Algorithm 3: PartialCopy($q, \mathbf{z}, m, n2c$)

Input: q: node to copy, $\mathbf{z}$: variable assignment, m: depth of copy, $n2c$: map of nodes to copy

Result: constrained copy of q in $n2c$

```
1  E = ∅
2  X and Y are the partition variables of q
3  z_p = ∃_Y z ; z_s = ∃_X z
4  for i ← 1 to n do
5      if z_p ⊨ [p_i] ∧ z_s ⊨ [s_i] then
6          p' = p_i ; s' = s_i
7          if m > 0 or [p_i] ⊭ z_p then
8              if p_i ∉ n2c then
                   PartialCopy(p_i, z_p, m − 1, n2c)
9              p' = n2c[p_i]
10         if m > 0 or [s_i] ⊭ z_s then
11             if s_i ∉ n2c then
                   PartialCopy(s_i, z_s, m − 1, n2c)
12             s' = n2c[s_i]
13         E = E ∪ [(p', s')]
14 n2c[q] = NewNode(E)
```

A PARTIAL COPY OF A PSDD

A copy of a node creates a new fold for that node and its descendants up to a specified level (Algorithm 3, lines 8,11). The elements of the copy beyond the specified level redirect to nodes of the original PSDD (line 6). Optionally, the copy can be constrained to a partial assignment for some variables. In this case, only descendants that agree with the assignment are kept in the copy (line 5) and nodes beyond the specified level may have to be copied to enforce the constraint (lines 7,10).

B PROOF OF SYNTACTIC VALIDITY OF OPERATIONS

Definition 1 (Valid PSDD node)**.** *A PSDD node q that is normalized for a vtree node v is valid if: (1) all primes p_i are valid nodes and normalized for the left child of v; (2) all subs s_i are valid nodes and normalized for the right child of v; (3) the primes are mutual exclusive: $\forall i \neq j, [p_i] \wedge [p_j] = \bot$; (4) all elements are satisfiable: $\forall i, [p_i] \wedge [s_i] \neq \bot$.*

A valid operation keeps the PSDD syntactically sound and does not alter the base of the root node.

Lemma 1 (PartialCopy($q, \mathbf{z}, m, n2c$) is valid)**.** *If the following conditions are satisfied: (1) q is valid; (2) $n2c$ is valid (this means that it only contains entries $q \rightarrow q'$ where q and q' are normalized for the same vtree, valid and $[q'] = [q] \wedge \mathbf{z}_q$, where $\mathbf{z}_q$ is the projection of the assignment $\mathbf{z}$ to the variables in the vtree of node q); (3) $\mathbf{z}$ only contains variables in the vtree of q and is satisfiable in q: $\mathbf{z} \vDash [q]$.*

Proof. Proof by induction.

Note the following preconditions hold and we use them in our proof: (1) $n2c$ includes q; (2) $n2c$ is valid.

Base case: $m = 0$ and $[q] \rightarrow \mathbf{z}$. In the base case, only one decision node is added according to $n2c$ which is q', the copy of q. Because q and q' have the same elements, q' is valid and $[q] = [q']$. Because $\mathbf{z}$ is implied by $[q]$, $[q'] = [q] \wedge \mathbf{z}_q$.

Induction step: To use the inductive assumption, we first show that the preconditions hold for the calls of PartialCopy. Because q is valid, so are its primes and subs. By induction and precondition, $n2c$ is valid. Finally, $\mathbf{z}_p$ and $\mathbf{z}_s$ only contain the relevant variables because the others are forgotten using existential quantification.

The first postcondition is satisfied because q is added to $n2c$ in line 14. In terms of the second postcondition, we consider 3 cases: (i) The entry is already in $n2c$ when PartialCopy is called, then it is valid because of the precondition. (ii) The entry is added by a recursive call of PartialCopy, then it is valid because of induction. (iii) The entry is $q \rightarrow q'$, where q' has an element (p'_i, s'_i) for every element $(p_i, s_i) \in q$, except for those that do not agree with the assignment: $p_i \wedge \mathbf{z}_p = \bot$ or $s_i \wedge \mathbf{z}_s = \bot$. p'_i and s'_i are normalized for the correct vtrees because they either are the original children, or they come from $n2c$ which is valid by the precondition and induction.

We proceed to prove the mutual exclusivity of the copied primes, the satisfiability of the copied elements and the correctness of the base of the copied decision node.

The primes of q' are mutually exclusive:

$$\begin{aligned}[p'_i] \wedge [p'_j] &= [p_i] \wedge \mathbf{z}_p \wedge [p_j] \wedge \mathbf{z}_p \\ &= [p_i] \wedge [p_j] \wedge \mathbf{z}_p \\ &= \bot\end{aligned}$$

All elements of q' are satisfiable because all the elements of q are satisfiable and elements that would become unsatisfied by conditioning on $\mathbf{z}$ are removed.

