离散数学 期中课题报告

WFF 'N PROOF: The Game of Modern Logic 初探

fa_555

2022 年 5 月 11 日

目录

一、 概述

WFF 'N PROOF: The Game of Modern Logic 是一部由 Layman Allen 制作并发行于 1961 年的桌面游戏（下文简称“游戏”）。游戏基于离散数学中的命题逻辑领域，能够提升游玩者运用命题逻辑基本概念、进行抽象证明与解决实际问题的能力[1]。

游戏现存 21 个子游戏（早期版本有 24 个[2]），基于一种名为“WFF”（Well Formed Formulas 的缩写）的命题逻辑表达式的表记形式。前 2 个游戏中，玩家需构造前缀形式的 WFF；后 19 个游戏中，玩家通过 Fitch's method[3] 构造对于指定结论的证明[2]。

本文着重介绍了游戏中符号体系 WFF 'N PROOF 的基本原理及其衍生出的应用，还介绍了作者前人论文中出现的子游戏。

二、 WFF 'N PROOF 的基本内容

1. WFF 的基本构成

WFF 实际上是**前缀形式**的命题表达式，由**命题变量**和**逻辑运算符**组成[4]。

- 命题变量

  与离散数学中相同，多为以 `p` 为首的多个小写拉丁字母，其具体数量可能随具体版本不同。

  本文中我们认为有 4 个命题变量，分别为 `p`, `q`, `r` 和 `s`。每个命题变量都可以赋真值 T 或 F 之一。

- 运算符

  此处的逻辑运算符形式和离散数学中有较大区别，以 `C`, `A`, `K`, `E` 和 `N` 五个大写拉丁字母表示。其中前四个为双目运算符，最后一个为单目运算符。

- 命题表达式构造规则

  根据 WFF 前缀表达式的性质，有以下定义：

  1. `p`, `q`, `r` 和 `s` 是 WFF；
  2. 如果表达式 w 是 WFF，U 是单目运算符，那么 Uw 是 WFF；
  3. 如果表达式 w 和 x 均是 WFF，B 是双目运算符，那么 Bwx 是 WFF。

按照定义，WFF 的五种运算符的真值表如下：

p	q	Cpq	Apq	Kpq	Epq
T	T	T	T	T	T
T	F	T	T	F	F
F	T	F	T	F	F
F	F	T	F	F	T

p	Np
T	F
F	T

需要注意的是，前缀表达式中不需要考虑运算符优先级的问题。

把若干运算进行符合可以得到更加复杂的表达式。原作者论文中给出了这样一个例子[5]：

例

EKApqNpNCqp

将它转写为现在常用的中缀表达式形式为

$$((p \lor q) \land \neg p) \leftrightarrow (\neg(q \to p))$$

值得一提的是，这是一个重言式。

2. WFF 与当今常用表记法的对应关系

常用表记中，为便于阅读和理解，表达式一般采取中缀形式（而不同于 WFF 采用的前缀形式）。这就是说，对于双目运算符 $\oplus$ 和操作数 L, R，一般写作 $L \oplus R$ 而非 $\oplus LR$。

对于 WFF 中的运算，通过上述真值表，分别可以得到与常用中缀表达式的对应关系：

WFF 前缀表达式	常用中缀表达式
Cpq	$p \to q$
Apq	$p \lor q$
Kpq	$p \land q$
Epq	$p \leftrightarrow q$
Np	$\neg q$

3. WFF 'N PROOF 中介绍的推理规则

WFF 'N PROOF 中介绍并编号了一些常用的推理规则[4]。它们的形式和对应的中缀表达式如下表右栏（省略了由交换律立即可得的条目）。此外，在游戏过程中，有人会使用其他的二级规则，因数量繁多，此处略去不表。

