Information and Inference: A Journal of the IMA () , 1–9
https://doi.org/

Beyond Set Theory: Bunches

YIYANG ZHOU

[]

It is impossible to deny the importance of Set Theory in modern Mathematics. With only a few axioms, it has become the cornerstone of many different fields of Mathematics. However, this does not mean that Set Theory is the only way to conceptualize the idea of *a collection of things*. In this report, we examine an alternative structure that could be a good complement to Set Theory.

1. Introduction

If we take a closer look at how Set Theory conceptualize *a group of items*, we will realize, that the set of items is different from the items themselves. Indeed, a pile of potatoes is different from a set of potatoes, and all molecules in a droplet of water is different from a set that consists of those molecules. In other words, by putting items into a set, we are actually doing two distinct actions - grouping and packaging. This hints us to consider another possibility. What if we do only the grouping, but not the packaging of those items? Hehner explores this kind of data structure, calling them "bunches", in his book *a Practical Theory of Programming*[2]. He mainly uses them in his reasoning of program correctness, as a simplified version of Set Theory. We instead, will focus more on how to generalize various concepts and definitions in Set Theory to incorporate this so called *Bunch Theory*.

2. Expectations of bunches

Way before Set Theory is even a thing, people already know how to do complex arithmetic on all kinds of numbers, even though they were not able to fully justify their existence and prove why they should behave like that. Hence, it is helpful to us that we first develop some reasonable expected behavior of our new mathematical object from an intuitionistic point of view.

2.1. *What do you mean by* unpackaged*?*

In Set Theory, a set is not considered the same as its *contents*. When we say "the set of natural numbers", we are not actually referring to the "numbers" themselves. We speak of them as something different - an abstract entity that captures the common properties of those elements. In contrast, an *unpackaged* collection of "things", we shall now call them *bunches*, makes no distinction between the bunch and the contents of the bunch. When you put two piles of potatoes together, Set Theory will tell you that it is a set of two sets of potatoes. Bunch Theory, on the other hand, simply regards them as a larger pile of potatoes. As a result, a bunch of bunches should not be any different than just a "flat" bunch that is made up of the elements in those bunches. As another example, *strings* are also unpackaged collections of items (additionally, they are also *indexed*, but this is not important here). If you concatenate a string with another, the result will simply be a longer string, with no way to *single out* the first nor the second one from it.

2.2. *That just sounds like set union with extra steps*

In a sense, yes. But as we will see in a moment, that some operations, such as non-deterministic function applications, can be more easily and concisely explained with Bunch Theory than with Set Theory. An

important object that is unique to Bunch Theory is *null* (the empty bunch). *null* represents the idea of *true nothingness*. There is not anything, not even a container to encapsulate this emptiness (unlike $\emptyset$, which itself is still *something*). On the other end of the spectrum, thanks to the simplicity of bunches' structures, the Universal Bunch (the bunch that contains everything in the universe) does not lead to paradoxes like the sets do. Though it is not entirely useful to consider such a huge bunch in a Computer Scientist's practical works, we shall see whether we can extend our existing rules and notations to incorporate those large bunches.

3. An attempt to make something out of nothing

Set Theory starts out from almost nothing, only the empty set $\emptyset$. It remarkably proceeds to build the entire city of mathematics out of it. As for Bunch Theory, our structural simplicity has become our first curse - It is unclear whether we can go any further than just getting the empty bunch, without requiring us to make too much assumption about the existence of other bunches. Remember, that we do not think a bunch of bunches is any different than the union of those bunches, so putting the empty bunch in a bunch returns the empty bunch back to us.

Maybe it is just too ambitious to completely replace Set Theory with Bunch Theory. Maybe it is utterly impossible. But still, we should not abandon the thought so easily just because we suffered such a small setback. Even if we can't be as great as Set Theory, we can be an useful extension to it.

Therefore, our first axiom:

Axiom 1 (The Axiom of Universe) Every set is also a bunch.

A few things need to be clarified here. First, when we say "every set", we mean not to say the *contents* of that set. For example, the axiom says that the set $\{0,1,2\}$ is a bunch, but it does not say anything about $0,1,2$ (although we will soon add more axioms to make this reasonable-looking object also a bunch). In this case, we do really mean that there exists a bunch which consists of only a single element - the set $\{0,1,2\}$, or simply, 3. Yes, although a bunch of bunches is meaningless to talk about, we do allow sets to be the elements of bunches.

Bunches given rise by this axiom are so fundamental that we will give them a special name, *elementary bunch*. We will also sometimes use the word *element* instead as an abbreviation. The usage of the word "element" here and the one in the context of a set is not so ambiguous, as the word actually refers to the same thing in both contexts. A bunch of one element is no different from the element itself.

