Notes for Bachelor project report

**[important for writing]: Only cover one idea per subparagraph to make it more readable. Also make sure for every new idea, first introduce it then explain it then recap.**

**Pick the best section to talk about some of the implementation choices that can be made. So in some of the material given the max-heap property is being used while in the Lean 4 language min-heap property is used. Also the order of the children is different in the Lean 4 language. This ordering doesn’t affect the way the binomial heap can be used.**

**When we were checking if our predicates were right we found multiple errors on different occasions and thus there were several intermediate versions of the predicates before we reached the final versions which are almost certainly correct. One of the intermediate versions of the predicates can be seen below……..**

**Also because LEAN4 is still a work in progress, sometimes working code would end up not be working because the language was changed in the meantime so this meant that the existing code would have to be changed and updated to work with the new changes.**

**Talk about the rename\_i, that it is better to not use it but in some cases not using it would complicate the code more and the benefits wouldn’t outweigh the negatives (with split)**

**Talk about the variable(le …) added because it was needed and le didn’t give enough context.**

**Post the code to github and in the report also when talking about a piece of code describe the difficulties and the length so that it becomes clear how difficult it was.**

**Talk about the membership predicate needed in order to fully verify that the binomial heap implementation is correct and that for the scope of this project it would complicate things and isn’t worth it because it is pretty trivial by looking at the function that this will hold.**

**After an update unfolding min didn’t work anymore so it was necessary to create a theorem which basically unfolded min for us to split the cases of min.**

**Some of the code might be redundant and could be replaced by simply doing simp now because the language has changed but before we had to fix it with lemmas and do it manually**

**The code can probably be shortened quite a bit but this will make it less readable and my preference is to keep it more readable since the only benefit of shortening the code will be that there are less lines of code(ask Jannis for the word he used for shortening the code using for ex: <;>). Probably also mention that the readability could further be improved by using better naming.**

**When working on the IsHeap\_merge proof it was necessary to make a recursive call to the theorem itself, at that moment it was necessary to add (xs := t₁) or (ys := t₂) to the recursive call because lean couldn’t figure out which part of the goal it should fill in for ys. When going through the code to check for mistakes I tried to remove this part to check if the newer Lean4 versions would figure it out by itself which it did. This example is one of the many improvements that were made to the Lean4 language which made my proofs less complicated**

mutual

/-IsHeap rank heap :<=> loops through the trees in a heap and checks if the trees are ordered by rank

starting with the smallest. For each tree in the heap IsBintree needs to hold-/

inductive IsHeap : Nat → Heap α → Prop where

| empty\_heap: IsHeap rank (heap [])

| single\_tree: IsBinTree t → IsHeap rank (heap [t])

| non\_empty: rank < t.rank → IsBinTree t → IsHeap t.rank (heap ts) → IsHeap rank (heap (t::ts))

inductive IsBinTree : BinTree α → Prop where

| C: IsRankedTree 1 a.rank a.children.nodes → IsBinTree a

/-IsRankedTree n m ts :<=> the list ts contains the children of the parentnode of a binomial tree, IsRankedTree

assures that the order of the rank of the children is n, n+1, n+2,.....,m-1 and if n = m, then ts is empty-/

inductive IsRankedTree : Nat → Nat → List (BinTree α)  → Prop where

| no\_child: IsRankedTree n n []

| with\_child: t.rank == n  → IsRankedTree (n + 1) m ts → IsBinTree t → IsRankedTree n m (t::ts)

end

mutual

/-IsSearchForrest le heap :<=> loops throug the trees in heap and checks that for each tree in the

heap IsSearchTree holds-/

inductive IsSearchForrest (le : α → α → Bool) : Heap α → Prop where

| empty\_heap: IsSearchForrest le (heap [])

| non\_empty: IsSearchTree le n → IsSearchForrest le (heap ns) → IsSearchForrest le (heap (n::ns))

inductive IsSearchTree (le : α → α → Bool) : BinTree α → Prop where

| C: IsMinHeap le a.children.nodes a.val → IsSearchTree le a

/-IsMinHeap le ns val :<=> assures that val(value of parent node) is less or equal than the value

of the nodes in ns(the children). Maintains minimum heap property-/

inductive IsMinHeap (le : α → α → Bool) : List (BinTree α) → α → Prop where

| no\_child: IsMinHeap le [] val

| with\_child: le val n.val → IsMinHeap le ns val → IsSearchTree le n → IsMinHeap le (n::ns) val

# Abstract

First talk about broader scientific context and who the paper is for. Then something like programming languages are constantly being improved to enhance the usability and increase the compilation speed for example. These improvements are introduced in new versions of the language but with these changes there is also room for new errors in the implementation of the language. To ensure that the implementation is correct, a formal verification language can be used. Recently a new version of the Lean programming language has been released, Lean 4. In this paper the implementation of the binomial heap data structure in Lean 4 will be formally verified. The formal verification will be done with the Lean 4 language itself. In order to formally verify a data structure we need predicates that hold the properties that a binomial heap should have. To formally verify the implementation we need to proof that for every binomial heap in the language these predicates hold. In the process of formally verifying the binomial heap data structure some changes were made to the code, mainly to improve the readability of the code and making it more intuitive. However no major faults were found. The full formal verification proof of the binomial heap data structure in the Lean 4 language can be found ……….

