CS 3EA3: Sheet 9 Optional Assignment - The Importance of Algebraic Properties

James Zhu `001317457`

21 April 2017

1 Abstract

Algebraic properties (such as associativity and commutativity) may be defined and applied across many domains in order to provide optimisations to the problem-solving process. The properties of associativity and commutativity and the corresponding manipulations which they allow are defined in the domain of propositional formulas, and applications of these properties onto solutions in the computing industry and their associated benefits are explored. At the same time, the consequences of a lack of such properties is highlighted. Lastly, the importance of the consideration to employ such properties in all problem domains is emphasised.

2 Introduction

In the fields of formal logic and specification, especially as it is applied for the purpose of computing and the practise of the ***correct-by-construction*** programming paradigm, we may observe certain properties across the domain of the data over which one is working.

When working across such domains, which may be reduced to an instance of a defined and acceptable *language*, one may observe a set of properties known as the **algebraic properties**, in that they define some attribute of the symbols within the said language, which may be exploited in some way to manipulate the symbols and/or their relationship with other symbols within a *sentence*. [1]

A core subset of these *algebraic properties* will be explored in this report within the context of *propositional formulas* over *boolean-valued variables*. Specifically, an *informal*, colloquial definition will be provided for each property, and the significance of the *truthfulness* (or lack thereof) of the property within the domain of computing.

3 Background and Mathematical Preliminaries

Boolean variables may be defined as the set of values over the set of {*true*, *false* }. Note that they may also be defined as *ranging over the set* ***2***. [2]

Furthermore, the set of **propositional operators** may be defined as the operations $\neg$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$.

Now, with these two sets defined, the notion of a **propositional formula** may be introduced, the latter which is composed from the ***propositional operators***, along with a ***finite set V of boolean variables***. [3]

As stated above, the properties, applications, and implications of algebraic properties will be explored within the context of this domain.

4 What are Algebraic Properties?

4.1 Definition

Formally, **algebraic properties** may be defined as *"the fundamental laws by which reasoning is performed [within a certain domain]... and upon this foundation, establish the Science of Logic"*. [1] Informally, they may said to be the properties which apply to the members of the set over which a certain problem has established its domain. In the context of this paper, the domain is specified to be closed under *propositional formulas*.

4.2 Properties under Observation

Within this report, the algebraic properties of *associativity*, and *commutativity* will be discussed.

4.3 Algebraic Property: Associativity

4.3.1 Definition

The algebraic property of **associativity** may be defined for some *propositional (binary) operator* such that, given a propositional formula containing *three boolean-valued variables a, b* and *c*, which are composited by this operator, ***the order in which the operations are carried out does not affect the final result.*** [1]

Thus, given a binary propositional operator $\oplus$, the propositional formula: $a \oplus b \oplus c$ may be evaluated as *either*: $(a \oplus b) \oplus c$ *OR* $a \oplus (b \oplus c)$, *for all values in the domain of a, b, and c.*

4.3.2 Applications to Propositional Formulas

In establishing the significance of associativity within propositional logic, consider the following example:

Three *boolean-valued variables* are defined as such:

1. *a: Mary has brown hair.*
2. *b: David has brown hair.*
3. *c: John has brown hair.*

Assume that both a and b evaluate to *true*, while c evaluates to *false*.

Furthermore, the propositional operator $\wedge$ *(conjunction)* is said to satisfy the *associative property.*

Then, the propositional formula: (1) *Mary has brown hair* $\wedge$ *David has brown hair* $\wedge$ *John has brown hair* may be evaluated as:

1. *(Mary has brown hair* $\wedge$ *David has brown hair)* $\wedge$ *John has brown hair*

   OR

2. *Mary has brown hair* $\wedge$ *(David has brown hair* $\wedge$ *John has brown hair)*

would evaluate to the same result — *false*.

Thus, it has been established that within the above formula (1), the order in which the boolean conditions are considered and simplified does *not* matter.

In matters of application, this would allow for the evaluation of the proposition *without discrimination to the order that each term must be evaluated within the expression.* Thus, for any individual wishing to establish the truthfulness of this proposition, they may compare either the truthfulness of *Mary and David* or *David and John* having brown hair first, and then take the resulting answer and apply it to a comparison with the remaining individual, without affecting the result of the proposition.

