A Framework for Efficient Computations of Belief Theoretic Operations

Lalintha G. Polpitiya, Kamal Premaratne, Manohar N. Murthi and Dilip Sarkar

19th International Conference on Information Fusion,
Heidelberg, Germany, 2016

Acknowledgement: This work is based on research supported by the U.S. Office of Naval Research (ONR) via grant #N00014-10-1- 0140, and the U.S. National Science Foundation (NSF) via grant #1343430.

7 July 2016

Outline

Motivation

Wide applicability of Dempster-Shafer (DS) theory

- The Dempster-Shafer (DS) theory is a powerful general framework for reasoning under uncertainty.
- It has been identified as a framework for handling a wide variety of data imperfections. [Smets, 1999, Yager et al., 1994]
- As a consequence, DS theory has been transformed into an important computational tool for evidential reasoning in numerous application scenarios (e.g., expert systems). [Yager and Liu, 2008]

Motivation

Heavy computational burden it entails

- DS theory offers greater expressiveness and flexibility in evidential reasoning.
- However, these advantages come at a cost: DS theoretic (DST) operations involve an additional cost in terms of higher computational complexity.
- A major challenge for harnessing the advantages of DS theory in practice is to overcome this computational complexity, especially when working with large frames of discernment.

Previous work

for computations of belief theoretic operations

- The use of bit-strings[Thoma, 1989, Xu and Kennes, 1994] or integers[Liu, 2014] to represent focal elements.
- Approximations methods - provide lower bounds by removing some of the focal elements with or without redistributing the corresponding belief potentials [Voorbraak, 1989, Dubois and Prade, 1990, Tessem, 1993, Bauer, 1997, Harmanec, 1999].
- More sophisticated methods produce lower and upper bounds [Denœux, 2001, Haenni and Lehmann, 2002].
- Fast Möbius transform, which is analogous to the fast Fourier transform, has been developed toward efficient DST computations [Thoma, 1989, Kennes and Smets, 1990, Thoma, 1991].

What is still lacking

on computations of DST operations

- There is no widely accepted computationally feasible generalized framework to represent DST models and carry out DST operations.
- A thoughtful discussion about data structures and algorithms for efficient DST computations is still lacking.
- Current implementations, lack the ability to handle computations on larger frames.

Contributions

to fill the void between what DS theory can offer and its practical implementation

- We introduce a novel generalized computational framework where we develop three different representations — *DS-Vector*, *DS-Matrix*, and *DS-Tree* —
- Relevant data structures and algorithms for DST operations.
- Act as simple tools for visualization of DST models and the complex nature of the computations involved.
- A strategy, which we refer to as *REGAP (REcursive Generation of and Access to Propositions)*.
- We introduce an implicit index calculation mechanism to represent a focal element.
- Open source library implementation for DST operations.

Basic notions of Belief theory

Symbol	Meaning
Θ	*Frame of discernment (FoD),* i.e., the set of all possible mutually exclusive and exhaustive propositions.
θ_i	*Singletons,* i.e., the lowest level of discernible information, i.e., $\Theta = \{\theta_0, \ldots, \theta_{n-1}\}$, here $n = \lvert\Theta\rvert$. For computational ease, we start the indexing from 0.
$\overline{A}$	*Complement* of the proposition $A \subseteq \Theta$, i.e., those singletons that are not in A.
$m(\cdot)$	*Basic belief assignment (BBA)* or *mass assignment* $m : 2^{\Theta} \mapsto [0, 1]$ where $\sum_{A \subseteq \Theta} m(A) = 1$ and $m(\emptyset) = 0$.
Focal element	Singleton or composite (i.e., non-singleton) proposition that receives a non-zero mass.
$\mathfrak{F}$	*Core,* the set of focal elements.
$\mathcal{E}$	*Body of evidence (BoE)* represented via the triplet $\{\Theta, \mathfrak{F}, m\}$.
Subset propositions of A	Subsets of A and A itself, here $m = \lvert A\rvert$

REcursive Generation of and Access to Propositions

REGAP: Starting with $\emptyset$ element

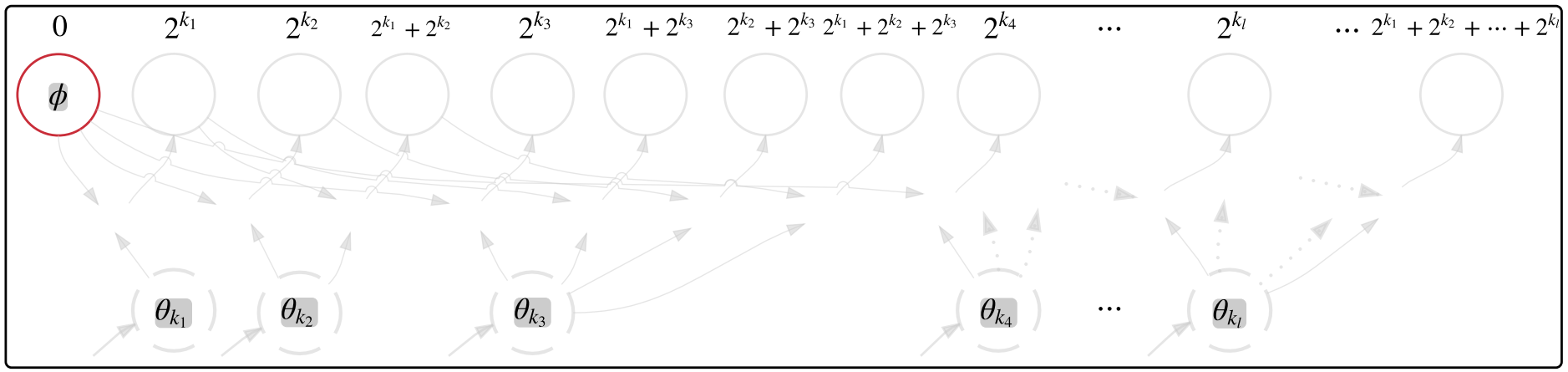

Figure: REGAP: *REcursive Generation of and Access to Propositions, Start with* $\emptyset$