With the Axiom of Universe set in place, we are now ready to construct more interesting bunches. Set Theory has many constructional axioms. We only need one thanks to the simplicity of our structure:

Axiom 2 (The Axiom Schema of Bunch Comprehension) For any predicate $\mathbf{P}$ defined on all elementary bunches, there exists a bunch B such that x is in B if and only if $\mathbf{P}(x)$. We denote B as $x\$\mathbf{P}(x)$ (pronounced "x such that $\mathbf{P}(x)$").

Notice that we very carefully avoid using any undefined symbol here. Even something as simple as $x \in B$ must not be written down before we fully define what it means for an element to be *in* a bunch.

Two bunches that worth special attention are:

- The null bunch: $null = x\$False$, and
- The Universal Bunch: $U = x\$True$.

3.1. *Enhanced bunch comprehension*

Sometime we may wish to write the elements in the bunch as an expression. For example, we may wish to use the expression $x+2\$x :\in \mathbb{N}$ to denote the bunch of all plural natural numbers. We define such expressions as follows:

Definition 1 Suppose $E(x)$ is a function from elements to elements, and $\mathbf{P}$ is a predicate of elements. Then

$$E(x)\,\$\,\mathbf{P}(x) = y\$\,\exists x, (P(x) \wedge y = E(x)).$$

4. Additional constructional operations

4.1. *Bunch union*

Definition 2 Let $A = x\$\mathbf{P}(x)$ and $B = x\$\mathbf{Q}(x)$ be two bunches. The *bunch union* of A and B, denoted by A,B, is the bunch $x\$\mathbf{P}(x) \vee \mathbf{Q}(x)$.

Note that this is not an axiom, meaning that it is still possible to express bunch union using only the Axiom of Bunch Comprehension. We define it separately here because of the notaional conciseness it provides. If A (or B) is an elementary bunch, we can think of $\mathbf{P}$ (or $\mathbf{Q}$) as $\mathbf{P}(x) : x = A$. The $=$ symbol here is the *set equality*. This is consistent with our definition that $\mathbf{P}$ must be defined on all elements of the universe (i.e. all sets).

Examples of bunch unions:

- If $A = 0$ and $B = 1$, then $A,B = 0,1$.
- If $A = 1,2,4,8$ and $B = 2,3,5,7$, then $A,B = 1,2,3,4,5,7,8$.
- $null,A = A$ for all bunch A.
- $U,A = U$ for all bunch A, where U is the Universal Bunch.

We deliberately choose the "," to be our union operator so that bunches and sets *look alike* when written down. A word of caution, though. This does not mean that those two symbols have the same meaning in every context. Consider this following example. Suppose D and R are some set:

$$f : D \longrightarrow R,$$
$$f(a,b) = \{a,b\}.$$

We may not substitute a, and b in the expression $\{a,b\}$ if they are not both sets (elements). In fact, if a or b were bunches, the result of applying f to them would be another bunch, but simple substitution yields a set (if such a set even exists). In this case, the mistake is obvious since the domain is given, but not all instructions are given explicitly as functions. We will learn more about this when we get to the section on ordered pairs of bunches.

4.2. *Unpackaging*

Definition 3 Let S be a set. We call $\sim S$ the *contents* of S. It is the bunch $x\$x \in S$.

Examples of unpackaging:

- $\sim\{0,1,2\} = 0,1,2$.
- $\sim\emptyset = null$.
- $\sim\mathbb{N} = nat$, where *nat* is the *natural bunch*.

4.3. *Packaging*

Definition 4 Let $B = x\$\mathbf{P}(x)$ be a bunch. If $S = \{x\,|\,\mathbf{P}(x)\}$ is a set, then we say $S = \{B\}$. The $\{\}$ is an one-place operator called *packaging*.

Examples of packaging:

- $\{0,1,2\} = \{0,1,2\}$, where the left-hand side is the packaging of the bunch $0,1,2$, while the right-hand side is the set 3. Not very surprising, is it?
- $\{null\} = \emptyset$.
- $\{U\}$ does not exist,

5. Bunch inclusion and extensionality

Unlike set membership, which is a basic property of all sets, we must give a clear definition of bunch membership. Nevertheless, bunch membership and bunch inclusion can still be defined in an fairly straightforward way.

Definition 5 Let A, B be bunches, we define the bunch membership $:\in$ as:

- If B is an element, then $A :\in B$ if and only if $A = B$. The $=$ here is the set equality.
- Otherwise, let $B = x\$\mathbf{P}(x)$, then $A :\in B$ if and only if $\mathbf{P}(A)$.

