On the Sequence Rule for the Floyd-Hoare Logic with Partial Pre- and Post-Conditions

Ievgen Ivanov, Mykola Nikitchenko

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

Abstract. Classical Floyd-Hoare logic is valid when total pre- and post-conditions are considered. In the case of partial conditions (predicates) the logic becomes invalid. This situation may be corrected by introducing additional constraints for the rules of the logic. But such constraints, especially for the sequence and while rules, are rather complicated. In this paper we propose a new simpler sequence rule formulated in an extended program algebra. The same considerations also allow to reformulate the while rule. The obtained results can be useful for software verification.

Keywords. Formal methods, software verification, Floyd-Hoare logic.

1 Introduction

The classical Floyd-Hoare logic [1–3] is a useful tool for proving partial correctness of sequential programs. It is based on properties of Floyd-Hoare triples (assersions) of the form $\{p\}f\{q\}$ which consists of pre- and post-conditions p, q (predicates) and a program f. This assersion is treated in the following way: when the program's input data d satisfies the pre-condition ($q(d)$ is true), and the program terminates on d, the program's output (the value $f(d)$) satisfies the post-condition ($q(d)$ is true).

Although in the classical Floyd-Hoare logic it is assumed that 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.

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

$\{\texttt{true}\}\ \texttt{a[1]:=0}\ \{\texttt{a[1]=0}\}$

where `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) `a`, and `a[1]=0` is a predicate stating that the 1-st element of `a` is zero. Logically, it makes sense to consider the latter predicate (the truth value of which depends on the content of `a`) as partial and assume that it is defined only when `a` has an element with

index/key 1 (e.g. it is undefined, when `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 `a[1]:=0` automatically allocates the space for the new value, if it is not allocated), but the post-condition predicate may be formally partial, because extraction from `a` is not always defined in the sense that it causes an error or other types of abnormal behavior. In particular, a situation of this kind occurs in Matlab [4] 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 [6] 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, 6, 8].

To deal with this issue in the previous works [9, 10] extensions of the classical Floyd-Hoare logic which allowed one to reason about partial correctness of sequential programs using Floyd-Hoare triples consisting of both 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 ($q(d)$ is defined and true), and the program terminates on d, and the post-condition is defined on the program's output (the value $q(f(d))$), 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).

An important fact is that the classical inference system for the Floyd-Hoare logic for the language WHILE [11] is not sound in the presence of partial pre- and post-conditions [9]. 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 is runs after f on the result of f).

This can be explained on the following simple example in Octave syntax. Suppose that `n` denotes 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,...) 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 are valid assertions (in the sense of weak Floyd-Hoare triples):

```
{n>=0} a=zeros(n,1) {a(1)==0}
```

`{a(1)==0} m=length(a) {m>0}`

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>=0} a=zeros(n,1); m=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, in [9] a new inference system for an extended Floyd-Hoare logic with partial pre-and post-conditions for a similar language was proposed. Besides a few other modifications, 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 $\neg p \vee q$ ($p \to q$) never takes a false value (i.e. it is always either true or undefined);
- PC is the *Predicate transformer composition* [10] 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).

The presence of a complicated constraint, however, makes application of this rule difficult in all but the most trivial cases.

In this paper we propose an alternative unconstrained sequence rule for Floyd-Hoare logic with partial predicates based on extended program algebra which can be useful for software verification.

2 Notation

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

- $f(d)\downarrow$ denotes that f is defined on $d \in D$;
- $f(d)\downarrow = d'$ denotes that f is defined on $d \in D$ with a value $d' \in D'$;
- $f(d)\uparrow$ denotes that f is undefined on $d \in D$;
- $\mathrm{dom}(f) = \{d \in D \mid f(d)\downarrow\}$ 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 $f_1(d_1) = f_2(d_2)$.

3 Constructing New Sequence Rule

Let D be a non-empty set. We will consider its elements mathematical models of computer data (which is used as input or output for programs of interest). An example of such models can be *nominative sets* [12, 13] (which are partial assignments of values to variable names) or *nominative data* [12–15] (which can be thought of as hierarchically constructed, nested nominative sets). The latter ones can be used to represent many forms of data commonly used in programming (e.g. multidimensional arrays, lists, trees, etc.) [12].

We will consider partial functions from D to D (denoted as $D\tilde{\to}D$) as semantic models of sequential programs operating on such data.

