Trits to Tetrits

Nathan T. Hayes, Sunfish Studio, LLC
P1788 Working Group

May 28, 2010

Abstract

This paper presents a view of tetrits that clarifies certain aspects of Motion 8 by providing explicit definitions and mappings. In particular, Level 1 definitions are provided as well as explicit Level 2 mappings. The requirements of a reliable exception handling mechanism for interval arithmetic are explored, and the useful role of tetrits in such a framework is examined.

1 Introduction

The interesting idea of a "tetrit" recently came up in the P1788 forum[1]. What exactly is a tetrit and how do tetrits fit into the role of exception handling for interval computations? These are questions to be answered, and discussions on the subject have already been generated.

Among the other perspectives being looked at, this paper presents one view of a tetrit aimed at simplifying and clarifying exception handling semantics described in [2]. At the same time it shows tetrits are a natural "completion" of the trit concept and fit nicely into exception handling semantics. Some examples are given in the last section.

2 Tetrits

2.1 Level 1 Definition of a Tetrit

For any real function $f : \mathbf{R}^n \to \mathbf{R}$, a tetrit is a pair (P^+, P^-) of two existentially-qualified propositions P^+ and P^- that make opposite assertions about some property P of the function f over the interval domain $X \in \overline{\mathbf{IR}}^n$, that is

$$P^+ \iff (\exists x \in X) : P(x), \tag{1}$$

$$P^- \iff (\exists x \in X) : \neg P(x). \tag{2}$$

[1] The concept of a tetrit was first introduced in the P1788 forum by Dan Zuras.

Each proposition P^+ and P^- must be true (T) or false (F), so a tetrit represents one of four possible states and is therefore an element of the set

$$\{(F,F),(F,T),(T,F),(T,T)\}. \tag{3}$$

By definitions (1)-(2) a tetrit will be (F,F) if and only if X is empty, otherwise a tetrit will be any remaining element of the set (3).

For interval X, the property P is "everywhere true" for tetrit (T,F) due to the identity

$$(\exists x \in X) : \neg P(x) \equiv \neg\left((\forall x \in X) : P(x)\right). \tag{4}$$

The left side of (4) is false because (2) is false, so $(\forall x \in X) : P(x)$ on the right side of (4) must be true. Moreover, $(\forall x \in X) : P(x)$ is not a vacuous truth since (1) is true and this means X is not empty. Similar deductions can be made for tetrit (F,T). The elements of (3) therefore have the meaning:

(P^+, P^-)	For interval X, property P is...
(T,F)	Everywhere true
(T,T)	Somewhere true, somewhere false
(F,T)	Everywhere false
(F,F)	Nowhere true, nowhere false

The notation $P(f,X)$ is shorthand for the tetrit (P^+,P^-) of the property P of the real function f over the interval domain $X \in \overline{\mathbf{IR}}^n$. The more explicit notation $P(f,X_1,X_2,\ldots,X_n)$ may also be used.

2.1.1 Natural Domain of a Function

If $D_f \subseteq \mathbf{R}^n$ is the natural domain of a real function $f : \mathbf{R}^n \to \mathbf{R}$, then for any interval $X \in \overline{\mathbf{IR}}^n$ the "domain" tetrit $D(f,X) = (D^+,D^-)$ is defined by the truth-values of the propositions

$$D^+ \iff (\exists x \in X) : x \in D_f \tag{5}$$

$$D^- \iff (\exists x \in X) : x \notin D_f. \tag{6}$$

2.2 Level 2 Exception Handling

The four states of a tetrit (P^+,P^-) are totally ordered by the priority mapping:

Priority	(P^+, P^-)	For interval X, property P is...
3	(T,F)	Everywhere true
2	(T,T)	Somewhere true, somewhere false
1	(F,T)	Everywhere false
0	(F,F)	Nowhere true, nowhere false

The priority of a tetrit represents a guarantee about the history of a computation up to and including the result of the most recent operation. More specifically, the priority of a tetrit is a guarantee that no operation in the history of

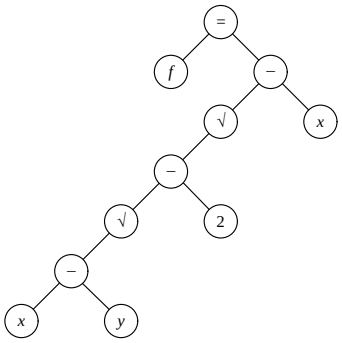

Figure 1: Syntax Tree

a computation produced a resulting tetrit of lower priority. When propagating tetrits through a computation, the resulting tetrit of any arithmetic operation is necessarily the infimum priority of all tetrits of the operands as well as the new tetrit of the operands w. r. t. the current operation.[2]

Definition 1 *If $P(f, X)$ is the tetrit (P^+, P^-) of the property P of the real function $f : \mathbf{R}^n \to \mathbf{R}$ over interval domain $X \in \overline{\mathbf{IR}}^n$, then for operands $(X_1, P_1), (X_2, P_2), \ldots, (X_n, P_n)$ of f, where each (X_i, P_i) is a decorated interval comprised of interval X_i and tetrit P_i, the resulting tetrit of the entire operation is* $\inf\left(\inf\left(P_1, P_2, \ldots, P_n\right), P\left(f, X_1, X_2, \ldots, X_n\right)\right)$.