Definition 6 Let A, B be bunches, we say $A :\subseteq B$ if and only if $\forall x, (x :\in A \to x :\in B)$.

Remark 1 *When A is an element, $\forall B$, $A :\in B \leftrightarrow A :\subseteq B$.*

And finally, we have the Axiom of Bunch Extensionality:

Axiom 3 (The Axiom of Bunch Extensionality) Let A, B be bunches, $A = B$ if and only if $A :\subseteq B \wedge B :\subseteq A$.

6. Ordered Pairs of Bunches

One reason why we care about Bunch Theory is its simplicity when dealing with non-determinism. Non-determinism, in essence, says that one input may correspond to multiple outputs, one pre-condition my correspond to many post-conditions. There are some in-real-life examples of this - the weather of any future day is a non-deterministic function from dates to weather conditions - but more applications can be found in computer program analysis.

Bunches themselves can be think of as a value that is non-deterministic. For example, if we let a variable $x = 0, 1, 2$, we can consider x to be in a state where it attains a value of 0, 1, or 2, but we are not sure which exactly. This kind of thinking is important in understand bunch relations and bunch functions, which we are going to explore next.

In Set Theory, ordered pair is denoted by a pair of parentheses and a comma between the left and right elements. Unfortunately this clashes with our notation of bunch union. Therefore, we prefer using the maplet symbol $\mapsto$ instead.

Definition 7 Let A and B be bunches. The ordered pair of bunches

$$A \mapsto B = z \$ \exists x \, \exists y, \; (x :\in A \wedge y :\in B \wedge z = (x, y)). \tag{6.1}$$

where $(\_, \_)$ is the pair construction operator on sets.

This definition essentially says that bunch pairs are distributive over bunch unions. For example, $(0,1) \mapsto (2,3) = 0 \mapsto 2, 0 \mapsto 3, 1 \mapsto 2, 1 \mapsto 3$. We can think of a pair of bunches as a non-deterministic pair of elements. Therefore every combination of two elements from both sides is a possible outcome.

One may try to define bunch pairs using the definition of ordered pairs in Set Theory (i.e. $(a,b) = \{\{a\}, \{a,b\}\}$), and draws the conclusion that $(0,1) \mapsto (2,3) = \{\{0,1\}, \{0,1,2,3\}\}$. But again, this is wrong since the comma in this expression is not bunch union, and the brackets are not bunch packaging operator either. It is a mistake to blindly substitute variables in expressions that are in the scope of only sets. Remember that $(\_, \_)$ and $\mapsto$ are the same operator, written differently, so just like the $(\_, \_)$ operator, the $\mapsto$ operator is actually a 2-place function from sets to sets, but now its domain is extended to include bunches as well. Therefore, the correct way to evaluate $(0,1) \mapsto (2,3)$ is to first use the distribute rule in (6.1) to turn it into the bunch $0 \mapsto 2, 0 \mapsto 3, 1 \mapsto 2, 1 \mapsto 3$, then we can substitute a and b with those numeric values if we so desire.

For tuples of higher arity, we use the right associative definition. That is, a 3-tuple of A, B and C is actually $A \mapsto (B \mapsto C)$. Note that this is different from the usual definition in Set Theory, where the tuples are left associative $(a,b,c) = ((a,b),c)$. Our definition has the benefit of making reasoning with curried functions more straight forward. Function currying is a common technique employed by many modern programming languages that turns a multivariate function into a chain of single-variate functions that returns a function which accepts the next argument.

7. Function application on bunches

As we have discussed, the pair construction is a special case of function application on bunches. We now give function application on bunches a more general meaning.

All functions defined on sets can be extended to be applicable on bunches as well:

Definition 8 Let A be a bunch. Suppose f is a function that maps sets to sets, then

$$f(A) = y \$ \exists \, x, \; (x :\in A \wedge y :\in f(x)). \tag{7.1}$$

Note the $:\in$ on the right hand side of the definition. One may wonder what difference it makes to not use an ordinary set equality $=$ instead. This will become relevant once we introduce the concept of non-deterministic functions, where we will see that it is possible for functions to not return an element, but a bunch.

We still use the notation $f(a,b)$ to denote a function application to two variables. Thus, when we want to apply a function to a bunch of two elements, we must write $f((a,b))$ instead.

8. Wholistic function application

Sometimes, we may still want to apply a function to the entire bunch as a whole. For example, finding the union over a bunch is actually a function that applies to the entirety of the bunch. Should we use our previous definition, this function must be distributed over every element of the bunch, which is not what we want. We have to find another way to define it.

