Shapes/Slotted E-graphs

Definition slotted e-graph

Similar to (regular) egraphs:

- function symbols f, g
- e-class ids a, b
- slots $s_1, s_2, \ldots$
- slotmap $m ::= [s_j \mapsto s_k, \ldots]$ bijection
- invocation $i ::= m * a$
- terms $t ::= f \mid f(t_1, \ldots, t_k) \mid s_j \mid \lambda s_j.t$
- e-nodes $n ::= f \mid f(i_1, \ldots, i_k) \mid s_j \mid \lambda s_j.i$
- e-classes $c ::= \{n_1, \ldots n_m\} :: \{s_{i_1}, \ldots, s_{i_k}\}$

We have a mapping $\text{Classes} : \text{Id} \to \text{Eclass}$ to interpret the invocations.

Additional definitions

slots(__)

We define a family of (overloaded) functions slots for e-class ids, invocations, terms and e-nodes to a set of slots $\{s_{i_1}, \ldots, s_{i_k}\}$.

slots(__) for E-Class Id and Invocations

- $\text{slots}(a) := \{s_1, \ldots, s_m\}$, given $\text{Classes}(a) = \{n_1, \ldots, n_k\} :: \{s_1, \ldots, s_m\}$
- $\text{slots}(m * a) := m \circ \text{slots}(a) = \{m(s_j) \mid s_j \in \text{slots(a)}\}$

slots(__) for Terms and E-Nodes Let $x, x_1, \ldots$ be either terms t or invocations i.

- $\text{slots}(f) := \emptyset$
- $\text{slots}(f(x_1, \ldots, x_k)) := \text{slots}(x_1) \cup \ldots \cup \text{slots}(x_k)$
- $\text{slots}(s_j) := \{s_j\}$
- $\text{slots}(\lambda s_j.x) := \text{slots}(x) \setminus \{s_j\}$

slots(t) on terms corresponds to the set of free variables.

The Action m * __

We define a family of (overloaded) functions $m * \_$ for e-class ids, invocations, terms and e-nodes.

Generally, $m * x$ is only defined, if $\text{slots}(x) \subseteq \text{dom}(m)$.

- For any x we have $m * (m' * x) = (m' * m) * x$
- with $m * m' := m \circ m' = \{x \mapsto z \mid x \mapsto y \in m', y \mapsto z \in m\}$

m * _ for E-Class Ids and Invocations

- $m * a$ is just $m * a$, there is no way to simplify it
- For Invocations $i = m * a$, we define $m' * i := (m' * m) * a$

m * _ for Terms and E-Nodes Let $x, x_1, \ldots$ be either terms t or invocations i.

- $m * f := f$
- $m * f(x_1, \ldots, x_k) := f(m * x_1, \ldots, m * x_k)$
- $m * s_j := m(s_j)$
- $m * (\lambda s_j.x) := \lambda s_j.(m * x)$, assuming s_j is neither in the domain nor codomain of m.

We follow the Barendregt convention: We assume that all bound slots are never colliding with anything else. And if they do, we just rename them.

(Note to future self: We also need the Barendregt convention for redundant slots)

We claim that this definition implies $\text{slots}(m * x) = m \circ \text{slots}(x)$ for all x.

Examples:

- $\text{slots}(\lambda s_1.f(s_1, s_2, s_3)) = \text{slots}(f(s_1, s_2, s_3)) \setminus \{s_1\} = \{s_2, s_3\}$
- $\lambda s_1.f(s_1, s_2, s_3) * (s_1 \mapsto s_2, s_2 \mapsto s_3, s_3 \mapsto s_1)$ does not typecheck
- $\lambda s_1.f(s_1, s_2, s_3) * (s_2 \mapsto s_3, s_3 \mapsto s_1) = \lambda s_1.f(s_1, s_2, s_3)$ needs freshness
- $[s_2 \mapsto s_3, s_{47} \mapsto s_2] * a = [s_47 \mapsto s_2, s_2 \mapsto s_3] * a; \text{slots}(a) = \{s_2, s_47\}$

Containment

We define an element relation $\in \subseteq \text{Enodes} \times \text{Invocations}$, defined recursively as:

- If an E-node n is contained in an e-class $\text{Classes}(a)$ (set containment), then $n \in a * id_{\text{slots}(a)}$, where $id_{\text{slots}(a)}(s_j) = s_j$ is the identity slotmap on the set of slots $\text{slots}(a)$.
- If $n \in i$, then $m * n \in m * i$.

Notes

- We need to be more precise about the slots of e-nodes and e-classes (union/intersection, etc)
- We allow the shortcut for ordered (instead of named) arguments, $[s_j, s_{j'}, s_{j''}, ...] * i := [s_k \mapsto s_j, s_{k'} \mapsto s_{j'}, s_{k''} \mapsto s_{j''}, ...] * i$, assuming $slots(i) = \{s_k, s_{k'}, s_{k''}, ...\}$ and $k < k' < k'' < ....$

Group Action

Let $G \leq \mathrm{Sym}(s_1, \ldots, s_m)$, then G acts on the set of enodes $n[s_1, \ldots, s_m]$ the obvious way $gn[s_1, \ldots, s_m] = n[gs_1, \ldots, gs_m]$.

Similarly, $G \subseteq \mathrm{Sym}(s_1, \ldots, s_m)$ acts on the e-class $c[s_1, \ldots, s_m] = \{n_1, \ldots, n_k\}$ element-wise, i.e. $gc = \{gn_1, \ldots, gn_k\}$ (Note: this doesn't type check, we need to consider the appropriate restrictions of the groups acting on the different sets of slots).

The automorphism group $\mathrm{Aut}(c)$ of an e-class is the largest subgroup $\mathrm{Aut}(c) \leq \mathrm{Sym}(s_1, \ldots, s_m)$, such that $gc = c$ for all $g \in \mathrm{Aut}(c)$.

Strong shape computation

- enodes must be hashable
- hash must be invariant of renamings
- weak shape: canonical naming $s_1, s_2, \ldots$
- egraph idea: congruence, i.e. if $a = b \Rightarrow f(a) = f(b)$ (in memory). This does not work in weak shapes:
- example: $\mathrm{Classes}(a) = \{(fs_1 0 s_1 1), (fs_1 1 s_1 0)\} :: \{s_1 0, s_1 1\}$ enodes: $(+[s_1 0 \to s_1, s_1 1 \to s_2] * a[s_1 0 \to s_2, s_1 1 \to s_1] * a)$ and $(+[s_1 0 \to s_1, s_1 1 \to s_2] * a[s_1 0 \to s_1, s_1 1 \to s_2] * a)$ concrete terms correspond to \$ (+ (f x y) (f y x))\$ and $(+(fxy)(fxy))$
- strong shape: lex-min of all equivalent weak shapes.
- Conjecture: this is the double coset contstructive orbit problem

Open questions

- Is λ above the most generic possible, or are there examples of languages where the binders cannot be expressed this way?