Integration of antichain algorithms in automata library

September 6, 2019

Hakim Boulahya

supervised by
Emmanuel Filiot and Guillermo A. Pérez

ULB

Motivations

Scenario

- A developer wants to implement automata antichain algorithms
- How to implement antichains ?
- Which tools to use for the automata representation ?

Motivations

- There exists libraries to use antichains, but not easy to integrate when using automata
- Having antichains in automata library *seems* to be needed

Ease the implementation of antichain-based algorithms for developers

Contributions

Objectives

- The goal of the thesis was to define a use case algorithm using antichains:
  Language universality check $L(A) \stackrel{?}{=} \Sigma^*$ using antichains
- Then implement it in an automata library:
  Owl: ω-automata library

At the end we should have something like:

```
owl -I myautomaton.hoa \
hoa --- complete --- universality-check --- string
```

My contributions

Content

- `poset` library, a `Java` library that provides interfaces to interact with antichains [Bou]
- Use case implementation: algorithm for checking universality of NFA using antichains [DWDHR06]
- a *User Guide* (this work) on how to integrate an algorithm in `Owl` [KMS18]

The main goal is to provide a framework to implement antichain algorithms in automata libraries

`poset` library

`poset` is a `Java` library providing interfaces for:
orders, bounds and antichains

- All functions are implemented using the `FunctionalInterface`: single method class using lambda expressions syntax
- Antichains (and closed sets) are implemented using `Set` interfaces
- Interfaces use generic types, and default implementations make use of builtin interfaces: `Set` for set of elements and `List` for vectors representation

poset functions

Orders

```
interface Order<T> extends BiPredicate<T, T>
```

- With implementations for: $\subseteq$, $\sqsubseteq$ and $\preceq$

Bounds

```
public interface Bound<T>{
  T compute(T a, T b, Order<T> order)
}
```

- With an implementation for the greatest lower bound $a \sqcap b$

Antichains

```java
interface Antichain<E> extends Set<E>
```

- With one default implementation using a `LinkedList`[1] to store the incomparable elements
- Available methods are the usual for the `Set` interface (add, remove, contains, etc.) and the union ($\cup$) and intersection ($\cap$) operations

Closed sets

- Mainly antichains where the containment $\in$ is tested against the lower closure $\downarrow\lceil L \rceil$

[1]preferred because it is more needed to add/remove elements than accessing them

poset examples

How it works

- Import the functions

```
import poset.orders.LEQ
import poset.bounds.GreatestLowerBound
import poset.AntichainList
```

- Create them

```
var leq = new LEQ() // The partial order
var glb = new GreatestLowerBound() // required to
    compute the intersection
var a1 = new AntichainList<>(leq, glb)
var a2 = new AntichainList<>(leq, glb)
```

`poset` examples

How it works

- Use it

```
a1.add(List.of(3, 1))
//  a1 ==> [[3, 1]]
a1.add(List.of(2, 2))
// a1 ==> [[3, 1], [2,
    2]]
a2.add(List.of(0, 2))
// a2 ==> [[0, 2]]
```

```
a1.union(a2)
// $ ==> [[3, 1], [2, 2]]
a2.add(List.of(2, 3))
// a2 ==> [[2, 3]]
a1.union(a2)
// $ ==> [[3, 1], [2, 3]]
a1.intersection(a2)
// $  ==> [[2, 2]]
```

`Owl` introduction

`Owl` description

`Owl` is a *"Java tool collection library for ω-words, ω-automata and linear temporal logic"*

- It can be used either as a command line interface or a library (`Java` and `C++` API)
- Its a library used mainly for ω-automata and LTL translations

How we use it

- We use `Owl` as a library to use the automata structures
- And we also use `Owl` as a CLI to run the algorithms

Integration in `Owl`

How to integrate ?

`Owl` uses a pipelines logic to execute the different modules

```
owl -I myautomaton.hoa \
hoa --- complete --- universality-check --- string
```

- The goal is to integrate the antichain-based language universality check of an automaton in `Owl`
- It provides a nice interface to integrate any kind of modules that can be chained together to produce a result

Integration in `Owl`


```
owl -I myautomaton.hoa \
hoa --- complete --- universality-check --- string
```

Decomposition of the modules

- `hoa`: Module reading the input automaton in Hanoi Omega Automata format[2]
- `complete`: built-in module to complete the input automaton
- `universality-check`: The integrated antichain algorithm
- `string`: the output module, writing `true` or `false` in this case

[2] ω-automata have the same structure as NFA so the HOA format can still be used to represent NFAs

Implementation of $L(A) \stackrel{?}{=} \Sigma^*$

- The `univerality-check` module is the implementation of the backward antichain algorithm for universality check of an automaton [DWDHR06]
- It uses `Owl` library to interact with automata structures
- It uses `poset` library to build the symbolic representation using antichains

Experimental evaluation

Comparison

- The main goal of implementating antichain algorithms is because they perform better than the usual state-of-the-art solutions for the problem
- Main comparison for $L(A) \stackrel{?}{=} \Sigma^*$: *antichain-based* vs *subset construction*

Experimental evaluation

What to compare

1. Owl (antichain) vs VCSN (subset)
   - VCSN is an automata library in C++ with Python bindings
   - It has been chosen for the comparison because it is easy to use and provide lots of functions:

```
a.determinize().complete().complement().is_useless()
```

