Empowering Domain Recursion in Symmetric Weighted First-Order Model Counting

12th January 2022

1 Basic Definitions

Things I might need to explain.

- notation: Im
- atom
- inequality constraint
- Vars, $\mathrm{Vars}(c) = \mathrm{Vars}(P) \cup \mathrm{Vars}(N) \cup \mathrm{Vars}(C)$
- Doms on both formulas and clauses. $\mathrm{Doms}(c) = \mathrm{Im}\,\delta_c$, and $\mathrm{Doms}(\phi) = \bigcup_{c \in \phi} \mathrm{Doms}(c)$.
- the hash codes of clauses and formulas. Introduce the # notation.
- substitution
- (strict) equality of clauses. It's important to mention that we check the number of variables but not their domains.
- size of a domain, how each domain is partitioned into two, and how we iterate over all possible integer partitions of length two.
- maybe: notation for partial function, notation for powerset
- notation for projection (or avoid it?)
- WMC

Let $\mathscr{V}$ be the set of circuit nodes.
TODO: merge κ and ι into one.
TODO: maybe π and κ are global enough to have more unique names.

Definition 1. A *domain* is a set with elements not used anywhere else.[1] Let $\mathscr{D}$ be the set of all domains (note that this set expands during compilation).

We now define two partial maps π and κ with the same domain $\mathrm{dom}(\pi) = \mathrm{dom}(\kappa) \subset \mathscr{D}$. First, let $\pi\colon \mathscr{D} \nrightarrow \mathscr{D}$ be a partial endomorphism on $\mathscr{D}$ that denotes the *parent* relation, i.e., if $\pi(d) = e$ for some $d, e \in \mathscr{D}$, then we call e the parent (domain) of d, and e a child of d. Intuitively, π arranges all domains into a forest—thus, we often use graph theoretical terminology to describe properties of and relationships between domains. Second, let $\kappa\colon \mathscr{D} \nrightarrow \mathscr{V}$ be a partial map that assigns a *cause node* to all non-root domains. Third, let $\iota\colon \mathscr{D} \nrightarrow \{\,0, 1\,\}$ unambiguously order the children of any internal node, i.e., $\iota(d) \neq \iota(e)$ whenever $\pi(d) = \pi(e)$ for any $d, e \in \mathscr{D}$.[2]

[1] In the context of functions, the domain of a function f retains its usual meaning and is denoted $\mathrm{dom}(f)$.
[2] Here, each internal node has at most two children.

Algorithm 1: A recursive function for checking whether one can reuse the circuit for computing WMC(ψ) to compute WMC(ϕ). Both ϕ and ψ are formulas, and $\rho\colon \mathrm{Doms}(\phi) \nrightarrow \mathrm{Doms}(\psi) \times 2^{\mathscr{V} \times \{\,0,1\,\}}$ is a partial map. TODO: explain more about ρ.

Function `identifyRecursion`(ϕ, ψ, $\rho = \varnothing$)**:**
- **if** $\phi = \psi = \varnothing$ **then return** ρ;
- **if** $\mid \phi \mid \neq \mid \psi \mid$ **or** $\#\phi \neq \#\psi$ **then return** `null`;
- **foreach** *clause* $c \in \phi$ **do**
  - **foreach** *clause* $d \in \psi$ *such that* $\#d = \#c$ **do**
    - **foreach** *bijection* $\beta\colon \mathrm{Vars}(c) \to \mathrm{Vars}(d)$ **do**
      - suitable $\leftarrow$ `true`;
      - **if** $\forall\, v \in \mathrm{Vars}(c).(\delta_c(v) \not\in \mathrm{dom}(\rho) \vee \pi_1(\rho(\delta_c(v))) = \delta_d(\beta(v)))$ **and** $c\beta = d$ **then**
        - $\rho' \leftarrow \rho$;
        - **foreach** $v \in \mathrm{Vars}(c)$ **do**
          - $H \leftarrow$ `findHistory`($\delta_c(v)$, $\delta_d(\beta(v))$);
          - **if** $H =$ `null` **then**
            - suitable $\leftarrow$ `false`;
            - break;
          - $\rho' \leftarrow \rho \cup \{\, \delta_c(v) \mapsto (\delta_d(\beta(v)), H)\,\}$;
        - **if** suitable **then**
          - $\rho'' \leftarrow$ `identifyRecursion`($\phi \setminus \{\, c \,\}$, $\psi \setminus \{\, d \,\}$, ρ');
          - **if** $\rho'' \neq$ `null` **then return** ρ'';
  - **return** `null`;

