Christian Pasero, BSc

Computation of Clustered Argumentation Frameworks via Boolean Satisfiability

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

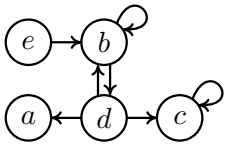
Johannes P. Wallner, Ass.Prof. Dipl.-Ing. Dr.techn. BSc.
Institute of Software Technology

Graz, March 7, 2024

Dev Notes

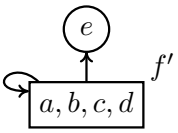
0.1 Concretizing less does not infer spuriousness

0.1.1 Concrete AF



Stable Sets: $\{d, e\}$

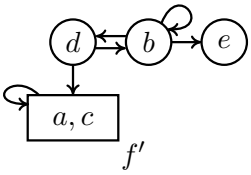
0.1.2 Abstract AF



Stable Sets: $\{f', e\}$, $\{e\}$

Abstract AF is **spurious** to concrete AF because set $\{e\}$.

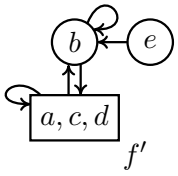
0.1.3 Concretized AF (b,d) Grounded



Stable Sets: $\{d, e\}$, $\{d, e, f'\}$

Concretized AF (b, d) is **spurious** to concrete AF because set $\{d, e, f'\}$.

0.1.4 Concretized AF (b)



Stable Sets: $\{f', e\}$

Concretized AF (e) is **faithful** to concrete AF.

Abstract

English abstract of your thesis

Kurzfassung

Deutsche Kurzfassung der Abschlussarbeit

Acknowledgements

Thanks to everyone who made this thesis possible

Contents

List of Figures

List of Tables

List of Acronyms and Symbols

1 Introduction

2 Theory

A reference to Figure 2.1, Table 2.1, and a book [Knu97].

Figure 2.1: A figure caption for the list of figures.

A	small
example	table

Table 2.1: A table caption for the list of tables.

3 Implementation

3.1 Concretizing Arguments

Concretizing a list of arguments is done iteratively by deep copying the abstract AF F' to create a new AF F'' and mutating it. The mutation is guided by five steps considering the unchanged abstract AF F' and the concrete AF F. To improve the understanding of each step, we accompany the explanation with the example depicted in 3.3, where we concretize the arguments a and b.

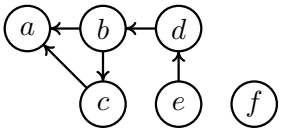


Figure 3.1: Concrete AF F

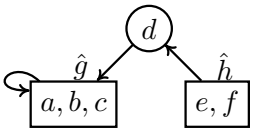


Figure 3.2: Abstract AF F'

Figure 3.3: Example: Concretization of arguments

Step 1: Each argument needing concretization is first removed from the parent cluster and added as a singleton in F''. If an argument is not part of a cluster or is invalid, we remove it and continue with the filtered valid list. We do not consider attacks in this step since they depend on the concrete- and abstract AFs.

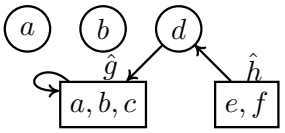


Figure 3.4: Concretized AF F'' after Step 1

Step 2: We add the new attacks from all concretized arguments to the remaining clusters. We must do this after removing the arguments from the clusters because if an argument a attacks argument b in the concrete AF, and b is part of the cluster F' in the abstract AF, by concretizing b, the attack (a, F') would not be valid anymore.

Step 3: After adding the new attacks, we need to check which attacks from F' are still valid in F''. If an attack is not valid anymore through the concretization, we remove it in

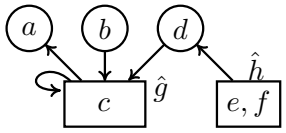


Figure 3.5: Concretized AF F'' after Step 2

F''. An attack is not valid anymore; if we remove one of the arguments being attacked or attacked by argument a from the cluster f and no other attack exists, s.t. a is attacked from/attacking an argument within f. Selfattacks of clusters could also change by the concretization of arguments. Therefore, we need to check the clusters from which the arguments are concretized.

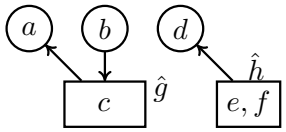


Figure 3.6: Concretized AF F'' after Step 3

Step 4: In this step we add the new attacks between the singletons. Due to the fact, that we copied all the attacks from F', we only have to take into consideration the attacks from or to the concretized singletons. So instead of iterating over all singletons of the AF, we can limit the attack creation to the concretized singletons.

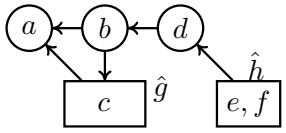


Figure 3.7: Concretized AF F'' after Step 4

Step 5: The last step is to clean up the argumentation framework F'' by removing all empty clusters and mutating the clusters with exactly one singleton to the mentioned singleton.