Definition 9 Let f be a function. For any bunch A such that $\{A\}$ exists, the wholistic function application $f.A$ is defined as:

$$f.A = \sim f(\{A\}).^{1} \tag{8.1}$$

Equation 8.1 *lifts* the normal set function application. One limitation of this definition is that it can't handle bunches that are "too large to fit into a set", even though sometimes the result makes intuitive sense. For example, it can be reasoned that the intersection over the Universal Bunch U *should* be *null*, and the union over U *should* be itself (For any set S, $\{S\}$ is also a set. All such sets are in U), but we cannot deduce those facts from our axioms since $\{U\}$ is not a set.

9. Non-deterministic functions

In Set Theory, a function is defined as a relation such that each input maps to only one unique output. Relations that do not satisfy this constraint are not regarded as functions. But from another point of view, those relations that maps one input to multiple outputs can be seen as *non-deterministic functions*, and therefore the ordinary functions are now *deterministic functions*.

Deterministic functions map an element to another unique element. Non-deterministic functions map an element to a bunch of elements. Just like their deterministic counterparts, non-deterministic functions can also be applied to bunches, in which case the same rule applies (equation 7.1).

Definition 10 Let R be a relation from sets to sets. For any set x (which may or may not be in the domain of R):

$$R(x) = y \,\$\, x \mapsto y \in R. \tag{9.1}$$

Examples: Suppose R is the relation $\{(m,n) \mid \exists\, m\, \exists\, n, (m > 0 \wedge n > 0 \wedge m \textit{ divides } n)\}$.

- $R(1) = 1,2,3,4,\ldots$
- $R(2) = 2,4,6,8,\ldots$
- $R(0) = null$
- $R((2,3)) = R(2), R(3) = 2,3,4,6,8,9,10,12,\ldots$

---

[1] I am really not sure if this is the only way to define wholistic function application. Sources I have read either makes the same compromise as I did there [3], or does not care to mention the limitation of Set Theory at all [1][2]. Seen from a functional programming's point of view, normal function applications on bunches *map* bunches to other bunches, while wholistic function applications *reduce* a bunch to an element. However, *reductions* are meaningful only when the collection is *countable*, which is an even smaller sub-bunch of U (though it is true that Computer Scientists usually does not care about uncountable sets).

Remarks: From now on, we no longer distinguish functions and relations anymore. Relations are simply non-deterministic functions, and all normal function application rules apply to them as well.

We also implicitly extend the domain of all function to the class of all sets, with those not originally in the domain mapped to *null*. Therefore:

$$f(U) = ran\ f. \tag{9.2}$$

10. Set membership in non-deterministic contexts

The set membership is a fundamental property of sets. We can now extends this property to be applicable to bunch too. We will stick with the notation $\in$.

Note the difference between $:\in$ and $\in$. The former is the bunch membership, which can only be used when the left operand is an element. The latter is an extension of the set membership, which we are going to define as:

Definition 11 Let E, S be bunches. If both $\{E\}$ and $\{S\}$ exist, we then say E is in S, or S *non-deterministicly contains* E if and only if:

$$\{E\} \subseteq \bigcap \{S\}. \tag{10.1}$$

We denote this fact by $E \in S$. Note that when E and S are both elements:

$$\{E\} \subseteq \bigcap \{S\} \iff E \in S. \tag{10.2}$$

where the $\in$ is the normal set membership. Therefore, this definition is compatible with traditional Set Theory.

This definition essentially says that no matter how the non-determinism on both sides of the relation is resolved, the left-hand side will always be a member of the right-hand side.

Examples:

- $\emptyset \in \mathbb{N}$.
- $0,1,2,3 \in \{0,1,2,3,4,5,6\}$.
- $b \in \{a,b\},\{b,c\}$.
- $0,1 \in 2,3$.
- $1 \mapsto 2 \in ((\{0,1\},\{1,2\}) \times (\{2,3\},\{2,4\}))$.

A worth-mentioning difference between $\in$ and $:\in$:

$$x :\in E,F \iff X :\in E \vee x :\in F \tag{10.3}$$

where x must be an element, while

$$x \in E,F \iff x \in E \wedge x \in F. \tag{10.4}$$

Incidentally, the non-deterministic set inclusion (subset) is also extended to include bunches.