If we suppose that the comparator is a student named Daniel who is in a class with *Mary* and *David* in the morning, and with *John* in the afternoon, assuming that neither Mary nor David are in this latter class, and that neither of the individuals being compared are previously known to Daniel, then the *associativity property* would enable Daniel to track down Mary and David in the morning, and carry out the evaluation of that respective part of the expression, and then apply this result in an evaluation with John in the afternoon to get a result.

However, in the case that the associativity property does *not* hold for $\wedge$, namely that $(\ a \wedge b) \wedge c \neq a \wedge (b \wedge c)$ then, assuming that it always associates to the *right*, Daniel must first wait for the afternoon class to do the evaluation with David and John (even as he was in a class with Mary and David in the morning), and then track down Mary to complete the evaluation of the proposition.

As Mary may have already departed by that time, Daniel will then have to wait for the next day in order to get a result.

Thus, the *associativity property* thus allows for the evaluation of an expression (and any associated sub-expressions) without the prior need to transform or specify the preposition in some pre-existing form or order. Thus, the most "effective" order (as described in the above example) may be selected. [1]. This may lead to (often substantial) improvements with regards to the *running time* of any task (or computer program) which utilises combinations of $\wedge$ in this way.

4.3.3 Applications to Computing - Parsing

As established above, ***associative property over propositional formulas*** may result in significant time savings in any program requiring the evaluation of the latter. This property may also be applied to gain practical improvements in the computing domain.

For instance, consider a **parser** evaluating an expression containing n terms, stored as a linked-list: $t_1 \wedge t_2 \wedge t_3$ .... $\wedge t_n$. If the associative property were to *hold*, then the parser may simply conduct the simplification of sub-expressions as it encounters them (from left to right). This would require n *accesses* and $n-1$ *conjunction operations*. However, if the associative property were to *not hold*, namely, if as above, it were to be *right-associative* — then the parser must first *traverse through the entire expression* prior to the beginning of evaluation. Thus, this introduces an additional n access operations. In implementation, should the terms of the expression be stored within a *singly-linked-list* a *stack* may also be required in order to hold the terms as the parser encounters them, and then *pop* and process them in the reverse order. This also introduces additional space requirements.

Whilst this may not introduce significant overhead for expressions with a smaller number of terms, consider a situation in which a several *trillion* terms are within the expression — for instance, if some simulation must verify some property of all the cells within a virtual human body (estimated at about 30 trillion). In this case, the additional n accesses required for traversal would significantly increase the *practical time* required to compute the result, even as the complexity remains at $O(n)$.

If we consider, for example, that the parsing is being conducted on a cluster of server farm which is capable of ***1 billion access operations per second***. Assuming that the solution is *time-critical*, and that there is *per-cluster cost* of $ 10 000, regardless of the number of computations carried out, then the introduction of ***30 trillion extra accesses*** would result in a significant financial penalty to the user.

4.3.4 Conclusion

Inasmuch as the benefits of ***associativity*** have been established in real-world applications, the implications of such an algebraic property *not* holding, have also been brought to light. Especially in the case for relatively-simple operations which may be computed *many** (see above) times within common computing solutions (such as multiplication), the advantages brought to industry by the existence and application of the algebraic property can be very clearly felt.

5 Algebraic Property: Commutativity

5.1 Definition

The algebraic property of **commutativity** may be defined for some *propositional (binary) operator* such that, given a propositional formula containing *two boolean-valued variables* a and b, which are composited by this operator, ***the order in which the operands appear does not affect the final result.*** [4]

Thus, given a a binary propositional operator $\oplus$, the propositional formula: $a \oplus b$ *is equivalent to* $b \oplus a$, *for all values in the domain of of* a *and* b.

5.1.1 Applications to Propositional Formulas

In establishing the significance of the commutative property within propositional logic, the example provided in **Section 4.3.2** is once again considered. However this time, the following proposition is made: (2) *Mary has brown hair* $\wedge$ *David has brown hair.*

If the ***algebraic property of commutativity holds over the*** $\wedge$ ***operator***, then the expression may be rearranged and evaluated in the order: *David has brown hair* $\wedge$ *Mary has brown hair*, while producing the exact same result as (2) — *true.*

Thus, it has been established that within the above formula (2), the order in which the boolean operands appear does not matter.

