "coord ⇒ bool"    //触边界谓词 hit_last
      where "hit_last coord ≡ hit_last_row coord ∨ hit_last_col coord"
end
```

Fig.4 Locale fcomp and the definition of predicate hit_last_row in it

图 4 Locale fcomp 及其下的触边界谓词定义

Isabelle/HOL 使用函数式的建模方法.形式化一种算法,意味着将其实现为函数式语言风格的“程序”.而后,“程序”中的定义式如递归式经 Isabelle 内部处理后将被接受为基本事实,供我们后面证明算法的性质时使用.Isabelle 提供了很多自动证明工具,帮助用户从基本事实开始构造复杂性质,如正确性、可终止性的证明.但一个复杂证明的框架、所用到的引理等仍需用户构造,自动证明工具在这方面的作用有限.Isabelle 更多地扮演着对证明进行检查的角色,这也正是交互式定理证明工具的特征.

由第 3.1 节可知,fcomp 算法核心是围绕编辑距离矩阵对数组 *last_d*,*script* 和上下边界变量 *upper*,*lower* 等 4 个数据结构的迭代.迭代初始,编辑距离 *d*=0,以后每步加 1.当某步迭代得到 $last\_d[K_0]=m$(对角线 $K_0=n-m$),即碰到矩阵右下角元素 *D*[*m*,*n*]时,迭代过程终止.迭代结束时的 *d* 为所求 *A* 到 *B* 的编辑距离,*script*[K_0]给出最短编辑脚本.基于这一过程,我们将把 fcomp 形式化为一个尾递归函数.首先定义所需的类型.

```
type_synonym diag = int.    //对角线编号类型
type_synonym last_d = "diag ⇒ coord".    //last_d 数组类型
type_synonym scripts = "diag ⇒ script".    //script 数组类型
```

假设 *d* 为当前迭代步对应的编辑距离,仍用 *last_d*,*script*,*upper* 和 *lower* 表示对应的数组和边界变量,图 5 给出了形式化后的 4 个迭代函数,其中用到的辅助函数定义如图 6 所示.

辅助函数 slide_down 表示连续应用对角线下滑规则,直到碰到相异元素或触矩阵边界不能继续下滑为止;up_trig_slide 是右移规则和对角线下滑规则的组合,先应用一次下移规则,再沿对角线尽可能地下滑;down_trig_slide 则先应用一次右移规则,再沿对角线尽可能地下滑.函数 next 按照在 up_trig_slide 和 down_trig_slide 中选择最有利动作的原则,实现了 *last_d* 数组的迭代更新.next_s 实现最短编辑脚本 *script* 数组的迭代更新.这两个迭代过程与原 fcomp 算法完全相同.

```
//数组 last_d 的迭代
definition "next" :: "last_d ⇒ nat ⇒ diag ⇒ diag ⇒ last_d"
where "next last_d d lower upper k ≡ if inRange k lower upper then
                                    (if shouldDown last_d d k then down_trig_slide (last_d (k + 1))
                                else up_trig_slide (last_d (k - 1)))
                            else last_d k"
//数组 script 的迭代
definition next_s :: "last_d ⇒ last_s ⇒ nat ⇒ diag ⇒ diag ⇒ last_s"
where "next_s last_d script d lower upper k ≡ if inRange k lower upper then
                                    (if shouldDown last_d d k then script (k + 1) @ [down_trig_command (last_d (k + 1))]
                                        else script (k - 1) @ [up_trig_command (last_d (k - 1))])
                                            else script k"
//下边界变量 lower 的迭代
function next_lower :: "last_d ⇒ diag ⇒ diag ⇒ diag"
where "next_lower last_d lower upper = (if upper < lower then lower else
                                    (if hit_last_row (last_d upper) then upper + 2 else next_lower last_d lower (upper - 2)))"
definition lower_it :: "last_d ⇒ diag ⇒ diag ⇒ diag"
where "lower_it last_d lower upper ≡ if upper < lower then lower + 1 else next_lower last_d lower upper - 1"
//上边界变量 upper 的迭代
definition upper_it :: "last_d ⇒ diag ⇒ diag ⇒ diag"
where "upper_it last_d lower upper ≡- symm.lower_it (trans last_d) (-upper) (-lower)"
```

Fig.5 Iteration of arrays *last_d*, *script* and bounds *upper*, *lower*

图 5 数组 *last_d*,*script* 和边界变量 *upper*,*lower* 的迭代函数

```
definition inRange :: "diag ⇒ diag ⇒ diag ⇒ bool"   //检查对角线 k 是否在更新范围
