(Pre)Thesis draft

Hakim Boulahya

May 8, 2018

Contents

Contents **1**

1 Introduction **5**
1.1 Motivation 5
1.2 Related work 5

2 Data Structures **7**
2.1 Binary relations 7
2.2 Partially ordered set 8
2.3 Antichains and pseudo-antichains 8

3 Implementation **11**
3.1 Summary of objectives 11
3.2 Existing implementation 11
3.3 New implementation 11

4 Conclusion **13**
4.1 Possible extensions 13

Bibliography **15**

Todo list

cite the original papers of those problems (w/ complexity) 5
Talk about the problems, complexity and alternative (Safra vs antichain) 5
Include difference betweens BDDs and Antichains 5
How are antichains implemented in Acacia+ ? Is it AaPAL or another impl. ? 5
Discuss impl. specifics etc in chapter Implementation 6
What operations are implemented in those papers ? 6
What domain is is used in those papers ? 6
Research other possible related works 6
is this Total def correct ? 7
Include definition of least upper bound 8
Meaning of — vs . vs : in set definition ? 8
Includes limitation of Java built-in and different possible solution for antichains found on stack overflow and others 11
Referring to [Hoe] 11
Is actually [Hoe] what we what to do, and if not, what will be the differences ? 11
Fill-in bib correctly! 13

Chapter 1

Introduction

1.1 Motivation

Automata theory is used in various field in computer science and has shown to be an interesting way to resolve important problems, such as synthesis of computer systems or the universality problem. It has been proved that those problems are PSPACE-complete [DWRLH06].

cite the original papers of those problems (w/ complexity)

More efficient algorithms to resolve those problems have been implemented using antichain based-algorithms. Antichains are data structures that allow to represent a partially ordered set, in a more compact way.

Talk about the problems, complexity and alternative (Safra vs antichain)

The goal of this thesis is to provide an efficient implementation of different data structures that allow to compactly represent partially ordered sets, specifically antichains and pseudo-antichains. The first step is to implement in Java, classes that will be provided to the `Owl` library [Sal16]. `Owl` is a LTL to deterministic automata translations tool-set written in `Java`. A second step will be to implement antichain-based algorithms using the new antichains implementation and study the performance.

1.2 Related work

Include difference betweens BDDs and Antichains

`AaPAL` is a generic library is a that was implemented in the frame of Aaron Bohy's PhD thesis [Boh14] to provide an antichain library. It is implemented in `C`.

How are antichains implemented in Acacia+ ? Is it AaPAL or another impl. ?

An implementation of antichains in `Java` have been done by De Causmaecker and De Wannemacker in [DCDW]. The algorithms to find the ninth Dedekind number uses antichains and they needed to implement a representation of antichains. To improve efficiency and performances, Hoedts in [Hoe] has extended [DCDW] antichains implementation by using bit sequence instead of tree reprensentation.

Discuss impl. specifics etc in chapter Implementation

What operations are implemented in those papers ?

What domain is is used in those papers ?

Research other possible related works

Chapter 2

Data Structures

In this section, we will provide formal definitions of the data structures that we will implement. We recall the notion of binary relations and important propreties of such relations. We then define partially ordered set, totally order set and closed set. Finally we give a formal definition for antichains and pseudo-antichains.

The definitions and examples for this section are based on [Boh14].

2.1 Binary relations

A binary relation for an arbitrary set S is a set of pair $R \subseteq S \times S$. There are five important properties: reflexitivity, transitivity, symmetry, antisymmetry and total.

A relation R on S is said to be:

- Reflexive: iff $\forall s \in S$ it holds that $(s, s) \in R$
- Transitive: iff $\forall s_1, s_2, s_3 \in S$, if $(s_1, s_2) \in R$ and $(s_2, s_3) \in R$ then it holds that $(s_1, s_3) \in R$
- Symmetric: iff $(s_1, s_2) \in R$ then $(s_2, s_1) \in R$.
- Antisymmetric: iff $(s_1, s_2) \in R$ and $(s_2, s_1) \in R$ then $s_1 = s_2$
- Total: iff $\forall s_1, s_2 \in S$ then $(s_1, s_2) \in R$ or $(s_2, s_1) \in R$ is this Total def correct ?

Orders A *partial order* is a binary relation that is *reflexive*, *transitive* and *antisymmetric*. We note a partial order relation by R. We note $s_1 R s_2$ to show the belonging of a binary relation to a partial order, which is equivalent to $(s_1, s_2) \in R$. A *total order* is a partial order that is *total*.

2.2 Partially ordered set

An arbitrary set S associated with a partial order $\preceq$ is called a *partially ordered set* or *poset*. It is denoted by the pair $\langle S, \preceq\rangle$.

Comparable Let $s_1, s_2 \in S$ and $\langle S, \preceq\rangle$ a poset. The two elementes s_1 and s_2 are called *comparable* if either $s_1 \preceq s_2$ or $s_2 \preceq s_1$. If neither of those two comparaisons are correct, then s_1 and s_2 are called *uncomparable*.