- Consider the FoD $\Theta = \{\theta_0, \theta_1, \ldots, \theta_{n-1}\}$.
- Suppose we desire to determine the belief potential associated with $A = (\theta_{k_1}, \theta_{k_2}, \ldots, \theta_{k_\ell}) \subseteq \Theta$.
- The REGAP property allows us to recursively generate the propositions that are relevant for this computation: Start with $\emptyset$.

REcursive Generation of and Access to Propositions

REGAP: Inserting singleton θ_{k_1}

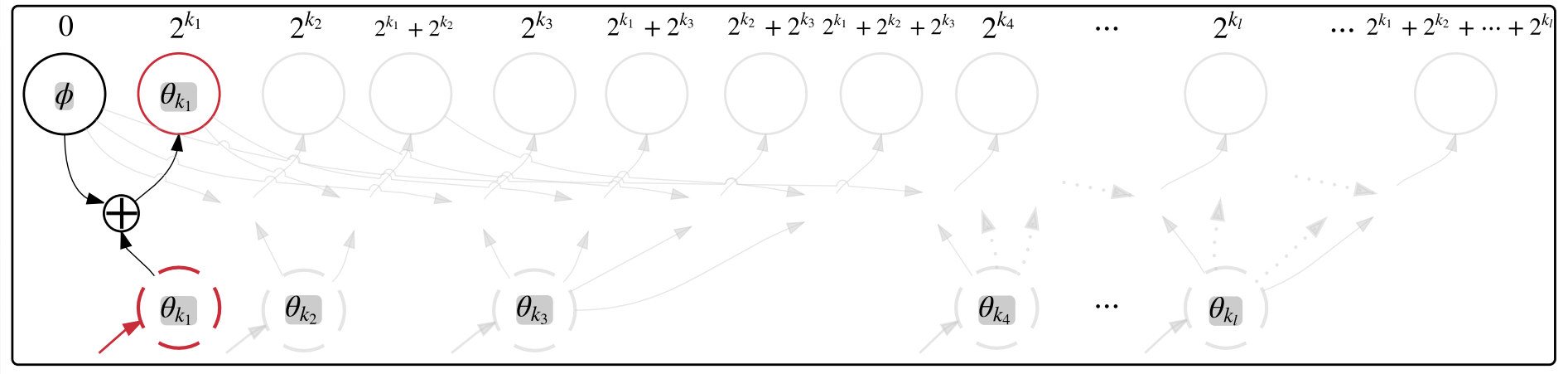

Figure: REGAP: *REcursive Generation of and Access to Propositions, inserting* θ_{k_1}

- First insert the singleton $\theta_{k_1} \in A$. Only one proposition is associated with this singleton, viz., $\emptyset \cup \theta_{k_1} = \theta_{k_1}$ itself.

REcursive Generation of and Access to Propositions

REGAP: Inserting singleton θ_{k_2}

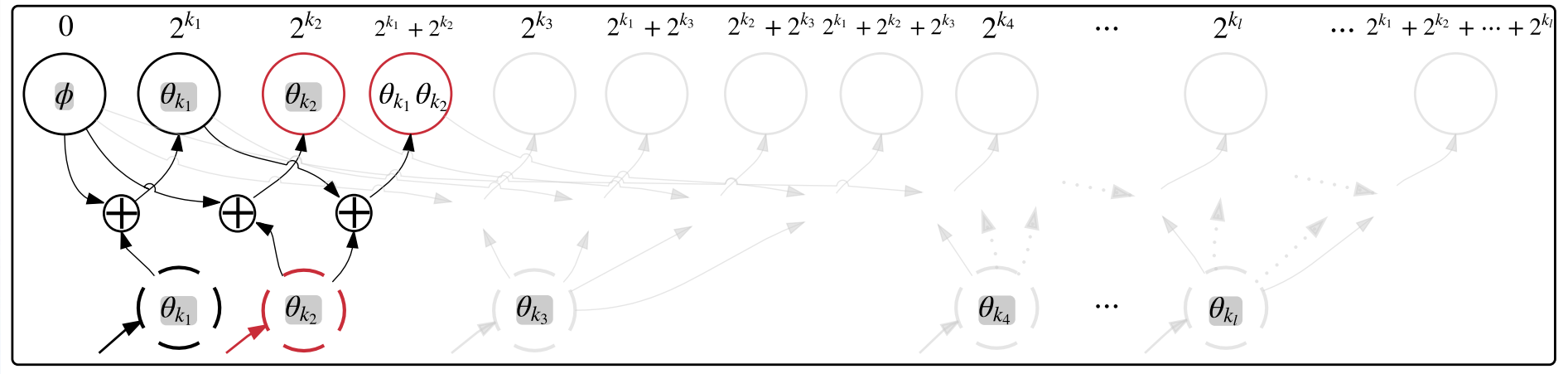

Figure: REGAP: *REcursive Generation of and Access to Propositions, inserting* θ_{k_2}

- Next insert another singleton $\theta_{k_2} \in A$. The new propositions that are associated with this singleton are $\emptyset \cup \theta_{k_2} = \theta_{k_2}$ and $\theta_{k_1} \cup \theta_{k_2} = (\theta_{k_1}, \theta_{k_2})$.