The base of q' is the base of q constraint by $\mathbf{z}$:

$$[q'] = \bigvee_{i \in q : \mathbf{z}_p \vDash [p_i] \wedge \mathbf{z}_s \vDash [s_i]} [p'_i] \wedge [s'_i]$$

$$= \bigvee_{i \in q} [p_i] \wedge \mathbf{z}_p \wedge [s_i] \wedge \mathbf{z}_s$$
$$= \mathbf{z} \wedge \bigvee_{i \in q} [p_i] \wedge [s_i]$$
$$= \mathbf{z} \wedge [q]$$

□

Proposition 4 (Split(q, i, Zs, m) is valid)**.** *If the following conditions are satisfied: (1) q is valid. (2) All $\mathbf{z} \in Zs$ only contain variables of the left children of q's vtree and are satisfiable in the ith element of q: $\mathbf{z} \vDash [p_i] \wedge [s_i]$. (3) All $\mathbf{z} \in Zs$ are mutually exclusive and exhaustive.*

Proof. Note that the following postconditions hold and we use them in our proof: (1) q is valid; (2) the base of q is not altered: $[q] = [q_{\text{old}}]$.

The primes and subs of q are normalized for the correct vtree because q is valid and $n2c$ is valid (Lemma 1).

The primes of q are mutually exclusive if: (i) the original primes are mutually exclusive, (ii) the new primes are mutually exclusive and (iii) every pair of an original prime and a new prime is mutually exclusive.

All the elements of q are satisfiable, because the precondition states that all the assignments must be satisfiable in the split element.

The original base of q is $[q_{\text{old}}] = \bigvee_j [p_j] \wedge [s_j]$. After the split, the base is:

$$[q] = \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \bigvee_{\mathbf{z} \in Zs} [p_{i,\mathbf{z}}] \wedge [s_i]$$
$$= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \bigvee_{\mathbf{z} \in Zs} [p_i] \wedge \mathbf{z} \wedge [s_i]$$
$$= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \Big([p_i] \wedge [s_i] \wedge \bigvee_{\mathbf{z} \in Zs} \mathbf{z} \Big)$$
$$= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \Big([p_i] \wedge [s_i] \Big)$$
$$= [q_{\text{old}}]$$

□

Proposition 5 (Clone(q, P, m) is valid)**.** *If the following conditions are satisfied: (1) q is valid; (2) $\forall (\pi, i) \in P$, q is either $p_{\pi,i}$ or $s_{\pi,i}$.*

Proof. Note the following postconditions hold and we use them in our proof: (1) $\forall (\pi, i) \in P$, π is valid; (2) $\forall (\pi, i) \in P$, the base of π is not altered: $[\pi] = [\pi_{\text{old}}]$. Because of lemma 1 and the preconditions, q' is a valid node with the same vtree and base as q. Redirecting the parents to this node therefore keeps the parents valid and also remain the base as unaltered. □

C IMPLEMENTATION DETAILS

We discuss implementation details of LEARNPSDD.

Data In The Nodes The training data is explicitly kept in the PSDD nodes during learning. Every node contains a bitset that indicates which examples agree with the context of that node. This speeds up parameter estimation and log-likelihood calculations, which are needed for every execution and simulation of an operation. For simulation of an operation, a bitmask is used to represent the examples that are moved to a copy.

Unique Node Cache To avoid duplicate calculations when doing inference, the PSDD should not have duplicate nodes. This is accomplished using the unique-node technique, where a cache of the nodes is kept and it is checked every time before creating a new node (Meinel and Theobald, 2012). In general, two nodes are considered equal if they have the same (p, s, θ) elements. During learning, however, we adapt this by considering two nodes different if they might evolve to a different structure, based on the training data that it contains. There are two reasons for a node not to change. First, if the node's base is a complete assignment, i.e. if all descendants of this node have only one element, then there are possible LEARNPSDD operations. A clone would be useless in this case because all the parameters would remain as 1. Second, if the node contains no data. Such a node cannot contribute to the log-likelihood and has therefore no reason to change.

The number of added nodes is no longer a local characteristic of an operation, as it depends on the nodes available in the cache. To cope with this, we consider nodes that can be cached as free nodes: they are not counted in the score. This makes sense because if the node is already in the cache, it does not need to be added, otherwise adding it to the cache can make subsequent operations less expensive to simulate or execute.

SDDs In The Nodes SDDs are kept in the nodes to represent their base. This is not really needed, because the base is implied by the structure of the PSDD. However, during structure learning, PSDDs grow bigger, while SDDs do not. Therefore, if the base needs to be checked, doing this on the SDD is more efficient. Note that before any structure learning is done, the SDD is larger than the PSDD because SDD's primes need to be exhaustive and therefore the SDD may have elements for subs that represent false. However, PSDDs are expected to grow larger than the corresponding SDDs during structure learning.