Implementing data structures for Partially Ordered Set

Hakim Boulahya

May 15, 2018

Contents

Todo list

1 Introduction

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 [**?**].

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.

The goal of this preparatory work is to motivate the interest that are made in the antichains data structure, and why we need an efficient implementation. It also defines the desired objectives and propose and state-of-the-art of the existing implementation.

Motivation An example that highlights the use of antichains in automata theory is the universality problem. It is the problem that for an automaton `A`, is the language of `A` equivalent to the language of all the words on the alphabet, that is, is $L(A) = \Sigma^*$. The language of `A` is universal if and only if the language of the complementary of `A` accepts no words, that is, $L(A) = \Sigma^*$ if and only if $L(\bar{A}) = \emptyset$. The only way to compute the complementary of `A` is to first determinise it, which is hard. Now if $L(\bar{A})$ is the empty set, it means that there is no set of states that for which there exists a word such that this word is accepted by $\bar{A}$. A function can be defined to check the existence of such sets. Let the function $EAcc^A(w) = X = \{P \subseteq Q, P' \subseteq Q, P' \cap \bar{F} = \emptyset | \exists w \in \Sigma^* s.t. P \xrightarrow{w} P'\}$. The goal is to show that if there exists no set of states P such that $P \cap F = \emptyset$ i.e. $X = \emptyset$, then it is shown that $L(\bar{A}) = \emptyset$, therefore A is universal. The difficulty of the problem arise at the computation of X. Checking intersection between the accepting states set and all the subsets is costly. For a subset $P \subseteq Q$, for all subsets p such that $p \subseteq P$, it is known that, computing the work w starting from p or P, will lead to the same final state. Therefore it is more interesting to compute the maximal sets $\lceil X \rceil$, which correspond to an antichain of set of states that are $\subseteq$-uncomparable.

In the antichain algorithms paper [DWRLH06], the algorithm proposed follow an idea equivalent by using game theory. The universality problem is reduce to a two-player reachability game, which can be done in polynomial time. The objective of the game is for the protagonist to establish that the automaton A is not universal. To this end, the protagonist will provide a word, a letter at a time, and find a strategy that try to show that A ends in a rejecting state. The protagonist only has a strategy to win the game if and only if A is not universal.

Objective 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,

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

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

We have to define complementary with det. computation to understand why we use set of state

Define F bar, final state of A bar

EAcc function def is not correct, how to correctly formulate it ?

EAcc, set or function ?

Costly vs expensive vs complexe vs hard etc..

Include that UP is a problem important because verif-related 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.

Related work

`AaPAL` is a generic library is a that was implemented in the frame of Aaron Bohy's PhD thesis [Boh14b] to provide an antichain library. It is implemented in `C`.

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.

Include difference betweens BDDs and Antichains

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

Discuss impl. specifics etc in chapter Implementation

What operations are implemented in those papers ?

What domain (of sets) is is used in those papers ?

Research other possible related works

Structure of the work This work is the introduction to next year thesis. Therefore the content concern only the prelimenaries, the goal is to properly define the subject and existing implementation. In Section **??**, we formally define antichains, and give examples of such data structures. In Section **??**, we cite and summarize the work that has been done by the community for antichains implementation. In 4, we propose an overview of next year work and possibilities.

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 [Boh14b].

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

2.3.1 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 $L_1 \setminus L_2$ does not result necessarily to a closed set.

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 $\lfloor L \rfloor = \{l \in L | \forall l' \in L : l' \preceq l \Rightarrow l = l'\}$.

Meaning of — vs . vs : in set definition ?

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 $\downarrow L = L$.

Antichain 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 $\downarrow \alpha = L$.

2.4 Operations on antichains

Cite original paper, FJR11 from Bohy' phd

Proposition 1. *Let $\alpha_1, \alpha_2 \subseteq S$ two antichains and $s \in S$:*

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

Give complete definition for interesection

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.

Our goal is to find a way to not keep all the closure all element of the antichain in memory, but be able to retrieve those elements or check the belonging of a closure element from the uncomparable elements of the antichain.

3.2 Existing implementation

3.2.1 AaPAL

Bohy's **A**ntichain **a**nd **P**seudo-**A**ntichain **L**ibrary [Boh14a] is an open-source `C` library for the manipulation of antichains and pseudo-antichains data structures.

Antichain representation An antichain is represented by a `struct`, containing as attributes the size of the antichain, and the incomparable elements of the antichains, as a list. The list is manipulated using the `GSList` object from the `glib` library. To allow modularity, the type of the elements is `void`.

Operations The operations implemented in `AaPAL` are described in this section. An interesting remark is that most of the complexity is given as a paramater to the functions. For example the function to compare two elements in an antichain is given as a parameter. It means that the complexity to define the domain of the antichain, must be implemented in the compare function.

Operations such as creating an antichain, adding an element to an antichain, checking emptiness or cloning an antichain are implemented in `AaPAL`. In addition to that
create add elem clone
interesection Union
compare 2 antichains
containing element closure size

3.2.2 ABD

3.2.3 Hoedt's ABD modifications

3.3 Difficulties

Referring to [Hoe]

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

What is the difference beetween domain, univers and set ?

One of the difficulties is to define the domain of the elements that the antichains should work on. In `AaPAL` the all complexity is implemented in the `compare_elements` that must be given to the library functions. It that case, all the complexity of the must be implemented by the user to define the domain of the antichains.

3.3.1 Tree vs Bitarray representations

3.4 Possible solutions

In the case of the domain specifications, a good approach could be to provide an interface/abstract class to the users, to let them provide their own implementation. In addition to that, usual domain such a natural numbers or others well known domain used, could be directly implemented in the library.

3.5 New implementation

4 Next year

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!

References

[Boh14a] Aaron Bohy. Aapal website `http://lit2.ulb.ac.be/aapal`, 2014.

[Boh14b] 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.