REcursive Generation of and Access to Propositions

REGAP: Inserting singleton θ_{k_3}

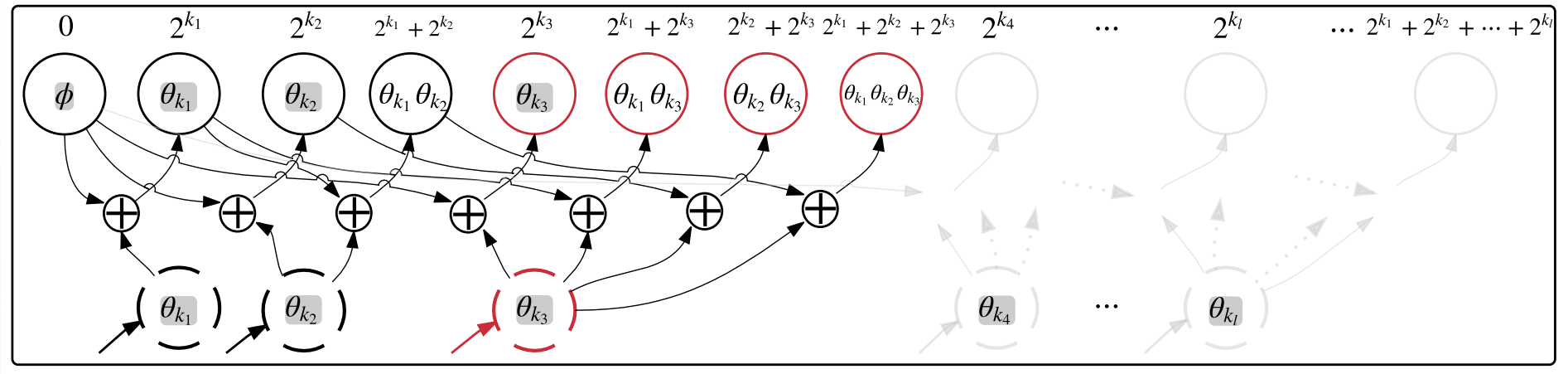

Figure: REGAP: *REcursive Generation of and Access to Propositions, inserting* θ_{k_3}

- Inserting another singleton $\theta_{k_3} \in A$ brings the new propositions $\emptyset \cup \theta_{k_3} = \theta_{k_3}$, $\theta_{k_1} \cup \theta_{k_3} = (\theta_{k_1}, \theta_{k_3})$, $\theta_{k_2} \cup \theta_{k_3} = (\theta_{k_2}, \theta_{k_3})$, and $(\theta_{k_1}, \theta_{k_2}) \cup \theta_{k_3} = (\theta_{k_1}, \theta_{k_2}, \theta_{k_3})$.
- In essence, when a new singleton is added, new propositions associated with it can be recursively generated by adding the new singleton to each existing proposition.

REcursive Generation of and Access to Propositions

REGAP: Generalized representation

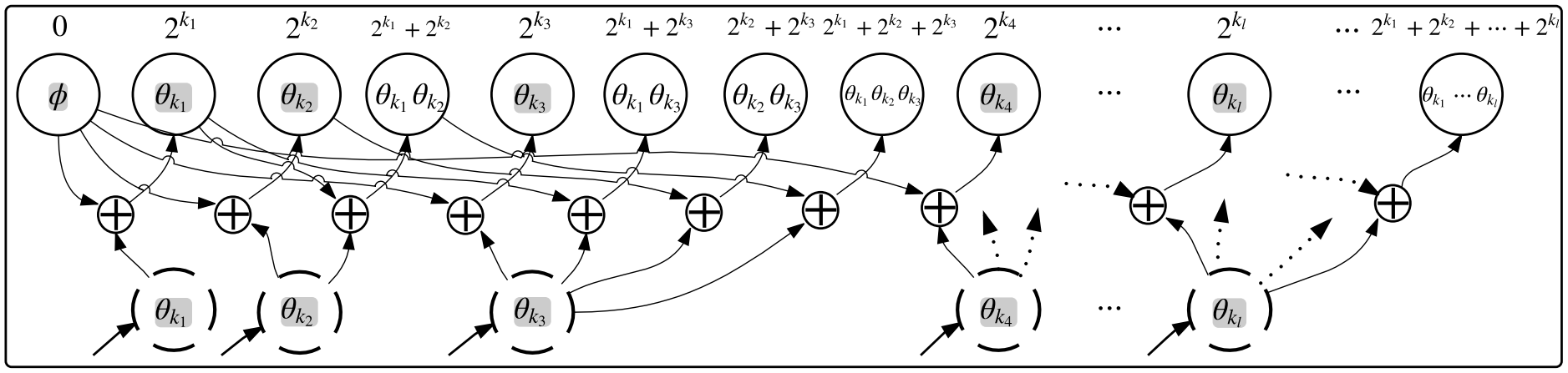

Figure: REGAP: *REcursive Generation of and Access to Propositions*

- We refer to this recursive scheme as *REGAP*, which stands for *REcursive Generation of and Access to Propositions*.
- All propositions of interest within the FoD Θ can be generated when $A = \Theta$.
- These recursively generated propositions can be formulated as a vector, a matrix or a tree, and utilized to represent a dynamic BoE.