Let $f, g : D\tilde{\to}D$. Denote by $f \bullet g$ a function $D\tilde{\to}D$ such that

- $(f \bullet g)(d) = g(f(d))$, if $g(f(d))$ is defined;
- $(f \bullet g)(d)$ is undefined, otherwise.

We will call $f \bullet g$ the *sequential composition* of f and g.

Denote by T, F the logical values *true* and *false*.

We call partial functions from an arbitrary set X to $Bool = \{T, F\}$ *partial predicates* on X.

Let $p, q : D\tilde{\to}Bool$.

Denote by $p \vee q$ the partial predicate $D\tilde{\to}Bool$ such that

- $(p \vee q)(d) = T$, if $p(d) \downarrow= T$ (i.e. $p(d)$ is defined and has the value T), or if $q(d) \downarrow= T$;
- $(p \vee q)(d) = F$, if $p(d) \downarrow= F$ and $q(d) \downarrow= F$;
- $(p \vee q)(d)$ is undefined, otherwise.

Denote by $p \wedge q$ the partial predicate $D\tilde{\to}Bool$ such that

- $(p \wedge q)(d) = T$, if $p(d) \downarrow= T$ and $q(d) \downarrow= T$;
- $(p \wedge q)(d) = F$, if $p(d) \downarrow= F$ or $q(d) \downarrow= F$;
- $(p \wedge q)(d)$ is undefined, otherwise.

Denote by $\neg p$ the partial predicate $D\tilde{\to}Bool$ such that

- $(\neg p)(d) = T$, if $p(d) \downarrow= F$;
- $(\neg p)(d) = F$, if $p(d) \downarrow= T$;
- $(\neg p)(d)$ is undefined, if $p(d)$ is undefined.

Operations (compositions) $\vee, \wedge, and \neg$ are respectively disjunction, conjunction, and negation operations on partial predicates. These operations are defined according to the truth tables of Kleene's strong logic of indeterminacy [16]. The set of all partial predicates on D together with these operations forms a Kleene algebra (a De Morgan algebra which satisfies the normality axiom) [17].

Denote by $\sim p$ the partial predicate $D\tilde{\to}Bool$ such that

- $(\sim p)(d) = T$, if $p(d)$ is undefined;
- $(\sim p)(d)$ is undefined, if $p(d)$ is defined.

It means that that we extend a program algebras, and in particular predicate algebra, with new operation $\sim$.

Definition 1. *A semantic weak Floyd-Hoare triple is a tuple* (p, f, q)*, where* $f : D \tilde{\to} D'$*,* $p : D \tilde{\to} Bool$*,* $q : D' \tilde{\to} Bool$ *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.

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 $(r_1 \vee r_2)(d'') = T$. □

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 [9, 10] (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 potentially applied in the case mentioned in the introduction:

```
{n>=0} a=zeros(n,1) {a(1)==0}
```

{a(1)==0} m=length(a) {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 `m=length(a)` when the predicate `(a(1)==0)` is undefined. Under the interpretation assumed in Introduction, this undefinedness means that the attempt of evaluation of `(a(1)==0)` leads to an abnormal/error state. If `a` is a defined vector, this happens exactly when `a` has zero length (is empty). If `a` is undefined, then an attempt of evaluation of `length(a)` causes an error. Thus we can state that

{∼(a(1)==0)} m=length(a) {m=0}.

where ∼ is not a part of Octave syntax, but just a notation to represent the statement that the expression `(a(1)==0)` is undefined (causing an abnormal/error state). Thus by the *R_USEQ* rule:

{n>=0} a=zeros(n,1); m=length(a) {m>0 ∨ m=0}.

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

4 Application to the While Loop Analysis

Let r be a partial predicate on D and $f : D \tilde{\to} D$.

Denote by $WH(r, f)$ the function $D \tilde{\to} D$ 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 $i = 0, 1, ..., n-1$ and $r(f^{(n)}(d)) \downarrow = F$, where $f^{(i)}$ denotes $\underbrace{f \bullet f \bullet ... \bullet f}_{i}$ 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.

