Extending Floyd-Hoare Logic for Partial Pre- and Postconditions

Andrii Kryvolap[1], Mykola Nikitchenko[1], and Wolfgang Schreiner[2]

[1] Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
krivolapa@gmail.com, nikitchenko@unicyb.kiev.ua
[2] Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

Abstract. Traditional (classical) Floyd-Hoare logic is defined for a case of total pre- and postconditions while programs can be partial. In the chapter we propose to extend this logic for partial conditions. To do this we first construct and investigate special program algebras of partial predicates, functions, and programs. In such algebras program correctness assertions are presented with the help of a special composition called Floyd-Hoare composition. This composition is monotone and continuous. Considering the class of constructed algebras as a semantic base we then define an extended logic – Partial Floyd-Hoare Logic – and investigate its properties. This logic has rather complicated soundness constraints for inference rules, therefore simpler sufficient constraints are proposed. The logic constructed can be used for program verification.

Keywords. Program algebra, program logic, partial predicate, soundness, composition-nominative approach.

1 Introduction

Program logics are the main formalisms used for proving assertions about program properties. A well-known classical Floyd-Hoare logic (here also referred to as CFHL) [1, 2] is an example of such logics. Semantically, this logic is defined for the case of total predicates though programs can be partial (non-terminating). In this case program correctness assertions can be presented with the help of a special composition over total predicates called Floyd-Hoare composition (FH-composition). However, a straightforward extension of CFHL for partial predicates meets some difficulties. The first one is that the FH-composition will not be monotone with respect to partial predicates. Monotonicity is an important property that gives the possibility to reason about the correctness of the program based on the correctness of its approximations.

That is why the need of a modified definition of the classical Floyd-Hoare logic for the case of partial mappings arises. We construct such logics in this paper and called them Partial Floyd-Hoare Logics (PFHL). Here we will consider only a special case of partial mappings (predicates, ordinary functions, and program functions) defined over sets of named values (nominative sets). Mappings over classes of such sets are

called quasiary mappings [3] and corresponding program algebras are called quasiary program algebras. They form the semantic component of PFHL.

The syntactic component of such logics is presented by their languages and systems of inference rules. We study the possibility to use classical rules for modified logics with a monotone Floyd-Hoare composition. Systems of such inference rules should be sound to be of a practical use. This could be achieved by adding proper restrictions (constraints) to the inference rules of the classical Floyd-Hoare logic that fail to be sound. Thus, the proposed scheme permits to define a Floyd-Hoare-like logic for partial pre- and postconditions.

The rest of the chapter is structured as follows. In Section 2 we analyze the traditional Floyd-Hoare logic and its potential to be extended for partial predicates. In Section 3 we describe program algebras of quasiary predicates and functions at different levels of abstraction, define a modified Floyd-Hoare composition and specify the syntax for the modified logic. In Section 4 we study the soundness of the system of inference rules for the introduced program algebras and define constraints for the rules of the systems. We prove that obtained logic is indeed an extension of classical Floyd-Hoare logic. Also simpler constraints are formulated. In Section 5 we describe related work, and finally, we formulate conclusions in Section 6.

2 Analysis of Classical Floyd-Hoare Logic

We first analyze the CFHL constructed for a very simple imperative language **WHILE** [4]. The grammar of the (slightly modified) language is defined as follows:

$a ::= k \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \mid (a)$

$b ::= T \mid F \mid a_1 = a_2 \mid a_1 \leq a_2 \mid b_1 \vee b_2 \mid \neg b \mid (b)$

$S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \mid \text{begin } S \text{ end}$

where:

- k ranges over integers $Int = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$,
- x ranges over variables (names) $V = \{ N, R, X, Y, Z, \ldots\}$
- a ranges over arithmetic expressions *Aexpr*,
- b ranges over Boolean expressions *Bexpr*,
- S ranges over statements (programs) *Stm*.

Semantics of arithmetical expressions is denoted as $[\![a]\!]$ and of Boolean expressions as $[\![b]\!]$. Program states (also called data) are considered as collections of named values. Program correctness assertions (referred to simply as assertions) are Floyd-Hoare triples of the form $\{p\}S\{q\}$ where p, q are predicates of some basic predicate logic and S is a statement. An assertion $\{p\}S\{q\}$ is said to be valid [4] if the following proposition holds: if S is started in a state satisfying p, and if S eventually terminates in some final state, then this final state will satisfy q.

Analyzing this definition of assertion validity we should admit that it permits semantic treatment of assertion $\{p\}S\{q\}$ as a certain predicate defined on states. This treatment of assertions will not be monotone under predicate extension. Indeed, consider informally the following assertion:

$$\{T\} \text{ while } T \text{ do skip } \{F\}.$$

This Floyd-Hoare triple will be true on all states because the infinite loop is undefined on all states, and thus on all states the condition of validity for this assertion is satisfied. Now consider a triple $\{T\}$ skip $\{F\}$ that is false on all states. However, the mapping 'skip' is an extension of 'while T do skip'. Thus, monotonicity of assertion validity fails.

Now we make a short analysis of the inference system for **WHILE** presented by rules of Table 1 [4].

Table 1. WHILE inference system for concrete syntax.

$\{p[x \mapsto [\![a]\!]]\}\ x := a\ \{p\}$	*R_as*
$\{p\}\text{skip}\{p\}$	*R_skip*
$\dfrac{\{p\}S_1\{q\},\{q\}S_2\{r\}}{\{p\}S_1;S_2\{r\}}$	*R_seq*
$\dfrac{\{[\![b]\!] \wedge p\}S_1\{q\},\{\neg[\![b]\!] \wedge p\}S_2\{q\}}{\{p\}\text{ if } b \text{ then } S_1 \text{ else } S_2\{q\}}$	*R_if*
$\dfrac{\{[\![b]\!] \wedge p\}S\{p\}}{\{p\}\text{ while } b \text{ do } S\{\neg[\![b]\!] \wedge p\}}$	*R_wh*
$\dfrac{\{p'\}f\{q'\}}{\{p\}f\{q\}}$ if $p \to p'$ and $q' \to q$	*R_cons*

These rules are oriented on the concrete syntax of **WHILE,** and moreover they use semantic mappings $[\![a]\!]$ and $[\![b]\!]$. We adopt the semantic-syntactic style, thus, we present these rules as constructed over special semantic program algebra. In this algebra (see the formal definitions in the next section) we semantically treat program structuring constructs as special operations called compositions. For **WHILE** the following compositions are introduced (we use notation of [3, 5]):

- superposition S_P^x;
- assignment As^x;
- sequential execution $\bullet$;
- conditional *IF*;
- cycle (loop) *WH*.

In the sequel pre- and postconditions are denoted (possibly with indexes) as *p*, *q*, *r*; ordinary functions as *h*, *s*; program functions (semantics of statements) as *f*, *g*. Statement 'skip' is semantically represented by identity function *id*. Data (states) are usually denoted as *d*.

Note, that we do not make an explicit distinction between a formula and its interpretation. Thus, in the assertion $\{p\}f\{q\}$ we treat p and q syntactically as formulas of the logic language and semantically as predicates of the program algebra.

According to the introduced notations the inference system can be presented by rules of Table 2.

Table 2. WHILE inference system for semantic program algebra.

$\{S_P^x(p,h)\}AS^x(h)\{p\}$	*R_AS*
$\{p\}id\{p\}$	*R_SKIP*
$\dfrac{\{p\}f\{q\},\{q\}g\{r\}}{\{p\}f \bullet g\{r\}}$	*R_SEQ*
$\dfrac{\{r \wedge p\}f\{q\},\{\neg r \wedge p\}g\{q\}}{\{p\}\,IF(r,f,g)\,\{q\}}$	*R_IF*
$\dfrac{\{r \wedge p\}f\{p\}}{\{p\}\,WH(r,f)\,\{\neg r \wedge p\}}$	*R_WH*
$\dfrac{\{p'\}f\{q'\}}{\{p\}f\{q\}}$ if $p \to p'$ and $q' \to q$	*R_CONS*

In the next sections we define a class of algebras of partial predicates (semantics of logic) and modify the inference system for such predicates, thus obtaining Partial Floyd-Hoare Logics.

3 Quasiary Program Algebras

To modify the classical Floyd-Hoare logic for partial quasiary mappings, we will use semantic-syntactic scheme [3, 5]. This means that we will first define the semantics in the form of classes of quasiary program algebras. Then the language of the logic will be defined as well as the interpretation mappings.

To emphasize a mapping's *partiality/totality* we write the sign $\xrightarrow{p}$ for partial mappings and the sign $\xrightarrow{t}$ for total mappings. Given an arbitrary partial mapping $\mu: D \xrightarrow{p} D'$, $d \in D$, $S \subseteq D$, $S' \subseteq D'$ we write:

- $\mu(d)\downarrow$ to denote that μ is defined on d;
- $\mu(d)\downarrow = d'$ to denote that μ is defined on d with a value d';
- $\mu(d)\uparrow$ to denote that μ is undefined on d;
- $\mu[S] = \{\mu(d) \mid \mu(d)\downarrow, d \in S\}$ to denote the image of S under μ;
- $\mu^{-1}[S'] = \{d \mid \mu(d)\downarrow, \mu(d) \in S'\}$ to denote the preimage (inverse image) of S' under μ.