DS-Vector: Vector representation of a dynamic BoE

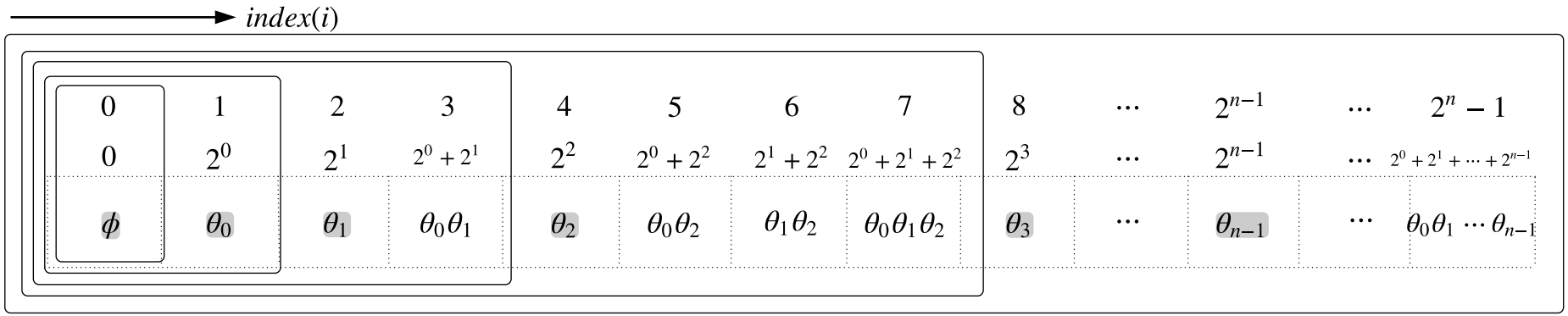

Figure: DS-Vector: Vector representation of a dynamic BoE.

- In figure, rectangles represent the recursive steps of dynamic BoE generation.
- Propositions are represented by implicit contiguous indexes. So, no memory allocation is needed to store a proposition.
- Memory allocation is needed only to store the required belief potentials (or, more generally, mass, belief, plausibility, or commonality values)

DS-Matrix: Matrix representation of a dynamic BoE

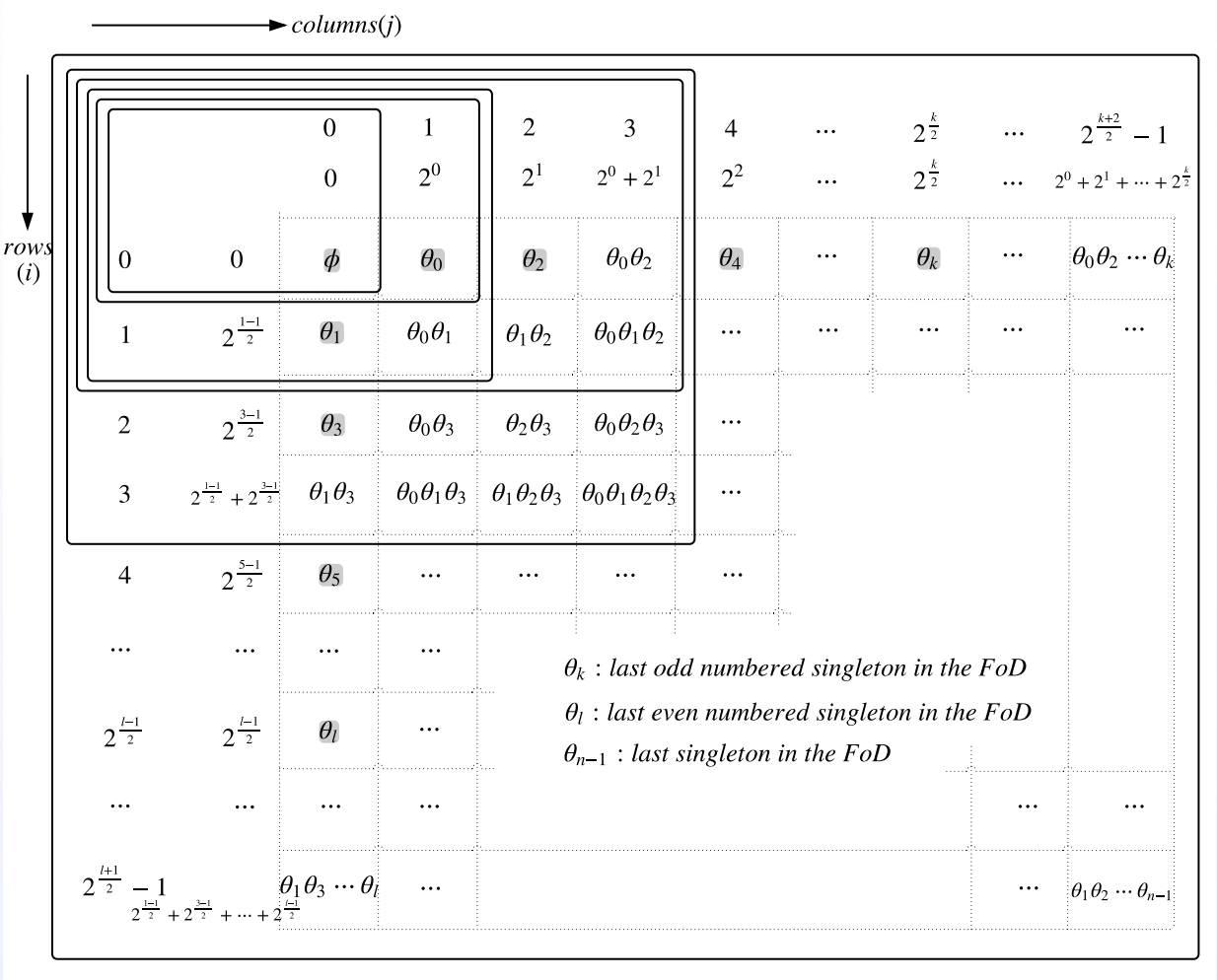

Figure: DS-Matrix: Matrix representation of a dynamic BoE.

