




Inference Rules for the Partial Floyd-Hoare Logic Based on Composition of Predicate Complement

Ievgen Ivanov and Mykola Nikitchenko^(✉) 

Taras Shevchenko National University of Kyiv, 64/13, Volodymyrska Street,
Kyiv 01601, Ukraine
ivanov.eugen@gmail.com, nikitchenko@unicyb.kiev.ua

Abstract. Classical Floyd-Hoare logic is sound when total pre- and post-conditions are considered. In the case of partial conditions (predicates) the logic becomes unsound. This situation may be corrected by introducing additional constraints to the rules of the logic. But such constraints, in particular, for the sequence and while rules, are rather complicated. In this paper we propose new simpler rules formulated in a program algebra extended with the composition of predicate complement. The obtained logic is called the Complemented Partial Floyd-Hoare Logic (CPFHL). The predicate component of this logic is related to three-valued logic. We prove the soundness theorem for CPFHL and discuss further investigations of the problem. The obtained results can be useful for software verification.

Keywords: Formal methods · Software verification · Partial predicate · Floyd-Hoare logic

1 Introduction

The research presented in this paper¹ is devoted to generalization of the classical Floyd-Hoare logic [2–4] to the case of partial pre- and post-conditions. This logic is a useful tool for proving partial correctness of sequential programs. It is based on properties of a Floyd-Hoare triple (assertion) of the form $\{p\}f\{q\}$ which consists of pre- and post-conditions p , q (predicates) and a program f . This assertion is treated in the following way: when the program's input data d satisfies the pre-condition ($p(d)$ is true), and the program terminates on d , the program's output (the value $f(d)$) satisfies the post-condition ($q(f(d))$ is true).

Although in the classical Floyd-Hoare logic it is assumed that a program may not terminate and its output may be undefined on a particular input d ($f(d)$ is undefined), it is also assumed that the pre- and post-conditions p , q are predicates which are defined on all possible data, i.e. that they are total predicates.

¹ This paper is a refined and extended version of [1].

Partiality of these predicates can arise naturally, if they are expressed using operations which can cause errors, nontermination, or non-well-defined behavior. For example, one may want to consider the Floyd-Hoare triple written in Octave syntax [5]

$$\{\text{true}\} \mathbf{a}(1)=0 \{\mathbf{a}(1)==0\}$$

where $\mathbf{a}(1)=0$ denotes the operation of assignment of the value 0 to the 1-st element of the array (or a more sophisticated key-value map) \mathbf{a} , and $\mathbf{a}(1)==0$ is a predicate stating that the 1-st element of \mathbf{a} is zero. Logically, it makes sense to consider the latter predicate (the truth value of which depends on the content of \mathbf{a}) as partial and assume that it is defined only when \mathbf{a} has an element with index/key 1 (e.g. it is undefined when \mathbf{a} has no content). Note that depending on the rules of the programming language, the assignment operation may be always well-defined (e.g. if the assignment $\mathbf{a}(1)=0$ automatically allocates the space for the new value, if it is not allocated), but the post-condition predicate may be partial, because extraction from \mathbf{a} is not always defined in the sense that it causes an error or another type of abnormal behavior. In particular, a situation of this kind occurs in Matlab [6] and Octave [5] languages for element insertion and extraction operations for vectors (these languages and the corresponding environments are widely used in scientific computing).

In the literature [7] one finds discussions of the ways of emulating partial predicates and functions in software specifications using total predicates and/or functions, however, almost all approaches which try to avoid partiality have some drawbacks which are described in [7–9].

To deal with this issue, in the previous works [10–13] extensions of the classical Floyd-Hoare logic which allowed one to reason about partial correctness of sequential programs using Floyd-Hoare triples with partial programs and partial pre- and post-conditions were investigated.

Here a Floyd-Hoare triple $\{p\}f\{q\}$ means that when the program's input data d satisfies the pre-condition ($p(d)$ is defined and true), and the program terminates on d , and the post-condition is defined on the program's output (the value $q(f(d))$ is defined), then the program's output satisfies the post-condition ($q(f(d))$ is true). We call this interpretation of a triple with partial pre- and post-conditions a *weak Floyd-Hoare triple* (the reason is that it does not require the post-condition to be defined, if the pre-condition is defined; one alternative is to require that the post-condition is defined whenever the pre-condition is defined which we call the *strong Floyd-Hoare triple*, but which we do not consider in this paper). The logic itself will be called Partial Floyd-Hoare Logic (PFHL).

An important fact is that the classical inference system for the Floyd-Hoare logic for the language WHILE [14] is not sound in the presence of partial pre- and post-conditions [10]. One reason of this is unsoundness of the *sequence rule* when p, q, r are partial predicates:

$$R_SEQ \frac{\{p\} f \{q\}, \{q\} g \{r\}}{\{p\} f \bullet g \{r\}}$$

where $f \bullet g$ denotes the sequential composition of programs f and g (i.e. g runs after f on the result of f).

This can be explained on the following simple example in Octave syntax. Let n be an integer-valued variable. The expression `zeros(n,1)` evaluates to a $n \times 1$ vector of zeros. If the variable a contains a vector, the expression `length(a)` evaluates to the length of a . The i -th ($i=1,2,\dots$) component can be accessed using the expression `a(i)` which raises an error, if the value of i is not a valid index, e.g. if the length of a is less than the value of i . Assignment is denoted as `=`, equality test is denoted as `==`, and comparisons are denoted as `>=`, `>`. Then we can assume that the following assertions are valid (in the sense of weak Floyd-Hoare triples):

$$\begin{aligned} &\{n \geq 0\} \ a = \text{zeros}(n,1) \ \{a(i) == 0\}, \\ &\{a(i) == 0\} \ m = \text{length}(a) \ \{m > 0\} \end{aligned}$$

We assume that in the first triple the post-condition is undefined, if the length of a is zero (a is empty), which happens if n is zero. However,

$$\{n \geq 0\} \ a = \text{zeros}(n,1); \ m = \text{length}(a) \ \{m > 0\}$$

is not a valid assertion (in sense of weak Floyd-Hoare triples), since if n is zero, then a is a zero-length vector (is empty) and m is zero.

Because of unsoundness of some rules of the classical inference system in the presence of partial predicates, new inference systems for Partial Floyd-Hoare Logic should be constructed. In our previous works we have considered two ideas of such construction. The first idea consists in restricting the class of assertions to T -increasing assertions [11]; in this case the rules preserve their validity and no other changes are required. The second idea consists in introducing constraints which restrict applicability of the rules of logic. Such a system with constrained rules was proposed in [10]. In this system the regular sequence rule was replaced with the following *constrained sequence rule*:

$$R_SEQ, \frac{\{p\} f \{q\}, \{q\} g \{r\}}{\{p\} f \bullet g \{r\}}, p \models PC(f \bullet g, r)$$

where

- $f \bullet g$ denotes the function $d \mapsto g(f(d))$ which is the result of sequential composition of f and g ;
- $p \models q$ means that each interpretation of the formula $\neg p \vee q$ (i.e. $p \rightarrow q$) never takes the false value (i.e. it is always either true or undefined);
- PC is the *Predicate transformer composition* [11] such that $PC(f, q)$ is a partial predicate r such that for any data d , $r(d) = q(f(d))$, if $f(d)$ (i.e. program value) and $q(f(d))$ (i.e. the value of the predicate q on the result of f on data d) are defined, and $r(d)$ is undefined otherwise (i.e. if $f(d)$ or $q(f(d))$ are undefined).