3.1 Classes of Quasiary Mappings

Let V be a set of *names (variables)*. Let A be a set of *basic values*. Given V and A, the class ${}^VA$ of *nominative sets* is defined as the class of all partial mappings from V to A, thus, ${}^VA = V \xrightarrow{p} A$. Informally speaking, nominative sets represent states of variables.

Though nominative sets are defined as mappings, we follow mathematical traditions and also use a set-like notation for these objects. In particular, the notation $d = [v_i \mapsto a_i \mid i \in I]$ describes a nominative set d where $v_i \mapsto a_i \in_n d$, which means that $d(v_i)$ is defined and its value is a_i ($d(v_i)\downarrow = a_i$). The main operation for nominative sets is the binary *total overriding operation* $\nabla: {}^V\!A \times {}^V\!A \xrightarrow{t} {}^V\!A$ defined by the formula $d_1 \nabla d_2 = [\, v \mapsto a \mid v \mapsto a \in_n d_2 \vee (v \mapsto a \in_n d_1 \wedge \neg\exists a'(\, v \mapsto a' \in_n d_2\,))]$. Intuitively, given d_1 and d_2 this operation yields a new nominative set which consists of named pairs of d_2 and those pairs of d_1 whose names do not occur in d_2.

Let $Bool = \{F, T\}$ be the set of Boolean values. Let $Pr^{V,A} = {}^V\!A \xrightarrow{p} Bool$ be the set of all partial predicates over ${}^V\!A$. Such predicates are called *partial quasiary predicates*. Let $Fn^{V,A} = {}^V\!A \xrightarrow{p} A$ be the set of all partial functions from ${}^V\!A$ to A. Such functions are called *partial quasiary ordinary functions*. Here 'ordinary' means that the range of such functions is the set of basic values A. Let $FPrg^{V,A} = {}^V\!A \xrightarrow{p} {}^V\!A$ be the set of all partial functions from ${}^V\!A$ to ${}^V\!A$. Such functions are called *bi-quasiary functions*.

Quasiary predicates represent conditions which occur in programs, quasiary ordinary functions represent the semantics of program expressions, and bi-quasiary functions represent program semantics.

The terms 'partial' and 'ordinary' are usually omitted. In a general term, elements of $Pr^{V,A}$, $Fn^{V,A}$, and $FPrg^{V,A}$ are called *quasiary mappings*.

3.2 Hierarchy of Quasiary Program Algebras and Logics

Based on algebras with three carriers ($Pr^{V,A}$, $Fn^{V,A}$, and $FPrg^{V,A}$) we can define logics of three types (see details in [3, 5]):

1) *pure quasiary predicate logics based on algebras with one sort*: $Pr^{V,A}$;
2) *quasiary predicate-function logics based on algebras with two sorts*: $Pr^{V,A}$ and $Fn^{V,A}$;
3) *quasiary program logics based on algebras with three sorts*: $Pr^{V,A}$, $Fn^{V,A}$, and $FPrg^{V,A}$.

The basic compositions of logics of the first type are disjunction $\vee$, negation $\neg$, renomination $R\frac{\bar{v}}{\bar{x}}$, and quantification $\exists x$.

The basic compositions of logics of the second type additionally include superpositions $S_F^{\bar{v}}$ and $S_P^{\bar{v}}$, and denomination function $'x$.

The basic compositions of logics of the third type additionally include the following program compositions: the parametric assignment composition $AS^x : Fn^{V,A} \xrightarrow{t} FPrg^{V,A}$, the composition of sequential execution $\bullet : FPrg^{V,A} \times FPrg^{V,A} \xrightarrow{t} FPrg^{V,A}$, the conditional composition $IF : Pr^{V,A} \times FPrg^{V,A} \times FPrg^{V,A} \xrightarrow{t} FPrg^{V,A}$, the cyclic (loop) composition $WH : Pr^{V,A} \times FPrg^{V,A} \xrightarrow{t} FPrg^{V,A}$, and identity function id: $FPrg^{V,A}$. Also we need

compositions that describe properties of the programs. The Floyd-Hoare composition $FH : Pr^{V,A} \times FPrg^{V,A} \times Pr^{V,A} \xrightarrow{t} Pr^{V,A}$ is the most important of them. Its formal definition will be given in the next subsection.

3.3 Formal Definition of the Floyd-Hoare Composition

The required definition stems from the analysis of Floyd-Hoare assertions with total predicates (see, for example, [4]). Namely, an assertion $\{p\}f\{q\}$ is said to be valid if and only if

for all d from ${}^{V}A$ if $p(d) = T$, $f(d)\downarrow = d'$ for some d' then $q(d') = T$.

This definition permits to treat $\{p\}f\{q\}$ as a predicate because it is a pointwise definition. Rewriting this definition for different cases we get the following matrices (Table 3) specifying the logical values of $\{p\}f\{q\}$ for an arbitrary d.

Table 3. Logical values of $\{p\}f\{q\}$ for total predicates.

a) $f(d)$ is defined

$p(d) \setminus q(f(d))$	F	T
F	T	T
T	F	T

b) $f(d)$ is undefined

$p(d)$	$\{p\}f\{q\}(d)$
F	T
T	T

The example given in Section 2 demonstrates that for partial predicates monotonicity fails for the case when $f(d)$ is undefined and $p(d) \downarrow = T$. Therefore, to define a monotone interpretation of Floyd-Hoare triples for partial predicates we should change the value of $\{p\}f\{q\}$ for this case. Also, we should specify the logical values of $\{p\}f\{q\}$ for the cases when pre- or postconditions are not defined. In Table 4 such unspecified logical values are denoted by the question marks.

Table 4. Logical values of $\{p\}f\{q\}$ for partial predicates, where question mark represents values that should be changed to a Boolean values.

a) $f(d)$ is defined

$p(d) \setminus q(f(d))$	F	T	*undefined*
F	T	T	?
T	F	T	?
undefined	?	?	?

b) $f(d)$ is undefined

$p(d)$	$\{p\}f\{q\}(d)$
F	T
T	?
undefined	?

While defining a required composition we adopt the following requirements:

- partiality of mappings;
- monotonicity of a composition on all its arguments;
- maximal definiteness of the obtained predicates (we call this as Kleene's requirement).

To do this we use techniques for non-deterministic semantics described in [6]. We will treat the case when a predicate is '*undefined*' as non-deterministic values T and

F. Thus, we can use Boolean values given in Table 4 to evaluate a set of values for cases with question marks. The obtained results are presented in Table 5.

Table 5. Logical values of $\{p\}f\{q\}$ for partial predicates presented as sets of Boolean values.

a) $f(d)$ is defined

$p(d) \setminus q(f(d))$	$\{F\}$	$\{T\}$	$\{F,T\}$
$\{F\}$	$\{T\}$	$\{T\}$	$\{T\}$
$\{T\}$	$\{F\}$	$\{T\}$	$\{F,T\}$
$\{F,T\}$	$\{F,T\}$	$\{T\}$	$\{F,T\}$

b) $f(d)$ is undefined

$p(d)$	$\{p\}f\{q\}(d)$
$\{F\}$	$\{T\}$
$\{T\}$	$\{F,T\}$
$\{F,T\}$	$\{F,T\}$

Now, replacing non-deterministic results $\{F, T\}$ on *undefined* we get the final results (Table 6).

Table 6. Logical values of $\{p\}f\{q\}$ for partial predicates with undefined values.

a) $f(d)$ is defined

$p(d)\setminus q(f(d))$	F	T	*undefined*
F	T	T	T
T	F	T	*undefined*
undefined	*undefined*	T	*undefined*

b) $f(d)$ is undefined

$p(d)$	$\{p\}f\{q\}(d)$
F	T
T	*undefined*
undefined	*undefined*

The obtained matrices define an interpretation of $\{p\}f\{q\}$ for partial predicates. As was said earlier, we formalize such triples as a Floyd-Hoare composition $FH : Pr^{V,A} \times FPrg^{V,A} \times Pr^{V,A} \xrightarrow{t} Pr^{V,A}$ ($p, q \in Pr^{V,A}$, $f \in FPrg^{V,A}$, $d \in {}^{V}A$):

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

3.4 Formal Definition of Program Algebra Compositions

In the previous subsection the formal definition of FH-composition was presented. In this subsection we give definitions of other compositions (see details in [3, 5]).