    where "inRange k lower upper ≡ lower ≤ k ∧ k ≤ upper ∧ 2 dvd (upper - k)"
definition shouldDown :: "last_d ⇒ nat ⇒ diag ⇒ bool"   //确定对角线 k 更新应该使用下移还是右移规则
    where "shouldDown last_d d k ≡ k = - (int d) ∨ (k ≠ int d ∧ fst (last_d (k + 1)) ≥ fst (last_d (k - 1)))"
function slide_down :: "coord ⇒ coord"   //沿对角线持续下滑,直到碰到相异字符或触到矩阵边界
    where "slide_down (i,j) = (if (hit_last (i,j) ∨ A ! i ≠ B ! j) then (i,j) else slide_down (i+1,j+1))"
definition up_trig_slide :: "coord ⇒ coord"   //先右移,再 slide_down
    where "up_trig_slide coord ≡ let (i,j) = coord in slide_down (i,j+1)"
primrec up_trig_command :: "coord ⇒ command"   //对应右移动作的插入命令
    where "up_trig_command (i, j) = Ins i (B ! j)"
definition down_trig_slide :: "coord ⇒ coord"   //先下移,再 slide_down
    where "down_trig_slide coord ≡ converse (symm.up_trig_slide (converse coord))"
primrec down_trig_command :: "coord ⇒ command"   //对应下移动作的删除命令
    where "down_trig_command (i, j) = Del i"
```

Fig.6 Auxiliary functions for the main iteration

图 6 迭代过程用到的辅助函数

下边界变量 *lower* 的迭代包含 next_lower 和 lower_it 两个函数,与原 fcomp 算法对 *lower* 更新步骤一致,计算结果将是所有触下边界对角线的最大值加 1,无触下边界对角线时则为当前 *lower* 减 1;*upper* 的迭代过程取 *lower* 迭代过程的对称定义,表示所有触右边界对角线的最小值减 1,无触右边界对角线时为当前 *upper* 加 1;trans 表示数组的转置,trans *last_d k*=*converse*(*last_d*(–*k*)).这里特别规定了 *lower* 和 *upper* 翻转时的情况,一旦 *lower*>*upper*,后面将一直如此,这样做是为了方便算法可终止性证明.

再来定义迭代初值和终值.当 *d*=0 时,由于所有的 0 元素都在对角线 0 上,所以 *lower* 和 *upper* 的初值均为 0.相应地,数组 *last_d* 在对角线 0 处的值应为 *slide_down*(0,0),对应 *A*,*B* 的最长公共字头,数组 *script* 迭代初始为一个空命令串,见图 7 关于 $last\_d_0$ 和 $script_0$ 的定义.以初值为起点,不断应用迭代函数,直到满足终止条件,输出算法结果.这里,算法结果是一个序偶(见图 7 中 Distance 定义),序偶第一分量为 *A* 到 *B* 编辑距离,第二分量为最短编辑脚本.图 5 列出的 4 个迭代函数在图 7 中被合并成一个迭代过程——尾递归函数 Distance_aux.

观察图 7 的定义,函数 Distance_aux 可能在两种情况下终止:

(1) 当某步迭代开始时,*upper*<*lower*∨¬ valid *last_d d lower upper*;

(2) 在下一步迭代之前,*upper′*<*lower′*(*′*为变量的下一步迭代值).

```
//计算编辑距离和最短编辑脚本的辅助函数
function Distance_aux :: "last_d ⇒ last_s ⇒ nat ⇒ diag ⇒ diag ⇒ nat × script"
    where "Distance_aux s t d l u = (if u < l ∨ ¬ valid s d l u then (d, t K₀) else
                        let l' = lower_it s l u; u' = upper_it s l u; s' = next s (Suc d) l' u'; t' = next_s s t (Suc d) l' u'
                                in (if u' < l' then (d, t K₀) else Distance_aux s' t' (Suc d) l' u'))"
//初始 last_d
definition last_d₀ :: last_d
    where "last_d₀ k ≡ (if k = 0 then (slide_down (0, 0)) else (0, 0))"
//初始 script
definition script₀ :: "last_s"
    where "script₀ k ≡ []"
//最终的编辑距离和最短编辑脚本
definition Distance :: "nat × script"
    where "Distance ≡ Distance_aux last_d₀ script₀ 0 0 0"
```

Fig.7 Functions for finding edit distance and a shortest edit script

图 7 编辑距离和最短编辑脚本计算函数

谓词 valid 对矩阵元素和边界变量的合法性做了一些简单约束,实际上是 *last_d* 和 *lower*,*upper* 在迭代中满足的一个循环不变式,第 4.5 节再详细介绍.这里只需指出,由于初始状态(*d*=0)*last_d* 和 *lower*,*upper* 满足 valid 谓词,且 *lower*≤*upper*,所以最终迭代只可能停止在第 2 种情况,也就是上下边界变量即将翻转时.

