www.imdea.org

Jan 14, 2019 [PADL-2019]

Incremental Evaluation of Lattice-Based Aggregates in Logic Programming Using Modular TCLP

J. Arias[1,2] M. Carro[1,2]

[1]IMDEA Software Institute, [2]Universidad Politécnica de Madrid

madrid institute for advanced studies in software development technologies

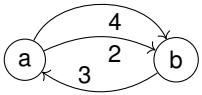
Introduction

- An aggregate function computes a single result from separate data items.
- Common aggregates include:
  - *min*: the smallest value in a set. $\min(\{2,4,7\}) = 2$
  - *set*: the set of answers returned to a query . $\text{set}(x|p(x)) = \{a,b,c\}$
  - *sum*: the sum of a collection of numbers. $\text{sum}(\langle 1,2,5\rangle) = 8$
- A naïve evaluation collects all the elements and computes the aggregate: *efficiency and termination challenged*.
- Some aggregates can be computed incrementally:
  - Increased efficiency (memory / speed).
  - Termination in more cases (model with aggregates finite while original model infinite).
- Moreover, standard LP semantics not well-suited to programs with aggregates.

Aggregates in Logic Programming

- Use case: reachability + distance in a graph with cycles has an infinite model.

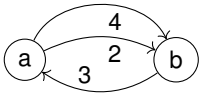
```
1 edge(a,b,4).
2 edge(a,b,2).
3 edge(b,a,3).
4 dist(X,Y,D) :- edge(X,Y,D).
5 dist(X,Y,D) :- edge(X,Z,D1), dist(Z,Y,D2), D is D1 + D2.
```



Aggregates in Logic Programming

- Use case: reachability + distance in a graph with cycles has an infinite model.

```
1  edge(a,b,4).
2  edge(a,b,2).
3  edge(b,a,3).
4  dist(X,Y,D) :- edge(X,Y,D).
5  dist(X,Y,D) :- edge(X,Z,D1), dist(Z,Y,D2), D is D1 + D2.
```



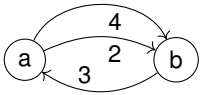
But reachability with shortest path has a finite model:

```
{ edge(a,b,4), edge(a,b,2), edge(b,a,3),
               dist(a,a,5), dist(b,b,5), dist(a,b,2), dist(b,a,3) }
```

Aggregates in Logic Programming

- Use case: reachability + distance in a graph with cycles has an infinite model.

```
1  edge(a,b,4).
2  edge(a,b,2).
3  edge(b,a,3).
4  dist(X,Y,D) :- edge(X,Y,D).
5  dist(X,Y,D) :- edge(X,Z,D1), dist(Z,Y,D2), D is D1 + D2.
```



But reachability with shortest path has a finite model:

```
{ edge(a,b,4), edge(a,b,2), edge(b,a,3),
               dist(a,a,5), dist(b,b,5), dist(a,b,2), dist(b,a,3) }
```

`dist(X,Y,D)` has been aggregated using *minimum* for `D`.

Naïve vs. Incremental Evaluation

- Compare: compute model and then aggregate results (left) with incrementally aggregating results (right).

Naïve vs. Incremental Evaluation

- Compare: compute model and then aggregate results (left) with incrementally aggregating results (right).

```
{ edge(a,b,4), edge(a,b,2),
  edge(b,a,3), dist(a,b,4),
  dist(a,b,2), dist(b,a,3) }
```

```
{ edge(a,b,4), edge(a,b,2),
  edge(b,a,3),
  dist(a,b,2), dist(b,a,3) }
```

1st iteration: `dist(a,b,4)` is not added
because `dist(a,b,2)` is their *minimum*.

Naïve vs. Incremental Evaluation

- Compare: compute model and then aggregate results (left) with incrementally aggregating results (right).

```
{ edge(a,b,4), edge(a,b,2),
  edge(b,a,3), dist(a,b,4),
  dist(a,b,2), dist(b,a,3),
  dist(a,a,7), dist(a,a,5),
  dist(b,b,7), dist(b,b,5) }
```

```
{ edge(a,b,4), edge(a,b,2),
  edge(b,a,3),
  dist(a,b,2), dist(b,a,3),
               dist(a,a,5),
               dist(b,b,5) }
```

2^{nd} iteration:

- `dist(a,a,7)` is not derived since `dist(a,b,4)` is not in the model
- `dist(b,b,7)` is not added because `dist(b,b,5)` is their *minimum*.