In a similar way some other rules were modified.

However, the presence of complicated constraints makes application of such rules difficult in all but the most trivial cases. Therefore in [11] such constraints were even called trifling constraints.

Thus, the above mentioned ideas lead to PFHL limitation (T -increasing assertions) or do not lead to practical inference systems (constrained rules). This implies that further investigation of PFHL is necessary and the inference rules have to be based on some other ideas.

In this paper we propose an extension of the predicate language with a new composition called predicate complement. Introduction of this composition permits us to modify the rules in such a way that they become sound and no constraints are required. The obtained inference system for PFHL based on extended program algebra can be useful for software verification.

2 Notation

The symbol $\overset{\sim}{\rightarrow}$ will denote partial functions and \rightarrow will denote total functions (e.g. $f : A \overset{\sim}{\rightarrow} B$ means that f is a partial function on a set A with values in a set B ; $f : A \rightarrow B$ means that f is a total function from A to B). We will use the symbol $\overset{n}{\rightarrow}$ for partial functions with finite graph. For $f : D \overset{\sim}{\rightarrow} D'$:

- $f(d) \downarrow$ denotes that f is defined on $d \in D$;
- $f(d) \downarrow = d'$ denotes that f is defined on $d \in D$ and has the value $d' \in D'$;
- $f(d) \uparrow$ denotes that f is undefined on $d \in D$;
- $\text{dom}(f) = \{d \in D \mid f(d) \downarrow\}$ is the domain of a function (note that there are different definitions of the domain of a partial function in different branches of mathematics); we use the convention from recursion theory).

The notation $f_1(d_1) \cong f_2(d_2)$ means the *strong equality*, i.e. that $f_1(d_1) \downarrow$ if and only if $f_2(d_2) \downarrow$, and if $f_1(d_1) \downarrow$, then $f_1(d_1) = f_2(d_2)$.

We use the following notations for predicate $p : D \overset{\sim}{\rightarrow} \text{Bool}$:

- $p^T = \{d \mid p(d) \downarrow = T\}$ is the truth domain of a predicate p ;
- $p^F = \{d \mid p(d) \downarrow = F\}$ is the falsity domain of p .

3 Program Algebras with Predicate Complement

In accordance with the composition-nominative approach [15–18] we consider program logics which are based on program algebras. Such algebras are constructed in the following way:

- first, a set D of data processed by programs is defined; in our case we treat D as hierarchical nominative data [19];
- then, classes of predicates $Pr = D \overset{\sim}{\rightarrow} \text{Bool}$ and functions $Fn = D \overset{\sim}{\rightarrow} D$ are defined;
- at last, operations (compositions) over Pr and Fn are specified.

This scheme leads to two-sorted program algebras. In our previous works [10,11] we considered program algebras with traditional compositions. But the

problem of defining unconstrained rules requires new compositions. In this paper we introduce a program algebra extended with a composition of predicate complement.

3.1 Hierarchical Nominative Data

Informally speaking, simple hierarchical nominative data are constructed inductively over sets of basic names V and values A and can be presented as trees, the arcs of which are labeled by elements of V and leafs are labeled by elements of A . Here we consider a more complex case in which names can be presented as sequences of elements of V of the form $v_1v_2\dots v_n \in V^+$. Introduction of complex names from V^+ requires additional restrictions induced by the *principle of unambiguous associative naming*. This principle demands that for any hierarchical nominative data d with complex names for any two paths (u_1, u_2, \dots, u_k) and (v_1, v_2, \dots, v_l) in d , neither of which is a prefix of another, the words $u_1u_2\dots u_k$ and $v_1v_2\dots v_l$ are incomparable in the sense of the prefix relation. Such data are also called *complex-named data* [20]. An example of such data is $[uv \mapsto 1, w \mapsto [uw \mapsto 3]]$, $u, v, w \in V$. We denote this class of data as $ND_{CC}(V, A)$. Formal definitions can be found in [19, 21].

3.2 Basic Operations on Nominative Data

The basic operations on nominative data are the operations of

- *denaming* (taking the value of a name),
- *naming* (assigning a new value to a name),
- *overlapping*.

We define these operations for the class $ND_{CC}(V, A)$ of nominative data with complex names and values ($d \in ND_{CC}(V, A)$).

Definition 1 ([19], Denaming). *The (associative) denaming is an operation $v \Rightarrow_a$ with a parameter $v \in V^+$ defined by induction on the length of v :*

- if $v \in V$, then $v \Rightarrow_a (d) \cong \begin{cases} d(v), & \text{if } d(v) \downarrow; \\ d/v, & \text{if } d(v) \uparrow \text{ and } d/v \neq \emptyset; \\ \text{undefined,} & \text{if } d(v) \uparrow \text{ and } d/v = \emptyset, \end{cases}$
where $d/u = [v_1 \mapsto d(v) \mid d(v) \downarrow, v = uv_1, v_1 \in V^+]$;
- if $v \in V^+ \setminus V$, then $v \Rightarrow_a (d) \cong v_2 \Rightarrow_a (v_1 \Rightarrow_a (d))$, *where v_1 is the first symbol of v and v_2 is the suffix, i.e. v_1, v_2 are (unique) words such that $v = v_1v_2$ and $v_1 \in V$.*

Definition 2 ([19], Naming). *Naming is an unary operation $\Rightarrow v$ with a parameter $v \in V^+$ such that $\Rightarrow v(d) = [v \mapsto d]$.*

Overlapping is a kind of updating operation which updates the values of names in its first argument with the values of names in its second argument. For different types of nominative data different overlapping operations can be considered. Here we will define two kinds of overlapping: global and local overlapping. Global (associative or structural) overlapping ∇_a updates several values in the first argument while the local one ∇_a^v (with a parameter $v \in V^+$) updates only one value which is associated with the name v .

Global overlapping can be used, e.g. for formalizing procedures calls, while the local overlapping can be used as a formalization of the assignment operator in programming languages.

Definition 3 ([19], **Global overlapping**). *This is a binary operation ∇_a defined by induction on the rank of the first argument as follows.*

Let $ND_{CC_k}(V, A)$ be the class of data with the rank less or equal to k .

Induction base of the definition. If $d_1 \in ND_{CC_0}(V, A)$, then

$$d_1 \nabla_a d_2 \cong \begin{cases} d_2, & \text{if } d_1 = \emptyset \text{ and } d_2 \in ND_{CC}(V, A) \setminus A; \\ \text{undefined,} & \text{if } d_1 \in A \text{ or } d_2 \in A. \end{cases}$$

Induction step of the definition. Assume that the value $d_1 \nabla_a d_2$ is already defined for all d_1, d_2 such that $d_1 \in ND_{CC_k}(V, A)$. Let

$$d_1 \in ND_{CC_{k+1}}(V, A) \setminus ND_{CC_k}(V, A).$$

Then $d_1 \nabla_a d_2 = d$, where d is defined for each name $u \in V^+$ as follows:

- (1) $d(u) = d_2(u)$, if $u \in \text{dom}(d_2)$ and u does not have a proper prefix which belongs to $\text{dom}(d_1)$;
- (2) $d(u) = d_1(u) \nabla_a (d_2/u)$, if $d_1(u)$ is defined and does not belong to A and u is a proper prefix of some element of $\text{dom}(d_2)$, where

$$d_2/u = [v_1 \mapsto d_2(v) \mid d_2(v) \downarrow, v = uv_1, v_1 \in V^+];$$
- (3) $d(u) = d_2/u$, if $d_1(u)$ is defined and belongs to A and u is a proper prefix of some element of $\text{dom}(d_2)$;
- (4) $d(u) = d_1(u)$, if $d_1(u)$ is defined and u is not comparable (in the sense of the prefix relation) with any element of $\text{dom}(d_2)$;
- (5) $d(u) \uparrow$, otherwise.

