Informatics and Control Problems 39 Issue 2 (2019)
journal homepage: www.icp.az

# Implementation of three-state logic using the specialized SmartBool structure in the C# programming language

**E.R. Aliyev, D.B. Gakh*, Kh.Kh. Abdullayev**

*Institute of Control Systems of ANAS, Baku, Azerbaijan*

**ARTICLE INFO**

*Article history:*
Received 17.04.2019
Received in revised form 19.06.2019
Accepted 20.09.2019
Available online 30.12.2019

*Keywords:*
Ternary logic
C# programming language

**ABSTRACT**

*The standard logical type in the C# programming language has only two states — "true" and "false". However, this is not enough in some situations. The authors describe the development and practical experience of using a logical type with the three states "true", "false" and "unknown" — the so-called ternary logic.*

## 1. Introduction

Typically, programming languages implement Boolean logic with two states, "true" and "false". For situations requiring the introduction of a third state, one or another programming technique is used. Thus, modern versions of the C# programming language allow defining structures that, in addition to their values, can take the null value. These are so-called "nullable types" [1].

It should be noted that "nullable types" are not intended for implementing ternary logic, and therefore its implementation by this property of the language has certain drawbacks. As part of the development of the www.GoMap.Az web portal [2], a specialized SmartBool data structure was created that successfully implements the tasks without the drawbacks inherent in the methods for implementing ternary logic using "nullable types".

This paper discusses the experience of creating and operating a SmartBool data structure that implements ternary logic.

## 2. The need for ternary logic

The presence of Boolean logic with two states, “true” and “false”, in the programming languages is dictated by the capabilities of the hardware operating in the “0” and “1” binary system. In view of this, ternary logic is not a standard language tool, but it can be implemented in one way or another. It should be noted that such characteristics of code as performance, compactness and ease of programming will depend on the way of implementing ternary logic.

The need to implement a specialized ternary logic structure within the framework of the

*Corresponding author.
*E-mail addresses*: elchin.aliyev@sinam.net (E.R. Aliyev), dmgakh@gmail.com (D.B. Gakh), khanmurad.abdullayev@sinam.net (Kh.Kh. Abdullayev)

www.icp.az/2019/2-12.pdf

www.GoMap.Az project arose due to the need to configure data processing tools and data processing logic itself. Since the www.GoMap.Az project processes a large amount of data from a geographic information system, including data from the territories of Azerbaijan, Georgia and Turkey (according to SINAM as of March 3, 2019), the experience of implementing such volumes of data can be useful for other areas of human activity.

Database management systems have a value indicating the absence of data in the cell. In SQL, this value is referred to as "NULL" and can be interpreted as a missing, inaccessible, or unknown value. A similar situation occurs when users configure data processing tools through text configuration files. If there is no value for the logical key, the read function should signal this absence. The function that returns the value of ternary logic can organically signal this lack of value.

Ternary logic can also be used when comparing numerical values, implying the absence of a value. An example of such values can be considered as floating-point numbers capable of taking the "NAN" values ("Not a Number") [3, 4]. When comparing such values, the application of ternary logic is more informative than the application of binary logic. Thus, comparing any value with a NAN value yielding an "unknown" value is more consistent with the semantics of the analysis than more specific "true" and "false" values corresponding to binary logic.

## 3. Nullable Boolean type in C#

As noted, the nullable type based on the bool type (hereinafter referred to as the "Nullable Boolean") can be used to implement ternary logic in C#. Semantically, the null value of a Nullable Boolean type variable is completely equivalent to the NULL value in database management systems and in the SQL language. At the same time, it should be noted that the Nullable Boolean type is not designed to support ternary logic, as it is implemented on the basis of a "generic", which aims to provide any "value type" with the possibility of assigning a null value [1, 5]. Thus, the Nullable Boolean type can be adapted to implement ternary logic using the mentioned property of the C# programming language.