Naïve vs. Incremental Evaluation

- Compare: compute model and then aggregate results (left) with incrementally aggregating results (right).

```
{ edge(a,b,4), edge(a,b,2),
  edge(b,a,3), dist(a,b,4),
  dist(a,b,2), dist(b,a,3),
  dist(a,a,7), dist(a,a,5),
  dist(b,b,7), dist(b,b,5),
  dist(a,b,11), ... }
```

```
{ edge(a,b,4), edge(a,b,2),
  edge(b,a,3),
  dist(a,b,2), dist(b,a,3),
               dist(a,a,5),
               dist(b,b,5) }
```

3^{rd} iteration: incremental evaluation terminates.
Model wo. aggregates model will not, because it is infinite.

Issues with the Meaning of Aggregates

- For the program below, the model without aggregates is {`p(0)`, `p(1)`}.

```
1 p(1).
2 p(0) :- p(1).
```

- If we aggregate `p(X)` using *minimum* for `X`:

Intuitively, only the `p(X)` with the (single) smallest value for `X` should be in the model.
However, neither {`p(0)`} nor {`p(1)`} are consistent with the standard semantics.

Issues with the Meaning of Aggregates

- For the program below, the model without aggregates is {`p(0)`, `p(1)`}.

```
1 p(1).
2 p(0) :- p(1).
```

- If we aggregate `p(X)` using *minimum* for `X`:

Intuitively, only the `p(X)` with the (single) smallest value for `X` should be in the model.
However, neither {`p(0)`} nor {`p(1)`} are consistent with the standard semantics.

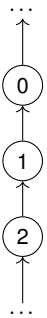
Standard least fixpoint semantics not well-suited for programs with aggregates [Kemp and Stuckey 1991; Pelov et al. 2007; Vandenbroucke et al. 2016].

A Solution Proposal within the LFP Framework

- Some relevant aggregates are consistent with the partial order of a lattice ($\sqsubseteq$).
- In these cases, literals (e.g., `p/1`) can be associated to points in the lattice.

A Solution Proposal within the LFP Framework

- Some relevant aggregates are consistent with the partial order of a lattice ($\sqsubseteq$).
- In these cases, literals (e.g., `p/1`) can be associated to points in the lattice.
- E.g., *minimum* of numbers.



$$min(a, b) = a \longleftrightarrow b \geq a$$

$$b \sqsubseteq a$$

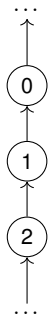
`p(1)` $\sqsubseteq$ `p(0)`

Entailment-Based Aggregates

- Simple but useful aggregates can be defined based on the $\sqsubseteq$ operation of the lattice.

  The aggregate of a multiset S under $\sqsubseteq$ is the subset of more general values of S:

$$Agg_{\sqsubseteq}(S) = \{x \in S \mid \nexists y \in S, y \neq x \cdot x \sqsubseteq y\}$$



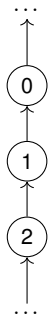
$$Agg_{min}(\{0,1,2\}) = \{0\}$$

Entailment-Based Aggregates

- Simple but useful aggregates can be defined based on the $\sqsubseteq$ operation of the lattice.

  The aggregate of a multiset S under $\sqsubseteq$ is the subset of more general values of S:

$$Agg_{\sqsubseteq}(S) = \{x \in S \mid \nexists y \in S, y \neq x \cdot x \sqsubseteq y\}$$



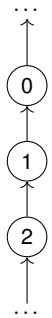
$$Agg_{min}(\{0,1,2\}) = \{0\}$$
$$Agg_{min}(\{0,5,7,\ldots\}) = \{0\}$$

Entailment-Based Aggregates

- Simple but useful aggregates can be defined based on the $\sqsubseteq$ operation of the lattice.

  The aggregate of a multiset S under $\sqsubseteq$ is the subset of more general values of S:

$$Agg_{\sqsubseteq}(S) = \{x \in S \mid \nexists y \in S, y \neq x \cdot x \sqsubseteq y\}$$



$$Agg_{min}(\{0,1,2\}) = \{0\}$$
$$Agg_{min}(\{0,5,7,\dots\}) = \{0\}$$

Intended Semantics

Different initial models may have the same aggregate. After aggregating, the link with the aggregate-less model is lost.

If $Agg_{\sqsubseteq}(S) = p(k)$ then $p(x) \longleftrightarrow x \sqsubseteq k$.