为什么用边界变量的翻转代替原算法中的"*last_d*[K_0]=*m*"作为迭代停止条件?因为在图 5 给出的定义下,这两个条件是等价的,并且前者更方便证明.参照图 2 仔细考察 *lower* 和 *upper*:每一步,[*lower*,*upper*]标出了求取最终目标所依赖的对角线区间.从初值 0 开始,起初边界在向外扩张,每步 *upper* 加 1、*lower* 减 1,分别向边缘对角线 *n* 和–*m* 逼近,[*lower*,*upper*]涵盖了所有 *d* 值元素可能出现的对角线.之后,若区间内某条对角线下滑时触到右边界,则 *upper* 将转而向内收缩(减小),将那些对目标求取无用的对角线排除在外.并且这一过程不可逆转,一旦开始就会一直收缩,直到碰到对角线 K_0,这是因为触右边界对角线下面的那条对角线更新后依然触到右边界.边界 *lower* 的情况与此类似.这样,我们就可以将一个边界变量的迭代过程区分为两个阶段——先扩张再收缩.扩张时逐步向边缘对角线 *n* 或–*m* 逼近,收缩时逐步向对角线 K_0 逼近.注意,只有 K_0 右边的对角线才能触矩阵右边界,也只有 K_0 左边的对角线才能触下边界,而 K_0 则将同时触两个边界,意味着碰到矩阵右下角元素[*m*,*n*],此时根据图 5 定义,*lower* 和 *upper* 的下一步迭代值就会出现翻转,导致 *upper*′<*lower*′.并且,也只有 K_0 触边界才可能导致这样的翻转.如此,碰到元素[*m*,*n*]这一终止信号就转化成了上下边界变量即将翻转的信号.并且,边界变量先扩张再收缩是一个两阶段单调变化过程,借助其单调性即可证明迭代过程一定能够停止.

3.4 算法可终止性证明

为了证明算法正确,首先必须证明迭代过程能够终止,即 *lower* 和 *upper* 终将翻转.事实上,为了确保系统的一致性,Isabelle/HOL 强制要求所有函数都是全函数,也就是说,图 7 中的 Distance_aux 函数必须在定义之后提供终止性证明,否则 Isabelle 不能接受.

Isabelle/HOL 为证明递归终止性提供了一种称为 relation 的证明方法.使用该方法,我们需要在 Distance_aux 函数的自变量上定义一个关系,并证明:(1) 该关系是良基的;(2) Distance_aux 中递归调用的参数满足该关系.换句话说,就是必须证明递归调用的参数以某种方式在渐渐"变小".

根据第 3.3 节末尾对上下边界变量的分析,*lower* 和 *upper* 在翻转之前都呈现一个扩张-收缩的两阶段变化过程.以 *lower* 为例,起初是每步减 1,向边缘对角线 *k*=–*m* 逼近;然后,当出现对角线触下边界事件后,进入第 2 阶段,严格递增,从 K_0 的左侧向 K_0 逼近.这提示我们使用⟨阶段,距离⟩二元组来编码 *lower* 的变化过程:扩张阶段编码为 1,收缩阶段编码为 0;当处于扩张阶段时,*lower* 与对角线 *k*=–*m* 的绝对差作为距离度量;当处于收缩阶段时,*lower* 与对角线 K_0 的绝对差作为距离度量.容易看出,随着迭代的逐步进行,⟨阶段,距离⟩二元组以字典序在逐步减小,直至减到最小值⟨0,0⟩(K_0 触边界),迭代结束.由⟨阶段,距离⟩的字典序所确定的 Distance_aux 函数自变量上的关系即可作为所要找的良基关系.

图 8 给出了 Distance_aux 函数的终止性证明及其所依赖的编码函数定义和相关引理.phase 是阶段编码函

数,根据下边界变量 *lower* 与迭代值 *d* 的相反数是否相等判断当前处于扩张还是收缩阶段.距离编码函数 space 根据当前所处的阶段计算与参考点之间的距离度量,扩张阶段取边缘对角线-*m* 为参考点,收缩阶段取 K_0 为参考点.引理 1 证明了迭代中阶段编码(非严格)单调递减,引理 2 则证明同一阶段内距离编码严格单调减,两个引理合起来等于证明了 Distance_aux 的递归调用参数满足⟨阶段,距离⟩编码字典序所确定的关系.这里,fcomp_valid, fcomp_valid_it 是 Locale fcomp 的子 Locale,详见下一节.