Propositional compositions are defined as follows ($p, q \in Pr^{V,A}$, $d \in {}^{V}A$):

$$(p \vee q)(d) = \begin{cases} T, & \text{if } p(d)\downarrow = T \text{ or } q(d)\downarrow = T, \\ F, & \text{if } p(d)\downarrow = F \text{ and } q(d)\downarrow = F, \\ \text{undefined in other cases.} \end{cases} \quad (\neg p)(d) = \begin{cases} T, & \text{if } p(d)\downarrow = F, \\ F, & \text{if } p(d)\downarrow = T, \\ \text{undefined if } & p(d)\uparrow. \end{cases}$$

Unary parametric composition of existential quantification $\exists x$ with the parameter $x \in V$ is defined by the following formula ($p \in Pr^{V,A}$, $d \in {}^{V}A$):

$$(\exists x\ p)(d) = \begin{cases} T, & \text{if } b \in A \text{ exists: } p(d\nabla x \mapsto b)\downarrow = T, \\ F, & \text{if } p(d\nabla x \mapsto a)\downarrow = F \text{ for each } a \in A, \\ \text{undefined in other cases.} \end{cases}$$

Here $d\nabla x \mapsto a$ is a shorter form for $d\nabla[x \mapsto a]$.

Parametric n-ary superpositions with $\bar{x} = (x_1, ..., x_n)$ as the parameter are defined by the following formulas ($h, s_1, \ldots, s_n \in Fn^{V,A}$, $p \in Pr^{V,A}$, $d \in {}^{V}A$):

$$(S_F^{\bar{x}}(h, s_1, \ldots, s_n))(d) = h(d\nabla[x_1 \mapsto s_1(d), \ldots, x_n \mapsto s_n(d)]),$$

$$(S_P^{\bar{x}}(p, s_1, \ldots, s_n))(d) = p(d\nabla[x_1 \mapsto s_1(d), \ldots, x_n \mapsto s_n(d)]).$$

Null-ary parametric denomination composition with the parameter $x \in V$ is defined by the following formula ($d \in {}^{V}A$): $'x(d) = d(x)$.

Identical program composition $id \in FPrg^{V,A}$ is simple: $id(d) = d$ ($d \in {}^{V}A$).

Assignment composition is defined as follows ($h \in Fn^{V,A}$, $d \in {}^{V}A$):

$$AS^x(h)(d) = d\nabla[x \mapsto h(d)].$$

Composition of sequential execution is introduced in the ordinary way ($f, g \in FPrg^{V,A}$, $d \in {}^{V}A$):

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

Note, that we define $\bullet$ by commuting arguments of conventional functional composition: $f \bullet g = g \circ f$.

Conditional composition depends on the value of the first argument which is the condition itself ($p \in Pr^{V,A}$, $f, g \in FPrg^{V,A}$, $d \in {}^{V}A$):

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

Cycle (*loop*) *composition* is defined by the following formulas: $WH(p, f)(d) = d_n$, if there exists a sequence $d_0, ..., d_n$ such that $d_0 = d$, $f(d_0)\downarrow = d_1$, …, $f(d_{n-1})\downarrow = d_n$, $p(d_0)\downarrow = T$, … , $p(d_{n-1})\downarrow = T$, $p(d_n)\downarrow = F$ ($p \in Pr^{V,A}$, $f \in FPrg^{V,A}$, $d \in {}^{V}A$).

It means that we have defined the following *quasiary program algebra*:

$QPA(V, A) = < Pr^{V,A}, Fn^{V,A}, FPrg^{V,A}; \vee, \neg, S_F^{\bar{v}}, S_P^{\bar{v}}, 'x, \exists x, id, AS^x, \bullet, IF, WH, FH >$.

The class of such algebras is the main object of our investigation.

3.5 Formal Definition of Program Algebra Terms

Terms of the algebra $QPA(V, A)$ defined over sets of predicate symbols *Ps*, ordinary function symbols *Fs*, program symbols *Prs*, and variables V specify the syntax (the language) of the logic. We now give inductive definitions for terms $Tr(Ps, Fs, Prs, V)$, formulas $Fr(Ps, Fs, Prs, V)$, program texts $Pt(Ps, Fs, Prs, V)$, and Floyd-Hoare assertions $FHFr(Ps, Fs, Prs, V)$.

First we will define terms:

- if $F \in Fs$ then $F \in Tr(Ps, Fs, Prs, V)$;
- if $v \in V$ then $'v \in Tr(Ps, Fs, Prs, V)$;
- if $F \in Fs$, $t_1, \ldots, t_n \in Tr(Ps, Fs, Prs, V)$, and $v_1, \ldots, v_n \in V$ $(n \geq 0)$ are distinct variables then $S_F^{v_1, \ldots, v_n}(F, t_1, \ldots, t_n) \in Tr(Ps, Fs, Prs, V)$.

Then we will define program texts:

- if $Pr \in Prs$ then $Pr \in Pt(Ps, Fs, Prs, V)$;
- $id \in Pt(Ps, Fs, Prs, V)$;
- if $v \in V$ and $t \in Tr(Ps, Fs, Prs, V)$ then $AS^v(t) \in Pt(Ps, Fs, Prs, V)$;
- if $pr_1, pr_2 \in Pt(Ps, Fs, Prs, V)$ then $pr_1 \bullet pr_2 \in Pt(Ps, Fs, Prs, V)$;
- if $pr_1, pr_2 \in Pt(Ps, Fs, Prs, V)$ and $p \in Fr(Ps, Fs, Prs, V)$ then $IF(p, pr_1, pr_2) \in Pt(Ps, Fs, Prs, V)$;
- if $pr \in Pt(Ps, Fs, Prs, V)$ and $p \in Fr(Ps, Fs, Prs, V)$ then $WH(p, pr) \in Pt(Ps, Fs, Prs, V)$.

Finally, formulas and Floyd-Hoare triples are defined:

- if $P \in Ps$ then $P \in Fr(Ps, Fs, Prs, V)$;
- if $\Phi, \Psi \in Fr(Ps, Fs, Prs, V)$ then $\Phi \vee \Psi \in Fr(Ps, Fs, Prs, V)$;
- if $\Phi \in Fr(Ps, Fs, Prs, V)$ then $\neg\Phi \in Fr(Ps, Fs, Prs, V)$;
- if $P \in Ps$, $t_1, \ldots, t_n \in Tr(Ps, Fs, Prs, V)$, and $v_1, \ldots, v_n \in V$ $(n \geq 0)$ are distinct variables then $S_P^{v_1, \ldots, v_n}(P, t_1, \ldots, t_n) \in Fr(Ps, Fs, Prs, V)$;
- if $\Phi \in Fr(Ps, Fs, Prs, V)$ and $v \in V$ then $\exists v\Phi \in Fr(Ps, Fs, Prs, V)$;
- if $f \in Pt(Ps, Fs, Prs, V)$ and $p, q \in Fr(Ps, Fs, Prs, V)$ then $\{p\} f \{q\} \in FHFr(Ps, Fs, Prs, V)$.

After syntax and semantics have been defined, we need to specify the interpretation mappings, assuming that interpretation mappings for the predicate symbols $I_{Ps} : Ps \xrightarrow{t} Pr^{V,A}$, function symbols $I_{Fs} : Fs \xrightarrow{t} Fn^{V,A}$, and program symbols $I_{Prs} : Prs \xrightarrow{t} FPrg^{V,A}$ are given. Let $J_{Fr} : Fr(Fs, Ps, Prs, V) \xrightarrow{t} Pr^{V,A}$ denote an interpretation mapping for formulas, $J_{Tr} : Tr(Fs, Ps, Prs, V) \xrightarrow{t} Fn^{V,A}$ denote an interpretation mapping for terms and $J_{Pt} : Pt(Fs, Ps, Prs, V) \xrightarrow{t} Prg^{V,A}$ denote an interpretation mapping for programs (statements). They are all defined in a natural way, only the case with assertions needs special consideration:

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

Thus, an interpretation *J* is defined by some algebra *QPA*(*V*, *A*) and interpretation mappings I_{Ps}, I_{Fs}, and I_{Prs}. An assertion is said to be *valid* (*irrefutable*) *in an inter-*

pretation J (denoted $J \models_{IR} \{p\}f\{q\}$ or simply $\models \{p\}f\{q\}$) if a predicate obtained under interpretation J is *not refutable*. An assertion is said to be *valid* (denoted $\models \{p\}f\{q\}$) if for any interpretation J we have $J \models \{p\}f\{q\}$. In this chapter we do not define interpretations explicitly expecting that they are clear from the context. Thus, in the text the main reasoning steps are described for program algebras though to be precise we had to define an interpretation first and then consider predicates of the corresponding program algebra.

3.6 Monotonicity and Continuity of the Floyd-Hoare Composition

In the previous subsections a function-theoretic style of composition definitions was used. To prove properties of the FH-composition, it is more convenient to use a set-theoretic style of definitions.

The following sets are called respectively *truth*, *falsity*, and *undefinedness domains* of the predicate p over D:

$$p^T = \{d \mid p(d) \downarrow = T\},$$
$$p^F = \{d \mid p(d) \downarrow = F\},$$
$$p^\perp = \{d \mid p(d) \uparrow\}.$$

The following definitions introduce various images and preimages involved in Floyd-Hoare composition:

$$q^{-T,f} = f^{-1}[q^T],$$
$$q^{-F,f} = f^{-1}[q^F],$$
$$q^{-\perp,f} = f^{-1}[q^\perp],$$
$$p^{T,f} = f[p^T],$$
$$p^{F,f} = f[p^F],$$
$$p^{\perp,f} = f[p^\perp].$$

Using these notations we can define FH-composition by describing the truth and falsity domains of the predicate that is the result of the composition application:

$$FH(p,f,q)^T = p^F \cup q^{-T,f},$$
$$FH(p,f,q)^F = p^T \cap q^{-F,f}.$$

Validity of formulas (predicates) is considered as irrefutability, that is

$$\models p \Leftrightarrow p^F = \varnothing.$$

From this follows that

$$\models FH(p,f,q) \Leftrightarrow p^T \cap q^{-F,f} = \varnothing.$$

Let us give a formal definition of a monotone composition.