2.2.1 The "Domain" Tetrit

P1788 decorations may have a "domain" tetrit (D^+, D^-). As mentioned above, the resulting domain tetrit of any arithmetic operation is the infimum priority of all domain tetrits of the operands as well as the new tetrit of the operands w. r. t. the current operation. A few detailed examples are given in the next section.

3 Rationale

Exception handling as it pertains to interval computations is the mechanism of propagating "exceptional" events through a computation. Consider the syntax tree depicted in Figure 1 of the example

$$f(x, y) = sqrt(sqrt(x - y) - 2) - x. \tag{7}$$

Starting at the leaf nodes of the tree, operands are provided as input, operations are performed, and results are propagated up the tree to the root.

[2] Not all decoration attributes and their respective tetrits have yet been studied by P1788, e. g., the "bounded" or "continuous" attributes. Some assumption is therefore made for the time being.

A useful exception handling mechanism follows this concept by propagating exceptional information up the tree as well as numeric results. For example, information about operations evaluated outside their natural domain may be tagged as an exceptional event, and this information will propagate up the tree to the root. For an exception handling mechanism to be reliable, the exceptional information must not be lost in the history of the computation once an exception occurs; hence the notion that propagating exception information is "sticky."

3.1 Trits and Tetrits

Motion 8 trits are elements of the totally ordered set $\{-1, 0, 1\}$ characterizing the history of a computation. The values -1 and 1 make opposite certainty claims about an associated property, and the value 0 indicates a lack of certainty about the property.

Trits are elements of decorations. An arithmetic operation on decorated intervals returns a decorated interval whose interval is the result of the operation on the argument intervals, and whose decorations are computed from the arguments such that they retain the most informative and valid information about the result.

The total ordering of trits is implied by the fact that one cannot avoid getting less and less informative decorations if different decorations are to be combined, much like overestimation of intervals cannot be avoided in many types of interval computations. The meaning of the word "informative" depends on the trit. For example, consider the isDefined trit as it propagates through the computation of (7) depicted in Figure 1 for the intervals $X = [1, 3]$ and $Y = [2, 4]$:

Operation	Result	Decoration
$[1, 3] - [2, 4]$	$[-3, 1]$	isDefined
$sqrt([-3, 1])$	$[0, 1]$	possiblyDefined
$[0, 1] - 2$	$[-2, -1]$	possiblyDefined
$sqrt([-2, -1])$	$\varnothing$	notDefined
$\varnothing - [1, 3]$	$\varnothing$	notDefined

In these respects, a tetrit is the same as a trit but with an extra state. Unlike trits, however, tetrits have a Level 1 definition. The following table summarizes similarities between trits and tetrits:

Priority	Tetrit	Trit	For interval X, property P is...
3	(T, F)	isDefined	Everywhere true
2	(T, T)	possiblyDefined	Somewhere true, somewhere false
1	(F, T)	notDefined	Everywhere false
0	(F, F)		Nowhere true, nowhere false

The most distinguishing characteristic of a tetrit is that any operation on the empty set gives the result (F, F), which has no counterpart in the trit model. Perhaps this may be a suitable Level 2 representation of the empty set in the forthcoming standard. This should be given a closer look.

3.1.1 C++ bool_set

Level 2 semantics of tetrits are the same as the multi-valued logic of bool_set as proposed in [1]. The following table summarizes similarities between tetrits and bool_set:

Priority	Tetrit	bool_set	For interval X, property P is...
3	(T, F)	{true}	Everywhere true
2	(T, T)	{true,false}	Somewhere true, somewhere false
1	(F, T)	{false}	Everywhere false
0	(F, F)	$\{\varnothing\}$	Nowhere true, nowhere false

The infimum of two tetrits is the same as the "&" operation between two bool_set values as depicted in Table 1 of [1].

3.1.2 Empty vs. NaI

Motion 8 introduced a framework to unify the concept of "Not-an-Interval" and the empty set $\varnothing$ in a semantically correct way. For example, given any non-empty bare interval X, one probably wants to have

$$X \cup \varnothing = X. \tag{8}$$

However, some applications may also need

$$X \cup \mathrm{NaI} = \mathrm{NaI}. \tag{9}$$

These different semantics can be neatly handled with decorations, for example

$$(X, \mathrm{isDefined}) \cup (\varnothing, \mathrm{isDefined}) = (X, \mathrm{isDefined}) \tag{10}$$

is an unambiguous version of (8), and

$$(X, \mathrm{isDefined}) \cup (\varnothing, \mathrm{notDefined}) = (X, \mathrm{notDefined}) \tag{11}$$

is an unambiguous version of (9). The semantics depend only on the decorations, not the bare interval portions of the decorated intervals.