```
//阶段编码函数
fun phase :: "last_d * last_s * nat * diag * diag ⇒ nat"
where "phase (s, t, d, l, u) = (if (l = - int d) then 1 else 0)"
//距离编码函数
fun space :: "last_d * last_s * nat * diag * diag ⇒ nat"
    where "space (s, t, d, l, u) = (if (l = - int d) then length A - d else nat (K0 - l))"
引理 1[intro]. "⟦fcomp_valid_it  A B s d l u l' u' s';  ¬ phase (s', t', (Suc d), l', u') < phase (s, t, d, l, u) ⟧ ⟹ phase (s',
t', (Suc d), l', u') = phase (s, t, d, l, u)".
引理 2[intro]. "⟦ fcomp_valid_it  A B s d l u l' u' s'; ¬ u' < l'; ¬ phase (s', t', (Suc d), l', u') < phase (s, t, d, l, u) ⟧ ⟹
space (s', t', (Suc d), l', u') < space (s, t, d, l, u)".
//Distance_aux 函数的终止性证明
termination Distance_aux
apply(relation "measures [phase, space]")
by(auto simp add: fcomp_valid_it_def fcomp_valid_it_axioms_def fcomp_valid_def)
```

Fig.8 Termination proof of the function Distance_aux, encoding functions and some involved lemmas

图 8 Distance_aux 函数终止性证明、编码函数定义和相关引理

最后,termination 语句开启函数终止性证明.应用 relation 方法,并指示 Isabelle 使用 measures 函数按照[phase,space]二元编码字典序生成 Distance_aux 参数上的一个关系.自动证明工具 auto 证明了该关系是良基的,并在引理 1、引理 2 的帮助下证明了 Distance_aux 递归调用的参数满足该关系.

3.5 算法正确性证明

从 Hoare 演算的角度来说,假定迭代语句正确性的推理规则表示为 $\dfrac{\{\Phi \wedge B\}C\{\Phi\}}{\{\Phi\}\ while\ (B)C\{\Phi \wedge \neg B\}}$,其中,$\Phi$为循环不变式,$B$ 为循环条件,为证明循环结束后具有状态 Ψ,我们需要证明:(1) Φ确为循环不变式($\{\Phi \wedge B\}C\{\Phi\}$);(2) 循环初始时$\Phi$成立;(3) $\Phi \wedge \neg B \Longrightarrow \Psi$.

Isabelle/HOL 中的验证策略基本相同,只除了一点:由于算法建模使用了函数式风格的递归过程,没有变量记录,从而也就无法引用迭代的中间状态,因此第 3 步只能采用变通的方法证明 Ψ.基于此,我们增加一个步骤,用 4 步来证明算法正确性:第 1 步,证明一组循环不变式;第 2 步,证明迭代初始值($last\_d_0$,$script_0$)满足不变式;第 3 步,为了证明算法输出结果 Distance 的性质,导出一条 Distance 引入规则,该规则使我们不必直接证明 Distance 具有某性质 *P*,而是可以间接证明满足循环不变式的所有结构组合 *last_d*,*script*,*lower* 和 *upper* 在边界变量即将翻转时(¬*B*)均满足 *P*;第 4 步,使用 Distance 引入规则间接证明算法将输出预期结果(定理 4).

定理 4. "Distance = (d, script) ⇒ d = distance A B ∧ shortest_s_of script A B".

工作量主要集中在第 1 步.迭代中的数组 *last_d*,*script* 和边界变量 *upper*,*lower* 保持哪些不变性质呢?首先,从取值合法性上看,两个边界变量总是相差一个偶数,它们标出的数组有效区域应该不超过[–*d*,*d*]范围,且该有效区域内数组 *last_d* 记录的每一个坐标应与其对应对角线一致,是矩阵合法坐标.这些简单的合法性约束可以用谓词 valid(定义 4)来表示;其次,按照算法设计,数组 *last_d* 在每个迭代步 *d* 记录了矩阵上距离右下角最近的那些 *d* 值元素坐标.也就是说,如果 *last_d* 记录了坐标[*i*,*j*],则位于矩阵第 *i* 行第 *j* 列的元素一定是 *d*,并且只要[*i*,*j*]没有触到矩阵边界,其右方、下方、右下方相邻元素都应该大于 *d*.这一关于编辑距离的语义约束可以用谓词 D_valid(定义 5)表示;最后,数组 *script* 中记录了与 *last_d* 对应的最短编辑脚本,这一关于编辑脚本的语义约束可以用谓词 S_of(定义 2)表示.

定义 3. valid_matrix :: "last_d ⇒ diag ⇒ diag ⇒ bool" where "valid_matrix s l u ≡ ∀k. inRange k l u → Diag

(s k) = k ∧ legal (s k)".

