Software Engineering and Applications 软件工程与应用, 2020, 9(1), 14-21
Published Online February 2020 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2020.91003

Hans汉斯

Tactics for Proving Separation Logic Assertions in Coq

Siran Lei, Mengqi Cheng, Jianguo Jiang*

School of Mathematics, Liaoning Normal University, Dalian Liaoning
Email: siranlei0922@gmail.com, cmq215box@sina.com, *jjgbox@sina.com

Received: Dec. 23rd, 2019; accepted: Jan. 6th, 2020; published: Jan. 13th, 2020

Abstract

The verification of the correctness of large programs is an unmanageable but important endeavor. We are interested in verifying C programs with formal methods; the logic is separation logic, a Hoare-style program logic. In this paper, we present a simple extension of the syntax of separation logic assertion on existing verification system in Coq proof assistant to make assertions more versatile and flexible to describe the state of programs. Moreover, we develop several tactics for proving some related assertions to reduce manual proof as much as possible and improve the efficiency of verification.

Keywords

Formal Method, Program Verification, Automated Reasoning, Coq, Interactive Theorem Proving

分离逻辑断言的Coq证明策略

雷斯然，程梦奇，江建国*

辽宁师范大学数学学院，辽宁 大连
Email: siranlei0922@gmail.com, cmq215box@sina.com, *jjgbox@sina.com

收稿日期：2019年12月23日；录用日期：2020年1月6日；发布日期：2020年1月13日

摘　要

大型程序的验证是一项十分复杂但又极其重要的工作。本文以使用Hoare风格的分离逻辑验证C程序为目的，在基于Coq的现有的验证系统中，添加分离蕴含以扩展其分离逻辑断言语法，使得分离逻辑断言可以更灵活的描述程序状态。此外，我们开发了一些相关的自动证明策略，尽可能的通过减少人工证明来

*通讯作者。

文章引用: 雷斯然, 程梦奇, 江建国. 分离逻辑断言的 Coq 证明策略[J]. 软件工程与应用, 2020, 9(1): 14-21.
DOI: 10.12677/sea.2020.91003

提高验证效率。

关键词

形式化方法，程序验证，分离逻辑，Coq，交互式定理证明

1. 引言

随着科技的发展，软件系统的应用日益广泛，其正确性也越来越重要，尤其是安全攸关[1]，商业攸关以及任务攸关等系统。由于 C 语言具有编译效率高、可移植性好等特点，大多数软件系统较底层的开发是由它实现的。验证 C 程序的正确性可采用形式化方法，该方法是通过使用数学方法，对软件或硬件系统进行描述、开发和验证的一种技术。然而，大型 C 程序的验证是一个十分巨大且困难的工作，并且该问题已被证明是一个不可判定性问题，即不可以完全自动化验证。因此，交互式的证明 C 程序的正确性是一个很好的选择。

自 20 世纪 60 年代以来，Hoare 提出的 Hoare 逻辑是一直被广泛应用的程序验证逻辑系统[2]，用于对命令式程序进行推理验证，是程序逻辑研究领域的一个重大进展。但是 Hoare 逻辑对于带有指针的程序的验证仍然十分繁琐。

2002 年，Reynolds 和 O'Hearn 将 Hoare 逻辑拓展为可推理验证操作动态分配内存程序的分离逻辑[3] [4]，进一步发展了程序逻辑理论。近二十年以来，分离逻辑已被拓展至多种版本，其使用范围更加广泛，在程序验证中占据着越来越重要的地位。分离逻辑具有更强的表达力和验证能力，例如并发程序的验证等。然而表达力越强的程序逻辑，其验证过程越复杂。分离逻辑已被证明是一个不可判定的程序逻辑，因此使用分离逻辑验证 C 程序的过程无法达到完全自动化。然而大型的 C 程序证明十分复杂，手动证明不但需要大量的人力物力，可信度也不是很高，因此人机交互式的证明 C 程序是不可避免的。