Definition 4 ([19], **Local overlapping**). *This is a binary operation ∇_a^v with a parameter $v \in V^+$ defined as follows: $d_1 \nabla_a^v d_2 \cong d_1 \nabla_a (\Rightarrow v(d_2))$.*

3.3 Compositions

Now we define compositions of program algebras over nominative data with complex names and values.

Let V and A be fixed sets of basic names and values. Denote

$$Pr_{CC}(V, A) = ND_{CC}(V, A) \xrightarrow{\sim} \{T, F\},$$

$$Fn_{CC}(V, A) = ND_{CC}(V, A) \xrightarrow{\sim} ND_{CC}(V, A).$$

We will assume that T and F do not belong to $ND_{CC}(V, A)$.

We will call the elements of $Pr_{CC}(V, A)$ (*partial nominative predicates*) and the elements of $Fn_{CC}(V, A)$ (*partial nominative functions*).

Let us denote by \bar{U} the set of all tuples (u_1, u_2, \dots, u_n) , $n \geq 1$ of complex names from V^+ such that whenever $i \neq j$, u_i and u_j are incomparable in the sense of the prefix relation.

In the following definitions $d \in ND_{CC}(V, A)$, $f, g, f_1, \dots, f_n \in Fn_{CC}(V, A)$, $p, p_1, p_2 \in Pr_{CC}(V, A)$.

Definition 5 (Compositions).

- *Composition of superposition into a predicate*

$$S_{PC}^{u_1, \dots, u_n} : Pr_{CC}(V, A) \times (Fn_{CC}(V, A))^n \rightarrow Pr_{CC}(V, A)$$

with a parameter $(u_1, \dots, u_n) \in \bar{U}$ is defined as follows:

$$S_P^{u_1, \dots, u_n}(p, f_1, \dots, f_n)(d) \cong p(\dots(d\nabla_a^{u_1} f_1(d)) \dots \nabla_a^{u_n} f_n(d)).$$

We will also use the following notation for this composition: $S_P^{\bar{u}}$.

- *Composition of superposition into a function*

$$S_F^{u_1, \dots, u_n} : Fn_{CC}(V, A) \times (Fn_{CC}(V, A))^n \rightarrow Fn_{CC}(V, A)$$

with a parameter $(u_1, \dots, u_n) \in \bar{U}$ is defined as follows:

$$S_F^{u_1, \dots, u_n}(f, f_1, \dots, f_n)(d) \cong f(\dots(d\nabla_a^{u_1} f_1(d)) \dots \nabla_a^{u_n} f_n(d)).$$

We will also use the following notation for this composition: $S_F^{\bar{u}}$.

- *Assignment composition $AS^u : Fn_{CC}(V, A) \rightarrow Fn_{CC}(V, A)$ with a parameter $u \in V^+$ is defined as follows:*

$$(AS^u(f))(d) \cong d\nabla_a^u f(d).$$

- *Sequential composition of functions (denoted using the infix notation) $\bullet : Fn_{CC}(V, A) \times Fn_{CC}(V, A) \rightarrow Fn_{CC}(V, A)$ is defined as follows:*

$$(f \bullet g)(d) \cong g(f(d)).$$

- *Branching composition $IF : Pr_{CC}(V, A) \times Fn_{CC}(V, A) \times Fn_{CC}(V, A) \rightarrow Fn_{CC}(V, A)$ is defined as follows:*

$$IF(p, f, g)(d) = \begin{cases} f(d), & \text{if } p(d) \downarrow = T \text{ and } f(d) \downarrow; \\ g(d), & \text{if } p(d) \downarrow = F \text{ and } g(d) \downarrow; \\ \text{undefined} & \text{in other cases.} \end{cases}$$

- *Cycle composition $WH : Pr_{CC}(V, A) \times Fn_{CC}(V, A) \rightarrow Fn_{CC}(V, A)$ is defined as follows:*

$WH(p, f)(d) \downarrow = f^{(n)}(d)$, if there exists $n \geq 0$ such that $p(f^{(i)}(d)) \downarrow = T$ for all $i \in \{0, 1, \dots, n-1\}$ and $p(f^{(n)}(d)) \downarrow = F$, where $f^{(n)}$ is a n -times sequential composition of f with itself ($f^{(0)}$ is the identity function), and $WH(p, f)(d)$ is undefined otherwise.

- Identity null-ary composition (function) $id : Fn_{CC}(V, A) \rightarrow ND_{CC}(V, A)$ ($id : ND_{CC}(V, A) \rightarrow ND_{CC}(V, A)$) is defined as follows:

$$id(d) = d.$$

- Composition of disjunction $\vee : Pr_{CC}(V, A) \times Pr_{CC}(V, A) \rightarrow Pr_{CC}(V, A)$ is a composition defined as follows:

$$(p_1 \vee p_2)(d) = \begin{cases} T, & \text{if } p_1(d) \downarrow = T \text{ or } p_2(d) \downarrow = T; \\ F, & \text{if } p_1(d) \downarrow = F \text{ and } p_2(d) \downarrow = F; \\ \text{undefined} & \text{in other cases.} \end{cases}$$

- Composition of negation $\neg : Pr_{CC}(V, A) \rightarrow Pr_{CC}(V, A)$ is a composition such that

$$(\neg p)(d) = \begin{cases} T, & \text{if } p(d) \downarrow = F; \\ F, & \text{if } p(d) \downarrow = T; \\ \text{undefined} & \text{in other cases.} \end{cases}$$

- Composition of existential quantification over hierarchical data is a unary composition $Pr_{CC}(V, A) \rightarrow Pr_{CC}(V, A)$ with a parameter $x \in V^+$ such that

$$(\exists x p)(d) = \begin{cases} T, & \text{if } p(d\nabla_a^x d') \downarrow = T \text{ for some } d' \in ND_{CC}(V, A), \\ F, & \text{if } p(d\nabla_a^x d') \downarrow = F \text{ for all } d' \in ND_{CC}(V, A), \\ \text{undefined} & \text{in other cases.} \end{cases}$$

- Composition of predicate complement is a composition $\sim : Pr_{CC}(V, A) \rightarrow Pr_{CC}(V, A)$ such that

$$(\sim p)(d) = \begin{cases} T, & \text{if } p(d) \text{ is undefined;} \\ \text{undefined} & \text{in other cases.} \end{cases}$$

Having compositions, we can define a special program algebra investigated in this paper.

Definition 6. A complemented program algebra over hierarchical nominative data with complex names and values $CPAND_{CC}(V, A)$ is a two-sorted algebra $CPAND_{CC}(V, A) = (Pr_{CC}(V, A), Fn_{CC}(V, A))$;

$$AS^u, id, \bullet, IF, WH, S_F^{\bar{u}}, S_P^{\bar{u}}, \Rightarrow v, v \Rightarrow_a, \vee, \neg, \exists x, \sim,$$

where $v, u, x \in V^+$, $\bar{u} \in \bar{U}$.