定义 4. valid :: "last_d ⇒ nat ⇒ diag ⇒ diag ⇒ bool" where "valid s d l u ≡ l ≥– int d∧u ≤ int d∧2 dvd (u – l)∧valid_matrix s l u ∧(¬ u < l∧l ≠ - int d → hit_last_row (s l))∧(¬ u < l∧u ≠ int d → hit_last_col (s u))".

定义 5. D_valid :: "last_d ⇒ nat ⇒ diag ⇒ bool" where "D_valid s d k ≡ D (s k) = d (let (i,j)=s k in (¬ hit_last_row (i, j) → d < D (i+1, j))∧(¬ hit_last_col (i, j) → d < D (i, j+1))∧(¬ hit_last (i, j) → D (i+1, j+1) = Suc (Suc d)))".

综上,我们将循环不变式分解为 3 个谓词.为了完成这些循环不变式的证明,我们还在 Locale fcomp 的基础上额外定义了 4 个 Locale,这些 Locale 的参数、假设和它们之间的逻辑关系如图 9 所示.3 个循环不变式分别在 fcomp_valid,fcomp_D_valid,fcomp_S_valid 这 3 个 Locale 的假设部分作为已知条件引入.图中另外两个带阴影的 Locale,将用于算法时间复杂性分析.

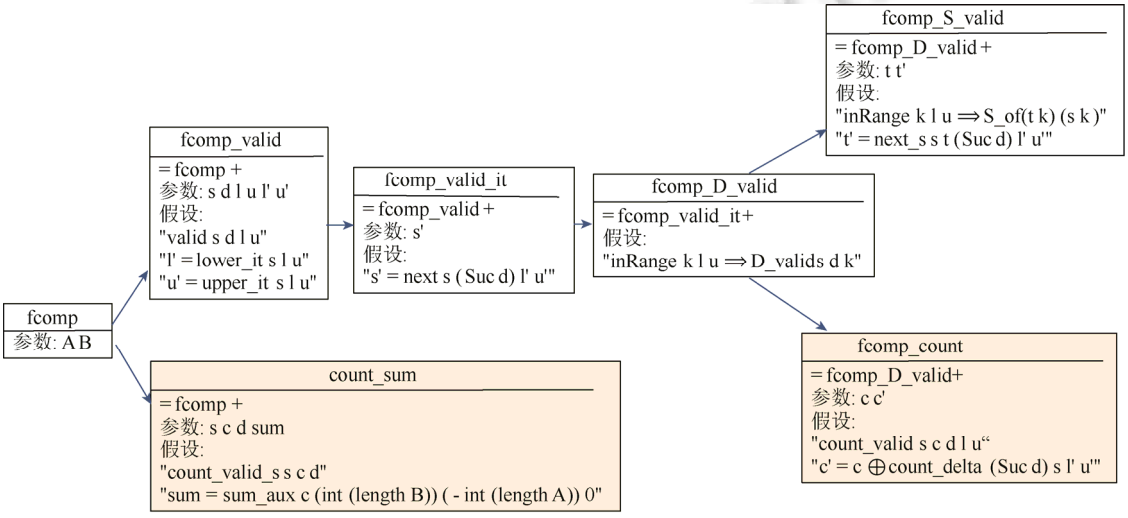


Fig.9 Locales for formalization

图 9 形式化中定义的 Locale 结构

谓词 valid,D_valid 和 S_of 所断言的关系在迭代后的 *last_d*,*script* 和上下边界变量 *lower*,*upper* 上仍然满足,即上述 3 个谓词确实为循环不变式的证明在各 Locale 中完成,如图 10 中的引理 4、引理 5、引理 7 所示.同时证明的还有 K_0 相关的重要性质:引理 3 断定当上下边界变量即将翻转(迭代结束)时,对角线 K_0 恰好被更新且触到了矩阵边界;引理 6、引理 8 则断定当对角线 K_0 被更新且触边界时,当前迭代值 *d* 就是 *A*,*B* 的编辑距离,*script*[K_0]中存储的即为 *A* 到 *B* 的最短编辑脚本.

```
Locale fcomp_valid:
//当上下边界变量即将翻转时,K0 被更新且触边界
引理 3. "⟦¬ u < l; u' < l'⟧ ⟹ inRange K0 l u ∧ hit_last (s K0)".
Locale fcomp_valid_it:
引理 4. "valid s‘ (Suc d) l' u'".  //valid 确是一个循环不变式
Locale fcomp_D_valid:
引理 5. "inRange k l' u' ⟹ D_valid s' (Suc d) k".  //D_valid 确是一个循环不变式
//当对角线 K0 被更新且触边界时,当前迭代步数 d 即为 A,B 编辑距离
引理 6. "⟦inRange K0 l u; hit_last (s K0)⟧ ⟹d = distance A B".