在过去的几十年中，有关交互式定理证明器实现与应用分离逻辑的进展有很多，例如[5] [6] [7] [8] [9]。Coq [10]是一个被广泛应用的人机交互式的定理证明器，其基本理论是归纳结构的演算。一些由[11]提供的 Coq 实用策略也是用于基于分离逻辑的 C 程序验证，这些高度自动化的策略可以仅使用单行证明脚本完成对特定 C 程序的验证，例如，该系统只需使用 1 行仅有 4 个单词的证明脚本来证明经典的就地反转链表算法，而 McCreight 的 Coq 策略[12]需要 68 行 400 个单词，Charge 的策略[13]需要 25 行 105 个单词。本文将分离逻辑中的分离蕴含运算符添加到该系统的分离逻辑断言语法中，并提供一些相关的自动证明策略。

2. 程序逻辑和证明助手

2.1. Hoare 逻辑

Hoare 逻辑是使用严格的数理逻辑推理来验证命令式程序正确性的逻辑系统，它使用逻辑断言来描述程序的状态。

Hoare 使用霍尔三元组来描述程序的行为，是 Hoare 逻辑的中心特征，其形式如下：

$$\{P\}\ c\ \{Q\},$$

其中断言 P 是前置条件，断言 Q 是后置条件，c 是程序。前置条件和后置条件是分别描述执行程序 c 前后的程序状态的断言。一个有效的霍尔三元组表示：如果程序 c 执行前，内存状态满足断言 P，那么程序执行后，内存状态满足断言 Q。考虑以下霍尔三元组：

$$\{x=3\}\ x:=x*2\ \{x>4\},$$

它显然是有效的，由于程序 $x:=x*2$ 执行前 x 的值为 3，执行后可得出 x 的值为 6，因此 $x>3$ 为真。然而上述霍尔三元组并不够精准，如果更换其后置条件为 $x>5\wedge x<10$，则霍尔三元组为 $\{x=3\}\ x:=x*2\ \{x>5\wedge x<10\}$，因为 $x=6$ 蕴含 $x>5\wedge x<10$，所以该霍尔三元组也是有效的。显然 $x>5\wedge x<10$ 相比于 $x>3$ 更加精准，但是相对于 $x=6$ 还是要弱一些，像 $x=6$ 这样的后置条件通常被称之为最强后置条件。一般的，对于给定的霍尔三元组{P} c {Q}，对于所有的 Q′，如果有{P} c {Q′}，且 Q ⇒ Q′，则称 Q 为对应的 P 和 c 的最强后置条件。与最强后置条件类似，如果对于所有的 P′，有 t {P′} c {Q}，且 P′ ⇒ P，那么称 P 为最弱前置条件。在霍尔逻辑中，下面的规则可以根据对应的语句与后置条件生成最弱前置条件：

$$\frac{}{\{Q[a/x]\}\ x:=a\ \{Q\}} \quad (\text{赋值规则})$$

$$\frac{\{P\}\ c_1\ \{Q\}\quad \{Q\}\ c_2\ \{R\}}{\{P\}\ c_1;c_2\ \{R\}} \quad (\text{顺序规则})$$

$$\frac{\{P\}\ c_1\ \{Q\}\quad \{P\}\ c_2\ \{Q\}}{\{P\}\ \textbf{IF}\ b\ \textbf{THEN}\ c_1\ \textbf{ELSE}\ c_2\ \{Q\}} \quad (\text{条件规则})$$

$$\frac{\{P\wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{WHILE}\ b\ \textbf{DO}\ c\ \textbf{END}\ \{P\wedge\neg b\}} \quad (\text{循环规则})$$

其中 Q[a/x]表示使用 a 替换 Q 中 x 的每次自由出现。此外，推论规则(如下)可以加强前置条件或者减弱后置条件：

$$\frac{P'\Rightarrow P\quad \{P\}\ c\ \{Q\}\quad Q\Rightarrow Q'}{\{P'\}\ c\ \{Q'\}}$$

自 1960 年起，Hoare 逻辑被广泛的应用于命令式程序语言的验证中，然而，在验证带有指针的程序时，其证明过程仍然十分复杂，极具挑战性。2002 年，分离逻辑作为霍尔逻辑的拓展，简化了此类证明过程。

2.2. 分离逻辑

分离逻辑是 Hoare 霍尔逻辑的拓展，它引入了两个重要的运算符分别是分离合取(∗)和分离蕴涵(−∗)，它们的形式语义如下：

$$s,h\models p_0 * p_1 \text{ iff } \exists h_0,h_1$$
$$h_0\perp h_1 \text{ and } h_0\cdot h_1=h \text{ and } s,h_0\models p_0 \text{ and } s,h_1\models p_1$$