# Introduction

Talk a little about the Lean language and binomial heaps

# The problem

# The main idea

To answer the research question: Is the binomial heap data structure correctly implemented in the Lean 4 language?, the Lean 4 language itself will be used. As mentioned earlier the Lean 4 language is not only a programming language but also a theorem prover that allows to formally verify processes. To achieve this goal It will be necessary to proof that each instance of a binomial heap in the language is indeed a binomial heap and thus conforms to all the criteria that a binomial heap needs to meet in order to be a binomial heap. These criteria [reference 1]

The main idea behind this research paper is to check whether the implementation of binomial heaps in the programming

# Process of verifying the implementation

After the first meeting with Jannis I received learning material that I would need in order to start with the project. The material I received consisted of

# Mistakes and improvements to the code

* When I was first studying the material given to me by Jannis I also looked at the code of the binomial heap implementation in Lean 4, It seemed to me as if the structure that represents a node in a binomial tree was implemented in not the most intuitive way. As Lean 4 was a new language to me I assumed that I just didn’t understand it properly. I tried to explain it to Jannis but I think I wasn’t clear enough so I left it as it was. But as I continued with the project and started making the predicates needed to proof that the implementation of the binomial heap was correct I still found the implementation confusing. This time Jannis agreed with me that the children of a heapnode shouldn’t be a list of heaps but a list of heapnodes which comes down to the fact that the children can be seen as a heap.
* A binomial heap can be implemented in more than one way. One of the decisions that must be made is if it is going to be a min-heap or a max-heap. In a max-heap each parent node has a value greater or equal than it’s child nodes, and in a min-heap each parent node has to have a value smaller or equal than it’s child nodes. One of the comments in the implementation of the binomial heap mentions “Additionally, the value of each child must be less than or equal to the value of its parent node.” , this comments suggest that a max-heap is implemented. However, looking at the code I realised that a min-heap was implemented which makes this comment invalid.

# References

1. Algorithms, 3rd Edition, in Java, Parts 1-4: Fundamentals, Data Structures, Sorting, and Searching. Robert Sedgewick, Addison-Wesley 2002. §9.7

**Table of contents**

Title : verified binomial heaps in lean 4

Table of contents:

Introduction : 2 pages (motivation)

Include research question, maybe add two:

-Is lean 4 suitable for verification of data structures

-Is the binomial heap data structure implemented correctly in lean 4

Background : 2 pages

Binomial heaps : 1 page (describe in general)

Lean : 1 page

Definition of the data structure : 2-3 pages

Basic operations : 1 page

Merging : 3 - 4 pages

Deletion : 3 page

Related work : 1 page

Conclusion/discussion : 1 page

Questions:

* Can I add a section where I mention the limitations of the work?
* The basic operations, merging and deletion sections are all sections on the proofs, can I make them sub sections of a section that is for example called proofs?
* Is help predicate a good name

Use monospace font

Create own style for code in style section

Don’t put the tactic snippets in but functions and goals mergenodes and possibly some of the goals.

Call the helper theorems lemmas and the main ones theorems

The def instead of the ind: with the ind you can do induction on the predicate itself

In the related work find papers that are close to this one in at least one dimension so qua data structure if it is similar to the binomial heap and if the language is similar to the Lean 4 language qua dependent type theory

@-modifier

Questions:

* You placed a comment about what the dynamic operations are, they mention it in the book you gave me but should I use a reference for that or should I just leave it out and name the operations?
* Is it okay to leave the full name of the sections in the overview I give at the end of the introduction?
* I removed the quotes but can I then still use the reference for my own description? A more detailed description….
* How important are references because I tried to find a source about binomial heaps but I would mostly find papers about improved versions
* Should I mention the line number of where the code can be found?
* Do you know why by\_cases was necessary because I forgot
* Is saying that a node complies with the IsHeap predicate correct
* Is it okay to add the findMin proofs as a sub-section to deletion and ask about extra proof

Questions:

* Can I use: git push origin main –force , to push my work on to github?
* Ask if papers found are good and if I want to tell the differences between my predicates and theirs I should thus learn some Isabelle and coq?
* In the conclusion is one of those pieces good to conclude the paper
* If it is really necessary to change the names of the predicates because I will have to change the names of theorems as well
* Comments on the predicates good?
* Are the references used good

Related work: also difference between predicates..

Rephrase: the aim of Lean is to close the gap between automated….

Talk about specific features or tactics that were good or bad