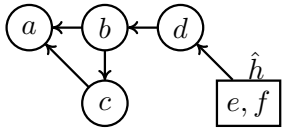
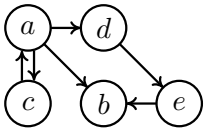


Figure 3.8: Concretized AF F'' after Step 5

4 Examples

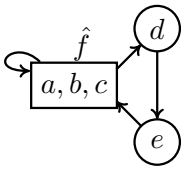
4.1 Basic AF

4.1.1 Concrete AF



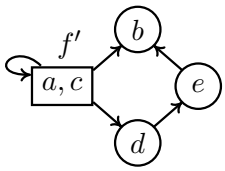
Stable Sets: $\{\}$, $\{a, e\}$, $\{b, c, d\}$

4.1.2 Abstract AF



Stable Sets: $\{\}$, $\{\hat{f}, e\}$, $\{\hat{f}, d\}$
concrete with main abstract → `FAITHFUL`

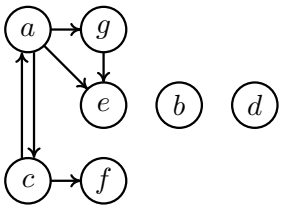
4.1.3 Abstract AF with Concretized Argument b



4.2 Basic Example

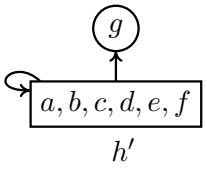
4.2.1 Concrete AF

Let $X = (ARG, ATT)$ be a concrete AF with the following arguments and attacks. Then the stable sets $ST(X)$ would be $\{\}, \{a, b, d, f\}, \{b, c, d, g\}$.



4.2.2 Abstract AF

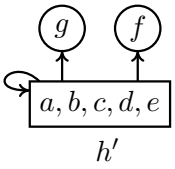
If we now abstract the concrete AF X to X', we obtain the following stable sets $\{\}, \{h'\}, \{h', g\}, \{g\}$. This would lead to a spurious abstraction, due to set $\{g\}$.



Now let X' be the input to our `CONCRETIZER` program and we parse as concretizer list the argument f.

4.2.3 Concretized Abstract AF (f)

We obtain the following AF X'' with the following stable sets $\{\}$, $\{h'\}$, $\{f, g, h'\}$, $\{g, h'\}$, $\{f, h'\}$, $\{f, g\}$. Which would lead to a spurious abstraction, due to the sets $\{h'\}$, $\{f, g, h'\}$, $\{f, g\}$.



4.2.4 Concretizing until Faithfulness

Since we want to obtain a faithful abstraction of the AF X' with the concretized argument f, we create all possible combinations of further concretization. Therefore, we need the spurious sets of X'' i.e. $\{h'\}$, $\{f, g, h'\}$, $\{f, g\}$. Since we are in the stable semantics, the depth of the concretizer search is 2 (i.e. if an argument x is spurious, we investigate all its attackers, and the attacker of the attackers and the same for the defenders (=the arguments which x attacks)).

Pre Filtering

The spurious sets of X'' can also have clusters in the sets. Since we relate to the attackers and defenders of the concrete AF X we can filter them out (because the concrete AF has no clusters). We then obtain the following sets: $\{f, g\}$, $\{f, g\}$, which can be reduced to $\{f, g\}$.

Attacker and Defender Depth 2

We now iterate over the filtered sets and check for each attacker a, the attackers of the attacker a_x. We also check, if a or a_x is in a cluster, because if they are not, we can not concretize them. Furthermore we add all the elements c from the concretizer list (if not already present) and create the following list of sets: $[\{a, c\}$, $\{a, a_0\, c\}$, $\{a, a_1\, c\}$, ..., $\{a, a_n, c\}]$ which in the current example would lead to the following list: $[\{c, f\}, \{a, c, f\}, \{a, f, g\}, \{a, c, f, g\}]$. The exact same is done with the defenders, where the list is $[\{e, f, g\}]$.

Combining Sets

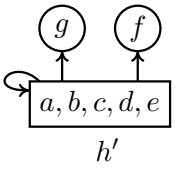
We now create each possible combination out of the 5 lists. This leads to a total of $\sum_{k=1}^{5} \binom{5}{k} = 31$ solutions. Since f.e. the combination $\{c, f\}$ and and $\{a, c, f\}$ are already covered in $\{a, c, f\}$ we remove the duplicates and obtain the following seven sets: $\{c, f\}$, $\{a, f\}$, $\{e, f\}$, $\{a, c, f\}$, $\{c, e, f\}$, $\{a, e, f\}$, $\{a, c, e, f\}$.

4.3 Problem

This approach works well for very small AF. But once we have more spurious sets, the list of the combinations 4.2.4 grows vastly. I had one instance, were 11 spurious sets led to 120 combinations which would then lead to $\sum_{k=1}^{120} \binom{120}{k}$ combinations, which is simply not feasible. Since conflict free sets produce a lot of sets, this case is not abstract and quite common.

4.4 Thoughts

If we consider the "larger" combinations first and once they result into faithfulness, we reduce the search to the selected set and try to concretize further each argument one by one. This would return a faithful solution. But I am not sure if it holds, that if the "larger" concretization AF is spurious, its fragmentation has to be spurious as well. To explain further what I mean: Let's take the previous example, where we tried to concretize the argument f.



Since this was spurious, we created the concretizer list $\{c, f\}$, $\{a, f\}$, $\{e, f\}$, $\{a, c, f\}$, $\{c, e, f\}$, $\{a, e, f\}$, $\{a, c, e, f\}$. Instead of creating the complete concretizer list (which is not feasible for a large amount of solutions as explained before) we produce a single set that contains all the unique singletons of the combinations, so in this example $\{a, c, e, f\}$. This is faithful in this case, so we focus only on this set and try to concretize its combinations. So in this case: $\{a, f\}$, $\{c, f\}$, $\{e, f\}$, $\{a, c, f\}$, $\{a, e, f\}$, $\{c, e, f\}$. If one of these is faithful, we found a better solution than the "larger" one. If all of these combinations are spurious, we just return the "larger" one.

For this approach I would simply compare each spurious solution, create one list of all the unique arguments. Instead of creating the combination list, I extend the spurious solution list with the attackers (and its attackers) and defenders (and its defenders).

Bibliography

[Knu97] Donald Ervin Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997.