其中 s 表示栈，h 表示堆，$h_0\perp h_1$ 表示堆 h_0 和 h_1 不相交，$h_0\cdot h_1$ 表示堆 h_0 和 h_1 的并集。[P * Q] s h 断言整个堆 h 分为两个不相交的部分 h_0 和 h_1，对于子堆 h_0，断言 P 为真，对于子堆 h_1，断言 Q 为真。

$$s,h\models p_0 -\!\!* p_1 \text{ iff } \forall h'$$
$$\left(h'\perp h \text{ and } s,h'\models p_0\right) \text{implies } s,h\cdot h'\models p_1$$

也就是说，如果堆的扩展 h'满足断言 p_0，并且断言 p_0 对 h'是正确的，则对于扩展后的堆 $h \cdot h'$ 满足断言 p_1。

分离逻辑通过表达显示分离的逻辑连接词以及相应的规则消除了共享的可能，具有较强的表达能力。分离逻辑用于程序验证，简化了程序的验证工作，例如带有数组和指针的程序，并支持局部推理，其分离思想和对应的框架规则使得局部推理可以很好地应用到程序验证中。较比 Hoare 逻辑，分离逻辑具有更强的表达能力与验证能力，因此继 Hoare 逻辑之后，分离逻辑成为程序验证的一种重要方法。

2.3. Coq 证明助手

交互式定理证明工具 Coq，为交互式形式化验证提供了一个丰富的环境。Coq 的规范语言是 Galina，它支持开发数学定理以及证明程序规范，其命令语言为 *vernacular*。

通常，Coq 是一个开发数学证明的环境，其应用包括法伊特－汤普森(Feit-Thompson)定理的证明，实现了 CompCert 编译器[14]以及 VST [11] [15]等。它虽然是交互式的定理证明器，不能达到完全自动化，但是它拥有着强大的判定过程以及自动化证明策略库，还有一种编写自动证明策略的语言，称为 Ltac [16]。本文旨在使用 Coq 证明助手验证 C 程序的正确性，其中的证明策略使用 Ltac 语言实现。

3. 分离逻辑断言

分离逻辑引入分离合取和分离蕴涵等运算符扩展断言语法，从而可以对堆进行更简洁而灵活的描述。本节将添加分离蕴含到原有系统中，扩展系统的断言语法[17]。

下面阐述如何逐步将分离蕴含添加到原有系统中，首先，扩充原有断言语法如下：

扩充前 :(Asrt) p ::=
emp|true|false|a↦v|e=v|x@a
|p*p|Exx.p|p∧p|p∨q|⟨p⟩
⇓
扩充后 :(Asrt) p ::=
emp|true|false|a↦v|e=v|x@a
|p*p|p--*p|Exx.p|p∧p|p∨q|⟨p⟩

在 Coq 中，添加构造子 Awand，这意味着将分离蕴含加入到 Asrt 的归纳定义中：

```
| Awand (P:asrt)(Q:asrt)
```

随后定义

```
Infix '' --* '':= Awand.
```

意味着 P −∗ Q 代表 (Awand P Q)。下一步，在 Coq 中描述 s |= P −∗ Q 的含义如下：

```
exists M1 M2 M,
 M = get_mem s ∧ MemMod.join M1 M2 M ∧
 (sat(substmem s M1)p → sat(substmem s M)q)
```

其中 (sat s P) 表示 s| = P，即内存 s 满足断言 P。显然，M，M1 和 M2 代表堆，并且参照章节 2.2 中[P −∗ Q] s h 的定义，(MemMod.join M1 M2 M)表示 M1 ⊥ M2 并且 M1·M2 = M 。

当然，仍有一些关于添加分离蕴含到系统中的细节，我们在这里不关注。

4. 策略

从本节开始，我们将在 Coq 中证明工作中使用的部分推理规则的可靠性并介绍证明带有分离蕴含的分离逻辑断言的策略。

4.1. 推理规则

以下是两个重要的推理规则，分别称为柯灵规则和柯灵逆反：

$$\frac{P * Q \Rightarrow R}{P \Rightarrow (Q \text{ --}* R)} \text{ (柯灵规则)} \quad \frac{P \Rightarrow (Q \text{ --}* R)}{P * Q \Rightarrow R} \text{ (柯灵逆反规则)}$$