Access a belief potential in a DS-Matrix

- Input parameters are passed as A_e and A_o; A_e includes even numbered singletons and A_o includes odd singletons of the proposition of interest.
- $Power[i]$ is a lookup table, which includes 2 to the power of indexes.
- Algorithm to access a belief potential in a DS-Matrix

```
1:  procedure AccessPotential(EvenSingletons A_e, OddSingletons A_o)
2:      row ← 0
3:      col ← 0
4:      for each θ_i in A_o do
5:          row = row + power[i]
6:      end for
7:      for each θ_i in A_e do
8:          col = col + power[i]
9:      end for
10:     Return potential[row][col]
11: end procedure
```

- Analysis and other access algorithms are in the paper.

DS-Tree: Perfectly balanced binary tree representation of a dynamic BoE

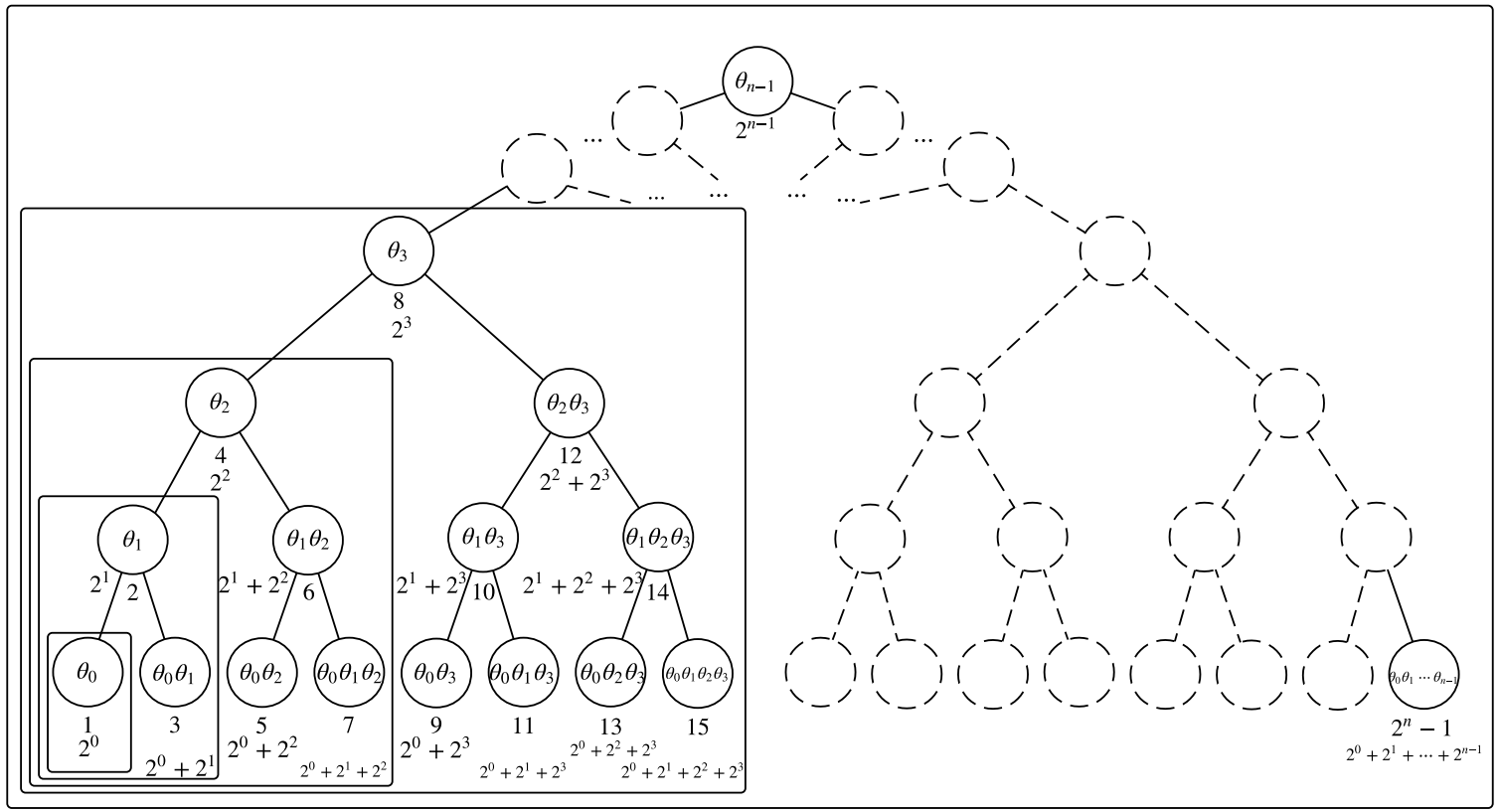

Figure: DS-Tree: Perfectly balanced binary tree representation of a dynamic BoE.