A Consistent Semantics for Lattice-Based Aggregates

- For the program below, the expected model if we aggregate `p/1` using *minimum* is {`p(0)`}.

```
1 p(1).
2 p(0) :- p(1).
```

A Consistent Semantics for Lattice-Based Aggregates

- For the program below, the expected model if we aggregate `p/1` using *minimum* is $\{$`p(0)`$\}$.

```
1 p(1).
2 p(0) :- p(1).
```

- Under our semantics:
  - the model $\{$`p(0)`$\}$ means that `p(X)` s.t. `X` ≥ 0 is true.
  - `p(1)` is true ($1 \geq 0$).
  - `p(0):- p(1)` can derive `p(0)` without contradictions.

A Consistent Semantics for Lattice-Based Aggregates

- For the program below, the expected model if we aggregate `p/1` using *minimum* is {`p(0)`}.

```
1 p(1).
2 p(0) :- p(1).
```

- Under our semantics:
  - the model {`p(0)`} means that `p(X)` s.t. `X` ≥ 0 is true.
  - `p(1)` is true ($1 \geq 0$).
  - `p(0):- p(1)` can derive `p(0)` without contradictions.

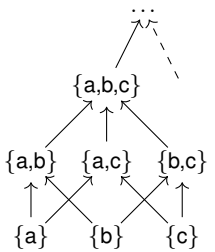
`p(1)` can be used to support `p(0)`.

Join-Based Aggregates

- More complex aggregates are defined using the join operation $\sqcup$.
- They generate new elements based on previous elements.

Join-Based Aggregates

- More complex aggregates are defined using the join operation $\sqcup$.
- They generate new elements based on previous elements.
- E.g., the union of *sets* of values.



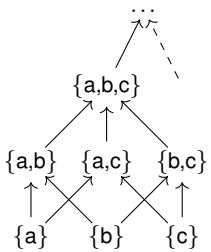
$$set(S_a, S_b) = S_c \iff S_c = S_a \cup S_b$$

$$S_a \sqcup S_b = S_c \ \wedge \ S_a \sqsubseteq S_c \ \wedge \ S_b \sqsubseteq S_c$$

$$\texttt{p([b])} \sqsubseteq \texttt{p([a,b])}$$

Join-Based Aggregates

- More complex aggregates are defined using the join operation $\sqcup$.
- They generate new elements based on previous elements.
- E.g., the union of *sets* of values.



$$set(S_a, S_b) = S_c \iff S_c = S_a \cup S_b$$

$$S_a \sqcup S_b = S_c \ \wedge \ S_a \sqsubseteq S_c \ \wedge \ S_b \sqsubseteq S_c$$

$$\texttt{p([b])} \sqsubseteq \texttt{p([a,b])}$$

Properties of $\sqcup$

Associativity, commutativity and idempotence.

State of the Art

- Tabling engines compute specific aggregates incrementally by means of:
  - Mode tabling [Guo and Gupta 2008; Zhou et al. 2010]. E.g., B-Prolog and Yap.
  - Answer subsumption [Swift and Warren 2010]. E.g., XSB.
- However, their behavior is at odds with LFP semantics:

E.g., given the program[1]:

```
1  :- table p(min).
2  p(3).
3  p(2).
4  p(1) :- p(2).
5  p(0) :- p(3).
```

[1] Example taken from [Vandenbroucke et al. 2016], where `:-table p(min)` denotes that `p(X)` should be aggregated using `min` for `X`

State of the Art

- Tabling engines compute specific aggregates incrementally by means of:
  - Mode tabling [Guo and Gupta 2008; Zhou et al. 2010]. E.g., B-Prolog and Yap.
  - Answer subsumption [Swift and Warren 2010]. E.g., XSB.
- However, their behavior is at odds with LFP semantics:

  E.g., given the program[1]:

```
1  :- table p(min).
2  p(3).
3  p(2).
4  p(1) :- p(2).
5  p(0) :- p(3).
```

  For the query `?-p(X)`

[1] Example taken from [Vandenbroucke et al. 2016], where `:-table p(min)` denotes that `p(X)` should be aggregated using `min` for `X`

State of the Art

- Tabling engines compute specific aggregates incrementally by means of:
  - Mode tabling [Guo and Gupta 2008; Zhou et al. 2010]. E.g., B-Prolog and Yap.
  - Answer subsumption [Swift and Warren 2010]. E.g., XSB.
- However, their behavior is at odds with LFP semantics:

E.g., given the program[1]:

```
1 :- table p(min).
2 p(3).
3 p(2).
4 p(1) :- p(2).
5 p(0) :- p(3).
```

For the query `?-p(X)`

Intuitively expected answer:

`X=0`

[1] Example taken from [Vandenbroucke et al. 2016], where `:-table p(min)` denotes that `p(X)` should be aggregated using `min` for `X`

State of the Art

- Tabling engines compute specific aggregates incrementally by means of:
  - Mode tabling [Guo and Gupta 2008; Zhou et al. 2010]. E.g., B-Prolog and Yap.
  - Answer subsumption [Swift and Warren 2010]. E.g., XSB.
- However, their behavior is at odds with LFP semantics:

E.g., given the program[1]:

```
1  :- table p(min).
2  p(3).
3  p(2).
4  p(1) :- p(2).
5  p(0) :- p(3).
```

For the query `?-p(X)`

Intuitively expected answer:

`X=0`

XSB and B-Prolog return:

`X=1`

Yap returns:

`X=3; X=2; X=1` for the first call.

`X=1` for subsequent calls.

[1] Example taken from [Vandenbroucke et al. 2016], where `:-table p(min)` denotes that `p(X)` should be aggregated using `min` for `X`

The ATCLP Framework

- ATCLP reformulates lattice-based aggregates inside a constraint system and computes them using Modular TCLP [Arias and Carro 2016, 2018].
- The ATCLP interface allows the programmer to define aggregates using two operations:
  - `entails(Agg,A,B)` for entailment-based aggregates. Succeeds iff $A \sqsubseteq_{Agg} B$.
  - `join(Agg,A,B,New)` for join-based aggregates: $New = A \sqcup_{Agg} B$.

The ATCLP Framework

- ATCLP reformulates lattice-based aggregates inside a constraint system and computes them using Modular TCLP [Arias and Carro 2016, 2018].
- The ATCLP interface allows the programmer to define aggregates using two operations:
  - `entails(Agg,A,B)` for entailment-based aggregates. Succeeds iff $A \sqsubseteq_{Agg} B$.
  - `join(Agg,A,B,New)` for join-based aggregates: $New = A \sqcup_{Agg} B$.

```
1  :- use_package(tclp_aggregates).
2  :- table dist(_,_,min).
3
4  entails(min,A,B) :- A >= B.
5
6  edge(a,b,...). % Graph definition
```

```
7  dist(X,Y,D) :-
8       edge(X,Z,D1),
9       dist(Z,Y,D2),
10      D is D1 + D2.
11 dist(X,Y,D) :-
12      edge(X,Y,D).
```

Implementation of `dist(_,_,min)` under ATCLP

Inside the Aggregate TCLP Interface

- User program transformed by substituting aggregated arguments with attributed variables.
- Generic TCLP interface handles aggregates by calling the user-defined `entails/3` and `join/4`.

Inside the Aggregate TCLP Interface

- User program transformed by substituting aggregated arguments with attributed variables.
- Generic TCLP interface handles aggregates by calling the user-defined `entails/3` and `join/4`.

```
1  store_projection(V, (Agg,A))                    :- get(V, (Agg,A)).
2  call_entail((_  ,_), (_  ,B))                   :- var(B),!.
3  call_entail((Agg,A), (Agg,B))                   :- entails(Agg,A,B).
4
5  answer_compare((Agg,A), (Agg,B),'=<')                 :- entails(Agg,A,B),!.
6  answer_compare((Agg,A), (Agg,B), '>')                 :- entails(Agg,B,A),!.
7  answer_compare((Agg,A), (Agg,B),'$new'((Agg,New)))    :- join(Agg,A,B,New).
8
9  apply_answer(V, (Agg,B))          :- get(V,(Agg,A)), \+ ground(A), A=B,!.
10 apply_answer(V, (Agg,B))          :- get(V,(Agg,A)), entails(Agg,A,B).
```

Generic TCLP interface for aggregates.
(Automatically injected by the compiler to connect ATCLP and TCLP.)

Non-lattice aggregates

- Some aggregates not consistent with the properties of $\sqsubseteq$ and / or $\sqcup$ can still be defined using ATCLP.
- Their execution may not conform to the intended semantics!

Non-lattice aggregates

- Some aggregates not consistent with the properties of $\sqsubseteq$ and / or $\sqcup$ can still be defined using ATCLP.
- Their execution may not conform to the intended semantics!
- E.g., *sum* can be defined as ...

```
1 entails(sum, _A, _B) :- fails.
2 join(sum, A, B, New) :- New is A + B.
```

However:

- It does not have a sound definition for entailment.
- The join operator is not idempotent.