柯灵规则及其逆反规则表述了分离合取和分离蕴涵之间的关系，并且由此可以推导出一系列推理规则。我们从几个基本示例开始阐述如何在 Coq 中证明它们是正确的。考虑下面的推断规则，它可由柯灵规则推出：

$$\frac{}{\text{emp} \Rightarrow (P \text{ } -\!\!* P)}$$

证明过程如下：

```
Theorem awand_emp : forallP, emp⇒P--*P.
```

Theorem 是一个命令，它表示正确性证明的开始，其证明过程从以下状态开始：

```
1subgoal

=================
forall   P  :  asrt,  emp⇒P--*P
       Proof. introP.

       P:asrt
       =================
       emp⇒P--*P
```

proof 是指启动证明脚本，而 intro P 将 P 转变为自由变量。这里，我们可以应用柯灵规则如下：

```
apply CURRYING with (Q:=P)(R:=P).
P:asrt
================
emp*P⇒P
apply  astar_elim.
```

astar_elim 是一个证明定理，其内容是：对所有 P，emp∗P⇒P 成立。应用 astar_elim 后，则不再有子目标，这意味着证明已完成。

```
Qed.
```

最后，**Qed** 检查证明是否完成。

注意到上面证明的规则可以被用来消去一些分离蕴含，那么以这样一个简单的设想为例，实现一个简单的策略，以展示如何使用交互式的机器证明代替手动证明。该策略称为 sclearwand：

```
Ltac sclearwand:=
match goal with
 ||-?s |= ?A⇒ match find_awand A with
              | some ?n⇒sep_lift n;
                match goal with
                | H : _ |=_ |-_|=(?B --* ?B) * ?C ⇒
                  apply astar_l_aemp_intro in H;
                  eapply astar_mono;[eapply awand_emp |
                  intros s'H';exact H'| auto];sclearwand
                |_⇒idtac
                end
              |_⇒idtac
              end
|_⇒fail
end.
```

首先匹配目标的模式，如果目标是类似于第 2 行的模式，则在第 3 行中，调用 find_awand 以查找 A 中分离蕴涵的位置。在第 3 行中，sep_lift 将其移至 A 的左侧，随后证明目标将在第 5~11 行中解决，其中 astar_l_aemp_intro 和 astar_mono 分别是如下规则：

$$\frac{}{P \Rightarrow emp * P} \qquad \frac{P \Rightarrow P' \quad Q \Rightarrow Q'}{P * Q \Rightarrow P' * Q'}$$

sclearwand 可以消除(P −∗ P)，之后“emp”将在证明目标中取代它，例如：

```
Goal forall(A B:asrt),A ⇒ A*(B-*B).
Proof. intros. sclearwand. Qed.
```

现在，考虑下面的推理规则并在 Coq 中证明其整确定性

$$\frac{P' \Rightarrow P \quad Q \Rightarrow Q'}{P \text{ --* } Q \Rightarrow P' \text{ --* } Q'}$$

```
Theorem awand_pro:forallP P'Q Q', P'⇒P→Q⇒Q'→P--*Q⇒P'--*Q'.
Proof.
  intros. simplin*; mytac.
  do3eexists; mytac; eauto.
Qed.
```

mytac 是原有系统中用于断言的自动证明策略，而上面的规则可以使用 mytac 证明，说明分离蕴含已被很好的添加到系统中。

仍然有很多关于分离蕴含的推理规则如下，在这里省略对他们的可靠性证明：

$$\frac{}{Q * (Q \text{ --* } P) \Rightarrow P} \qquad \frac{}{P \Rightarrow Q * (Q \text{ --* } P)}$$

$$\frac{}{(P * R) \Rightarrow (P * (Q \text{ --* } (Q * R)))}$$

$$\frac{P_0 \Rightarrow (Q \text{ --* } R) \quad P_1 \Rightarrow (R \text{ --* } S)}{P_0 * P_1 \Rightarrow (Q \text{ --* } S)}$$

$$\frac{P \Rightarrow P'}{P * (P' \text{ --* } Q) \Rightarrow Q}$$

4.2. 资源检测

在[5]提供的实用策略可以仅使用一行证明脚本证明整个程序，即“repeat hoare forward; sep pure”，其中 repeat 表示重复执行某项策略，这里指重复执行“hoare forward; sep pure”，sep pure 是证明关于纯断言的策略，而 hoare forward 可以证明霍尔三元组⊢{P} c {Q}，是验证程序的的关键。

hoare forward 的第一步是资源检测(check_resource)，它检测前置条件是否具有语句执行所需的信息。例如：

$$\{y \rightarrowtail \_\} \; y := x \; \{x \rightarrowtail \_ * y \rightarrowtail \_\}$$