编号	WFF 前缀表达式	常用中缀表达式
Co	`KCpqp` $\implies$ `q`	$(p \to q) \land p \implies q$
Ci [C 的定义]		
Ao	`KKCpsCqsApq` $\implies$ `s`	$(p \to s) \land (q \to s) \land (p \lor q) \implies s$
Ai	`p` $\implies$ `Apq`	$p \implies (p \lor q)$
Ko	`Kpq` $\implies$ `p`	$(p \land q) \implies p$
Ki [K 的定义]		
Eo	`Epq` $\implies$ `Cpq`	$(p \leftrightarrow q) \implies (p \to q)$
Ei	`KCpqCqp` $\implies$ `Epq`	$(p \to q) \land (q \to p) \implies (p \leftrightarrow q)$
No	`KCqNpNq` $\implies$ `p`	$(q \to \neg p) \land \neg q \implies p$
Ni	`KCqpNq` $\implies$ `Np`	$(q \to p) \land \neg q \implies \neg p$
Rp	`p` $\implies$ `Cqp`	$p \implies (q \to p)$

三、 *WFF 'N PROOF: The Game* 中的问题实例

1. 拼凑 WFF

在游戏中，每掷一次骰子，玩家将得到若干个命题和运算符，此时玩家需要将它们拼凑成一个合法的 WFF。本例改变自 POJ 的一道算法题[6]，即是基于这个背景，同时这可能是 *WFF 'N PROOF: The Game* 中的第一个游戏。

例

给出一个任意大小的包含若干命题和运算符的可重集合，其中命题变量在 `p`, `q`, `r` 和 `s` 中，运算符在 `C`, `A`, `K`, `E` 和 `N` 中，所有元素（命题和运算符）的含义均和前文相同。

假定每个元素的长度都是 `1`，计算由这个可重集合中的元素能组成的最长的合法的 WFF 的长度，并给出任意一个该长度的 WFF。

解答

我们不关心构造出的 WFF 的具体内容，只关心它的长度。因此我们将所有元素按照性质分为三类：命题变量、双目运算符、单目运算符，而不需要考虑它们具体是什么。

容易发现，只要命题变量的个数不为零，那么单目运算符就一定可以全部追加到某个合法 WFF 之前。这样我们只需处理另外两类元素。

另外两类元素的数量是相互制约的。每个双目运算符一定有两个操作数，将其看做一个二叉树，则有且仅有叶子节点是命题变量，非叶子节点都是双目运算符。

设多重集中和合法 WFF 中命题变量的数量分别为 V, V'，双目运算符的数量为 B, B'，则根据上述说明，有 $V' = B' + 1$。因此可取 $V' = \min(V, B + 1)$, $B' = \min(B, V - 1)$。

由此，任取原可重集内 V' 个命题变量和 B' 个双目运算符，按照双目运算符形成一条链的特殊情况构造，最后连接上所有单目运算符即可。时间复杂度与原多重集合的大小成线性。

以下是该算法的 `C++` 实现。

```cpp
// since C++20

#include <array>
#include <cassert>
#include <iostream>
#include <string>

constexpr char kSigma[] = "pqrsCAKEN";

constexpr std::array<int, 123> prework() {
  std::array<int, 123> ret;
  // the ascii of the maximum visible character is 122
  for (std::size_t i = 0; i < std::size(kSigma); ++i)
    ret[kSigma[i]] = i;
  return ret;
}

int main() {
  constexpr auto z = prework();

  std::string s;
  std::array<int, 10> cnt = {};
  std::array<int, 3> type = {};
  /*
```

```
   * cnt[i]: the amount of kSigma[i]
   * type[]: the amount of variables, binary operators,
   *   and unary operators respectively
   */

  std::cin >> s;
  for (int c : s) {
    assert(z[c] || c == *kSigma);
    ++cnt[z[c]];
    ++type[z[c] / 4];
  }
  if (!type[0]) {
    std::cout << "No WFF possible." << std::endl;
    return 0;
  }

  std::string ans(cnt[z['N']], 'N');
  if (type[1] > type[0] - 1)
    type[1] = type[0] - 1;
  if (type[0] > type[1] + 1)
    type[0] = type[1] + 1;
  for (int i = 7; i >= 0; --i)
    while (cnt[i] && type[i / 4]) {
      --cnt[i];
      --type[i / 4];
      ans += kSigma[i];
    }
  std::cout << ans << std::endl;
}
```

2. 转换成中缀形式并判断重言式

WFF 所采用的前缀表达式在当今看来是不容易阅读和理解的，因此我们尝试将其转换为中缀表达式。

同时，我们常要通过 WFF 证明复杂命题。假设 w, x 均为 WFF，则 $w \implies x$ 成立等价于 $\mathtt{C}wx$ 是重言式。由于游戏中至多有 4 个命题变量，通过直接计算真值表的方法判断命题是否为重言式是简便的。