Composition $C : (FPrg^{V,A})^n \times (Pr^{V,A})^k \times (Fn^{V,A})^m \xrightarrow{t} Pr^{V,A}$ (with the class of predicates as its range) is called *monotone* if the following condition holds for all arguments of C:

$$f_1 \subseteq g_1, \ldots, f_n \subseteq g_n, p_1 \subseteq q_1, \ldots, p_k \subseteq q_k, h_1 \subseteq s_1, \ldots, h_m \subseteq s_m \Rightarrow$$
$$C(f_1, \ldots, f_n, p_1, \ldots, p_k, h_1, \ldots, h_m) \subseteq C(g_1, \ldots, g_n, q_1, \ldots, q_k, s_1, \ldots, s_m).$$

Here relation of partial order $\subseteq$ is defined as inclusion of graphs of the arguments (which are mappings) of this relation.

Theorem 1. Floyd-Hoare composition is monotone for every argument.

Let us prove monotonicity for every argument separately, examining their truth and falsity domains. For the first argument (precondition) we have:

$$p_1 \subseteq p_2 \Rightarrow\ p_1^{\,F} \subseteq p_2^{\,F} \Rightarrow\ p_1^{\,F} \cup q^{-T,f} \subseteq p_2^{\,F} \cup q^{-T,f} \Rightarrow$$
$$FH(p_1, f, q)^T \subseteq FH(p_2, f, q)^T$$

and

$$p_1 \subseteq p_2 \Rightarrow\ p_1^{\,T} \subseteq p_2^{\,T} \Rightarrow\ p_1^{\,T} \cap q^{-F,f} \subseteq p_2^{\,T} \cap q^{-F,f} \Rightarrow$$
$$FH(p_1, f, q)^F \subseteq FH(p_2, f, q)^F .$$

Thus, $p_1 \subseteq p_2 \ \Rightarrow\ FH(p_1, f, q) \subseteq FH(p_2, f, q)$.

For the third argument (postcondition) the proof is similar:

$$q_1 \subseteq q_2 \ \Rightarrow\ q_1^{\,T} \subseteq q_2^{\,T} \ \Rightarrow\ q_1^{-T,f} \subseteq q_2^{-T,f} \ \Rightarrow\ p^F \cup q_1^{-T,f} \subseteq p^F \cup q_2^{-T,f} \ \Rightarrow$$
$$FH(p, f, q_1)^T \subseteq FH(p, f, q_2)^T$$

and

$$q_1 \subseteq q_2 \ \Rightarrow\ q_1^{\,F} \subseteq q_2^{\,F} \ \Rightarrow\ q_1^{-F,f} \subseteq q_2^{-F,f} \ \Rightarrow\ p^T \cap q_1^{-F,f} \subseteq p^T \cap q_2^{-F,f} \ \Rightarrow$$
$$FH(p, f, q_1)^F \subseteq FH(p, f, q_2)^F .$$

Thus, $q_1 \subseteq q_2 \Rightarrow FH(p, f, q_1) \subseteq FH(p, f, q_2)$.

Let us show the monotonicity of FP-composition for the second argument. For the truth domains we have:

$$f_1 \subseteq f_2 \ \Rightarrow\ q^{-T,f_1} \subseteq q^{-T,f_2} \Rightarrow\ p^F \cup q^{-T,f_1} \subseteq p^F \cup q^{-T,f_2}$$
$$\Rightarrow\ FH(p, f_1, q)^T \subseteq FH(p, f_2, q)^T .$$

Similar, for the falsity domains:

$$f_1 \subseteq f_2 \ \Rightarrow\ q^{-F,f_1} \subseteq q^{-F,f_2} \ \Rightarrow\ p^T \cap q^{-F,f_1} \subseteq p^T \cap q^{-F,f_2} \ \Rightarrow$$
$$FH(p, f_1, q)^F \subseteq FH(p, f_2, q)^F .$$

Therefore, $f_1 \subseteq f_2 \Rightarrow FH(p, f_1, q) \subseteq FH(p, f_2, q)$.

Thus, it was shown that the composition is monotone for every component, what was needed to prove.

For the constructed composition even stronger result is true, it is continuous. To show this, the following definitions are made and the notion of continuity is given (see, for example, [4]).

An infinite set of indexed mappings $\{\mu_0, \mu_1, \ldots\}$ with $\mu_i \subseteq \mu_{i+1}, i \in \omega$ is called a *chain* of mappings.

The supremum of the above-mentioned set of indexed mappings is called *limit* of the chain, denoted as $\coprod_i \mu_i$.

The composition $C : (Prg^{V,A})^n \times (Pr^{V,A})^m \times (Fn^{V,A})^l \xrightarrow{\ t\ } Pr^{V,A}$ is called *continuous* on the first argument if for arbitrary chain $\{f_i \mid i \in \omega\}$ the following property holds:

$$C(\coprod_i f_i, g_2, \ldots, g_n, p_1, \ldots p_m, h_1, \ldots h_l) = \coprod_i C(f_i, g_2, \ldots, g_n, p_1, \ldots p_m, h_1, \ldots h_l).$$

Continuity on the other arguments is defined in a similar manner.

Theorem 2. Floyd-Hoare composition is continuous on every argument.

Though this result follows from the general consideration, we give here its direct proof. Let us show the continuity on the first argument. In the case of other arguments the proof will be similar.

Consider a chain of predicates $\{p_i \mid i \in \omega\}$. Since Floyd-Hoare composition is monotone, $\{FH(p_i, f, q) \mid i \in \omega\}$ will also be a chain. We need to show that $FH(\coprod_i p_i, f, q) = \coprod_i FH(p_i, f, q)$. To do this we demonstrate that $FH(\coprod_i p_i, f, q)(d)$ is defined iff $(\coprod_i FH(p_i, f, q))(d)$ is defined, and in this case $FH(\coprod_i p_i, f, q)(d) = (\coprod_i FH(p_i, f, q))(d)$ for the arbitrary data d.

There are two different possibilities – $(\coprod_i p_i)(d) \uparrow$ and $(\coprod_i p_i)(d) \downarrow$.

In the first case none of the elements of the chain is defined on d. Therefore $(\coprod_i FH(p_i, f, q))(d)$ is defined iff $q(f(d)) \downarrow = T$. But this means that $(\coprod_i FH(p_i, f, q))(d)$ is defined with the same value.

In the second case ($(\coprod_i p_i)(d) \downarrow$), an element of the chain that is also defined on this data could be found. Otherwise the limit would have been undefined on d, what is guaranteed by the inclusion relation on the elements of the chain. Thus, there exists k such that $p_k(d) \downarrow$. Therefore

$$FH(\coprod_i p_i, f, q)(d) = FH(p_k, f, q)(d) \text{ and}$$

$$FH(p_k, f, q)(d) = (\coprod_i FH(p_i, f, q))(d),$$

since for any i: $i > k, p_i(d) \downarrow = p_k(d)$ by the definition of the chain.

The following equality is obtained: $FH(\coprod_i p_i, f, q)(d) = (\coprod_i FH(p_i, f, q))(d)$. Since the data was chosen arbitrary, we get $FH(\coprod_i p_i, f, q) = \coprod_i FH(p_i, f, q)$, what was needed to prove.

The proof for the other arguments is similar. Thus, the monotone Floyd-Hoare composition is continuous on every argument.

The theorems 1 and 2 often permit to consider instead of programs with cycles their cycle-free approximations.

4 Inference System for PFHL

In this section we investigate possibility to use inference rules of Table 2 for PFHL.

4.1 Soundness of Classical Inference System for PFHL

Analysis of inference rules shows that soundness fails for the rules *R_SEQ*, *R_WH*, and *R_CONS*. This can be demonstrated with the following examples.

First, let us show that for some interpretation there can be such p, q, r, f, and g that $\models \{p\} f \{q\}, \models \{q\} g \{r\} \Rightarrow \models \{p\} f \bullet g \{r\}$ is false.

Consider $p(d)\downarrow = T$, $q(d)\uparrow$ and $r(d)\downarrow = F$ for arbitrary d and $f = g = id$. In this case $\models \{p\} f \{q\}$ and $\models \{q\} g \{r\}$ because $q^T = q^F = \varnothing$, but $\not\models \{p\} f \bullet g \{r\}$. Thus, $\models \{p\} f \{q\}, \models \{q\} g \{r\} \Rightarrow \models \{p\} f \bullet g \{r\}$ is false.