Example I: Performance

- Programming problem presented at the ICLP 2015 LP/CP contest.
  - You have to play *n* games at least once. Some are more fun than others.
  - You have to manage your *money* to extract the most *fun* from the games.
- ATCLP aggregates *money* and *fun* using *max*.
- It does **not** evaluate states worse than other already evaluated.
- Search space reduction!

Example I: Performance

- Programming problem presented at the ICLP 2015 LP/CP contest.
  - You have to play *n* games at least once. Some are more fun than others.
  - You have to manage your *money* to extract the most *fun* from the games.
- ATCLP aggregates *money* and *fun* using *max*.
- It does **not** evaluate states worse than other already evaluated.
- Search space reduction!

	Prolog	**Tabling**	**ATCLP**
game_data_01	8062.49	14.66	**2.89**
game_data_02	$>$ 5 min.	37.59	**4.87**
game_data_03	$>$ 5 min.	1071.26	**19.61**
game_data_04	$>$ 5 min.	4883.00	**23.21**

Table: Run time (ms) comparison for *Games* with different scenarios.

Example II: Expressiveness

```
1 :- use_module(library(sets)).
2 entails(set, SetA, SetB) :- ord_subset(SetA, SetB).
3 join(set, SetA, SetB, NewSet) :- ord_union(SetA, SetB, NewSet).
```

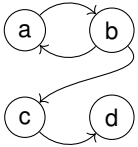
Example II: Expressiveness

```
1  :- use_module(library(sets)).
2  entails(set, SetA, SetB) :- ord_subset(SetA, SetB).
3  join(set, SetA, SetB, NewSet) :- ord_union(SetA, SetB, NewSet).
```

- Computing the set of reachable nodes from a given node in a graph under ATCLP.

```
1  :- table path(_,set).
2  path(X,[Y]) :- edge(X,Y).
3  path(X, Ys) :- edge(X,Z),
4                 path(Z,Ys).
```

```
5  edge(a,b).
6  edge(b,c).
7  edge(b,a).
8  edge(c,d).
```



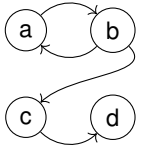
Example II: Expressiveness

```
1  :- use_module(library(sets)).
2  entails(set, SetA, SetB) :- ord_subset(SetA, SetB).
3  join(set, SetA, SetB, NewSet) :- ord_union(SetA, SetB, NewSet).
```

- Computing the set of reachable nodes from a given node in a graph under ATCLP.

```
1  :- table path(_,set).
2  path(X,[Y]) :- edge(X,Y).
3  path(X, Ys) :- edge(X,Z),
4                 path(Z,Ys).
```

```
5  edge(a,b).
6  edge(b,c).
7  edge(b,a).
8  edge(c,d).
```



`?-path(a,Nodes)` returns `Nodes=[a,b,c,d]`. an ordered list without repetitions

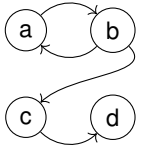
Example II: Expressiveness

```
1 :- use_module(library(sets)).
2 entails(set, SetA, SetB) :- ord_subset(SetA, SetB).
3 join(set, SetA, SetB, NewSet) :- ord_union(SetA, SetB, NewSet).
```

- Computing the set of reachable nodes from a given node in a graph under ATCLP.

```
1 :- table path(_,set).
2 path(X,[Y]) :- edge(X,Y).
3 path(X, Ys) :- edge(X,Z),
4                path(Z,Ys).
```

```
5 edge(a,b).
6 edge(b,c).
7 edge(b,a).
8 edge(c,d).
```



`?-path(a,Nodes)` returns `Nodes=[a,b,c,d]`. an ordered list without repetitions

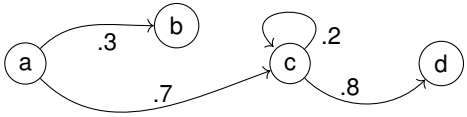
`?-path(X,[a,d])` returns `X=a` and `X=b`. Prolog/tabling can't if `setof/3` is used

Example III: Termination

- *Probability of a path in a graph:* the probability of reaching node `N` from node `a` is the *sum* of the transition probabilities of all paths from `a` to `N`.
  - Edges within loops can be traversed an unlimited number of times.
  - Pragmatic decision: discard hops whose contribution is below a certain *threshold*.

Example III: Termination

- *Probability of a path in a graph:* the probability of reaching node `N` from node `a` is the *sum* of the transition probabilities of all paths from `a` to `N`.
  - Edges within loops can be traversed an unlimited number of times.
  - Pragmatic decision: discard hops whose contribution is below a certain *threshold*.