2. Owl (antichain) vs NuSMV and CUDD (antichain)
   - NuSMV and CUDD are the tools used in the paper [DWDHR06] that introduced the antichain based algorithm for our use case

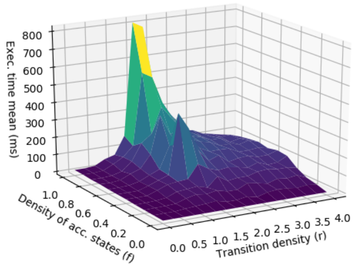
- We want to check that the Owl implementation is better that the subset construction
- and is at least as efficient as the one introduced in the original paper

Random automata generation as made in the original paper

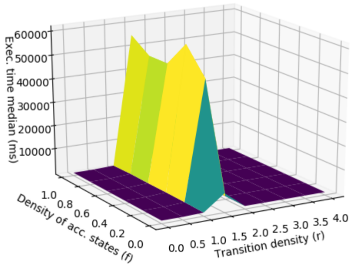
- $\Sigma = \{0, 1\}$
- r, the transition density
- $f = \frac{m}{|Q|}$, the density of accepting states

Experimental evaluation
Results

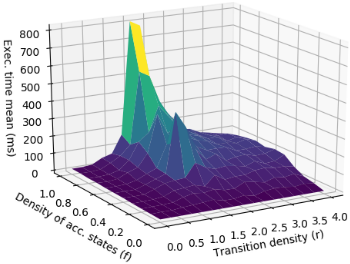
Owl



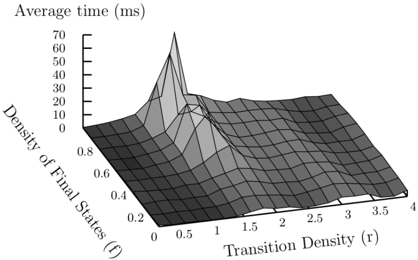
VCSN



Owl



NuSMV*



First results[3]: 10x slower... Why ?

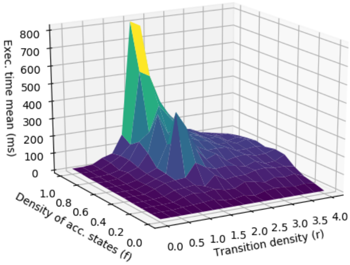
***Those results are taken from [DWDHR06] and were not rerun**

[3] Those are the results given in the thesis paper

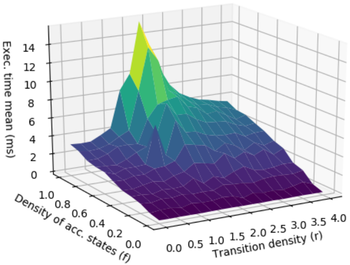
Experimental evaluation

Results

Owl

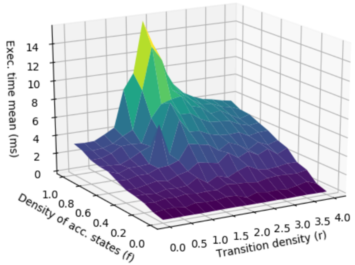


Owl with transitions stored in `HasMap`

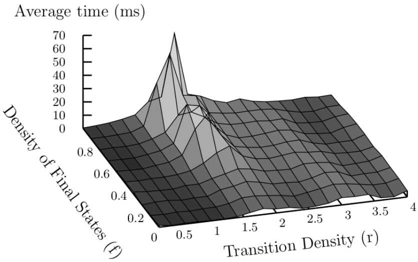


Successors and predecessors were computed on the fly

Owl


Exec. time mean (ms)
0
2
4
6
8
10
12
14
Density of acc. states (f)
1.0
0.8
0.6
0.4
0.2
0.0
Transition density (r)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

NuSMV*


Average time (ms)
70
60
50
40
30
20
10
0
Density of Final States (f)
0.8
0.6
0.4
0.2
0
Transition Density (r)
0.5
1
1.5
2
2.5
3
3.5
4

Conclusion

Last year objectives

- Provide an API and implement it against `Owl` ✓
- Provide some implementation for antichains depending on the universe of the sets ✓
- Define the algorithms to test our antichains implementation against ✓
- Study the performance of those implementations ✓

Future work

`poset` library

- Improving the default implementation of `poset` library
- Add support for closure operations

Integration

- A better support of NFA in `Owl`
- `Owl` can be more user-friendly
- Integrate antichains in `VCSN` since it is more user-friendly
- `Owl` has LTL support, so there are other possibilities of implementation and integration such as *antichain based algorithms for the synthesis of reactive systems* [Boh14]

References I

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

[Bou] Hakim Boulahya.
hqwisen / poset ů GitLab.
`https://gitlab.com/hqwisen/poset`.

[DWDHR06] M. De Wulf, L. Doyen, T. A. Henzinger, and J. F. Raskin.
Antichains: A New Algorithm for Checking Universality of Finite Automata.
In *Computer Aided Verification*, volume 4144, pages 17–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

References II

[KMS18] Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert.
Owl: A library for \omega -words, automata, and LTL.
In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, pages 543–550, 2018.

Thanks for listening