Function `findHistory`(c, d)**:**
- $H \leftarrow \varnothing$;
- **while** $d \neq c$ **and** $d \in \mathrm{dom}(\pi)$ **do**
  - $H \leftarrow H \cup \{\, \kappa(d) \,\}$;
  - $d \leftarrow \pi(d)$;
- **if** $d = c$ **then return** H;
- **return** `null`;

Definition 2. A *clause* is a triple $c = (P, N, C, \delta_c)$, where P and N are sets of atoms interpreted as positive and negative literals respectively, C is a set of inequality constraints, and $\delta_c\colon \mathrm{Vars}(c) \to \mathscr{D}$ is a function that maps all variables in c to their domains. Two clauses c and $d = (P', N', C', \delta_d)$ are *equivalent* (written $c \equiv d$) if there is a bijection $\beta\colon \mathrm{Vars}(c) \to \mathrm{Vars}(d)$ such that $c\beta = d\beta$. TODO: we will always use this subscript notation for the δ's.

A *formula* is a set of clauses.

2 Identifying Possibilities for Recursion

TODO: explain what the second return statement is about and why a third one is not necessary.

TODO: mention which one is the main function, what each function takes and returns.

Example 1. TODO: explain how the algorithm establishes a recursive relationship from

$$\forall\, X \in a'.\forall\, Y \in b^{\perp}.\forall\, Z \in b^{\perp}.Z \neq Y \Rightarrow \neg p(X, Y) \vee \neg p(X, Z)$$
$$\forall\, X \in a'.\forall\, Y \in b^{\perp}.\forall\, Z \in a'.X \neq Z \Rightarrow \neg p(X, Y) \vee \neg p(Z, Y)$$

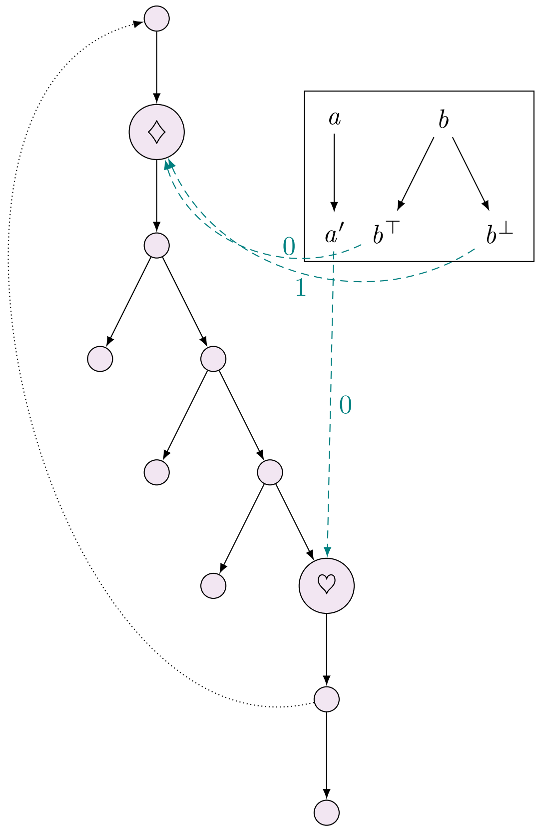

Figure 1: A circuit (outside the box) and a forest of domains (inside the box). The dotted line on the left will be added if `identifyRecursion` returns a non-`null` mapping. The dashed edges with their labels represent the κ function. The circuit nodes with symbols in them are the nodes that introduce new (sub)domains.

to

$$\forall X \in a. \forall Y \in b. \forall Z \in b. Y \neq Z \Rightarrow \neg p(X, Y) \vee \neg p(X, Z)$$
$$\forall X \in a. \forall Y \in b. \forall Z \in a. X \neq Z \Rightarrow \neg p(X, Y) \vee \neg p(Z, Y)$$

and mention Fig. 1. Solution map (stored as the edge label):

$$\rho(a') = (a, \{ (\diamondsuit, 0) \})$$
$$\rho(b^{\perp}) = (b, \{ (\heartsuit, 1) \})$$

TODO: conclude with a description of the inference rule and the node/edge type.

3 New Node Types

3.1 Improved Domain Recursion

The original version of domain recursion is here [1].

3.2 Constraint Removal

TODO: improved domain recursion (how it's different from the earlier version) and constraint removal.

4 Other Topics

- domains, smoothing, and avoiding infinite cycles
- new rules that don't create nodes

5 Circuit Evaluation

TODO: new node types, their algebraic/graphical representation, what info they hold, and how they're created.

TODO: describe evaluation of: and, counting, constraint removal, ref, unit, contradiction, improved domain recursion. Most of this will be from [2].

References

[1] Van den Broeck, G. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain* (2011), J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., pp. 1386–1394.

[2] Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., and De Raedt, L. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011* (2011), T. Walsh, Ed., IJCAI/AAAI, pp. 2178–2185.