Derived compositions like conjunction \wedge and universal quantification $\forall x$ are defined in a traditional way.

Let us discuss briefly predicate compositions of this algebra. Operations (compositions) \vee , \neg , and $\exists x$ are defined according to the truth tables of Kleene's strong logic of indeterminacy [22]. Please note that \vee , \wedge and \neg on the set of all partial predicates form a Kleene algebra (a De Morgan algebra which satisfies the normality axiom) [23].

In contrast to these compositions, the composition of predicate complement is more complicated. First, it does not have the monotonicity property, second, it does not have nice distributivity properties. Even more, it is not computable in the sense that the set of partial recursive predicates is not closed under this composition.

Introduction of the composition of predicate complement makes investigation of the corresponding logics more difficult. In this case methods developed for three-valued logics can be used.

Nevertheless, the algebra $(Pr_{CC}(V, A); \vee, \wedge, \neg, \sim)$ has certain properties which make it useful in program partial correctness proofs:

- it can be proven that the algebra $(Pr_{CC}(V, A); \vee, \wedge, \neg, \sim)$ has the same (up to the names of operations) set of identities as the algebra

$$(\{-1, 0, 1\}; \max(\cdot, \cdot), \min(\cdot, \cdot), x \mapsto -x, x \mapsto 1 - |x|);$$

- all scalar functions of n variables expressible in the latter algebra are non-expanding maps from $\{-1, 0, 1\}^n \rightarrow \{-1, 0, 1\}$ with respect to Chebyshev distance $dist((x_1, \dots, x_n), (y_1, \dots, y_n)) = \max_{i=1}^n |x_i - y_i|$.

Definition 7. *A semantic weak Floyd-Hoare triple is a tuple (p, f, q) , where $f : D \rightarrow D'$, $p : D \rightarrow Bool$, $q : D' \rightarrow Bool$ for some D, D' such that for each $d \in D$, if $p(d) \downarrow = T$ and $f(d) \downarrow$ and $q(f(d)) \downarrow$, then $q(f(d)) = T$.*

We will use the following notation:

- $\{p\}f\{q\}$ means that (p, f, q) is a semantic weak Floyd-Hoare triple.

Please note that a semantic weak Floyd-Hoare triple induces ternary Floyd-Hoare composition $FH : Pr_{CC}(V, A) \times Fn_{CC}(V, A) \times Pr_{CC}(V, A) \rightarrow Pr_{CC}(V, A)$ [10, 11], but in this paper we do not include it into program algebras in order to not make them overcomplicated.

We start with new inference rules for the sequential composition. These rules are valid for any set of data D .

4 New Inference Rules for Sequential Composition

Theorem 1. *Assume that $\{p\}f\{q\}$, $\{q\}g\{r_1\}$, and $\{\sim q\}g\{r_2\}$.*

Then $\{p\}f \bullet g\{r_1 \vee r_2\}$.

Proof. Let $d \in D$. Assume that $p(d) \downarrow = T$, $(f \bullet g)(d) \downarrow$, and $(r_1 \vee r_2)((f \bullet g)(d)) \downarrow$.

Then $f(d) \downarrow$ and $g(f(d)) \downarrow$. Denote $d' = f(d)$ and $d'' = (f \bullet g)(d) = g(f(d))$.

Let us show that $(r_1 \vee r_2)(d'') = T$.

Suppose that $(r_1 \vee r_2)(d'') \downarrow = F$. Then $r_1(d'') \downarrow = F$ and $r_2(d'') \downarrow = F$. We have that either $q(d') \downarrow$, or $q(d') \uparrow$.

Consider the case when $q(d') \downarrow$. Then $q(f(d)) \downarrow$, and since $p(d) \downarrow = T$ and $\{p\}f\{q\}$, we have $q(f(d)) = q(d') = T$. Then since $r_1(g(d')) \cong r_1(d'') \downarrow$ and

$\{q\}g\{r_1\}$, we have $r_1(g(d')) = r_1(d'') = T$, but this contradicts the above mentioned statement $r_1(d'') = F$.

Consider the case when $q(d') \uparrow$. Then $(\sim q)(d') \downarrow = T$. Then since $r_2(g(d')) \cong r_2(d'') \downarrow$ and $\{\sim q\}g\{r_2\}$, we have $r_2(g(d')) = r_2(d'') = T$, but this contradicts the above mentioned statement $r_2(d'') = F$.

In both cases we have a contradiction, so $(r_1 \vee r_2)(d'') \downarrow = F$ is impossible. But $(r_1 \vee r_2)(d'') \cong (r_1 \vee r_2)((f \bullet g)(d)) \downarrow$ by the assumption, so $(r_1 \vee r_2)(d'') = T$. \square

This result can be used as a semantic foundation of the following unconstrained inference rule for sequential composition for the inference system for Floyd-Hoare logic with partial pre- and post-conditions (which involves the \sim operation on predicates):

$$R_USEQ \frac{\{p\} f \{q\}, \{q\} g \{r_1\}, \{\sim q\}g\{r_2\}}{\{p\} f \bullet g \{r_1 \vee r_2\}}$$

In the special case of coinciding r_1 and r_2 , it can be rewritten as:

$$R_SSEQ \frac{\{p\} f \{q\}, \{q\} g \{r\}, \{\sim q\}g\{r\}}{\{p\} f \bullet g \{r\}}$$

Theorem 1 implies that addition of the rules R_USEQ and/or R_SSEQ to the inference system AC proposed in [10, 11] (with the proper extension of syntax of pre- and post-condition predicate formulas to accommodate the symbol of \sim operation) does not change its soundness.

As an informal example, consider how these rules can be applied in the case mentioned in the Introduction:

$$\{n>=0\} \mathbf{a=zeros}(n,1) \{\mathbf{a}(1)==0\},$$

$$\{\mathbf{a}(1)==0\} \mathbf{m=length}(\mathbf{a}) \{\mathbf{m}>0\}.$$

These two assumptions alone are not sufficient to establish the triple concerning the sequential composition. A missing piece of information is the triple describing the behavior of the instruction $\mathbf{m=length}(\mathbf{a})$ when the predicate $(\mathbf{a}(1)==0)$ is undefined. Under the interpretation assumed in the Introduction, this undefinedness means that an attempt of evaluation of $(\mathbf{a}(1)==0)$ leads to an abnormal/error state. If \mathbf{a} is a defined vector, this happens exactly when \mathbf{a} has zero length (is empty). If \mathbf{a} is undefined, then an attempt of evaluation of $\mathbf{length}(\mathbf{a})$ causes an error. Thus we can state that

$$\{\sim(\mathbf{a}(1)==0)\} \mathbf{m=length}(\mathbf{a}) \{\mathbf{m}=0\},$$

where \sim is not a part of Octave syntax, but just a notation to represent the statement that the expression $(\mathbf{a}(1)==0)$ is undefined (causes an abnormal/error state). Thus by the R_USEQ rule:

$$\{n>=0\} \mathbf{a=zeros}(n,1); \mathbf{m=length}(\mathbf{a}) \{\mathbf{m}>0 \vee \mathbf{m}=0\}.$$

Again, here \vee is not a part of Octave syntax, but a notation to represent the disjunction of two predicates.

5 New Inference Rule for Cycle Composition

Let r be a partial predicate on D and $f : D \dashrightarrow D$.