Arbitrary belief computations

from a minimum number of operations

Definition(Belief)

Given a BoE $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$, the belief assigned to $A \subseteq \Theta$ is $Bl : 2^{\Theta} \mapsto [0, 1]$ where

$$Bl(A) = \sum_{B \subseteq A} m(B).$$

■

- Shafer has stated, *"It remains to be seen how useful the fast Möbius transform will be in practice. It is clear, however, that it is not enough to make arbitrary belief function computations feasible."* [Shafer, 1990, p.348].
- Exactly for this purpose, employing REGAP, we propose a new approach to identify propositions relevant to a given belief computation.
- This technique can be used to calculate arbitrary belief, plausibility, and commonality function values from a minimum number of operations.

Arbitrary belief computations

Employing REGAP to generate relevant propositions

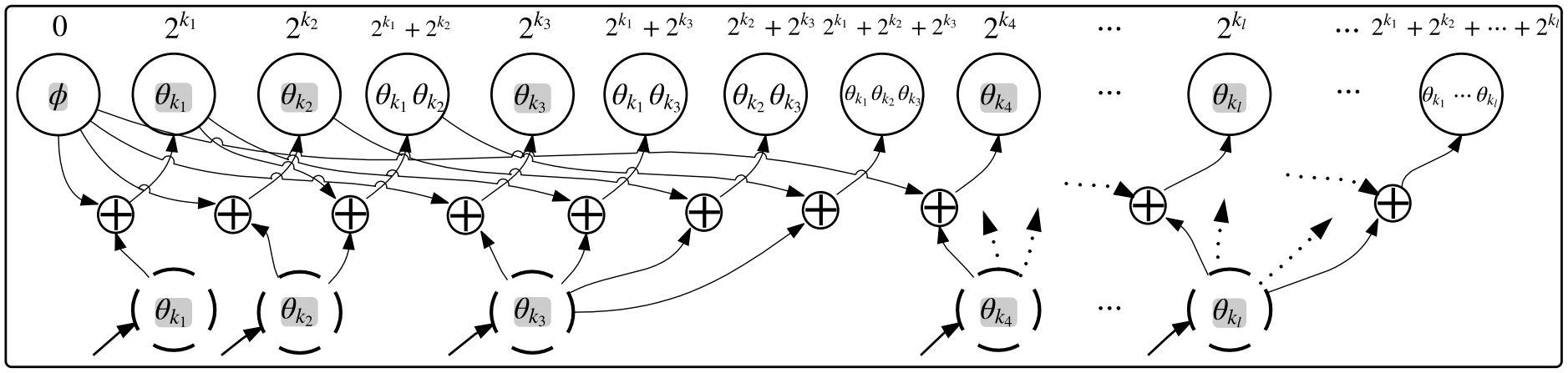

Figure: REGAP: *REcursive Generation of and Access to Propositions*

- The REGAP strategy generates required propositions relevant to the computation of $Bl(A)$, where $A = (\theta_{k_1}, \theta_{k_2}, \ldots, \theta_{k_\ell}) \subseteq \Theta$.
- Belief computation is performed by accessing only the subset propositions.
- The maximum number of subset propositions that one would have to access is about 2^m, where $m = |A|$.

Computing belief in a DS-Matrix

- Algorithms to compute belief in a DS-Matrix

```
1:  procedure ComputeBelief(SingletonCoordinates A_P, Normalize Nlz)
2:      belief ← 0
3:      count ← 0
4:      for each pair p in A_P do
5:          index[count].row ← p.row
6:          index[count].col ← p.col
7:          temp ← count
8:          count ← count + 1
9:          for j ← 0, temp − 1 do
10:             index[count].row ← index[j].row + p.row
11:             index[count].col ← index[j].col + p.col
12:             count ← count + 1
13:         end for
14:     end for
15:     for i ← 0, power[|A_P|] − 2 do
16:         belief ← belief
17:         +potential[index[i].row][index[i].col]
18:     end for
19:     Return belief / Nlz
20: end procedure
```

- Analysis and other belief computation algorithms are in the paper.

Plausibility computation

Definition(Plausibility)

Given a BoE $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$, the plausibility assigned to $A \subseteq \Theta$ is $Pl : 2^{\Theta} \mapsto [0, 1]$ where

$$Pl(A) = 1 - Bl(\overline{A}).$$

■

It is easy to see that

$$Pl(A) = \sum_{\substack{B \subseteq \Theta \\ B \cap \overline{A} \neq \emptyset}} m(B). \tag{1}$$

- Plausibility $Pl(A)$ can be computed by applying belief computation algorithm to $\overline{A}$ and using the equation, $Pl(A) = 1 - Bl(\overline{A})$.

Commonality value

Definition(Commonality)

Given a BoE $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$, the commonality function of $A \subseteq \Theta$ is $Q : 2^{\Theta} \mapsto [0, 1]$ where

$$Q(A) = \sum_{A \subseteq B \subseteq \Theta} m(B).$$

■

- Propositions relevant to commonality $Q(A)$ calculation can be generated by applying REGAP to $\overline{A}$ and adding the proposition A to all the generated propositions.
- In this way, computations can also be performed with minor modifications to belief computation algorithms.