Next example concerns the rule *R_WH*. We need to show that for some r, f, p $\models \{r \wedge p\} f \{p\}$ and $\not\models \{p\} WH(r, f)\{\neg r \wedge p\}$.

In this case we need at least three different data (states) $d_1 \neq d_2 \neq d_3$. Then r, f, p are defined in the following manner:

$$r(d) = \begin{cases} T, \text{ if } d = d_1 \text{ or } d = d_2, \\ F, \text{ if } d = d_3, \\ \text{undefined in other cases.} \end{cases}$$

$$f(d) = \begin{cases} d_3, \text{ if } d = d_3, \\ d_2, \text{ if } d = d_1, \\ d_3, \text{ if } d = d_2, \\ \text{undefined in other cases.} \end{cases}$$

$$p(d) = \begin{cases} T, \text{if } d = d_1, \\ \text{undefined, if } d = d_2, \\ F, \text{if } d = d_3, \\ \text{undefined in other cases.} \end{cases}$$

It is not hard to prove that $\models \{r \wedge p\} f \{p\}$, because $(r \wedge p)(d)\downarrow = T$ only when $d = d_1$, but in this case $f(d)\downarrow = f(d_1)\downarrow = d_2$ and $p(d_2)\uparrow$. This means that there is no such d that $FH(r \wedge p, f, p)(d)\downarrow = F$ in the case of abovementioned interpretations.

Let us show that $\not\models \{p\} WH(r, f)\{\neg r \wedge p\}$. Consider the value of $FH(p, WH(r, f), \neg r \wedge p)(d_1)$.

We have that $p(d_1)\downarrow = T$, $WH(r, f)(d_1)\downarrow = d_3$, $(\neg r \wedge p)(d_3)\downarrow = F$.

Thus, $FH(p, WH(r, f), \neg r \wedge p)(d_1)\downarrow = F$. This gives $\not\models \{p\} WH(r, f)\{\neg r \wedge p\}$.

So, $\models \{r \wedge p\} f \{p\} \Rightarrow \models \{p\} WH(r, f)\{\neg r \wedge p\}$ is false.

The case with the rule *R_CONS* is similar to the previous ones. Consider $p(d)\downarrow = T$, $q'(d)\downarrow = q(d)\downarrow = F$ and $p'(d)\uparrow$ for arbitrary d. Then $\models \{p'\} id \{q'\}$, $p \to p'$ and $q' \to q$, but $\not\models \{p\} id \{q\}$.

Thus, $\{p'\} f \{q'\}, p \to p', q' \to q \Rightarrow \{p\} f \{q\}$ is false.

Given examples prove that additional constraints should be introduced in order for inference system to be sound in the case of partial predicates.

4.2 Composition of Preimage Predicate Transformer

To introduce constraints for the rules of PFHL we need new compositions. They are inspired by the weakest precondition and the strongest postcondition predicate transformers introduced by Dijkstra. But in the case of partial mappings there can be more than one definition what predicate should be considered as weakest (or strongest) therefore more adequate definitions are required. In this chapter we restrict ourselves by introducing only one composition called *composition of preimage predicate transformer* (*preimage composition*). This composition is a generalization of the weakest precondition predicate transformer and is defined in the following way:

$$PC(f,q)(d) = \begin{cases} T, \text{ if } f(d)\downarrow \text{ and } q(f(d))\downarrow = T, \\ F, \text{ if } f(d)\downarrow \text{ and } q(f(d))\downarrow = F, \\ \text{undefined in other cases.} \end{cases}$$

In set-theoretic terms this composition can be defined as follows:

$$PC(f,q)^T = q^{-T,f}, \; PC(f,q)^F = q^{-F,f}.$$

Semantically, $PC(f,q)$ can be treated as *backward predicate transformer.*

Introduction of this composition means that now we work with algebras of the form

$QPAT(V, A) = < Pr^{V,A}, Fn^{V,A}, FPrg^{V,A};$

$\vee, \neg, S_F^{\bar{v}}, S_P^{\bar{v}}, 'x, \exists x, id, AS^x, \bullet, IF, WH, FH, PC>.$

Also, the introduced composition permits to reformulate the assertion validity definition. Preliminary, we define $p \models q$ as $\models p \to q$.

Theorem 3. For any assertion $\{p\} f \{q\}$ the following equivalence holds:

$$\models \{p\} f \{q\} \Leftrightarrow p \models PC(f,q).$$

To prove this theorem we first recall that

$$\models \{p\} f \{q\} \Leftrightarrow p^T \cap q^{-F,f} = \varnothing.$$

Therefore $p \models PC(f,q) \Leftrightarrow \models p \to PC(f,q) \Leftrightarrow (p \to PC(f,q))^F = \varnothing \Leftrightarrow$

$$\Leftrightarrow p^T \cap PC(f,q))^F = \varnothing \Leftrightarrow p^T \cap q^{-F,f} = \varnothing \Leftrightarrow \models \{p\} f \{q\}.$$

4.3 Constraints for Partial Predicate Inference System

Analysis of the constraint problem demonstrates that for an inference rule different constraints can be formulated. We start with the constraints that practically are reformulations of conditions of assertion validity. Such constraints will be called *trifling constraints* because they do not give additional knowledge of assertion validity. Constraints will be formulated in terms of the preimage predicate transformer.

The examples showed that validity constraints are required for the rules *R_SEQ*, *R_WH*, and *R_CONS*. Trifling constraints are the following:

- $p \models PC(f \bullet g, r)$ for *R_SEQ*,
- $p \models PC(WH(r, f), \neg r \wedge p)$ for *R_WH*,
- $p \models PC(f, q)$ for *R_CONS*.

These constraints in a quite natural sense are necessary and sufficient. But in this form such constraints are not very useful, especially the constraint for *R_CONS* because it does not relate premises with conclusions. Therefore we formulate a more stronger constraint for this rule, which will be sufficient but not necessary.

At first we introduce two *special logical consequence relations*: *over the truth domain* $\models_T$ and *over the falsity domain* $\models_F$ in the following way:

- $p \models_T q$ iff $p_J^T \subseteq q_J^T$ for every interpretation *J*,
- $p \models_F q$ iff $q_J^F \subseteq p_J^F$ for every interpretation *J*.

In these terms a new constraint for *R_CONS* is $p \models_T p', q' \models_F q$. This gives us a PFHL inference system with constraints for **WHILE** presented in Table 7.

Table 7. PFHL **i**nference system for **WHILE** with constraints in backward form.

$\{S_P^x(p,h)\}AS^x(h)\{p\}$	*R_AS'*
$\{p\}id\{p\}$	*R_SKIP'*
$\dfrac{\{p\}f\{q\}, \{q\}g\{r\}}{\{p\}f \bullet g\{r\}}, p \models PC(f \bullet g, r)$	*R_SEQ'*
$\dfrac{\{r \wedge p\}f\{q\}, \{\neg r \wedge p\}g\{q\}}{\{p\}IF(r, f, g)\{q\}}$	*R_IF'*
$\dfrac{\{r \wedge p\}f\{p\}}{\{p\}WH(r, f)\{\neg r \wedge p\}}, p \models PC(WH(r, f), \neg r \wedge p)$	*R_WH'*
$\dfrac{\{p'\}f\{q'\}}{\{p\}f\{q\}}, p \models_T p', q' \models_F q$	*R_CONS'*

In this table a constrained rule consists of two parts: pure inference rule and rule constraint written on the right side of the pure rule.

Theorem 4. PFHL inference rules of Table 7 are sound. That means:

1. $\models \{S_P^x(p,h)\}AS^x(h)\{p\}$,

2. $\models \{p\}\, id\, \{p\}$,
3. $\models \{p\} f \{q\}, \models \{q\} g \{r\}, p \models PC(f \bullet g, r) \Rightarrow \models \{p\} f \bullet g \{r\}$,
4. $\models \{r \wedge p\} f \{q\}, \models \{\neg r \wedge p\} g \{q\} \Rightarrow \models \{p\} IF(r, f, g) \{q\}$,
5. $\models \{r \wedge p\} f \{p\},\ p \models PC(WH(r, f), \neg r \wedge p) \Rightarrow \models \{p\}\, WH(r, f) \{\neg r \wedge p\}$,
6. $\models \{p'\} f \{q'\}, p \models_T p', q' \models_F q \Rightarrow \models \{p\} f \{q\}$.

Let us prove this for each rule. Recall our assumption that such properties are proved for an implicitly given arbitrary interpretation *J*.

1. For $\models \{S_P^x(p,h)\} AS^x(h) \{p\}$ to hold it is required that

$$FH(S_P^x(p,h), AS^x(h), p)^F = (S_P^x(p,h))^T \cap p^{-F, AS^x(h)} = \varnothing .$$

Let d be any data such that $d \in (S_P^x(p,h))^T$. Then $p(d\nabla[x \mapsto h(d)]) \downarrow = T$. By definition of assignment composition it means that $d \in p^{-T, AS^x(h)}$. So, $(S_P^x(p,h))^T \cap p^{-F, AS^x(h)} = \varnothing$ and $\models \{S_P^x(p,h)\} AS^x(h) \{p\}$.