其中 $x \rightarrowtail v$ 表示变量 x 的值为 v。显然，如果将 x 的值赋给 y 我们首先需要 x 的值。这里，resource_check 在前置条件中没有找到有关 x 的信息，它将返回一条用户友好的消息，用户可以根据将相应的信息添加前置条件中，如下：

$$\{x \rightarrowtail \_ * y \rightarrowtail \_\} \; y := x \; \{x \rightarrowtail \_ * y \rightarrowtail \_\}$$

这就像开始验证前的预处理一样。然而，尽管如上一节所述分离蕴含已经被添加到系统中，并且前

置条件中具有足够的信息，resource_check 也无法检测到有关分离蕴含的信息，例如：

$$\{x\rightarrow\_ * (x\rightarrow v_1 \text{ --* } y\rightarrow\_)\}\, x := v_1; y := v_2\, \{ y\rightarrow v_2 \}$$

前置条件中虽然存在 $y \rightarrowtail \_$，但无法被识别。因此，我们需要调整 resource_check 的组件使得分离蕴含可以被识别并可以从中检测到需要的信息。

考虑 resource_check 的组件 find_match_retV，它可以识别“$*$”，但无法识别“$-\!*$”，如下：

```
Ltac find_match_retV Hp x:=
 1  match Hp with
 2    |?A *?B⇒ match find_match_retV A x with
 3            |some?v⇒constr:(some v)
 4            |_⇒ match find_match_retV B x with
 5                |some?v⇒ constr:(some v)
 6                |_⇒constr:(@None)
 7                end
 8             end
 9    |?A ⇒(*hoare_tactics.v*)
10  end.
```

将其更改如下：

```
Ltac find_match_retV' Hp x:=
 1  match Hp with
 2    |?A * ?B⇒ match find_match_retV A x with
 3              |some?v⇒constr:(some v)
 4              |_⇒ match find_match_retV B x with
 5                  |some?v⇒ constr:(some v)
 6                  |_⇒constr:(@None)
 7                  end
 8              end
 9    |?C --*?D⇒ match find_match_retV C x with
10              |some?v⇒ constr:(some v)
11              |_⇒(* hoare_tactics.v*)
12                  match find_match_retV D x with
13                  |some?v⇒constr:(some v)
14                  |_⇒constr:(@None)
15                  end
16              end
17    |?A⇒(*hoare_tactics.v*)
18  end.
```

将 resource_check 中的 find_match_retV 替换为 find_match_retV'之后，上述$y \mapsto _$即可被识别，因此 hoare forward 可以正常开始运行。

5. 一个简单的例子

本节将给出一个简单的示例来表明，我们可以证明程序状态由含有分离蕴含的断言描述的程序规范。

sclearwand in 是可以证明含有分离蕴含的断言的策略，例如，sclearwand in 策略可以证明下面的证明目标：

```
Goal forall(A B D E:asrt),
   (A * D * B) * ((D * A * B) --* E)⇒ E.
 Proof. intros. sclearwand_in H. Qed.
```

我们以一个简单的例子来说明它在程序验证中的作用，如下：

$$\{x\rightarrow v_1 * (x\rightarrow null \text{ --* } y\rightarrow v_2)\}$$
$$x := null;$$
$$\{y\rightarrow v_2\}$$

这里，运行“repeat hoare forward; sep pure”后得到一个子目标：

$$(x \rightarrowtail null * (x \rightarrowtail null \text{ --}* y \rightarrowtail v_2)) \Rightarrow (y \rightarrowtail v_2)$$

运行“sclearwand in H”，该子目标被解决，证明完成。

6. 总结

我们通过添加分离蕴含操作符到已有系统的分离逻辑断言语法中，拓展了该验证系统，并提供了一些相关的自动证明策略。但是由于分离逻辑断言的复杂多样性，我们现有的策略不足以解决包含分离蕴涵的断言的所有证明。将来，我们将扩展自动证明策略以提高程序验证的效率。

参考文献

[1] Hill, J. and Tilley, S. (2010) Creating Safety Requirement Traceability for Assuring and Recertifying Legacy Safety-Critical Systems. 2010 18*th IEEE International Requirements Engineering Conference*, Sydney, 27 September-1 October 2010, 297-302. https://doi.org/10.1109/RE.2010.42

[2] Hoare, C.A.R. (1969) An Axiomatic Basis for Computer Programming. *Communications of the ACM*, **12**, 576-580.