Recall that the cycle composition $WH(r, f)$ is defined as follows: it returns a function in $D \dashrightarrow D$ such that for each $d \in D$,

$$WH(r, f)(d) \downarrow = f^{(n)}(d),$$

if there exists an integer $n \geq 0$ such that $r(f^{(i)}(d)) \downarrow = T$ for all $i = 0, 1, \dots, n-1$ and $r(f^{(n)}(d)) \downarrow = F$, where $f^{(i)}$ denotes $\underbrace{f \bullet f \bullet \dots \bullet f}_i$ and $f^{(0)}$ is the identity

function on D (i.e. $f^{(0)}(d') = d'$ for all $d' \in D$); and $WH(r, f)(d) \uparrow$, otherwise.

WH is intended to capture the semantics of the loop of the form `while_do_` in imperative programming languages which support structured programming. Here r represents the semantics of the loop condition and f represents the semantics of the loop body.

In terms of WH the loop rule for the classical inference system for the Floyd-Hoare logic with total pre- and post-conditions [14] can be reformulated as follows [24, p. 15]:

$$R_WH \frac{\{r \wedge p\} f \{p\}}{\{p\} WH(r, f) \{\neg r \wedge p\}}$$

Here p represents the loop invariant.

This rule, generally, is not valid in the case of partial pre- and post-conditions, but can be replaced with a constrained rule [24, p. 16] (R_WH') in this case.

Applying the approach which we used in the statement and proof of Theorem 1, we can propose an alternative unconstrained rule for the while loop which is more convenient to apply.

Theorem 2. *Assume $\{r \wedge p\} f \{p\}$, $\{r \wedge (\sim p)\} f \{p\}$. Then $\{p\} WH(r, f) \{\neg r \wedge p\}$.*

Proof. Let $d \in D$. Assume that

$$p(d) \downarrow = T, WH(r, f)(d) \downarrow = d', \text{ and } (\neg r \wedge p)(WH(r, f)(d)) \downarrow.$$

Let us show that $(\neg r \wedge p)(WH(r, f)(d)) = (\neg r \wedge p)(d') = T$.

From the definition on WH it follows that there exists an integer $n \geq 0$ such that $r(f^{(i)}(d)) \downarrow = T$ for all $i = 0, 1, \dots, n-1$, and $r(f^{(n)}(d)) \downarrow = F$, and

$$d' = WH(r, f)(d) = f^{(n)}(d).$$

If $n = 0$, then $d' = d$, so $r(d') \downarrow = F$ and $p(d') \downarrow = T$, whence $(\neg r \wedge p)(d') = T$.

Now we will assume that $n \geq 1$.

Let us show by induction on $i \in \{0, 1, \dots\}$ that if $i \in \{0, 1, \dots, n-1\}$, then $p(f^{(i)}(d)) \downarrow = T$ or $p(f^{(i)}(d)) \uparrow$.

Base of induction: $p(f^{(0)}(d)) \cong p(d) \downarrow = T$, so the statement holds.

Inductive step. Assume that $i \in \{0, 1, \dots, n-1\}$. Assume that $p(f^{(i)}(d)) \downarrow = T$ or $p(f^{(i)}(d)) \uparrow$. Assume that $i+1 \in \{0, 1, \dots, n-1\}$. Note that $r(f^{(i)}(d)) \downarrow = T$ because $i < n$. Denote $d_1 = f^{(i)}(d)$.

Consider the case $p(f^{(i)}(d)) \downarrow = T$. Then since $r(d_1) \downarrow = T$ and $p(d_1) \downarrow = T$, we have $(r \wedge p)(d_1) \downarrow = T$. If $p(f(d_1)) \downarrow$, then since $\{r \wedge p\}f\{p\}$, we have $p(f^{(i+1)}(d)) \cong p(f(d_1)) \downarrow = T$. Otherwise, $p(f(d_1)) \uparrow$. In either case, either $p(f^{(i+1)}(d)) \downarrow = T$, or $p(f^{(i+1)}(d)) \uparrow$ holds.

Consider the case $p(f^{(i)}(d)) \uparrow$. Then $p(d_1) \uparrow$ and $r(d_1) \downarrow = T$. Further, $(\sim p)(d_1) \downarrow = T$, so $(r \wedge (\sim p))(d_1) \downarrow = T$. Since $\{r \wedge (\sim p)\}f\{p\}$, we have either $p(f^{(i+1)}(d)) \cong p(f(d_1)) \downarrow = T$, or $p(f^{(i+1)}(d)) \cong p(f(d_1)) \uparrow$.

In both cases $p(f^{(i+1)}(d)) \downarrow = T$ or $p(f^{(i+1)}(d)) \uparrow$, so the inductive step is completed.

Since $n \geq 1$ by our assumption, the proven statement implies that either $p(f^{(n-1)}(d)) \downarrow = T$, or $p(f^{(n-1)}(d)) \uparrow$. We have

$$(\neg r \wedge p)(f^{(n)}(d)) \cong (\neg r \wedge p)(WH(r, f)(d)) \downarrow.$$

Moreover, $r(f^{(n)}(d)) \downarrow = F$, so $(\neg r)(f^{(n)}(d)) \downarrow = T$, whence $p(f^{(n)}(d)) \downarrow$.

Consider the case $p(f^{(n-1)}(d)) \downarrow = T$. We have $r(f^{(n-1)}(d)) \downarrow = T$, therefore $(r \wedge p)(f^{(n-1)}(d)) \downarrow = T$. Then since $\{r \wedge p\}f\{p\}$ and $p(f(f^{(n-1)}(d))) \cong p(f^{(n)}(d)) \downarrow$, we have $p(f^{(n)}(d)) = T$.

Consider the case $p(f^{(n-1)}(d)) \uparrow$. Then because $f^{(n-1)}(d) \downarrow$, we have $(\sim p)(f^{(n-1)}(d)) \downarrow = T$. Moreover, we have $r(f^{(n-1)}(d)) \downarrow = T$, whence $(r \wedge (\sim p))(f^{(n-1)}(d)) \downarrow = T$. Then because $\{r \wedge (\sim p)\}f\{p\}$ and, moreover, $p(f(f^{(n-1)}(d))) \cong p(f^{(n)}(d)) \downarrow$, we have $p(f^{(n)}(d)) = T$.

In both cases we have $p(f^{(n)}(d)) = T$. Since $r(f^{(n)}(d)) \downarrow = F$, we have $(\neg r \wedge p)(WH(r, f)(d)) \cong (\neg r \wedge p)(f^{(n)}(d)) \downarrow = T$. \square

This result can be used as a semantic foundation of the following unconstrained inference rule (for the case of partial pre- and post-conditions):

$$R_UWH \frac{\{r \wedge p\} f \{p\}, \{r \wedge (\sim p)\} f \{p\}}{\{p\} WH(r, f) \{\neg r \wedge p\}}$$

6 Syntax and Interpretation of Complemented Partial Floyd-Hoare Logic

Algebra $CPAND_{CC}(V, A)$ has strong expressive power that is not required for our goal: to construct a special program logic. Therefore we restrict syntactically the class of terms of this algebra. The idea is to consider programs as special nominative functions (program functions) constructed with the help of compositions AS^x , id , \bullet , IF , WH , S_F^x . Functions of other types can be represented by functional expressions. Formulas represent partial predicates over nominative data. The signature of the constructed logic is $\Sigma = (V, Ps, FEs, Prgs)$ where Ps , FEs , $Prgs$ are sets of predicate, function, and program symbols respectively.

