Alternary Operators: Alternative Ternary Operators

Jim McCann[*]
TCHOW llc

Your Name Here

Your Name Here, Evan

A ? B : C

Figure 1: *The ternary traditional operator ("tradnary operator") evaluates to either B (if A evaluates to true) or C (otherwise).*

Abstract

Wizened old C++ programmers often refer reverently to "the" ternary operator – a wonderful language construct that allows in-line decision-making to compactly expressed.

But is the traditional ternary operator really *the* ternary operator, or are there other, as-yet-undiscovered, ternary operators that work similarly? In this paper, we axiomatize the notion of a ternary operators and explore the space of operators consistent with these axioms, demonstrating that there is, indeed, no one true ternary operator.

CR Categories: ___ [Blank]: Blankings—Blank

1 Introduction

The traditional ternary operator ("tradnary operator") is used in C-like languages [ISO 2017][†] to express the notion of in-line decision making:

```
A ? B : C
```

Where B and C must have compatible types. At runtime, first A is evaluated, and then either B or C are evaluated based on the truth-value of A (particularly, if A is true, then B is evaluated; otherwise C is evaluated). See also Figure 1.

This notion of fickle evaluation gives the tradnary operator a lot of power, on both sides of the equals sign.

For instance, one can use it to implement circular indexing:

```
next = (i < size ? arr[i] : arr[0]);
```

Provide a default value for missing data:

```
val = (pts[i] ? *pts[i] : 0.0f);
```

Pick a variable to increment:

```
(x > 0 ? pos_sum : neg_sum) += x;
```

Pick a vector to append to:

```
(x > 0 ? pos : neg).push(x);
```

Or even select a function to call:

```
(op == '+' ? add : sub)(a,b);
```

[*]e-mail: ix@tchow.com

[†]Though the word "ternary" doesn't actually appear in the specification. ISO/IEC calls it the conditional operator.

2 Which Operators are Ternary Operators?

What is it that makes the tradnary operator a member of the fraternity of ternary operators ("fraternary")? We believe that it must hew to several key ternary properties ("terperties"). These properties are largely non-surprising, and so are stated with minimal justification.

2.1 List of Fraternary Terperties

Aritude. To be ternary, an operator must have three operands. So while it might be tempting to define an operator of the form `A ? B : C : D`, this would not fit our definition. As they say in the culture: "to be a cool dude, you gotta have the right aritude."

Infixulation. Like all C operators, ternary operators should be written in infix notation using characters that are disallowed in variable names. Ideally, the punctuation should be such that a lexer can tokenize the operator without resorting to recursion or complicated pattern matching. This may seem like a limitation in the creation of additional operators, but – even though the current specification seems to reccommend it – in practice, most compilers[‡] reject non-ASCII-letters in variable names, leaving whole swaths of codepoints available for productive use, including !, ¿, and U+203D.

Indeed, operator overloading also allows the re-use of the same punctuation as the tradnary operator; though this concept does not persist throughout all the languages that support the operator, so alternate punctuation may be preferred.

Fickleness. Of course, what makes any ternary truly interesting is its mercurial nature. The fact that the operator evaluates only some of its operands, in a way that cannot – in general – be determined at compile time, is integral to its nature.

Uniquivity. Finally, it should be the case that any ternary operator would be cumbersome to duplicate with other language syntax. Impossibility is too much to ask, since – as we

[‡]As in, both of the ones we tried.

show later – even the tradnary operator can be replaced with a templated functor.

3 Alternative Ternary Operators

With these properties in hand, we can finally consider what alternative ternary operators ("alternary operators") may exist that are Fickle, Infixulated, Uniqued, and have the right Aritude. Below, we review some possible operators and include implementation notes.

3.1 The Race Condition

`A # B` 🏁 `C`

This ternary operator evaluates `A` and `B` in parallel, and takes the value of whichever one finishes fastest. The expression `C` is evaluated only in cases where `A` completes before `B`.

3.2 The Reverse Tradnary

`A : B ? C`

If evaluating `C` after `A` would produce a true value, then evaluate `A`, otherwise evaluate `B`.

NOTE: a general implementation likely requires transactional memory and/or use of observable speculative execution (e.g. the spectre "bug"); while limiting the operator to compile-time evaluation only would, of course, violate *fickleness*. A practical middle-ground would be to require `A` to be an expression for which the compiler can statically determine all referenced variables.

3.3 The Iternary

`A @ B : C`

If `B` is true, evaluate `A` while `B` remains true. The value of the expression is the last value of `A`. Otherwise, the value of the expression is the value of `C`.

3.4 The Internary

`A` 🙃 `B : C`

The Internary is a cheaper Tradnary operator that only functions for a period of up to six months, negotiated at compile time and subject the to the availability of the Internary. Recompiling the code requires the Internary's contract to be re-negotiated. Internarys may return as a Tradnary operator after completing their Tradnary schooling.

3.5 The Caternary

`A ⊎ B : C`

The result of `A` is used as a hash to select containers of different sizes and materials, labeled `B` and `C`. A cat is placed in a room with containers `B` and `C`. When a cat sits in a container, the corresponding expression is evaluated. The initial cat may choose to bring other cats into the room ("concatination"), in which case expressions `B` and `C` may be evaluated multiple times at non-deterministic intervals. This operator may deadlock due to lack of interest from the selected cats.