Experiments

- We have developed a novel belief computation library using the C++ programming language.
- This includes the implementation of data structures and algorithms for carrying out DST operations in the three representations DS-Vector, DS-Matrix, and DS-Tree.
- Experiments were simulated on a Macintosh desktop computer (iMac) running Mac OS X 10.11.3, with 2.9GHz Intel Core i5 processor and 8GB of 1600MHz DDR3 RAM

Experiments

CPU time of accessing a proposition (μs)

- Average computational times for accessing arbitrary propositions are listed in table for different implementations.
- Results were obtained by executing the algorithms for 100000 randomly chosen propositions from the FoD and noting the average CPU time.
- A random set of focal elements were generated in the core for each FoD size.

FoD Size	Max. $\lvert\mathfrak{F}\rvert$	DS-Vector	DS-Matrix	List Struct.
2	3	0.379	0.393	0.465
4	15	0.400	0.412	0.510
6	63	0.410	0.454	0.739
8	255	0.443	0.449	1.541
10	1023	0.433	0.496	4.632
12	4095	0.465	0.493	16.906
14	16383	0.465	0.527	67.242
16	65535	0.495	0.517	268.443
18	262143	0.529	0.560	1124.0600
20	1048575	0.575	0.629	4609.3700

Table: CPU Time of Accessing a Proposition (μs)

Experiments

CPU time of belief Computation (μs)

- Average computational times of randomly chosen belief computations are listed in the table for different implementations.
- Results were obtained by executing the algorithms for 100000 randomly chosen propositions from the FoD and noting the average CPU time.
- A random set of focal elements were generated in the core for each FoD size.

FoD Size	Max. $\lvert\mathfrak{F}\rvert$	DS-Vector	DS-Matrix	List Struct.
2	3	0.373	0.362	0.450
4	15	0.378	0.376	0.531
6	63	0.415	0.450	0.833
8	255	0.453	0.508	1.779
10	1023	0.525	0.663	5.529
12	4095	0.655	0.923	20.757
14	16383	0.884	1.314	81.196
16	65535	1.340	2.159	325.930
18	262143	2.107	3.510	1373.110
20	1048575	3.963	6.210	5448.170

Table: CPU Time of Belief Computation (μs)

Concluding Remarks

- This research work provides a novel generalized computational framework along with data structures and efficient algorithms for DST computations.
- — *DS-Vector*, *DS-Matrix*, and *DS-Tree* — also act as simple tools for visualization of DST models and operations.
- The proposed REGAP strategy allows one to identify the exact subsets relevant to a given belief computations, and also to develop dynamic BoE representations.
- Introduce implicit index calculation mechanism is useful to improve the memory usage efficiency.
- As an outcome of this research work, a belief computation library in C++ which includes all the important operations to work with belief computations is made available. Now this library is available as a open source repository in GitHub. The url is, `https://github.com/ProFuSELab/Belief-Computation-Library`

Future research

- How to handle dynamic FoDs. Removing one singleton from a FoD removes half the propositions that need to be considered. Thus, from a computational perspective, the ability to add, remove, and change the FoD is highly important.
- Efficeint algorithms for DST conditional [Fagin and Halpern, 1991], the conditional update equation [Wickramarathne et al., 2011], and other operations associated with DST algorithms.

References I

Bauer, M. (1997).
Approximation algorithms and decision making in the dempster-shafer theory of evidencean empirical study.
Int. J. Approx. Reasoning, 17(2):217–237.

Denœux, T. (2001).
Inner and outer approximation of belief structures using a hierarchical clustering approach.
Int. J. Uncertainty, Fuzziness Knowledge-Based Syst., 9(4):437–460.

Dubois, D. and Prade, H. (1990).
Consonant approximations of belief functions.
Int. J. Approx. Reasoning, 4(5):419–449.

Fagin, R. and Halpern, J. Y. (1991).
A new approach to updating beliefs.
In Bonissone, P. P., Henrion, M., Kanal, L. N., and Lemmer, J. F., editors, *Proc. Conf. 6th Uncertainty in Artificial Intelligence (UAI)*, pages 347–374. Elsevier Science, New York.

Haenni, R. and Lehmann, N. (2002).
Resource bounded and anytime approximation of belief function computations.
Int. J. Approx. Reasoning, 31(1):103–154.

Harmanec, D. (1999).
Faithful approximations of belief functions.
In *Proc. Conf. 15th Uncertainty in Artificial Intelligence (UAI)*, pages 271–278, San Francisco, CA. Morgan Kaufmann.

References II

Kennes, R. and Smets, P. (1990).
Fast algorithms for dempster-shafer theory.
In Bouchon-Meunier, B., Yager, R. R., and Zadeh, L. A., editors, *Uncertainty in Knowledge Bases*, pages 14–23. Springer-Verlag, Berlin.

Liu, L. (2014).
A relational representation of belief functions.
In Cuzzolin, F., editor, *Belief Functions: Theory and Applications*, pages 161–170. Springer, Switzerland.

Shafer, G. (1990).
Perspectives on the theory and practice of belief functions.
Int. J. Approx. Reasoning, 4(5-6):323–362.

Smets, P. (1999).
Practical uses of belief functions.
In Laskey, K. B. and Prade, H., editors, *Proc. Conf. 15th Uncertainty in Artificial Intelligence (UAI)*, pages 612–621, San Francisco, CA. Morgan Kaufmann.

Tessem, B. (1993).
Approximations for efficient computation in the theory of evidence.
Artificial Intell., 61(2):315–329.

Thoma, H. M. (1989).
Factorization of Belief Functions.
PhD thesis, Harvard University, Cambridge, MA.