2. $\models \{p\} id \{p\}$ follows from the definition of *id*:

$$FH(p, id, p)^F = p^T \cap p^{-F, id} = p^T \cap p^F = \varnothing .$$

3. Soundness condition for rule *R_SEQ'* is obvious by theorem 3.

4. Let us prove $\models \{r \wedge p\} f \{q\}, \models \{\neg r \wedge p\} g \{q\} \Rightarrow \models \{p\} IF(r, f, g) \{q\}$.

Since $\models \{r \wedge p\} f \{q\},\ \models \{\neg r \wedge p\} g \{q\}$ we have:

$$(r \wedge p)^T \cap q^{-F, f} = \varnothing; (\neg r \wedge p)^T \cap q^{-F, g} = \varnothing .$$

We need to show that $p^T \cap q^{-F, IF(r, f, g)} = \varnothing$.

Let *d* be any data such that $p(d) \downarrow = T$ and $IF(r, f, g)(d) \downarrow$. If $r(d) \downarrow = T$ then $q(f(d)) \downarrow = T$ by the first premise; if $r(d) \downarrow = F$ then $q(\mathrm{g}(d)) \downarrow = T$ by the second premise. Therefore $d \in q^{-T, IF(r, f, g)}$ and $\models \{p\} IF(r, f, g) \{q\}$.

5. Soundness condition for rule *R_WH'* is obvious by theorem 3.

6. Let us prove $\models \{p'\} f \{q'\}, p \models_T p', q' \models_F q \Rightarrow \models \{p\} f \{q\}$.

We have $\models \{p'\} f \{q'\}$ what means $p'^T \cap q'^{-F, f} = \varnothing$.

Also we have $p \models_T p'$ and $q' \models_F q$; using definitions we get $p^T \subseteq p'^T$ and $q^F \subseteq q'^F$.

We need to show that $p^T \cap q^{-F, f} = \varnothing$.

Let *d* be any data such that $p(d) \downarrow = T$, $f(d) \downarrow$, and $q'(f(d)) \downarrow$. By the second premise $p'(d) \downarrow = T$, by the first premise $q'(f(d)) \downarrow = T$. If $q(f(d)) \downarrow$ then $q(f(d)) \downarrow = T$ by the third premise; therefore $d \notin q^{-F, f}$. If $q(f(d)) \uparrow$ then also $d \notin q^{-F, f}$. Thus, in both cases $p^T \cap q^{-F, f} = \varnothing$.

So, all rules are inspected and the theorem is proved.

Now we need to show that for total predicates properties of the classical Floyd-Hoare logic will be preserved and that defined logic will be an extension of the Floyd-

Hoare logic. This means that for total predicates a derivation of a Floyd-Hoare assertion in PFHL can be transformed to a derivation of this assertion in CFHL and vice versa: a derivation in CFHL can be presented as derivation in PFHL. This property holds because constraints of rules *R_SEQ'* and *R_WH'* will be satisfied automatically in the case of total predicates; as to *R_CONS'* its constraint can be reduced to the constraint of the rule *R_cons* of CFHL. This will be granted by Theorem 5. But before that we show that for total predicates assertion validity in PFHL ($\models$) is equivalent to validity in CFHL ($\models_{CL}$).

If we recall definitions of the classical (denoted FH_{CL}) and monotone compositions *FH* we will have:

$$FH_{CL}(p, f, q) = \begin{cases} T, & \text{if } p(d) = F \text{ or } f(d)\uparrow \text{ or } (f(d)\downarrow \text{ and } q(f(d)) = T), \\ F, & \text{if } p(d) = T,\ f(d)\downarrow, \text{ and } q(f(d)) = F. \end{cases}$$

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

By the definitions, for total predicates $FH(p, f, q)^F = FH_{CL}(p, f, q)^F$.

Thus, $FH(p, f, q)^F = \varnothing \Leftrightarrow FH_{CL}(p, f, q)^F = \varnothing$. But

$$\models \{p\} f \{q\} \Leftrightarrow \models FH(p, f, q)^F = \varnothing \text{ and}$$
$$\models_{CL} \{p\} f \{q\} \Leftrightarrow \models_{CL} FH(p, f, q)^F = \varnothing.$$

So, we obtain $\models \{p\} f \{q\} \Leftrightarrow \models_{CL} \{p\} f \{q\}$. It means that for total predicates classes of valid assertions in PFHL and CFHL are the same.

Theorem 5. For total predicates the inference rules of PFHL (Table 7) can be reduced to the inference rules of CFHL (Table 2).

To prove the theorem we should demonstrate that for total predicates the constraints of *R_SEQ'* and *R_WH'* hold. It means that

$$\models \{p\} f \{q\}, \models \{q\} g \{r\} \Rightarrow p \models PC(f \bullet g, r) \text{ and}$$
$$\models \{r \wedge p\} f \{p\} \Rightarrow p \models PC(WH(r, f), \neg r \wedge p).$$

Let us prove that $\models \{p\} f \{q\}, \models \{q\} g \{r\} \Rightarrow p \models PC(f \bullet g, r)$.

This means that

$$p^T \cap q^{-F, f} = \varnothing; q^T \cap r^{-F, g} = \varnothing \Rightarrow p^T \cap r^{-F, f \bullet g} = \varnothing.$$

Indeed, $r^{-F, f \bullet g} = f^{-1}[r^{-F, g}]$. Since $q^T \cap r^{-F, g} = \varnothing$ and q is total, we have that $r^{-F, g} \subseteq q^F$. And since $p^T \cap q^{-F, f} = \varnothing$ we obtain that $p^T \cap r^{-F, f \bullet g} = \varnothing$.

Let us prove that $\models \{r \wedge p\} f \{p\} \Rightarrow p \models PC(WH(r, f), \neg r \wedge p)$.

Using the definition of validity we have: $(r \wedge p)^T \cap p^{-F, f} = \varnothing$ and $p^T \cap PC(WH(r, f), \neg r \wedge p)^F \neq \varnothing$.

By definition of *PC*,

$$p^T \cap PC(WH(r, f), \neg r \wedge p)^F = p^T \cap (\neg r \wedge p)^{-F, WH(r, f)}.$$

Let d be any data such that $p(d)\downarrow = T$ and $WH(r, f)(d)\downarrow$. Then there exists a sequence $d_0, d_1, \ldots, d_n$ such that $d_0 = d$, $f(d_0)\downarrow = d_1$, …, $f(d_{n-1})\downarrow = d_n$, $r(d_0)\downarrow = T$, … , $r(d_{n-1})\downarrow = T$, $r(d_n)\downarrow = F$. Also, $WH(r, f)(d)\downarrow = d_n$ and $p(d)\downarrow = T$. This gives $(r \wedge p)(d_0)\downarrow = T$. Also, $f(d_0)\downarrow = d_1$, thus $p(d_1)\downarrow = T$. With $r(d_1)\downarrow = T$ and $f(d_1)\downarrow = d_2$ we obtain $p(d_2)\downarrow = T$. By induction we have $p(d_n)\downarrow = p(WH(r, f)(d))\downarrow = T$ and $r(d_n)\downarrow = r(WH(r, f)(d))\downarrow = F$. Thus, $d \in (\neg r \wedge p)^{-T, WH(r, f)}$. Therefore $p^T \cap (\neg r \wedge p)^{-F, WH(r, f)} = \varnothing$ and consequently $p \models PC(WH(r, f), \neg r \wedge p)$.

4.4 Simpler constraints for Partial Predicate Inference System

The trifling constraints introduced for rules *R_SEQ'* and *R_WH'* of PFHL in some cases can be changed to more stronger but simpler constraints. Such simpler constraints considered here stem from the following observation for properties of assertion validity for total predicates. In this case $\models \{p\} f \{q\}$ implies $p^{T, f} \subseteq q^T$, $q^F \subseteq p^{F, f}$ and, dually, $p^T \subseteq q^{-T, f}$, $q^{-F, f} \subseteq p^F$ because $\models \{p\} f \{q\}$ means that $p^{T, f} \cap q^F = \varnothing$ and predicates are total.

In terms of special consequence relations these properties can be reformulated as $p \models_T PC(f, q)$, $PC(f, q) \models_F p$.

Using these properties we can strengthen constraints for *R_SEQ'* and *R_WH'*.

Theorem 6. For PFHL the following properties hold:

1. $\models \{p\} f \{q\}, \models \{q\} g \{r\}, p \models_T PC(f, q) \Rightarrow p \models PC(f \bullet g, r)$,
2. $\models \{p\} f \{q\}, \models \{q\} g \{r\}, PC(f, q) \models_F p \Rightarrow p \models PC(f \bullet g, r)$,
3. $\models \{p\} f \{q\}, \models \{q\} g \{r\}, q \models_F PC(g, r) \Rightarrow p \models PC(f \bullet g, r)$,
4. $\models \{p\} f \{q\}, \models \{q\} g \{r\}, PC(g, r) \models_F q \Rightarrow p \models PC(f \bullet g, r)$,
5. $\models \{r \wedge p\} f \{p\}, PC(f, p) \models_T (r \wedge p) \Rightarrow p \models PC(WH(r, f), \neg r \wedge p)$,
6. $\models \{r \wedge p\} f \{p\}, (r \wedge p) \models_F PC(f, p) \Rightarrow p \models PC(WH(r, f), \neg r \wedge p)$.