3.6 Schrödinger's Caternary

`A Ψ B : C`

This is similar to the Caternary operator except that there is only one container in a room labeled both `B` and `C` and the room is set up as a Schrödinger's Cat experiment. After a random interval determined by evaluating `A`, if the cat is found inside the box and is alive, `B` is evaluated. If the cat is found inside the box and is dead, `C` is evaluated. If the cat is found outside the box chasing a laser instead, the experiment is reset.

3.7 The Facebook^{TM} Sponsored Operators

Thanks to a generous and entirely fictitious[§] grant from Facebook^{TM}, we have been able to posit a set of alternary operators with deep social media integration:

`A` 👍 `B : C`
`A` ❤️ `B : C`
`A` 😆 `B : C`
`A` 😮 `B : C`
`A` 😢 `B : C`
`A` 😡 `B : C`

These operators post the result of evaluating `A` to your Facebook^{TM} feed and – after enough time has elapsed – check if the predominant reaction matches the one used in the operator. If so, the value of the expression is the result of evaluating `B`; if not, the value of the expression is the result of evaluating `C`.

NOTE: Lack of reactions may lead to deadlock. Overuse of these operators may lead to a lack of friends, followed by deadlock.

3.8 The Hardest Ternary Operator Ever

A tip o' the hat to George Boolos for inspiring this difficult operator:

`A □ B ◊ C`

Your operands will be evaluated by three gods, named True, False, and Random. You do not know which statement will be evaluated by which god, or in which order. True will evaluate the statement and take the result directly, False will evaluate the statement and return the opposite of the result,

[§]The generosity is, of course, real.

Random noes not evaluate the statement and returns a random result. The overall value of the expression will be either the string “`da`” or “`ja`”, but the mapping between the strings and the values `true` and `false` is not known.

3.9 Evan’s Prerequisite Dragon-Related Operator

`A $ B 🔥 C`

The result of evaluating `A` is presented to a dragon[¶] as tribute. If the dragon accepts your offering, it adds `A` to its hoard and evaluates `B`. Otherwise, your offering is deemed unacceptable and the dragon begins a rampage of burnination over the countryside while evaluating `C`.

4 General Ternary Operators

The ternary operators discussed in the previous section remain purely theoretical. However, the astute C++ programmers among you will have realized that all of the operators we have discussed conform to a relatively simple functor signature[||]:

```
template< char C1, char C2, typename TA,
typename TB, typename TC >
struct ternary;
```

Where specializations define an operator():

```
template< typename TR >
TR operator()(
  std::function< TA() > const &A,
  std::function< TB() > const &B,
  std::function< TC() > const &C
);
```

With the appropriate specializations defined, the compiler can simply ‘‘‘de-sugar” the expression

`A ? B : C`

into

```
   ternary< '?', ':', TA, TB, TC >()(
[&]()->TA{A}, [&]()->TB{B}, [&]()->TB{C})
```

So, for instance, the tradnary operator can be supported by providing the following specialization[**]:

```
template< typename TA, typename TR >
struct ternary<'?', ':', TA, TR, TR > {
  TR operator()(
    std::function< TA() > const &A,
    std::function< TR() > const &B,
    std::function< TR() > const &C
  ) {
    if (A()) return B();
    else return C();
  }
};
```

With, e.g., our second initial example de-sugaring as:

```
// (op == '+' ?  add :  sub)(a,b);
ternary< '?', ':', bool,
int(*)(int,int), int(*)(int,int) >()(
  [&]()->bool {return op == '+';},
  [&]()->int(*)(int,int) {return add;},
  [&]()->int(*)(int,int) {return sub;}
)(a,b);
```

Which is definitely enough to make one really, really appreciate syntactic sugar.

5 Future Work

- Quaternary Operators
- Veterinary Operators
- Veteranary Operators

6 Conclusion and/or Punchline

In this paper, we demonstrated that by axiomatizing “the” tradnary operator – that is, extracting a list of fraternary terperties describing it – a whole slew of new ternary operators could be defined. Like all good programming language constructs, these operators range from the practical to the absurd.

In addition, thanks to the power of modern C++ templating, we showed how at least one language could add support for new first-class ternary operators through desugaring. Though it’s worth noting that we glossed over the question of how a compiler is supposed to tokenize punctuation marks when it doesn’t know which marks are valid until after parsing a file. Command line options, I guess?

Regardless, it isn’t “the” ternary operator after all; it is simply one among many possibilities. So, in the end, we find that wizened old C programmers have been wrong all along.

References

ISO. 2017. *International Standard ISO/IEC 14882:2017(E) Programming Language C++*. International Organization for Standardization, Geneva, Switzerland, Mar. I haven’t actually read this and I probably should but you know how the time gets away from you.

[¶] Don’t have a dragon handy? Just e-mail `dragon@cmu.edu`.

[||] At least, as soon as memory transactions make it into the standard library.

[**] This does gloss over some type coercion that the actual tradnary operator performs. But we’re about 80% sure that one could express this with sufficient template wrangling.