Bounds Let $\langle S, \preceq\rangle$ a partially ordered set. We denote the *greatest lower bound* of the two elements $s_1, s_2 \in S$ by $s_1 \sqcap s_2 \in S$. The greatest lower bound is defined as follow: $s_1 \sqcap s_2 \preceq s_1$, $s_1 \sqcap s_2 \preceq s_2$ and for all $s' \in S$ we have that if $s' \preceq s_1$ and $s' \preceq s_2$ then $s' \preceq s_1 \sqcap s_2$.

Include definition of least upper bound

Lattices A *lower semilattice* is a poset $\langle S, \preceq\rangle$ where for all pair of elements $s_1, s_2 \in S$, we have that the greatest lower bound $s_1 \sqcap s_2$ exists.

2.3 Antichains and pseudo-antichains

Closed sets

A closed set is a set $L \subseteq S$ of a lower semilattice $\langle S, \preceq\rangle$ where $\forall l \in L$ we have that $\forall s \in S$ such that $s \preceq l$, then $s \in L$.

Note that for two closed sets $L_1, L_2 \subseteq S$, we have that $L_1 \cup L_2$ and $L_1 \cap L_2$ are also closed sets, but $L_1 \setminus \text{Ł}_2$ does not result necessarily to a closed set.

Meaning of — vs . vs : in set definition ?

Maximal/minimal elements We denote by $\lceil L\rceil$ the set of maximal elements of a closed set L which correspond to $\lceil L\rceil = \{l \in L | \forall l' \in L : l \preceq l' \Rightarrow l = l'\}$. Alternatively, to represent the set of minimal elements, the noation $\lfloor L\rfloor$ is used which has the following semantic $\lfloor L\rfloor = \{l \in L | \forall l' \in L : l' \preceq l \Rightarrow l = l'\}$.

Closure A *lower closure* of a set L on S noted $\downarrow L$ is the set of all elements of S that are *smaller or equal* to an element of L i.e. $\downarrow L = \{s \in S \mid \exists l \in L \cdot s \preceq l\}$. Note that for a closed set L we have that $\downarrow L = L$.

Antichains

An antichain of a poset $\langle S, \preceq\rangle$ is a set $\alpha \subseteq S$ where all element of α are uncomparable with respect to the partial order $\preceq$. Antichains allow to represent closed set in a more compact way. For a closed set $L \subseteq S$ we can retrieve all elements of L by using the antichain $\alpha = \lceil L\rceil$. With respect to the definition of the lower closure we have that $\downarrow \alpha = L$.

Operations Let $\alpha_1, \alpha_2 \subseteq S$ two antichains and $s \in S$:

- $s \in \downarrow \alpha_1$ iff $\exists a \in \alpha_1$ s.t. $\preceq a$
- $\downarrow \alpha_1 \subseteq \downarrow \alpha_2$ iff $\forall a_1 \in \alpha_1, \exists a_2 \in \alpha_2$ s.t. $a_1 \preceq a_2$
- $\downarrow \alpha_1 \cup \downarrow \alpha_2 = \downarrow \lceil \alpha_1 \cup \alpha_2 \rceil$
- $\downarrow \alpha_1 \cap \downarrow \alpha_2 = \downarrow \lceil \alpha_1 \sqcap \alpha_2 \rceil$

Chapter 3

Implementation

Java already provide built-in implementation for `Set`.

Includes limitation of Java built-in and different possible solution for antichains found on stack overflow and others

In this thesis we are more interested in partially ordered sets as totally ordered sets are already implemented in`Java` built-in `SortedSet`.

3.1 Summary of objectives

The main focus of the thesis is to be able to provide an efficient implementation of antichains and pseudo-antichains in `Java`. The first step is to provide an interface for the different operations that can be applied to antichains. We then give a description of the implementation. Antichains provide a way to represent in a compact way partially ordered set that are closed. Pseudo-antichains are an extension of antichains and provide a compact way to represent partially ordered sets. Pseudo-antichains does not specifically require closed set.

3.2 Existing implementation

Referring to [Hoe]

Is actually [Hoe] what we what to do, and if not, what will be the differences ?

3.3 New implementation

Chapter 4

Conclusion

4.1 Possible extensions

As we mainly focus on efficiency, it could be interesting to use a `C` implementation such as `AaPAL`, and provide bindings to `Java`. We could try this method as an alternative to a pure `Java` implementation and compare performances.

Fill-in bib correctly!

Bibliography

[Boh14] Aaron Bohy. *Antichain based algorithms for the synthesis of reactive systems*. PhD thesis, Université de Mons, 2014.

[DCDW] Patrick De Causemaecker and Stefan De Wannemacker. On the number of antichains of sets in a finite universe.

[DWRLH06] Martin De Wulf, Jean-François Raskin, Doyen Laurent, and Thomas A. Henzinger. Antichains: A new algorithm for checking universality of finite automata. 2006.

[Hoe] Pieter-Jan Hoedt. Parallelizing with mpi in java to find the ninth dedekind number.

[Sal16] Sickert Salomon. Owl website `https://www7.in.tum.de/ sickert/projects/owl/`, Last released in 2016.