The semantics (8) and (9) are also preserved with tetrits. Consider the disjoint intersection of two non-empty decorated intervals

$$([1, 2], (T, F)) \cap ([3, 4], (T, F) = (\varnothing, (T, F)).$$

The result $(\varnothing, (T, F))$ is a decorated interval that will have the semantics (8) in subsequent operations. For example, if $(X, (T, F))$ is a non-empty decorated interval, then

$$(X, (T, F)) \cup (\varnothing, (T, F)) = (X, (T, F)).$$

But if the decorated interval $(\varnothing, (F, F))$ is used to represent an uninitialized variable, missing data, or an invalid construction, then

$$(X, (T, F)) \cup (\varnothing, (F, F)) = (X, (F, F)).$$

In this latter case, the decoration (F, F) is propagated correctly through the remainder of the computation according to the semantics of (9).

Remark 2 *It is important to note that the union of two intervals is a set-theoretic operation and not an interval extension of a real function as given in Definition 1. For this reason, it doesn't make sense to consider the tetrit $D(f, X)$ when f is the set-theoretic union operation. Therefore if (X_1, P_1) and (X_2, P_2) are decorated intervals comprised of interval X_i and tetrit P_i, the resulting tetrit of $(X_1, P_1) \cup (X_2, P_2)$ is simply $\inf(P_1, P_2)$. The same is true for the intersection of two intervals.*

3.2 Divide and Conquer

Rigorous interval computations often require divide-and-conquer methods. For example, given the real vector $x = (x_0, x_1)$ and

$$\begin{aligned} g(x) &= \cos(x_0) + \sin(x_1/\exp(x_0)) + (x_0/x_1)^2 \\ f(x) &= \ln(g(x)), \end{aligned}$$

find a subset of $X = ([-4, 0], [1, 5])$ so that for all $x \in X$ in the subset

$$f(x) \leq 0$$

is guaranteed to be true. In particular $f(x) \in \mathbf{R}$ must also be true, to prevent some catastrophic disaster (such as a missile not hitting its intended target) if for some reason $f(x)$ is undefined. This requires exception-handling semantics to ensure no $f(x) = \varnothing$ is ever accepted into the solution subset.

The problem can be solved with a divide-and-conquer method, that is

if $outsideDomain(f(X'))$ then delete X'
else if $f(X') > 0$ then delete X'
else if $f(X') \leq 0$ then accept X'
else if $eps(X')$ then mark X' indeterminate
else bisect X'

Here X' is any sub-box of X and $eps(X')$ is a function to determine if X' meets some user-specified tolerance criteria or not. The method begins by initializing $X' = X$, examining the interval range of $f(X')$, and then either deleting X' from the solution, accepting X' into the solution, or recursively bisecting X' and repeating the method on each half of the bisection. If an X' cannot be deleted or accepted before the tolerance $eps(X')$ is reached, X' is marked indeterminate. The purpose of $outsideDomain(f(X'))$ will be explained in the following discussion.

Assume during this method that $g(X') = [-1, 0.3]$ for some X'. This results in $\ln([-1, 0.3])$, which is partially outside the natural domain $(0, +\infty)$ of the logarithm function. By Motion 8 semantics, the bare interval $[-1, 0.3]$ is intersected with the natural domain of the logarithm function to obtain

$$\ln([-1, 0.3] \cap (0, +\infty)) = \ln((0, 0.3]) = (-\infty, \ln(0.3)].$$

However, the resulting "domain" tetrit of $\ln([-1, 0.3])$ is (T, T). The function $outsideDomain(f(X'))$ checks to see if this "domain" tetrit is (F, T) or worse, that is it returns "true" if any operation in the history of the computation has been evaluated entirely outside its natural domain or with an empty operand. In this case, it returns "false," since $\ln([-1, 0.3])$ is only partially outside the natural domain. If $f(X')$ is evaluated with forgetful operators so that only bare decorations are returned when exceptions occur, then $f(X') > 0$ and $f(X') \leq 0$ also return "false." The consequence is all such X' will be recursively bisected until $eps(X')$ is finally true, that is none of these X' will be deleted from or accepted into the solution subset (as expected).

In the case $g(X') = [-1.3, -0.2]$ for some X', however, this results in $\ln([-1.3, -0.2])$, which is entirely outside the natural domain $(0, +\infty)$ of the logarithm function. By Motion 8 semantics, the bare interval $[-1.3, -0.2]$ is intersected with the natural domain of the logarithm function to obtain

$$\ln([-1.3, -0.2] \cap (0, +\infty)) = \ln(\varnothing) = \varnothing.$$

The entire interval box X' should therefore be deleted. This indeed happens, since the resulting "domain" tetrit of $\ln([-1.3, -0.2])$ is (F, T) and the function $outsideDomain(f(X'))$ returns "true." The sub-box X' is immediately deleted without any further work or wasted effort.

References

[1] Bronnimann, H. et. al., "Bool_set: multi-valued logic," http://www.open-std.org/JTC1/sc22/wg21/docs/papers/2006/n2046.pdf

[2] Hayes, N. and A. Neumaier, "Exception Handling for Interval Arithmetic," P1788 Motion 8.02, Oct. 13, 2009

[3] Walster, G. W., "Empty Intervals," Technical Report, Sun Microsystems, April 1998