Let us give definitions of the sets of formulas Fr^Σ , functional expressions FEx^Σ , program texts Pt^Σ , and Floyd-Hoare assertions $FHF r^\Sigma$.

The sets Fr^Σ , FE^Σ , and Pt^Σ are defined inductively (here we use the symbols of compositions, predicates and functions in the purely syntactic sense, i.e. they are currently not associated with semantics):

1. if $ps \in Ps$, then $ps \in Fr^\Sigma$;
2. if $fes \in FEs$, then $fes \in FE^\Sigma$;
3. if $prgs \in Prgs$, then $prgs \in Pt^\Sigma$;
4. if $\Phi, \Psi \in Fr^\Sigma$, then $\Phi \vee \Psi, \neg\Phi, \sim\Phi, \exists x\Phi \in Fr^\Sigma$;
5. $\Rightarrow v, v \Rightarrow_a \in FE^\Sigma$;
6. if $n \geq 1, \Phi \in Fr^\Sigma, fe_1, \dots, fe_n \in FE^\Sigma$, and $\bar{x} \in \bar{U}$, then $S_P^{\bar{x}}(\Phi, fe_1, \dots, fe_n) \in Fr^\Sigma$;
7. if $n \geq 1, fe, fe_1, \dots, fe_n \in FE^\Sigma$, and $\bar{x} \in \bar{U}$, then $S_F^{\bar{x}}(fe, fe_1, \dots, fe_n) \in FE^\Sigma$;
8. if $n \geq 1, prg \in Pt^\Sigma, fe_1, \dots, fe_n \in FE^\Sigma$, and $\bar{x} \in \bar{U}$, then $S_F^{\bar{x}}(prg, fe_1, \dots, fe_n) \in Pt^\Sigma$;
9. if $x \in V^+$ and $fe \in FE^\Sigma$, then $AS^x(fe) \in Pt^\Sigma$;
10. $id \in Pt^\Sigma$;
11. if $prg_1, prg_2 \in Pt^\Sigma$, then $pr_1 \bullet pr_2 \in Pt^\Sigma$;
12. if $\Phi \in Fr^\Sigma$ and $prg_1, prg_2 \in Pt^\Sigma$, then $IF(\Phi, prg_1, prg_2) \in Pt^\Sigma$;
13. if $\Phi \in Fr^\Sigma$ and $prg \in Pt^\Sigma$, then $WH(\Phi, prg) \in Pt^\Sigma$.

To avoid syntactical nondeterminism, parentheses can be used.

The set $FHFr^\Sigma$ is the set of all formulas of the form $\{p\}f\{q\}$, where $p, q \in Fr^\Sigma$ and $f \in Pt^\Sigma$.

Please note that we often use the same notation both for predicates and for formulas, e.g. depending on the context, p can be treated as a predicate or as a function; the same concerns functions and functional expressions.

Definition 8. Let $\Sigma = (V, Ps, FEs, Prgs)$ be a logic signature and A be a set. Then an interpretation J is a tuple $(CPAND_{CC}(V, A), I_{Ps}, I_{FEs}, I_{Prgs})$, where $I_{Ps} : Ps \rightarrow Pr_{CC}(V, A)$ is an interpretation mapping for predicate symbols, $I_{FEs} : FEs \rightarrow Fn_{CC}(V, A)$ and $I_{Prs} : Prs \rightarrow Fn_{CC}(V, A)$ are interpretation mappings for function and program symbols, respectively.

For any interpretation $J = (CPAND_{CC}(V, A), I_{Ps}, I_{FEs}, I_{Prgs})$ we denote by J_{Fr} , J_{FE} , and J_{Pt} the formula, function, and program text interpretation mappings

$$J_{Fr} : Fr^\Sigma \rightarrow Pr_{CC}(V, A),$$

$$J_{FE} : FE^\Sigma \rightarrow Fn_{CC}(V, A),$$

$$J_{Pt} : Pt^\Sigma \rightarrow Fn_{CC}(V, A),$$

which are the standard extensions of I_{Ps}, I_{FEs} , and I_{Prgs} to Fr^Σ, FE^Σ , and Pt^Σ respectively (defined by structural induction). Also, we denote by J_{FHFr} the interpretation mapping of Floyd-Hoare assertions $J_{FHFr} : FHFr^\Sigma \rightarrow Pr_{CC}(V, A)$ defined as follows:

$$J_{FHFr}(\{p\}f\{q\}) = FH(J_{Fr}(p), J_{Pt}(f), J_{Fr}(q)).$$

Here we will not define interpretations explicitly expecting that they are clear from the context. For any $P \in Fr^\Sigma$ or $P \in FHFr^\Sigma$ we will denote by P_J or $(P)_J$ the predicate that corresponds to P under interpretation J . We will omit the index J when it is clear from the context.

Definition 9. A formula $P \in Fr^\Sigma$ or a Floyd-Hoare assertion $P \in FHFr^\Sigma$ is valid (irrefutable) in an interpretation J (denoted as $J \models P$), if $P_J^F = \emptyset$.

Definition 10. A formula $P \in Fr^\Sigma$ or a Floyd-Hoare assertion $P \in FHFr^\Sigma$ is logically valid (denoted as $\models P$), if it is valid in every interpretation.

We will also need special logical truth-consequence and falsity-consequence relations [10] $\models_T, \models_F \subseteq Fr^\Sigma \times Fr^\Sigma$.

Definition 11. A formula $Q \in Fr^\Sigma$ is a truth-consequence of a formula $P \in Fr^\Sigma$ in an interpretation J (denoted as $P_J \models_T Q$), if $P_J^T \subseteq Q_J^T$. A formula $Q \in Fr^\Sigma$ is a logical truth-consequence of a formula $P \in Fr^\Sigma$ (denoted as $P \models_T Q$), if $P_J \models_T Q$ for every interpretation J .

Definition 12. A formula $Q \in Fr^\Sigma$ is a falsity-consequence of a formula $P \in Fr^\Sigma$ in an interpretation J (denoted as $P_J \models_F Q$), if $P_J^F \supseteq Q_J^F$. A formula $Q \in Fr^\Sigma$ is a logical falsity-consequence of a formula $P \in Fr^\Sigma$ (denoted as $P \models_F Q$), if $P_J \models_F Q$ for every interpretation J .

7 Inference System for a Complemented Partial Floyd-Hoare Logic

To make the program logic CPFHL which we have defined applicable to software verification problems it is necessary to present an inference system. Such an inference system could be based on the inference system for the classical Floyd-Hoare logic with total predicates for the language WHILE [14], but it is known to be unsound in the case of partial predicates [11] which is considered in the paper. For this reason we present new inference rules based on program algebras with the composition of predicate complement. Obtained system will be sound.

We will write $\vdash_X p$ to denote that a formula p is *derived* in some inference system X . An inference system X is *sound*, if $\vdash_X p \Rightarrow \models p$ for each formula p , and is *complete*, if $\models p \Rightarrow \vdash_X p$ for each p .