Locale fcomp_S_valid:
引理 7. "inRange k l' u' ⟹ S_of (t' k) (s' k)".  //S_of 确是一个循环不变式
//当对角线 K0 被更新且触边界时,script[K0]中存储的即为 A,B 最短编辑脚本
引理 8. "⟦inRange K0 l u; hit_last (s K0)⟧ ⟹shortest_s_of (t K0) A B".
```

Fig.10 Important lemmas proved in various Locales

图 10 各 Locale 中证明的重要引理

以上是第 1 步.接下来,第 2 步证明 *last_d*,*script*,*lower* 和 *upper* 的迭代初值满足循环不变式(如图 11 中的引理 9~引理 11 所示).

```
引理 9. "valid last_d0 0 0 0".   //迭代初值满足循环不变式 valid
引理 10. "D_valid last_d0 0 0".   //迭代初值满足循环不变式 D_valid
引理 11. "S_of (script0 0) (last_d0 0)".   //迭代初值满足循环不变式 S_of
//Distancey 引入规则
引理 12. "(∧ s t d l u s' t' l' u'. ⟦fcomp_S_valid A B s d l u l' u' s' t t'; ¬ u < l; u' < l'⟧ ⟹ P (d, t K0)) ⟹ P Distance".
```

Fig.11 Lemmas on iteration starting values and an introduction rule for Distance

图 11 关于迭代初值的引理和 Distance 引入规则

第 3 步,在前两步的基础上导出 Distance 引入规则(如图 11 中的引理 12 所示,其中谓词 fcomp_S_valid 为 3 个循环不变式的合取,其参数 *s*,*t*,*l*,*u* 分别对应 *last_d*,*script*,*lower* 和 *upper*,*s'*,*t'*,*l'*,*u'* 为其下一步迭代值,*d* 为当前迭代步编辑距离值).如前所述,我们不能直接证明 $\Phi \wedge \neg B \Longrightarrow \Psi$,而 Distance 引入规则的作用是提供间接证明方法.这条规则确保,为了证明 Distance(算法输出结果)满足某个性质 *P*(这里我们关心的 *P* 是算法正确性,即 Ψ),可以间接对边界变量即将翻转($\neg B$)时刻的任意 *last_d*,*script*,*lower* 和 *upper* 结构组合立论,如果满足 fcomp_S_valid(循环不变式 Φ)的这种结构组合全都满足性质 *P*,那么 Distance 也同样满足 *P*.

Distance 引入规则的证明需要用到引理 4、引理 5、引理 7 和引理 9、引理 10、引理 11.事实上,前两步关于循环不变式和迭代初始值的证明都可以看作是为这条引入规则证明所做的准备.根据尾递归函数 Distance_aux 定义(如图 7 所示),若下一步迭代边界变量将发生翻转,则函数直接返回当前 *d* 和 *script* K_0 组成的序偶作为结果,否则执行下一次迭代.用归纳法证明引入规则的正确性(Isabelle/HOL 根据 Distance_aux 定义自动生成了一条归纳证明规则).假设迭代一开始即终止,则意味着边界变量在第 2 步就会发生翻转,由于 *last_d*,*script*,*lower* 和 *upper* 的迭代初值 $last\_d_0$,$script_0$,0,0 满足循环不变式 fcomp_S_valid,故根据 Distance 引入规则的前提条件,此时的函数输出(0,$script_0$,K_0)具有性质 *P*.如果边界变量并不会在第 2 步发生翻转,则进入下一次迭代.由于循环不变式 fcomp_S_valid 在下一步迭代的 *last_d'*,*script'*,*lower'* 和 *upper'* 上仍然满足,所以根据归纳假设,迭代得到的结果仍然具有性质 *P*.

最后,结合 Distance 引入规则和 K_0 有关的事实(引理 3、引理 6、引理 8),即可完成定理 4 的证明,如图 12 所示.图中方框内为定理证明器输出结果.可以看出,应用 Distance 引入规则后,Isabelle 能够自动实例化规则中的性质 *P*,并由此将证明目标中的 Distance 改为满足 Φ和$\neg B$ 的任意结构组合.下一步给自动证明工具 auto 补充引理 3、引理 6、引理 8 作为事实,auto 即可自动消解证明目标,完成定理证明.