本例改编自 POJ 的一道算法题[7]，即是基于上述两个背景。

例

给出一个合法的 WFF，其中元素的组成与上例完全相同。判断这个 WFF 是否为重言式。

解答

这是一个经典的前缀表达式与中缀表达式互换的问题，可以考虑建出表达式树，在此不过多赘述，复杂度与 WFF 的长度的平方成线性。

在编程解决问题时，我们直接用递归模拟建树过程。同时受文章篇幅的限制，我们分别用 `implies(p, q)` 和 `p == q` 代替 $p \to q$ 和 $p \leftrightarrow q$，以此缩短代码行数。以下是该算法的 python 实现。

```python
#! /usr/bin/env python3
# -*- coding: utf-8 -*-
# since python3.6; pardon me for my violating PEP-8

kBinary = {'K': 'and', 'A': 'or', 'E': '=='}
kUnary = {'N': 'not'}
kFunctional = {'C': 'implies'}

def implies(p: bool, q: bool) -> bool:
  return not p or q

def transform() -> str:
  global s, pos
  if pos >= len(s):
    raise ValueError
  if s[pos].islower():
    pos += 1
    return s[pos - 1]
  else:
    op = s[pos]
    pos += 1
    if op in kUnary:
      param = transform()
      return f'({kUnary[op]} {param})'
    else:
      param1 = transform()
      param2 = transform()
      if op in kBinary:
```

```python
        return f'({param1} {kBinary[op]} {param2})'
      elif op in kFunctional:
        return f'{kFunctional[op]}({param1}, {param2})'
      else:
        raise ValueError

def check(expr: str) -> bool:
  for p in False, True:
    for q in False, True:
      for r in False, True:
        for s in False, True:
          if not eval(expr):
            return False
  return True

def main() -> None:
  global s, pos
  s, pos = input(), 0
  try:
    expr = transform()
    print(f'{expr} is{"" if check(expr) else " not"} tautology')
  except ValueError:
    print('Not a valid WFF')

if __name__ == '__main__':
  main()
```

3. 晚点公交问题

这是一个来自前人的关于 WFF 'N PROOF 的论文的例子[8]，这个问题用于命题逻辑教学，和原作游戏的目的相同。

例

给出以下三个前提：

(1) 如果 Bill 乘公交车，如果公交车晚点，那么 Bill 会错过面试。

(2) 如果 (a) Bill 错过面试，且 (b) Bill 不高兴，那么 Bill 不该回家。

(3) 如果 Bill 没找到工作，那么 (a) Bill 不高兴，且 (b) Bill 应该回家。

能否证明以下结论？

Q1 如果 Bill 乘公交车，如果公交车晚点，那么 Bill 能找到工作。

Q2 如果 (a) Bill 错过面试，且 (b) Bill 应该回家，那么 Bill 找到了工作。

Q3 如果公交车晚点，那么 (a) Bill 不乘公交车，或者如果 (b) Bill 没找到工作，那么 Bill 不会错过面试。

Q4 如果 (a) 公交车晚点，且 (b) Bill 没找到工作，那么 Bill 没有乘公交车。

Q5 如果 Bill 没有乘公交车，那么 (a) Bill 应该回家，并且 (b) Bill 没找到工作。

Q6 如果 (a) 公交车晚点，或 (b) Bill 错过了面试，那么 Bill 不高兴。

Q7 如果 Bill 找到了工作，那么 (a) Bill 不会不高兴，或 (b) Bill 不应该回家。

Q8 如果 (a) Bill 应该回家，且 Bill 乘了公交车，那么 (b) 如果公交车晚点，那么 Bill 不会不高兴。

解答

首先，我们假设“Bill 没有错过面试”和“Bill 找到了工作”等价（互为充要条件），否则这个问题将失去意义。

首先我们将出现的所有事件用命题变量表示：

命题变量	意义
`p`	Bill 乘公交车
`q`	公交车晚点
`r`	Bill 错过面试 / Bill 没有找到工作
`s`	Bill 不高兴
`t`	Bill 应该回家