Taking into consideration the obtained results we write the following inference rules ($v, x \in V^+, \bar{x} \in \bar{U}, p, p', q, q', r \in Fr^\Sigma, h, g_1, \dots, g_n \in FE^\Sigma, f, g \in Pt^\Sigma$):

$$\begin{array}{l}
 R_AS \frac{}{\{S_P^x(p, h)\} AS^x(h) \{p\}} \\
 R_SKIP \frac{}{\{p\} id \{p\}} \\
 R_SSEQ \frac{\{p\} f \{q\}, \{q\} g \{r\}, \{\sim q\} g \{r\}}{\{p\} f \bullet g \{r\}}
 \end{array}$$

$$\begin{array}{l}
 R_IF \frac{\{r \wedge p\} f \{q\}, \{\neg r \wedge p\} g \{q\}}{\{p\} IF(r, f, g) \{q\}} \\
 R_UWH \frac{\{r \wedge p\} f \{p\}, \{r \wedge (\sim p)\} f \{p\}}{\{p\} WH(r, f) \{\neg r \wedge p\}} \\
 R_SFID \frac{}{\{S_P^{\bar{x}}(p, g_1, \dots, g_n)\} S_F^{\bar{x}}(id, g_1, \dots, g_n) \{p\}} \\
 R_SF \frac{\{p\} S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f \{q\}}{\{p\} S_F^{\bar{x}}(f, g_1, \dots, g_n) \{q\}} \\
 R_CONSTF \frac{\{p'\} f \{q'\}}{\{p\} f \{q\}}, p \models_T p', q' \models_F q
 \end{array}$$

Let us make the following comments to these rules:

- rules R_AS , R_SFID and R_SF are the only rules oriented on the class of nominative data with complex names and values; other rules can be considered for any class of data D ;
- rules R_SKIP and R_IF are traditional rules for Floyd-Hoare logics; they do not require any changes;
- rules R_SSEQ and R_UWH were proposed and investigated in the previous sections;
- rules R_SFID and R_SF specify procedure calls;
- consequence rules can be formulated in different forms; here we use the rule R_CONSTF based on special consequence relations \models_T and \models_F . In the case of total predicates this rule will be equivalent to traditional consequence rule.

We denote the inference system presented by the above rules as RCN .

Theorem 3. *The inference system RCN is sound, i.e. for any Floyd-Hoare assertion $P \in FHF r^{\Sigma}$ we have that*

$$\vdash_{RCN} P \Rightarrow \models P.$$

Proof. We prove the theorem by induction on the length of inference of P . Let $J = (CPAND_{CC}(V, A), I_{Ps}, I_{Fes}, I_{Prgs})$.

- Consider the case when P has the form $\{S_P^{\bar{x}}(p, h)\} AS^x(h) \{p\}$ and P is inferred in RCN , i.e. $\vdash_{RCN} P$ ($q, p \in Fr^{\Sigma}, h \in FE^{\Sigma}$) by rule R_AS . Given an interpretation J we should prove that $J \models \{S_P^{\bar{x}}(p, h)\} AS^x(h) \{p\}$. This means (by the definition of weak Floyd-Hoare triple) that we should prove the following statement: for any $d, d' \in ND_{CC}(V, A)$ if $S_P^{\bar{x}}(p, h)_J(d) \downarrow = T$, $AS^x(h)_J(d) \downarrow = d'$, and $p_J(d') \downarrow$ then $p_J(d') = T$. By definition of the composition of superposition into a predicate we have that $S_P^{\bar{x}}(p, h)_J(d) = p_J(d \nabla_a^x h_J(d))$. By definition of assignment composition we have that $AS^x(h)_J(d) = d \nabla_a^x h_J(d)$. Since $p_J(d \nabla_a^x h_J(d)) = T$ and $d \nabla_a^x h_J(d) = d'$ we obtain that $p_J(d') = T$.
- The case when P has the form $\{p\} id \{p\}$ i.e. the rule R_SKIP is used to infer P is trivial.

- The case when P is obtained by rule R_SSEQ has been proved in Sect. 4 of this paper.
- The case when P is obtained by rule R_IF is a traditional one.
- The case when P is obtained by rule R_UWH has been proved in Sect. 5 of this paper.
- Consider the case when P is obtained by rule R_SFID . It means that P has the form $\{S_P^{\bar{x}}(p, g_1, \dots, g_n)\}S_F^{\bar{x}}(id, g_1, \dots, g_n)\{p\}$ ($\bar{x} = (x_1, \dots, x_n)$). Given an interpretation J we should prove that

$$J \models \{S_P^{\bar{x}}(p, g_1, \dots, g_n)\}S_F^{\bar{x}}(id, g_1, \dots, g_n)\{p\}.$$

This means (by the definition of weak Floyd-Hoare triple) that we should prove the following statement:

$$\text{for any } d, d' \in ND_{CC}(V, A) \text{ if } S_P^{\bar{x}}(p, g_1, \dots, g_n)_J(d) \Downarrow = T, \\ S_F^{\bar{x}}(id, g_1, \dots, g_n)_J(d) \Downarrow = d', \text{ and } p_J(d') \Downarrow \text{ then } p_J(d') = T.$$

By the definition of the composition of superposition into a predicate we have that $S_P^{x_1, \dots, x_n}(p, g_1, \dots, g_n)_J(d) = p_J(\dots (d\nabla_a^{x_1} g_{1J}(d)) \dots \nabla_a^{x_n} g_{nJ}(d))$. Obtained value is equal to T .

By the definition of the composition of superposition into a function we have that $S_F^{x_1, \dots, x_n}(id, g_1, \dots, g_n)_J(d) = id_J(\dots (d\nabla_a^{x_1} g_{1J}(d)) \dots \nabla_a^{x_n} g_{nJ}(d)) = d'$. Therefore, $p_J(d') \Downarrow = T$.

- Consider the case when P is obtained by rule R_SF . In this case P has the form $\{p\}S_F^{\bar{x}}(f, g_1, \dots, g_n)\{q\}$. Given an interpretation J we should prove that $J \models \{p\}S_F^{\bar{x}}(f, g_1, \dots, g_n)\{q\}$ under assumption $J \models \{p\}S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f\{q\}$. It means that we should prove the following statement:
for any $d, d' \in ND_{CC}(V, A)$ if $p_J(d) \Downarrow = T$, $S_F^{\bar{x}}(f, g_1, \dots, g_n)_J(d) \Downarrow = d'$, and $q_J(d') \Downarrow$ then $q_J(d') = T$ using inductive hypothesis.

First, let us prove that $(S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f)_J(d) \cong S_F^{\bar{x}}(f, g_1, \dots, g_n)_J(d)$ for any $d \in ND_{CC}(V, A)$.

$$\text{Indeed, } (S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f)_J(d) \cong f_J(S_F^{\bar{x}}(id, g_1, \dots, g_n)_J(d)) \cong \\ \cong f_J(id_J(\dots (d\nabla_a^{u_1} g_{1J}(d)) \dots \nabla_a^{u_n} g_{nJ}(d))) \cong \\ \cong f_J(\dots (d\nabla_a^{u_1} g_{1J}(d)) \dots \nabla_a^{u_n} g_{nJ}(d)) \cong S_F^{\bar{x}}(f, g_1, \dots, g_n)_J(d).$$

Let $p_J(d) \Downarrow = T$, $S_F^{\bar{x}}(f, g_1, \dots, g_n)_J(d) \Downarrow = d'$, and $q_J(d') \Downarrow$.

Since $(S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f)_J(d) \cong S_F^{\bar{x}}(f, g_1, \dots, g_n)_J(d)$ we have that $(S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f)_J(d) \Downarrow = d'$.