[3] O'Hearn, P.W., Reynolds, J.C. and Yang, H. (2001) Local Reasoning about Programs that Alter Data Structures. In: Fribourg, L., Ed., *Computer Science Logic. CSL* 2001. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 1-19. https://doi.org/10.1007/3-540-44802-0_1

[4] Reynolds, J.C. (2002) Separation Logic: A Logic for Shared Mutable Data Structures. *Proceedings of the* 17*th Annual IEEE Symposium on Logic in Computer Science*, Copenhagen, 22-25 July 2002, 55-74. https://doi.org/10.1109/LICS.2002.1029817

[5] Chlipala, A. (2013) The Bedrock Structured Programming System: Combining Generative Metaprogramming and Hoare Logic in an Extensible Program Verifier. *Proceedings of the* 18*th ACM SIGPLAN International Conference on Functional Programming*, September 2013, 391-402. https://doi.org/10.1145/2500365.2500592

[6] Jensen, J.B., Benton, N. and Kennedy, A. (2013) High-Level Separation Logic for Low-Level Code. *ACM SIGPLAN Notices*, **48**, 301-314. https://doi.org/10.1145/2429069.2429105

[7] Krebbers, R. (2015) The C Standard Formalized in Coq. Ph.D. Dissertation, Radboud University, Gelderland.

[8] Sergey, I., Nanevski, A. and Banerjee, A. (2015) Mechanized Verification of Fine-Grained Concurrent Programs. *ACM SIGPLAN Notices*, **50**, 77-87.

[9] Tuch, H., Klein, G. and Norrish, M. (2007) Types, Bytes, and Separation Logic. *ACM SIGPLAN Notices*, **42**, 97-108. https://doi.org/10.1145/1190215.1190234

[10] The Coq Development Team: The Coq Proof Assistant. http://coq.inria.fr

[11] Cao, Q., Beringer, L., Gruetter, S., Dodds, J. and Appel, A.W. (2018) VST-Floyd: A Separation Logic Tool to Verify Correctness of C Programs. *Journal of Automated Reasoning*, **61**, 367-422. https://doi.org/10.1007/s10817-018-9457-5

[12] McCreigh, A. (2009) Practical Tactics for Separation Logic. In: Berghofer, S., Nipkow, T., Urban, C. and Wenzel, M., Eds., *Theorem Proving in Higher Order Logics. TPHOLs* 2009. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 343-358. https://doi.org/10.1007/978-3-642-03359-9_24

[13] Bengtson, J., Jensen, J.B. and Birkedal, L. (2012) Charge-A Framework for Higher-Order Separation Logic in Coq. In: Beringer, L. and Felty, A., Eds., *Interactive Theorem Proving. ITP* 2012. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 315-331. https://doi.org/10.1007/978-3-642-32347-8_21

[14] Leroy, X. (2006) Formal Certification of a Complier Back-End, or: Programming a Complier with a Proof Assistant. *Proceedings of the* 33*th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Charleston, SC, 11-13 January 2006, 42-54. https://doi.org/10.1145/1111320.1111042

[15] Appel, A.W., Dockins, R., Hobor, A., Beringer, L., Dodds, J., Stewart, G., Blazy, S. and Leroy, X. (2014) Program Logics for Certified Compilers. University of Cambridge Press, New York.

[16] Delahaye, D. (2000) A Tactic Language for the System Coq. In: Parigot, M. and Voronkov, A., Eds., *Logic for Programming and Automated Reasoning. LPAR* 2000. *Lecture Notes in Artificial Intelligence*, Springer, Berlin, Heidelberg, 85-95. https://doi.org/10.1007/3-540-44404-1_7

[17] Cao, J., Fu, M. and Feng, X. (2015) Practical Tactics for Verifying C Programs in Coq. *Proceedings of the* 2015 *Conference on Certified Programs and Proofs*, Mumbai, India, 15-17 January 2015, 97-108. https://doi.org/10.1145/2676724.2693162