Definition 12 Let E, S be bunches. If both $\{E\}$ and $\{S\}$ exist, then:

$$E \subseteq S \iff \forall x, (x \in E \longrightarrow x \in S). \tag{10.5}$$

11. Non-deterministic Cartesian products

The very last example in the previous section shows an interesting result of our definition of non-deterministic set membership. In traditional Set Theory, we have the following axiom for all sets:

$$E \mapsto F \in (S \times T) \iff E \in S \wedge F \in T. \tag{11.1}$$

We now show that this axiom can be generalized to include non-elemental values.

Proof

$$E \mapsto F \in (S \times T) \tag{11.2}$$
$$\iff \{E \mapsto F\} \subseteq \bigcap\{(S \times T)\} \tag{11.3}$$
$$\iff \forall e :\in E, \forall f :\in F, (e \mapsto f \in \bigcap\{(S \times T)\}). \tag{11.4}$$
$$\iff \forall e :\in E, \forall f :\in F, \forall s :\in S, \forall t :\in T, (e \mapsto f \in (s \times t)). \tag{11.5}$$
$$\iff \forall e :\in E, \forall f :\in F, \forall s :\in S, \forall t :\in T, (e \in s \wedge f \in t). \tag{11.6}$$
$$\iff \forall e :\in E, \forall s :\in S, (e \in s) \wedge \forall f :\in F, \forall t :\in T, (f \in t). \tag{11.7}$$
$$\iff \{E\} \subseteq \bigcap\{S\} \wedge \{F\} \subseteq \bigcap\{T\} \tag{11.8}$$
$$\iff E \in S \wedge F \in T. \qquad \square$$

Note that both $\mapsto$ and $\times$ are actually 2-fold functions applied to bunches, so they follow the bunch function application rule.

12. Non-deterministic power set

We will now show that the Axiom of Power Set is also compatible with our definition.

As a reminder, in Set Theory, we have:

$$S \in \mathbf{P}(T) \iff S \subseteq T \iff \forall x, (x \in S \longrightarrow x \in T). \tag{12.1}$$

This axiom can be generalized to include bunches as well.

Proof

$$S \in \mathbf{P}(T) \tag{12.2}$$
$$\iff \{S\} \subseteq \bigcap\{\mathbf{P}(T)\} \tag{12.3}$$
$$\iff \forall x \in S, (x \in \bigcap\{\mathbf{P}(T)\}) \tag{12.4}$$
$$\iff \forall x \in S, \forall t \in T, (x \in \mathbf{P}(t)) \tag{12.5}$$
$$\iff \forall x \in S, \forall t \in T, (s \subseteq t) \tag{12.6}$$
$$\iff \forall x \in S, (x \subseteq \bigcap\{T\}) \tag{12.7}$$
$$\iff \forall s, (s \in S \longrightarrow s \in T). \qquad \square$$

13. Conclusions

The biggest difference between Bunch Theory and Set Theory is that Bunch Theory does not package the items after collecting them. Therefore, a bunch of one item is no different than the item itself. A bunch can also be considered as the *contents* of a set, but not all bunches can have a set to containerize them. The structural simplicity of bunches allows them to grow big yet not cause Russell's Paradox. On the other hand, the null bunch *null* is a concept that is new and not included in the traditional Set Theory, which represents the *true nothingness* of items. In addition, a bunch naturally represents non-deterministic values and operations. We have shown this in terms of non-deterministic function applications. We also defined non-deterministic set membership, while proving that some of the axioms of traditional Set Theory can be naturally extended to include bunches.

In this report, we mainly examined some of the properties of bunches that are closely related to Set Theory on which it is based. However, the usefulness of introducing such a mathematical structure is not illuminated. Some of the referenced works use it in the reasoning of Descriptive Theory[3], others incorporate it into Programming Theory[1][2]. *null* seems to play an important role in Descriptive Theory by conceptualizing the *undefined* term, while the non-deterministic nature of bunches models the process of a computer program very nicely. To illustrate those applications would require much researching into those respective fields, which would most likely take this report off-topic.

A good continuation of this report could be on how the null bunch *null* handles ambiguous terms in traditional Description Theory (for example, Russell's Theory of Descriptions); which article could then devote more pages to Descriptive Theory. Another option would be examining how Bunch Theory notations could be used in the grammars of programming languages that supports non-determinism.

References

1. Eric C. R. Hehner. Bunch theory: A simple set theory for computer science. *Inf. Process. Lett.*, 12:26–30, 1981.
2. Eric C. R. Hehner. A practical theory of programming. In *Texts and Monographs in Computer Science*, 1993.
3. Bill Stoddart, Frank Zeyda, and Steve Dunne. Bunch theory: working notes on applications, axioms and models. *arXiv: Logic in Computer Science*, 2019.