$P_0(d) = .7 * .8 = .56$
$P_1(d) = .7 * .2 * .8 = .112$
$P_2(d) = .7 * .2 * .2 * .8 = .0224$
...
$P_n(d) = .7 * .2^n * .8 = ...$

P(b) = .3

P(d) = .7

Example III: Termination

- *Probability of a path in a graph:* the probability of reaching node `N` from node `a` is the *sum* of the transition probabilities of all paths from `a` to `N`.
  - Edges within loops can be traversed an unlimited number of times.
  - Pragmatic decision: discard hops whose contribution is below a certain *threshold*.

```
1  :- table reach(_,sum).
2  :- table path(_,_,thr(0.001)).
3
4  entails(sum,_,_) :- fails.
5  join(sum, A, B, New) :-
6      New is A + B.
7  entails(thr(Epsilon), A, B):-
8      A < Epsilon * B.
```

```
9  reach(N,P) :- path(a,N,P).
10
11 path(X,Y,P) :-
12         edge(X,Y,P).
13 path(X,Y,P) :-
14         edge(X,Z,P1),
15         path(Z,Y,P2),
16         P is P1 * P2.
```

`?- reach(d,P)` returns `P=0.699776`.

Conclusions

- ATCLP is a framework that implements Lattice-Based Aggregates under a semantics consistent with the LFP semantics / tabling.
- We validate its flexibility and expressiveness through several examples.
- Its evaluation showed a positive balance between memory needs and speed.

Conclusions

- ATCLP is a framework that implements Lattice-Based Aggregates under a semantics consistent with the LFP semantics / tabling.
- We validate its flexibility and expressiveness through several examples.
- Its evaluation showed a positive balance between memory needs and speed.

Future Work

- Increase the performance and make the ATCLP interface more user-friendly.
- Include with ATCLP a library of commonly-used aggregate functions.

https://ciao-lang.org

Conclusions

- ATCLP is a framework that implements Lattice-Based Aggregates under a semantics consistent with the LFP semantics / tabling.
- We validate its flexibility and expressiveness through several examples.
- Its evaluation showed a positive balance between memory needs and speed.

Future Work

- Increase the performance and make the ATCLP interface more user-friendly.
- Include with ATCLP a library of commonly-used aggregate functions.

https://ciao-lang.org

Other Applications

- Abstract interpretation, the join operator is used to reach the fix point.
- Stream Data Reasonning [Arias 2016].

Thanks for your attention

Bibliography I

Arias, J. (2016). Tabled CLP for Reasoning over Stream Data. In *Technical Communications of the 32nd Int'l Conference on Logic Programming (ICLP'16)*, volume 52, pages 1–8. OASIcs. Doctoral Consortium.

Arias, J. and Carro, M. (2016). Description and Evaluation of a Generic Design to Integrate CLP and Tabled Execution. In *Int'l. Symposium on Principles and Practice of Declarative Programming*, pages 10–23. ACM.

Arias, J. and Carro, M. (2018). Description, Implementation, and Evaluation of a Generic Design for Tabled CLP. *Theory and Practice of Logic Programming (to appear)*.

Guo, H.-F. and Gupta, G. (2008). Simplifying Dynamic Programming via Mode-directed Tabling. *Software: Practice and Experience*, (1):75–94.

Kemp, D. B. and Stuckey, P. J. (1991). Semantics of Logic Programs with Aggregates. In *Int'l. Simposium on Logic Programming*, pages 387–401. Citeseer.

Pelov, N., Denecker, M., and Bruynooghe, M. (2007). Well-Founded and Stable Semantics of Logic Programs with Aggregates. *Theory and Practice of Logic Programming*, (3):301–353.

Bibliography II

Swift, T. and Warren, D. S. (2010). Tabling with answer subsumption: Implementation, applications and performance. In *Logics in Artificial Intelligence*, volume 6341, pages 300–312.

Vandenbroucke, A., Pirog, M., Desouter, B., and Schrijvers, T. (2016). Tabling with Sound Answer Subsumption. *Theory and Practice of Logic Programming, 32nd Int'l. Conference on Logic Programming*, (5-6):933–949.

Zhou, N.-F., Kameya, Y., and Sato, T. (2010). Mode-Directed Tabling for Dynamic Programming, Machine Learning, and Constraint Solving. In *Int'l. Conference on Tools with Artificial Intelligence*, number 2, pages 213–218. IEEE.