接下来我们尝试用 WFF 和已有的 5 个命题变量表达出所有的 3 个前提和 8 个结论。其中对于 3 个前提，我们同时写出与之对应的中缀表达式写法以便对照。

序号	WFF 表示	中缀表示
(1)	CpCqr	$p \to (q \to r)$
(2)	CKrsNt	$(r \wedge s) \to \neg t$
(3)	CKrsNt	$r \to (s \wedge t)$

序号	WFF 表示	序号	WFF 表示
Q1	CpCqNr	Q5	CNpKtr
Q2	CKrtNr	Q6	CAqrs
Q3	ACqNpCrNr	Q7	CNrANsNt
Q4	CKqrNp	Q8	CKtpCqNs

三个前提的析取为 `KKCpCqrCKrsNtCrKst`。据此，“结论 Q1 能被证明”等价于 `KKCpCqrCKrsNtCrKst` $\Longrightarrow$ `CpCqNr` 成立，又等价于 `CKKCpCqrCKrsNtCrKstCpCqNr` 是重言式。

同样地，对于所有的 8 个结论，我们可以写出其对应的命题，使得其能被证明当且仅当其对应的命题是重言式。

序号	对应命题
Q1	CKKCpCqrCKrsNtCrKstCpCqNr
Q2	CKKCpCqrCKrsNtCrKstCKrtNr
Q3	CKKCpCqrCKrsNtCrKstACqNpCrNr
Q4	CKKCpCqrCKrsNtCrKstCKqrNp
Q5	CKKCpCqrCKrsNtCrKstCNpKtr
Q6	CKKCpCqrCKrsNtCrKstCAqrs
Q7	CKKCpCqrCKrsNtCrKstCNrANsNt
Q8	CKKCpCqrCKrsNtCrKstCKtpCqNs

通过前一例中的给出的算法和代码，我们可以简便地判断以上的 8 个命题分别是否为重言式（注意此处有 5 个命题变量，所以需要对代码进行适当的调整）。根据运行结果，只有 Q5, Q6, Q7 不是重言式。由此我们得到，结论 Q1, Q2, Q3, Q4, Q8 能被证明；Q5, Q6, Q7 不能被证明。

四、 总结与致谢

至此，本文介绍了游戏中符号体系 WFF 'N PROOF 的基本原理及其衍生出的一些应用。根据最后一个例子可以看出，*WFF 'N PROOF: The Game of Modern Logic* 中的游戏大都十分有技巧性，难以用朴素方法解决。可能由于其具有的竞技性，该例的作者声称其受到了当时的学生们的喜爱[8]。至于该游戏为何能够训练学生使用命题逻辑解决问题的能力，从中也可见一斑。

最后，感谢中央民族大学的 07 灵契（化名）同学、西安交通大学的 ucw（化名）同学、中国矿业大学的寒鸽儿（化名）同学帮助我获取（下载）了以下众多参考文献中部分的内容。没有你们，这篇文章就不会有如此详尽的参考资料。

参考文献

[1] Layman E. Allen, Robert W. Allen, and James C. Miller. Programed games and the learning of problem-solving skills: The wff ’n proof example. *Journal of Educational Research*, 60:22–26, 1966.

[2] F. C. Oglesby. *The Journal of Symbolic Logic*, 30(1):105–105, 1965.

[3] Wikipedia contributors. Fitch notation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Fitch_notation&oldid=965316869`, 2020. [Online; accessed 10-May-2022].

[4] LA TRELLE PIERRE. Basic wff’n proof: A teaching guide. `https://www.oercommons.org/authoring/1364-basic-wff-n-proof-a-teaching-guide/view`, Jan. 2016.

[5] P. C. Rosenbloom. *The Mathematics Teacher*, 57(5):346–347, 1964.

[6] Waterloo Local Contest. Poj 3290 – wff ’n proof. `http://poj.org/problem?id=3290`, Sept. 2006.

[7] Waterloo Local Contest. Poj 3295 – tautology. `http://poj.org/problem?id=3295`, Sept. 2006.

[8] James Jeffryes. Let’s play ”wff’n proof”. *The Mathematics Teacher*, 62(2):113–117, 1969.