```
theorem[rule_format]: "Distance = (d, s) ⟶ d = distance A B ∧ shortest_s_of s A B"
apply(rule DistanceI)

goal (1 subgoal):
 1. ⋀sa da l u s' l' u' t t '.
      fcomp_S_valid A B sa da l u l' u' s' t t' ⟹
      ¬ u < l ⟹ u' < l' ⟹ (da, t K0) = (d, s) ⟶ d = distance A B ∧ shortest_s_of s A B

by(auto intro: fcomp_D_valid.lemma_6 fcomp_S_valid.lemma_8 dest: fcomp_valid.lemma_3
   simp add: fcomp_S_valid_deffcomp_D_valid_deffcomp_valid_it_def)

theorem Distance = (?d, ?s) ⟹ ?d = distance A B ∧ shortest_s_of ?s A B
```

Fig.12 Proof scripts of Theorem 4 vs. Isabelle's outputs

图 12 定理 4 证明脚本与 Isabelle 输出结果

3.6 算法时间复杂性分析

文件比较算法的计算量主要来自于文本比较.这里从字符比较的角度,对 fcomp 算法执行时间进行分析.

设置字符比较计数器.易知算法的字符比较动作仅发生在 slide_down 函数(如图 6 所示)中.用函数 count_slide_down 计算 slide_down 发生的字符比较次数,同时定义一个数组(last_count 类型),记录迭代过程中各条对角线上已经发生的字符比较次数.每一步迭代,发生了新的字符比较动作的对角线都会增加一个增量.迭代

结束时,矩阵各对角线上记录的比较次数之和(只需对[–*m*,*n*]范围内的合法对角线求和)就是算法执行的总比较次数.相关定义如图 13 所示.

```
function count_slide_down :: "coord ⇒ nat ⇒ nat"   //统计一次 slide_down 执行的字符比较次数
    where "count_slide_down (i, j) n = (if hit_last (i, j) then n else
            if A ! i ≠ B ! j then Suc n   else count_slide_down (i+1, j+1) (Suc n))"
//记录字符比较次数的数组类型定义
type_synonym last_count = "diag ⇒ nat"
//求字符比较计数器数组的辅助函数,每次迭代计数器数组增加一个增量
//count_delta 为一次迭代产生的增量, ⊕ 为计数器数组加运算
function Count_aux :: "last_d ⇒ last_count ⇒ nat ⇒ diag ⇒ diag ⇒ last_count"
    where "Count_aux s c d l u = (if u < l ∨ ¬ valid s d l u then c else   let l' = lower_it s l u;
                    u' = upper_it s l u; s' = next s (Suc d) l' u'; c' = c ⊕ count_delta (Suc d) s l' u'
                                                in (if u' < l' then c else Count_aux s' c' (Suc d) l' u'))"
definition count0 :: "last_count"   //字符比较计数器数组初值
    where "count0 k ≡ if k = 0 then count_slide_down (0, 0) 0   else 0"
definition Last_Count :: "last_count"   //迭代结束时字符比较计数器数组终值
    where "Last_Count = Count_aux last_d0 count0 0 0 0"
definition Sum :: "nat"   //算法执行的总字符比较次数,sum_aux 为统计求和函数
    where "Sum ≡ sum_aux Last_Count (int (length B)) (- int (length A)) 0"
```

Fig.13 Definitions involved in symbol comparison counters

图 13 字符比较计数器及相关定义

图 9 中定义了两个专用于此处复杂性分析的 Locale.Locale fcomp_count 假设中的 count_valid 谓词是计数器数组的循环不变式,主要对计数器数组的取值上限和有效范围做了约束;Locale count_sum 中的 count_valid_s 谓词是 count_valid 的弱化.基于 Locale fcomp_count 和 Locale count_sum,可以证明算法执行时间上限:最坏情况下不超过 *m*×*n* 次字符比较(定理 5);当 *A*,*B* 编辑距离较小时,算法执行时间几乎呈线性,比较次数不超过(*A*,*B* 编辑距离的 2 倍+1)×min(*m*,*n*)(定理 6).这一结果与文献[2]一致.

定理 5. "Sum≤length A * length B".

定理 6. "Sum≤(2 * distance A B+1) * min(length A)(length B)".

4 结 语

基于证明的形式验证技术无需逐一检查状态空间,适宜验证含有复杂数据结构的算法程序.fcomp 是一种高效的文件比较算法,以数组为主要数据结构,本质上是对抽象的编辑距离矩阵的填充和搜索.本文改正了 fcomp 算法中边界变量迭代的一个小错误,在交互式定理证明器 Isabelle/HOL 中对算法做了形式化,证明了改正后算法的可终止性和正确性,展示了 Isabelle/HOL 在形式验证复杂算法程序方面的优势.文件比较算法是一个应用极其广泛、变种繁多的算法族,而目前关于这类算法的形式验证工作还非常不够,本文为今后针对此类算法的形式验证提供了一个较好的基础.