To prove the first property recall, that $\models \{p\} f \{q\}$ means $p^T \cap q^{-F, f} = \varnothing$, $\models \{q\} g \{r\}$ means $q^T \cap r^{-F, g} = \varnothing$, $p \models_T PC(f, q)$ means $p^T \subseteq q^{-T, f}$, and $p \models PC(f \bullet g, r)$ means $p^T \cap r^{-F, f \bullet g} = \varnothing$. Thus, we should prove

$$p^T \cap q^{-F, f} = \varnothing,\ q^T \cap r^{-F, g} = \varnothing,\ p^T \subseteq q^{-T, f} \Rightarrow p^T \cap r^{-F, f \bullet g} = \varnothing.$$

Let d be any data such that $p(d)\downarrow = T,\ f \bullet g(d)\downarrow, r(f \bullet g(d))\downarrow$. By the first premise $q(f(d))\downarrow = T$. By the second premise $r(f \bullet g(d))\downarrow = T$. Thus, $d \notin r^{-F, f \bullet g}$. Therefore $p^T \cap r^{-F, f \bullet g} = \varnothing$.

Other properties related with *R_SEQ'* are proved in the same manner.

Consider properties related with *R_WH'*.

Property

$$\models \{r \wedge p\} f \{p\}, PC(f, p) \models_T (r \wedge p) \Rightarrow p \models PC(WH(r, f), \neg r \wedge p)$$

can be represented as

$$(r \wedge p)^T \cap p^{-F, f} = \varnothing,\ p^{-T, f} \subseteq (r \wedge p)^T,\ p^T \cap (\neg r \wedge p)^{-F, WH(r, f)} = \varnothing.$$

Let d be any data such that

$$p(d)\downarrow = T,\ WH(r, f)(d)\downarrow, (\neg r \wedge p)(WH(r, f)(d))\downarrow.$$

By the definition of the loop composition we have that there exists a sequence $d_0, d_1, \ldots d_n$ such that $d_0 = d,\ f(d_0)\downarrow = d_1, \ldots, f(d_{n-1})\downarrow = d_n$, and $r(d_1)\downarrow = T, \ldots, r(d_{n-1})\downarrow = T, r(d_n)\downarrow = F$. By induction on n taking into consideration the second premise we get that $p(d_0)\downarrow = T,\ p(d_1)\downarrow = T, \ldots, p(d_n)\downarrow = T$. That means that $(\neg r \wedge p)(d_n)\downarrow = T$. Therefore $p^T \cap (\neg r \wedge p)^{-F, WH(r, f)} = \varnothing$.

Another property related with *R_WH'* is proved in the same manner.

This theorem permits to consider

$$p \models_T PC(f, q),\ PC(f, q) \models_F p,\ q \models_F PC(g, r),\ PC(g, r) \models_F q$$

(or any their combination) as constraints for *R_SEQ'* and

$$PC(f, p) \models_T (r \wedge p),\ (r \wedge p) \models_F PC(f, p)$$

as constraints for *R_WH'*. These constraints are simpler than initial trifling constraints.

We can go further trying to identify cases in which these constraints hold automatically. In other words to find cases in which the pure part of the inference rules can be used in derivation without proving validity of constraints.

One of such cases is described by the following definitions.

Assertion $\{p\} f \{q\}$ is called *T-increasing* if $p \models_T PC(f, q)$ holds, and *F-decreasing* if $PC(f, q) \models_F p$ holds.

Theorem 7. Let assertion $\{p\} f \{q\}$ be T-increasing or F-decreasing. Then $\models \{p\} f \{q\}$.

Consider the case $p \models_T PC(f, q)$. It means that $p^T \subseteq q^{-T, f}$ therefore $p^T \cap q^{-F, f} = \varnothing$. Other cases are considered in the same manner.

Theorem 8. All pure (with constraints omitted) inference rules of PFHL except rule *R_CONS'* (Table 7) preserve the classes of T-increasing and F-decreasing assertions.

Proofs for both properties is similar, thus consider the class of T-increasing assertion.

1. For *R_AS'* the proof that $\{S_p^x(p,h)\}AS^x(h)\{p\}$ is T-increasing can be easily obtained from the proof of the corresponding item of theorem 4.

2. For *R_ID'* the proof is obvious.

3. For *R_SEQ'* we should prove

$$p \models_T PC(f,q),\ q \models_T PC(g,r) \Rightarrow\ p \models_T PC(f \bullet g, r).$$

This means $p^T \subseteq q^{-T,f}, q^T \subseteq r^{-T,g} \Rightarrow p^T \subseteq r^{-T,f \bullet g}$. The proof of this fact is trivial.

4. For *R_IF'* we need to prove

$$r \wedge p \models_T PC(f,q),\ \neg r \wedge p \models_T PC(g,q) \Rightarrow\ p \models_T PC(IF(r,f,g),q).$$

This means $(r \wedge p)^T \subseteq q^{-T,f}, (\neg r \wedge p)^T \subseteq q^{-T,g} \Rightarrow p^T \subseteq q^{-T,IF(r,f,g)}$. Let d be any data such that $p(d)\downarrow = T$ and $IF(r,f,g)(d)\downarrow$. If $r(d)\downarrow = T$ then $r(f(d)\downarrow = T$ by the first premise; if $r(d)\downarrow = F$ then $r(\mathrm{g}(d)\downarrow = T$ by the second premise. Therefore $d \in q^{-T,IF(r,f,g)}$ and $p \models_T PC(IF(r,f,g),q)$.

5. For *R_WH'* we need to prove

$$r \wedge p \models_T PC(f,p) \Rightarrow\ p \models_T PC(WH(r,f),p).$$

From this point the proof coincides with the corresponding part of the proof of theorem 5 therefore it is omitted. So, we can conclude that $p \models_T PC(WH(r,f))$.

The theorem is proved.

As to rule *R_CONS'* we can change it to rule *R_CONS''* with the following new constraint:

$$p \models_T p' \text{ and } q' \models_T q.$$

It is easy to prove that the rule *R_CONS''* with this constraint is sound, and being restricted on the class of total predicates it is reduced to the rule *R_CONS*.

Theorem 9. Rule *R_CONS''* preserves the class of T-increasing assertions.

The proof is obvious.

The proved theorems permit to write Table 8 for simple **WHILE** inference system which is valid and is an extension of the inference system given in Table 2. In the new system only rule *R_CONS''* has a constraint. Simplicity of this system is explained by the fact that rules *R_AS'* and *R_SKIP'* (being axioms) specify T-increasing assertions and the constraint of *R_CONS''* is simple sufficient constraint (though it is rather expressive being an extension of *R_cons*).

Table 8. Simple PFHL **i**nference system for **WHILE** with T-increasing assertions.

$\{S_p^x(p,h)\}AS^x(h)\{p\}$	*R_AS*
$\{p\}id\{p\}$	*R_SKIP*

$\dfrac{\{p\}f\{q\},\{q\}g\{r\}}{\{p\}f \bullet g\{r\}}$	*R_SEQ*
$\dfrac{\{r \wedge p\}f\{q\},\{\neg r \wedge p\}g\{q\}}{\{p\}IF(r,f,g)\{q\}}$	*R_IF*
$\dfrac{\{r \wedge p\}f\{p\}}{\{p\}WH(r,f)\{\neg r \wedge p\}}$,	*R_WH*
$\dfrac{\{p'\}f\{q'\}}{\{p\}f\{q\}}, p \models_T p', q' \models_T q$	*R_CONS''*

Identification and investigation of other simple inference systems should be continued. One of such cases is induced by acyclic programs.

4.5 PFHL for Acyclic Programs

If we consider acyclic programs (loop-free programs), the preimage predicate transformer composition can easily be presented via formulas of predicate logic. This simplifies constraints and reduces the problem of their validity to the validity problem of formulas of composition-nominative predicate logics. These problems were investigated in [3, 5, 7]. For the cyclic programs, their acyclic approximations can be considered. Details are not presented here.

5 Related Work

The seminal work on a logical characterization of programs by Floyd [1] and Hoare [2] was purely axiomatic, i.e., not yet backed by a formal semantics of programs. While also Dijkstra followed this tradition with his weakest precondition calculus [8], he also systematically investigated the necessary properties of his predicate transformer "wp". In particular, he explicitly required its *monotonicity* and realized (after a hint of J.C. Reynolds) the importance of its *continuity* for expressing effectively implementable calculations (by ruling out unbounded nondeterminism).