This approach has the following side effects:

- The "null" value has a different semantics than the "true" and "false" values;
- Operations with Nullable Boolean type variables cause an exception state when the value is "null". However, operations with the "true" and "false" values do not cause an exception state;
- Nullable Boolean variables in some operations require explicit conversion to the "bool" type;
- Nullable Boolean variables take up additional memory space for storing the "hasValue" status flag [1].

Examples of such side effects are given in the following code snippets:

```
bool? bn = null;
bool b2 = (bool)bn;
```

This code snippet shows the following side effects:

- Assigning the value of the "bn" variable to the "b2" variable causes an exception state;•
- Assigning the value of the "bn" variable to the "b2" variable requires an explicit conversion to the bool type;
- It should be noted that this effect cannot be considered a drawback, since "Nullable Boolean" is more informative than "bool";
- The "bn" variable is assigned the "null" value but not "unknown" or "undefined", which violates the semantics of the code.

```
bool? bn = null;

if ((bool)bn)
{
   Console.WriteLine("True");
}
else
{
   Console.WriteLine("False or Unknown");
}
```

This code snippet shows the following side effects:

- The if-statement causes an exception state;
- The if-statement requires explicit conversion to the bool type;
- The "False or Unknown" string will never be displayed if bn has the null value;
- The "bn" variable is assigned the "null" value but not "unknown" or "undefined", which violates the semantics of the code.

### 3.1. Nullable Boolean type extension

A Nullable Boolean type extension can improve the usability of the source code and improve its performance [6]. However, this does not solve all the problems of adapting the Nullable Boolean type to implement ternary logic.

## 4. The “SmartBool” structure in C#

The "SmartBool" structure is presented in the "SSTypeLT" library [7]. This structure allows implementing ternary logic in the most optimal and convenient way. The “SmartInt” structure, which is a 32-bit integer value, was developed in a similar way and has been successfully used in the www.GoMap.Az web portal for several years [2, 8].

The SmartBool structure is based on the "sbyte" ("signed byte") type. This approach made it possible to implement the "!" negation operator without the use of branching statements such as "if" reducing the predictability of the CPU pipeline, which adversely affects performance. Unfortunately, almost all operations of ternary logic require the use of branching statements and cannot be realized by a simple operation on binary hardware. The SmartBool structure was developed taking into account the minimum memory consumption and the maximum speed in the if-statements. The performance of the SmartBool and Nullable Boolean structures was measured and evaluated (see the Performance section below).

The behavior of the code using the SmartBool and Nullable Boolean structures is different. The following code snippet using the SmartBool structure logically matches the previous code snippet using the Nullable Boolean structure but has several advantages.

```
SmartBool sb = SmartBool.Unknown;
bool sb2 = (bool)sb;

if (sb)
{
   Console.WriteLine("True");
}
else
```

```
{
   Console.WriteLine("False or Unknown");
}
```

The advantages of this code compared to the previous snippet are as follows:

- The "sb" variable takes the SmartBool.Unknown value and not "null", which is semantically more correct;
- Assigning the value of the "sb" variable to the "sb2" variable does not cause an exception state;
- The if-statement works semantically correctly and does not cause an exception state.

## 5. True and false operators

The type for which the "true" operator is defined can be the type used as a condition in the conditional "if", "do", "while", "for" and "?:" statements [9].

The "true" operator returns a Boolean value of "true" indicating that the operand actually contains "true" value. The "false" operator returns a boolean value of "true" indicating that the operand actually contains "false". There is no guarantee that the "true" and "false" operators complement each other. Thus, for the same value, the "true" and "false" operators can return the Boolean value of "false" [9].The SmartBool structure implements the "true" and "false" operators as described. The following sample code will work as indicated in the comments.

```
if (sb1)
{
   // Executes if sb1 is True
   Console.WriteLine("True");
}
else
{
   // Executes if sb1 is False or Unknown
   Console.WriteLine("False or Unknown");
}
if (!sb2)
{
   // Executes if sb2 is False
   Console.WriteLine("False");
}
else
{
   // Executes if sb2 is True or Unknown
   Console.WriteLine("True or Unknown");
}
```

## 6. Truth tables for the SmartBool structure

The SmartBool structure supports standard truth tables for ternary logic. The following are truth tables for basic logical operations:

- The unary NOT operator (Fig. 1);
- The binary OR operator (Fig. 2);
- The binary AND operator (Fig. 3);

- Comparison operator (Fig. 4);
- Implication operator (Fig. 5);
- Exclusive OR operator (Fig. 6).

| A | !A |
|---|---|
| True | False |
| Unknown | Unknown |
| False | True |

**Fig. 1.** NOT operator (!)

| | True | Unknown | False |
|---|---|---|---|
| **True** | True | True | True |
| **Unknown** | True | Unknown | Unknown |
| **False** | True | Unknown | False |

**Fig. 2.** OR operator (|, ||)

| | True | Unknown | False |
|---|---|---|---|
| **True** | True | Unknown | False |
| **Unknown** | Unknown | Unknown | False |
| **False** | False | False | False |

**Fig. 3.** AND operator (&, &&)

| | True | Unknown | False |
|---|---|---|---|
| **True** | True | Unknown | False |
| **Unknown** | Unknown | Unknown | Unknown |
| **False** | False | Unknown | True |

**Fig. 4.** Comparison operator (==)

| | True | Unknown | False |
|---|---|---|---|
| **True** | True | Unknown | False |
| **Unknown** | True | Unknown | Unknown |
| **False** | True | True | True |

**Fig. 5.** Implication operator

| | True | Unknown | False |
|---|---|---|---|
| **True** | False | Unknown | True |
| **Unknown** | Unknown | Unknown | Unknown |
| **False** | True | Unknown | False |

**Fig. 6.** Exclusive OR operator (XOR /^)

## 7. Testing and quality assurance

As part of quality assurance, two types of testing were implemented: performance testing and unit testing. The source code of the corresponding tests was published as part of the SSTypes project. [7].

### 7.1. Performance

Performance tests show that the execution speed of the "!", "true" and "false" operations by the SmartBool structure is higher than by Nullable Boolean. However, the execution speed of the “|” and “&” operations by the SmartBool structure is slightly lower than by Nullable Boolean.

Comparison operators of the SmartBool structure do not show good performance compared to Nullable Boolean. This is due to the more complex logic of comparing SmartBool type values and to the fact that comparison operators also return a SmartBool type value and not a bool type value value unlike for Nullable Boolean. Thus, a decrease in performance is associated with an increase in functionality. To improve performance, instead of comparison operators, one can consider using the SmartBool.isTrue(), SmartBool.isFalse(), SmartBool.isUnknown() functions.

The following diagram (Fig. 7) shows the results of performance measurements of basic operations for three logical types. Values are relative and provided for comparison purposes only. Test results for the "true" and "false" operators are multiplied by 1000 [10]. Testing was performed using the BenchmarkDotNet library [11].

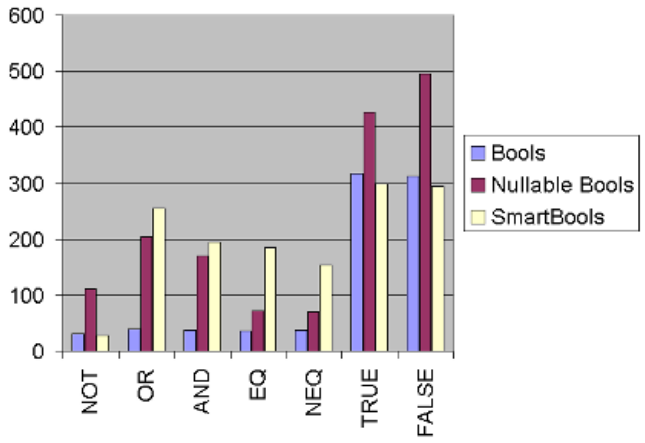


**Fig. 7.** Execution time for basic operations (μs)

### 7.2. Unit tests

Software testing is usually carried out at different levels. There are four levels of testing — unit testing, integration testing, system testing and acceptance testing. At the unit testing level, individual units are tested by functional and/or structural testing techniques. The purpose of unit testing is to ensure that the unit meets the functional specifications and/or its implementation structure matches the intended design [12].