In the realm of application, this algebraic property would thus allow for the evaluation of the proposition *without discrimination to the order in which each term appears as an operand.* Thus, taking the task undertaken by Daniel from the previous example once more, he may either check the hair colour of Mary OR David first, and then the other, and apply the conjunction correspondingly, arriving at the same result. Thus, if for instance, Daniel is in a class with Mary in the morning, and David in the afternoon, and he additionally knows beforehand that Mary will have departed at lunchtime, then Daniel can choose to evaluate the condition for Mary first, prior to David. Thus, once again, he would have saved an additional day (or more) of waiting to be in class with Mary again (even though David may be available all day!).

Again, the *commutative property* thus allows for the evaluation of an expression and any associated sub-expressions, with the operands in any order which may be **more convenient to suit the operands or purpose.** This may lead once more to significant improvements of in the run-time of tasks (or computer programs), whose operators exhibit this property.

5.1.2 Applications to Computing - Prioritisation for Efficiency

As established above, ***commutative property over propositional formulas*** may also be applied to gain practical improvements in computing solutions, similar to associativity.

Consider the **Cell Processing Problem** presented in **Section 4.3.3**. If commutativity were to hold for $\wedge$, then the processing program may select any of the cells to be verified *in any order* which is desirable, without changing the final result. Thus, for instance, if the property which is to be verified for each cell requires more processing power to satisfy for a certain type of cell compared to others, and the server farm charges for usage based on user-demand, then the researcher may select to verify these former cells at a time of day wherein the demand for processing power may be lower. This would lead to significant financial savings, as stated in Section 4, if the amount of cells required to be verified is in the magnitude of billions or trillions.

If however, *commutativity does not hold*, then in the above example, the cells may have to be processed in a *pre-defined* order, the latter which may not necessarily be the most efficient use of resources, dependent on the nature of the computing device.

5.1.3 Conclusion

As with *associativity*, the benefits of commutativity — as well as the drawbacks of operations which do not sport this aforementioned algebraic property is evident when considering solutions which may be used in the commercial realm. This is especially applicable to applications which are either time and/or

resource-sensitive. In fact, the latter may also be taken into consideration for instances wherein the amount of *"workers"* to complete a certain task is limited, and wherein a greater *amount of terms within the expression that is resolved* per unit-time provides greater benefit (in lieu of a final result). Thus, this may encourage the ordering of operands such that "smaller" sub-expressions (or which require less computational power) are processed first, instead of a larger sub-expression which may potentially delay the processing of other sub-expressions that appear later in the queue.

6 Conclusion

In the course of this paper, the algebraic properties of **associativity** and **commutativity** have been defined within the context of *propositional formulas*. Specifically, the ways in which the properties expose certain patterns of manipulation on the operators to which they apply, as well to the data which lie their respective domains, has been established. Following on this, the possibilities for the application of such algebraic properties to provide significant optimisations and improvements in efficiency to solutions in the computing and software industry (among others) were explored. Lastly, the (significant) consequences which may arise due to the *non-existence* of these properties for some operations was highlighted.

All in all, algebraic properties not only allow for expressions to be manipulated in such a way which best suits the domain (input, requirements, expected output, etc) of a specific problem, but more essentially, allows for the creative and dynamic resolution of these problems in ways which often provide much greater benefit in comparison to the "cost" that is required to apply them.

References

[1] R. Backhouse, *Program Construction: Calculating Implementations from Specifications*. Chichester, United Kingdom: John Wiley & Sons, Ltd, 2003.

[2] J. Zucker, Class Lecture, Topic: "Recursive Function Theory and Computability: Introduction; Mathematical Preliminaries." CS 3TC3, Department of Computing and Software, McMaster University, Hamilton, Canada, January 2017.

[3] H. Zantema, "SAT solving, SMT solving and Program Verification", Eindhoven University of Technology, Eindhoven, Netherlands, 2011.

[4] M. Al-hassy, Class Lecture, Topic: "Lattice Theory" CS 3EA3, Department of Computing and Software, McMaster University, Hamilton, Canada, Winter Term 2017.