The crucial importance of monotonicity and continuity of functions for the set-theoretic modeling of programs was exhibited by Scott's and Strachey's denotational semantics where unbounded repetition is modeled as the fixed point of a continuous functional [9,10]. Similar considerations of monotonicity and continuity play a role in those approaches to program semantics that are based on the formal representation of programs as state relations (predicates), e.g. Back's and White's Refinement Calculus [11], Hoare's and He's Unifying Theory of Programming [12] and Boute's Calculational Semantics [13]. However, this work was typically performed in a context where functions and predicates were basically assumed to be total, i.e., well-defined for all kinds of arguments (apart from the result of infinite loops which is usually represented by a special "non-termination" value).

From a logical perspective, *partial predicates* [14] give rise to *three-valued logics* where the additional value may represent "unknown" or "error". Depending on the exact interpretation of this additional value, numerous variants of such logics have been developed by Łukasiewicz [15], Kleene [16], Bochvar [17] and others, see e.g. Bergmann [18] for a survey. Moisil [19] provided by the "Łukasiewicz-Moisil Algebras" an axiomatic algebraic framework for their formalization. A particular interest in these non-standard logics arose in the context of the theory of computation (McCarthy [20]), the modeling of processes (Bergstra and Ponse [21]), and in particular in the formal specification and verification of computer programs (Blikle [22], Konikowska et al [23]).

Especially in the context of the algebraic specification of abstract datatypes [24], the handling of *partial functions* (whose execution may not terminate or yield an error) plays an important role [25, 26]. Within a classical framework these may be handled by explicitly restricting the domain of a partial function by a predicate and treat the function result for arguments outside the domain as a definite (but unknown) value in the range of the function; this was also the basis of the work of one of the author's of the present chapter [27, 28].

On the other hand, one may also introduce explicit support for partial functions within the logic itself such as in the Vienna Development Method (VDM) which introduces a corresponding "logic of partial functions" [29]. Broy and Wirsing devised in the CIP project the concept of "partial algebras" [30] where each carrier may contain unacceptable values (e.g. "undefined") where special rules are given to deal with the application of functions to unacceptable elements; thus even non-strict functions may be specified that produce acceptable results for unacceptable arguments. This concept has become the basis of a lot of subsequent work [31–33] and also forms the semantic basis of the "Common Algebraic Specification Language" CASL [34].

6 Conclusions

In the chapter we have considered questions concerning extension of traditional Floyd-Hoare logic for partial pre- and postconditions. We have adopted a semantic-syntactic style of logic definition. Therefore we first have constructed and investigated special program algebras of partial predicates, functions, and programs. In such algebras program correctness assertions can be presented with the help of a special composition called Floyd-Hoare composition. We have proved that this composition is monotone and continuous. Considering the class of constructed algebras as a semantic base we then have defined an extended logic – Partial Floyd-Hoare Logic – and investigated its properties. This logic has rather complicated soundness constraints for inference rules, therefore somewhat simpler but also sufficient constraints have been proposed. The logics constructed can be used for program verification.

This chapter can be considered as a first step in developing composition-nominative program logics. The major directions of further investigation are the question of relative completeness of the system of inference rules, invariants for cycles,

and types for variables and functions. Also the authors plan to construct a prototype of a program reasoning system oriented on the constructed logics.

References

1. Floyd R.W.: Assigning meanings to programs. Proceedings of the American Mathematical Society Symposia on Applied Mathematics, vol. 19, pp. 19-31 (1967)
2. Hoare C.A.R.: An axiomatic basis for computer programming. Communications of the ACM, issue 12, pp. 576-580,583 (1969)
3. Nikitchenko M.S., Shkilniak S.S.: Mathematical logic and theory of algorithms. Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, (in Ukrainian) (2008)
4. Nielson H.R., Nielson F.: Semantics with Applications: A Formal Introduction. John Wiley & Sons Inc, 240p. (1992)
5. Nikitchenko M.S., Tymofieiev V.G.: Satisfiability in Composition-Nominative Logics. Central European Journal of Computer Science, vol. 2, issue 3, pp. 194-213 (2012).
6. Avron A., Zamansky A.: Non-Deterministic Semantics for Logical Systems. Handbook of Philosophical Logic, vol. 16, pp. 227-304 (2011).
7. Nikitchenko M., Tymofieiev V.: Satisfiability and Validity Problems in Many-sorted Composition-Nominative Pure Predicate Logics. In: V. Ermolayev et al. (eds.): ICTERI 2012, CCIS 347, pp. 89–110. Springer, Heidelberg (2013).
8. Dijkstra E.W.: A Discipline of Programming, Prentice-Hall, Englewood Cliffs, New Jersey (1976).
9. Schmidt, D.A.: Denotational Semantics – A Methodology for Language Development. Allyn and Bacon, Boston, MA (1986).
10. Scott, D., and Strachey, C.: Towards a Mathematical Semantics for Computer Languages. Proc. Symp. on Computers and Automata, Polytechnic Institute of Brooklyn; also Tech. Mon. PRG-6, Oxford U. Computing Lab., pp. 19-46 (1971).
11. Back R.-J. and von Wright J.: Refinement Calculus: A Systematic Introduction. Springer, New York (1998).
12. Hoare C.A.R. and Jifeng He. Unifying Theories of Programming. Prentice Hall, London, UK (1998).
13. Boute, R.T.: Calculational Semantics: Deriving Programming Theories from Equations by Functional Predicate Calculus. ACM Transactions on Programming Languages and Systems, 28(4):747–793 (2006).
14. Wang, H: The Calculus of Partial Predicates and Its Extension to Set Theory. Zeitschr. f. math. Logik und Grundlagen d. Math., Vol. 7., p. 283-288 (1961).
15. Łukasiewicz, J: O logice trójwartościowej (in Polish). Ruch filozoficzny **5**:170–171. English translation: On three-valued logic, in L. Borkowski (ed.), Selected works by Jan Łukasiewicz, North–Holland, Amsterdam, pp. 87–88 (1970).
16. Kleene, S.C: On Notation for Ordinal Numbers. Journal Symbolic Logic 3, 150 – 155 (1938).
17. Bochvar, D.A. On a 3-valued Logical Calculus and its Application to the Analysis of Contradictions (in Russian)," Matematiceskij sbornik, vol. 4, pp. 287-308 (1939).
18. Bergmann M.: An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems, Cambridge University Press, Cambridge, UK (2008).
19. Moisil, G.: Recherches sur les logiques nonchrysippiennes. Ann. Sci. Univ. Jassy 26, 431-436 (1940).

20. McCarthy, J.: A Basis for a Mathematical Theory of Computation, pp. 33-70 in Computer Programming and Formal Systems, edited by P. Braffort and D. Hirshberg, North-Holland, Amsterdam (1963).
21. Bergstra, J.A. and Ponse, A.: Bochvar-McCarthy Logic and Process Algebra, Notre Dame Journal of Formal Logic, Volume 39, Number 4, pp. 464-484 (1988).
22. Blikle A.: Three-Valued Predicates for Software Specification and Validation, VDM '88: VDM – The Way Ahead. Volume 328 of Lecture Notes in Computer Science, Springer, New York, pp. 243-266 (1988).
23. Konikowska B., Tarlecki A., Blikle, A.: A Three-valued Logic for Software Specification and Validation. Fundam. Inform. 14(4): 411-453 (1991)
24. Sannella, D. and Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development, Monographs in Theoretical Computer Science, Springer (2012).
25. Cheng, J. H. and Jones, C.B.: On the Usability of Logics which Handle Partial Functions. In C. Morgan and J. C. P. Woodcock, editors, 3rd Refinement Workshop, 51–69 (1991).
26. Jones, C.B. Reasoning about Partial Functions in the Formal Development of Programs. Electronic Notes in Theoretical Computer Science, 145:3–2 (2006).
27. Schreiner W.: Computer-Assisted Program Reasoning Based on a Relational Semantics of Programs. In: Pedro Quaresma and Ralph-Johan Back (eds.): Proceedings First Workshop on CTP Components for Educational Software (THedu'11), July 31 2011, Wrocław, Poland, number 79 of Electronic Proceedings in Theoretical Computer Science (EPTCS), ISSN: 2075-2180, pp. 124-142, (2012).
28. Schreiner W.: A Program Calculus Technical Report. Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, http://www.risc.uni-linz.ac.at/people/schreine/papers/ProgramCalculus2008.pdf (2008).
29. Jones, C.B and Middelburg, C.A: A Typed Logic of Partial Functions Reconstructed Classically, Acta Informatica, 31(5):399-430 (1994).
30. Broy M. and Wirsing M. Partial Abstract Data Types, Acta Informatica, 18(1):47-64 (1982).
31. Burmeister, P. A Model Theoretic Oriented Approach to Partial Algebras. Akademie-Verlag (1986).
32. Kreowski H.-J.: Partial Algebra Flows from Algebraic Specifications. 14th Int. Colloquium on Automata, Languages and Programming, Vol. 267 of Lecture Notes in Computer Science, pp. 521-530, Springer, Berlin (1987).
33. Reichel H.: Initial Computability, Algebraic Specifications, and Partial Algebras. Oxford University Press (1987).
34. Mosses Peter D.: CASL Reference Manual: The Complete Documentation of the Common Algebraic Specification Language. Volume 2960 of Lecture Notes in Computer Science, Springer, Berlin (2004).