In [12] WH, considered as an operation on r and f, is called the *while composition*. It is intended to capture the semantics of the loop of the form

while r **do** f **end**

in imperative programming languages which support structured programming. Here 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 [11] can be reformulated as follows [18, 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 [18, 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 may be 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 $p(d) \downarrow= T$, $WH(r,f)(d) \downarrow= d'$, 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 $i = 0, 1, ..., n-1$, 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 $i \in \{0, 1, ...\}$ that if $i \in \{0, 1, ..., n-1\}$, 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 $i \in \{0, 1, ..., n-1\}$. Assume that $p(f^{(i)}(d)) \downarrow= T$ or $p(f^{(i)}(d)) \uparrow$. Assume that $i+1 \in \{0, 1, ..., n-1\}$. 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 $r(d_1) \downarrow= T$. Then $(\sim p)(d_1) \downarrow= T$, so $(r \wedge (\sim p))(d_1) \downarrow= T$. Then 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 $r(f^{(n-1)}(d)) \downarrow= T$, so $(r \wedge p)(f^{(n-1)}(d)) \downarrow= T$. 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 $(\neg r \wedge p)(WH(r,f)(d)) \cong (\neg r \wedge p)(f^{(n)}(d)) \downarrow= T$. □

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

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

5 Conclusion

We have proposed modifications of the inference system for an extension of the classical Floyd-Hoare logic to the case of partial pre- and post-conditions and partial programs proposed in [9, 10]. These modifications concern sequence and while rules and have been formulated in the extended program algebras. The addition of these rules does not change the soundness of the system. However, the new rules have no semantic constraints, unlike the sequence rule of the original system. The obtained results can be useful for verification of program 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. Floyd, R.: Assigning meanings to programs. Mathematical aspects of computer science **19**(19-32) (1967)
2. Hoare, C.: An axiomatic basis for computer programming. Commun. ACM **12**(10) (1969) 576–580
3. Apt, K.: Ten years of Hoare's logic: A survey – part I. ACM Trans. Program. Lang. Syst. **3**(4) (1981) 431–483
4. : MATLAB – MathWorks. `https://www.mathworks.com/products/matlab.html`
5. : GNU Octave. `https://www.gnu.org/software/octave/`
6. Jones, C.: Reasoning about partial functions in the formal development of programs. In: Proceedings of AVoCS'05. Volume 145., Elsevier, Electronic Notes in Theoretical Computer Science (2006) 3–25
7. Hähnle, R.: Many-valued logic, partiality, and abstraction in formal specification languages. Logic Journal of the IGPL **13**(4) (2005) 415–433
8. Gries, D., Schneider, F.: Avoiding the undefined by underspecification. Technical report, Ithaca, NY, USA (1995)
9. Nikitchenko, M., Kryvolap, A.: Properties of inference systems for Floyd-Hoare logic with partial predicates. Acta Electrotechnica et Informatica **13**(4) (2013) 70–78
10. Kryvolap, A., Nikitchenko, M., Schreiner, W.: Extending Floyd-Hoare logic for partial pre- and postconditions. In Ermolayev, V., Mayr, H., Nikitchenko, M., Spivakovsky, A., Zholtkevych, G., eds.: Information and Communication Technologies in Education, Research, and Industrial Applications. Volume 412 of Communications in Computer and Information Science. Springer International Publishing (2013) 355–378
11. Nielson, H., Nielson, F.: Semantics with applications – a formal introduction. Wiley professional computing. Wiley (1992)

12. 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.: Information and Communication Technologies in Education, Research, and Industrial Applications. Volume 469 of Communications in Computer and Information Science. Springer International Publishing (2014) 117–138
13. Nikitchenko, M., Shkilniak, S.: Mathematical logic and theory of algorithms. Publishing house of Taras Shevchenko National University of Kyiv, Ukraine (in Ukrainian) (2008)
14. 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, September 3-6, 2017. (2017) 237–244
15. Korniłowicz, A., Kryvolap, A., Nikitchenko, M., Ivanov, I. In: Formalization of the Nominative Algorithmic Algebra in Mizar. Springer International Publishing, Cham (2018) 176–186
16. Kleene, S.: Introduction to metamathematics. North-Holland Publishing Co., Amsterdam, and P. Noordhoff, Groningen (1952)
17. Korniłowicz, A., Ivanov, I., Nikitchenko, M.: Kleene algebra of partial predicates. Accepted for publication in Formalized Mathematics (2018)
18. Korniłowicz, 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, May 15-18, 2017. (2017) 504–523