References:

[1] Huth M, Ryan M. Logic in Computer Science: Modeling and Reasoning about Computer Systems. 2nd ed., Beijing: China Machine Press. 2005.

[2] Miller W, Myers EW. A file comparison program. Software, Practice and Experience, 1985,15(11):1025−1040. [doi: 10.1002/spe.4380151102]

[3] Khanna S, Kunal K, Pierce BC. A formal investigation of diff3. In: Arvind C, ed. Proc. of the FSTTCS 2007. LNCS 4855, New Delhi: Springer-Verlag, 2007. 485−496. [doi: 10.1007/978-3-540-77050-3_40]

[4] Moore J, Martinez M. A mechanically checked proof of the correctness of the Boyer-Moore fast string searching algorithm. In: Proc. of the Marktoberdorf Summer School: Engineering Methods and Tools for Software Safety and Security. 2009. 267−284.

[5] Boyer RS, Moore JS. A fast string searching algorithm. Communications of the ACM, 1977,20(10):762−772. [doi: 10.1145/359842.359859]

[6] Rytter W. A correct preprocessing algorithm for Boyer-Moore string searching. SIAM Journal on Computing, 1990,9(3):509−512.

[7] Besta M, Stomp F. A complete mechanization of a correctness proof of a string-preprocessing algorithm. Formal Methods in System Design, 2005,27(1-2):5−27. [doi: 10.1007/s10703-005-2243-0]

[8] Hunt JW, Mcillroy MD. An algorithm for differential file comparison. Technical Report, #41, Bell Laboratories, 1976.

[9] Hunt JW, Szymanski TG. A fast algorithm for computing longest common subsequences. Communications of the ACM, 1977, 20(5):350−353. [doi: 10.1145/359581.359603]

[10] Bergroth L, Hakonen H, Raita T. A survey of longest common subsequence algorithms. In: Proc. of the String Processing and Information Retrieval-SPIRE. 2000. 39−48. [doi: 10.1109/SPIRE.2000.878178]

[11] Amir A, Eisenberg E, Porat E. Swap and mismatch edit distance. Algorithmica, 2006,45(1):109−120. [doi: 10.1007/s00453-005-1192-8]

[12] Krusche P, Tiskin E. New algorithms for efficient parallel string comparison. In: Proc. of the ACM Symp. on Parallel Algorithms and Architectures-SPAA. 2010. 209−216. [doi: 10.1145/1810479.1810521]

[13] Wang Q, Korkin D, Shang Y. A fast multiple longest common subsequence (MLCS) algorithm. IEEE Trans. on Knowledge and Data Engineering, 2011,23(3):321−334. [doi: 10.1109/TKDE.2010.123]

[14] Ann HY, Yang CB, Tseng CT. Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion. Journal of Combinatorial Optimization, 2014,28(4):800−813. [doi: 10.1007/s10878-012-9588-2]

[15] Kruskal JB. An overview of sequence comparison: Time warps, string edits, and macromolecules. Siam Review, 1983,25(2): 201−237. [doi: 10.1137/1025045]

[16] Papachristou D, Baker SD. Longest-Common-Subsequence detection for common synonyms. U.S. Patent 8,001,136B1, 2011.

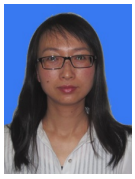
[17] Chandola V, Banerjee A, Kumar V. Anomaly detection for discrete sequences: A survey. IEEE Trans. on Knowledge and Data Engineering, 2012,24(5):823−839. [doi: 10.1109/TKDE.2010.235]

[18] Liao HJ, Yang YB, Tang LZ. Research on Levenshtein distance applying in computer-automated scoring. Journal of Guangzhou University (Natural Science Edition), 2012,11(4):79−83 (in Chinese with English abstract).

[19] Ballarin C. Tutorial to locales and locale interpretation. In: Lámban L, Romero A, Rubio J, eds. Contribuciones Científicas en Honor de Mirian Andrés. Servicio de Publicaciones de la Universidad de La Rioja, 2010. 123−140.

附中文参考文献:

[18] 廖宏建,杨玉宝,唐连章.改进的编辑距离计算及其在自动评分中的应用.广州大学学报(自然科学版),2012.11(4):79−83



宋丽华(1976－),女,河北高碑店人,博士,副教授,主要研究领域为机器定理证明,网络测量与性能分析.

季晓君(1979－),女,博士,讲师,主要研究领域为网络安全,形式化分析与验证.

王海涛(1976－),男,博士,副教授,主要研究领域为计算机网络,无线自组网,应急通信.

张兴元(1965－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为形式化方法机器定理证明.