Then, by induction hypothesis for $\{p\}S_F^{\bar{x}}(id, g_1, \dots, g_n) \bullet f\{q\}$ we obtain the required property $q_J(d') \Downarrow = T$.

- Consider the case when P is obtained by rule $R_CONSTRF$. It means that P has the form $\{p\}f\{q\}$. Given an interpretation J we should prove $J \models \{p\}f\{q\}$ under assumptions $J \models \{p'\}f\{q'\}$, $p_J \Vdash_T p'$ and $q'_J \Vdash_F q$.
Indeed, let $d \in ND_{CC}(V, A)$. Assume that $p_J(d) \Downarrow = T$, $f_J(d) \Downarrow = d'$ and $q_J(d') \Downarrow$ for some $d' \in ND_{CC}(V, A)$. Since $p_J \Vdash_T p'$ we have $p'_J(d) \Downarrow = T$. Since $f_J(d) \Downarrow = d'$ and $J \models \{p'\}f\{q'\}$ we have $q'_J(d') = T$ when $q'_J(d') \Downarrow$. Assume that $q_J(d') = F$. Since $q'_J \Vdash_F q$ we should have $q'_J(d') = F$. We have a contradiction with $q'_J(d') = T$. Therefore, $q_J(d') = T$.

□

In the system *RCN* new unconventional consequence relations \models_T and \models_F were used. Their main semantic properties were studied in [25]. Further investigation will permit one to substitute these consequence relations by the corresponding inference relations \vdash_T and \vdash_F . A detailed investigation of inference methods for CPFHL is planned for the forthcoming publications.

8 Conclusion

We have proposed a modified inference system for an extended Floyd-Hoare logic for partial pre- and post-conditions and partial programs studied in [10, 11, 26]. The modifications primarily concern the sequence and while rules and have been formulated in program algebras extended with the composition of predicate complement. The addition of these rules does not change the soundness of the system. Moreover, the new rules have no semantic constraints. The obtained results can be useful for verification of programs with respect to specifications which can contain partial operations.

In the future we plan to make detailed comparison of inference systems and propose new modifications to improve their efficiency.

References

1. Ivanov, I., Nikitchenko, M.: On the sequence rule for the Floyd-Hoare logic with partial pre-and post-conditions. In: CEUR Workshop Proceedings. Proceedings of the 14th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kyiv, Ukraine, 14–17 May 2018, vol. 2104, pp. 716–724 (2018)
2. Floyd, R.: Assigning meanings to programs. In: Mathematical Aspects of Computer Science, vol. 19, pp. 19–32 (1967)
3. Hoare, C.: An axiomatic basis for computer programming. *Commun. ACM* **12**(10), 576–580 (1969)
4. Apt, K.: Ten years of Hoare’s logic: a survey - part I. *ACM Trans. Program. Lang. Syst.* **3**(4), 431–483 (1981)
5. GNU: Octave. <https://www.gnu.org/software/octave/>
6. MathWorks: MATLAB. <https://www.mathworks.com/products/matlab.html>
7. Jones, C.: Reasoning about partial functions in the formal development of programs. *Electron. Notes Theor. Comput. Sci.* **145**, 3–25 (2006). Proceedings of AVoCS 2005. Elsevier
8. Hähnle, R.: Many-valued logic, partiality, and abstraction in formal specification languages. *Logic J. IGPL* **13**(4), 415–433 (2005)
9. Gries, D., Schneider, F.: Avoiding the undefined by underspecification. Technical report, Ithaca, NY, USA (1995)
10. Nikitchenko, M., Kryvolap, A.: Properties of inference systems for Floyd-Hoare logic with partial predicates. *Acta Electrotechnica et Informatica* **13**(4), 70–78 (2013)
11. Kryvolap, A., Nikitchenko, M., Schreiner, W.: Extending Floyd-Hoare logic for partial pre- and postconditions. In: Ermolayev, V., Mayr, H.C., Nikitchenko, M., Spivakovsky, A., Zholtkevych, G. (eds.) *ICTERI 2013. CCIS*, vol. 412, pp. 355–378. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03998-5_18

12. Kornilowicz, A., Kryvolap, A., Nikitchenko, M., Ivanov, I.: Formalization of the algebra of nominative data in Mizar. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, 3–6 September 2017, pp. 237–244 (2017)
13. Kornilowicz, A., Kryvolap, A., Nikitchenko, M., Ivanov, I.: Formalization of the nominative algorithmic algebra in Mizar. In: Świątek, J., Borzowski, L., Wilimowska, Z. (eds.) ISAT 2017. AISC, vol. 656, pp. 176–186. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67229-8_16
14. Nielson, H., Nielson, F.: *Semantics with Applications - a Formal Introduction*. Wiley, Hoboken (1992). Wiley professional computing
15. Nikitchenko, N.S.: A composition nominative approach to program semantics. Technical report, IT-TR 1998–020, Technical University of Denmark (1998)
16. Skobelev, V., Ivanov, I., Nikitchenko, M.: Nominative data with ordered set of names. *Comput. Sci. J. Moldova* **25**, 195–216 (2017)
17. Ivanov, I.: On representations of abstract systems with partial inputs and outputs. In: Gopal, T.V., Agrawal, M., Li, A., Cooper, S.B. (eds.) TAMC 2014. LNCS, vol. 8402, pp. 104–123. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06089-7_8
18. Ivanov, I.: An abstract block formalism for engineering systems. In: Ermolayev, V., et al. (eds.) Proceedings of the 9th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer, CEUR Workshop Proceedings, Kherson, Ukraine, 19–22 June 2013, vol. 1000, pp. 448–463. CEUR-WS.org (2013)
19. Skobelev, V., Nikitchenko, M., Ivanov, I.: On algebraic properties of nominative data and functions. In: Ermolayev, V., Mayr, H., Nikitchenko, M., Spivakovsky, A., Zholtkevych, G. (eds.) ICTERI 2014. CCIS, vol. 469, pp. 117–138. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13206-8_6
20. Nikitchenko, M., Ivanov, I.: Composition-nominative languages of programs with associative denaming. *Visnyk (Bull.) Lviv Univ. Ser. Appl. Math. Inform.* **16**, 124–139 (2010)
21. Nikitchenko, M., Ivanov, I., Skobelev, V.: Proving properties of programs on hierarchical nominative data. *Comput. Sci. J. Moldova* **24**(3(72)), 371–398 (2016)
22. Kleene, S.: *Introduction to Metamathematics*. North-Holland Publishing Co. and P. Noordhoff, Amsterdam and Groningen (1952)
23. Kornilowicz, A., Ivanov, I., Nikitchenko, M.: Kleene algebra of partial predicates. *Formalized Math.* **26**, 11–20 (2018)
24. Kornilowicz, A., Kryvolap, A., Nikitchenko, M., Ivanov, I.: An approach to formalization of an extension of Floyd-Hoare logic. In: Proceedings of the 13th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer, Kyiv, Ukraine, 15–18 May 2017, pp. 504–523 (2017)
25. Nikitchenko, M., Shkiliak, S.: Semantic properties of T-consequence relation in logics of quasiary predicates. *Comput. Sci. J. Moldova* **23**(2(68)), 102–122 (2015)
26. Nikitchenko, M., Ivanov, I., Kornilowicz, A., Kryvolap, A.: Extended Floyd-Hoare Logic over relational nominative data. In: Bassiliades, N., et al. (eds.) ICTERI 2017. CCIS, vol. 826, pp. 41–64. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76168-8_3