References III

Thoma, H. M. (1991).
Belief function computations.
In Goodman, I. R., Gupta, M. M., Nguyen, H. T., and Rogers, G. S., editors, *Conditional Logic in Expert Systems*, pages 269–308. North-Holland, Amsterdam.

Voorbraak, F. (1989).
A computationally efficient approximation of dempster-shafer theory.
Int. J. Man-Machine Stud., 30(5):525–536.

Wickramarathne, T. L., Premaratne, K., Murthi, M. N., Scheutz, M., Kuebler, S., and Pravia, M. (2011).
Belief theoretic methods for soft and hard data fusion.
In *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, pages 2388–2391, Prague, Czech Republic.

Xu, H. and Kennes, R. (1994).
Steps toward efficient implementation on dempster-shafer theory.
In Yager, R. R., Kacprzyk, J., and Fedrizzi, M., editors, *Advances in the Dempster-Shafer Theory of Evidence*, pages 153–174. Wiley, New York.

Yager, R. R., Kacprzyk, J., and Fedrizzi, M., editors (1994).
Advances in the Dempster-Shafer Theory of Evidence.
Wiley, New York.

Yager, R. R. and Liu, L., editors (2008).
Classic Works of the Dempster-Shafer Theory of Belief Functions.
Springer-Verlag, Heidelberg, Germany.

Questions ?

Thank you!

Access a belief potential in a DS-Vector

Algorithm 1 Access a belief potential in a DS-Vector

```
1: procedure AccessPotential(Singletons A)
2:     index ← 0
3:     for each θ_i in A do
4:         index ← index + power[i]
5:     end for
6:     Return potential[index]
7: end procedure
```

Access a belief potential in a DS-Matrix

Algorithm 2 Access a belief potential in a DS-Matrix

```
1:  procedure AccessPotential(EvenSingletons A_e, Odd-
    Singletons A_o)
2:      row ← 0
3:      col ← 0
4:      for each θ_i in A_o do
5:          row = row + power[i]
6:      end for
7:      for each θ_i in A_e do
8:          col = col + power[i]
9:      end for
10:     Return potential[row][col]
11: end procedure
```

Access a belief potential in a DS-Tree

Algorithm 3 Access a belief potential in a DS-Tree

```
1: procedure AccessPotential(FoD Θ, Singletons A, DS-Tree T)
2:     index ← 0
3:     level ← |Θ| − 1
4:     node ← T.root
5:     for each θ_i in A do
6:         index ← index + power[i]
7:     end for
8:     while index mod power[level] > 0 do
9:         if index/power[level] = 0 then
10:            node ← node.left
11:        else if index/power[level] = 1 then
12:            index ← index − power[level]
13:            if index = 0 then
14:                break the loop
15:            end if
16:            node ← node.right
17:        end if
18:        level ← level − 1
19:    end while
20:    Return node.potential
21: end procedure
```

Computing Belief in a DS-Vector

Algorithm 4 Computing Belief in a DS-Vector

```
1: procedure ComputeBelief(Singletons A, Normalize Nlz)
2:     belief ← 0
3:     count ← 0
4:     for each θ_i in A do
5:         index[count] ← power[i]
6:         temp ← count
7:         count ← count + 1
8:         for j ← 0, temp − 1 do
9:             index[count] ← index[j] + power[i]
10:            count ← count + 1
11:        end for
12:    end for
13:    for i ← 0, power[|A|] − 2 do
14:        belief ← belief + potential[index[i]]
15:    end for
16:    Return belief/Nlz
17: end procedure
```

Computing Belief in a DS-Matrix

Algorithm 5 Computing Belief in a DS-Matrix

```
1: procedure ComputeBelief(SingletonCoordinates A_P,
   Normalize Nlz)
2:     belief ← 0
3:     count ← 0
4:     for each pair p in A_P do
5:         index[count].row ← p.row
6:         index[count].col ← p.col
7:         temp ← count
8:         count ← count + 1
9:         for j ← 0, temp − 1 do
10:            index[count].row ← index[j].row + p.row
11:            index[count].col ← index[j].col + p.col
12:            count ← count + 1
13:        end for
14:    end for
15:    for i ← 0, power[|A_P|] − 2 do
16:        belief ← belief
17:        +potential[index[i].row][index[i].col]
18:    end for
19:    Return belief/Nlz
20: end procedure
```

Computing Belief in a DS-Tree

Algorithm 6 Computing Belief in a DS-Tree

```
1: procedure ComputeBelief(Singletons A, DSTree T,
   Normalize Nlz)
2:     belief ← 0
3:     count ← 0
4:     for each θ_i in A do
5:         index[count] ← power[i]
6:         temp ← count
7:         count ← count + 1
8:         for j ← 0, temp − 1 do
9:             index[count] ← index[j] + power[i]
10:            count ← count + 1
11:        end for
12:    end for
13:    for i ← 0, power[|A|] − 2 do
14:        level ← |Θ| − 1
15:        leaf ← T.root
```

Computing Belief in a DS-Tree

```
16:         index ← index[i]
17:         while index mod power[level] > 0 do
18:             if index/power[level] = 0 then
19:                 leaf ← leaf.left
20:             else if index/power[level] = 1 then
21:                 index ← index − power[level]
22:                 if index = 0 then
23:                     break the loop
24:                 end if
25:                 leaf ← leaf.right
26:             end if
27:             level ← level − 1
28:         end while
29:         belief ← belief + leaf.mass
30:     end for
31:     Return belief/Nlz
32: end procedure
```