The SmartBool structure is a unit. Therefore, in this case, it makes sense to implement only unit testing. For testing automation, the built-in Microsoft Visual Studio 2015 Community Edition tools were used. Test codes were implemented using elements from the Microsoft.VisualStudio.TestTools.UnitTesting namespace. This namespace is used by default by the built-in testing tools of Visual Studio [13].

Unit testing has confirmed that the SmartBool structure provides the necessary functionality. The necessary implementation structure is confirmed indirectly by the fact that the SmartBool structure is implemented similarly to the other two long and widely used structures of the SSTypeLT library: SmartInt and SmartDouble [7, 8].

## 8. Conclusions

The implementation of the SmartBool structure made it possible to implement operations using ternary logic with the most compact program code, i.e. fewer lines. Moreover, in comparison with other approaches, the correct code semantics is implemented. These advantages have simplified the writing of the code, its understanding in subsequent viewing, and as a result have led to minimizing the human factor and reducing the likelihood of errors made by developers.

Equally important is the solution to the problem of implicit exception states during code execution as it can happen when using the Nullable Boolean type. Using the SmartBool structure does not lead to such behavior, which significantly increases the stability of the program.

The most significant result was achieved in the www.GoMap.Az project [2] during the processing of geographic data and the preparation of electronic maps. The SmartBool structure made it possible to use a minimum number of lines to create a code that took into account three data states — "valid data", "indefinite data" and "invalid data". The implementation of the SmartBool structure allowed conducting further research on the implementation of operations with real and integer types (double and int, respectively), taking into account NaN values, in which ternary logic will be applied. Thus, in the case of applying ternary logic, comparison operations for real numbers will result in three values — "true", "false" and "uncertain", which will expand the scope of this data type in comparison with standard comparison operations where only two resulting values are implemented — "true" and

"false."

The structure is included in the SSTypeLT library and can be used along with other types included in this library [7]. The SSTypeLT library is an open source project [7]. This gives the possibility of using the types implemented in it in many projects around the world. In addition, this project can be used in educational institutions as an example and as a multicomponent exercise package [14].

To raise the awareness in the professional community, a description of the features of the SmartBool structure is published on the CodeProject.com website [10]. This opens up the possibility of feedback and discussions on the use of the SmartBool structure by community developers.

## References

[1] struct Nullable<T>, Microsoft Reference Source, https://referencesource.microsoft.com/#mscorlib/system/nullable.cs.
[2] http://www.gomap.az project
[3] M. Cowlishaw, "Decimal Arithmetic Encodings", IBM UK Laboratories, Version 1.01, 2009
[4] W. Kahan, "Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic", Elect. Eng. & Computer Science, University of California, Version of October 1, 1997 3:36 am, 1997
[5] Value types (C# Reference), https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/value-types.
[6] Ternary logic, https://rosettacode.org/wiki/Ternary_logic.
[7] SSTypes project. https://github.com/dgakh/SSTypes.
[8] D. Gakh, Accelerated .NET Types, https://www.codeproject.com/Articles/1088174/Accelerated-NET-Types.
[9] True and false operators (C# Reference), https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/true-false-operators.
[10] D. Gakh, SSTypesLT Library. Implementation of Ternary Logical Type SmartBool, https://www.codeproject.com/Articles/3135604/SSTypesLT-Library-Implementation-of-Ternary-Logica.
[11] BenchmarkDotNet, https://github.com/dotnet/BenchmarkDotNet.
[12] Y. Singh, Software Testing, Cambridge University Press, 2012.
[13] N. Kumar Satheesh, S. Subashni, Software Testing using Visual Studio 2012, Packt Publishing, 2013.
[14] D. Gakh, Open source software to support the education processes in software development in post-soviet countries, Proceedings of the eleventh international scientific-practical Conference Internet-Education-Science-2018, VNTU, 2018, ISBN 978-966-641-728-5, p.241-243

Reviewer: Associate Professor R.G. Alekberov, PhD (Engineering)