1 How do we check entailments?

1. Create set of "axioms" based on the program and the entailment to check
   (a) Datatype Declarations
      - Enumeration types: Variables, Classes, Fields
      - ADT: Path
   (b) Function Declarations
      - Declared: path-equivalence, instance-of, instantiated-by
      - Defined recursively: substitute
   (c) Calculus rules as all quantified formulas
      - Static Rules: C-Refl, C-Class, C-Subst
      - Template Rules: C-Prog
   (d) Assert entailment to be checked: $c_1, ..., c_n \vdash c \Rightarrow \neg(c_1 \wedge ... \wedge c_n \rightarrow c)$
2. Obtain Solution: Does the entailment contradict the rules?
   - If unsat: valid/correct entailment.
   - If sat: invalid entailment.

1.1 General First-Order Encoding Template

$$\texttt{Variable (...)} \tag{1}$$
$$\texttt{Field (...)} \tag{2}$$
$$\texttt{Class (...)} \tag{3}$$
$$\texttt{Path ((Variable) (Path Field))} \tag{4}$$
$$\equiv : \texttt{Path} \times \texttt{Path} \rightarrow \mathcal{B} \tag{5}$$
$$:: : \texttt{Path} \times \texttt{Class} \rightarrow \mathcal{B} \tag{6}$$
$$.\mathbf{cls} \equiv : \texttt{Path} \times \texttt{Class} \rightarrow \mathcal{B} \tag{7}$$
$$\__{\{\!\{\mapsto\}\!\}} : \texttt{Path} \times \texttt{Variable} \times \texttt{Path} \rightarrow \texttt{Path} \tag{8}$$
$$\forall p\!:\!\texttt{Path}.\ p \equiv p \tag{9}$$
$$\forall a\!:\!\mathcal{B}, p\!:\!\texttt{Path}, c\!:\!\texttt{Class}. \tag{10}$$
$$(a \rightarrow p.\mathbf{cls} \equiv c) \rightarrow (a \rightarrow p :: c) \tag{11}$$
$$\forall a\!:\!\mathcal{B}, p\!:\!\texttt{Path}, q\!:\!\texttt{Path}, r\!:\!\texttt{Path}, s\!:\!\texttt{Path}, x\!:\!\texttt{Variable}. \tag{12}$$
$$(a \rightarrow p_{\{\!\{x \mapsto r\}\!\}} \equiv q_{\{\!\{x \mapsto r\}\!\}} \wedge (a \rightarrow s \equiv r)) \rightarrow \tag{13}$$
$$(a \rightarrow p_{\{\!\{x \mapsto s\}\!\}} \equiv q_{\{\!\{x \mapsto s\}\!\}}) \tag{14}$$
$$\forall a\!:\!\mathcal{B}, p\!:\!\texttt{Path}, c\!:\!\texttt{Class}, r\!:\!\texttt{Path}, s\!:\!\texttt{Path}, x\!:\!\texttt{Variable}. \tag{15}$$
$$(a \rightarrow p_{\{\!\{x \mapsto r\}\!\}} :: c \wedge (a \rightarrow s \equiv r)) \rightarrow \tag{16}$$
$$(a \rightarrow p_{\{\!\{x \mapsto s\}\!\}} :: c) \tag{17}$$
$$\forall a\!:\!\mathcal{B}, p\!:\!\texttt{Path}, c\!:\!\texttt{Class}, r\!:\!\texttt{Path}, s\!:\!\texttt{Path}, x\!:\!\texttt{Variable}. \tag{18}$$
$$(a \rightarrow p_{\{\!\{x \mapsto r\}\!\}}.\mathbf{cls} \equiv c \wedge (a \rightarrow s \equiv r)) \rightarrow \tag{19}$$
$$(a \rightarrow p_{\{\!\{x \mapsto s\}\!\}}.\mathbf{cls} \equiv c) \tag{20}$$
$$\forall a\!:\!\mathcal{B}, p\!:\!\texttt{Path}.\ (a \rightarrow \bigwedge \_) \rightarrow (a \rightarrow p :: \_) \tag{21}$$

1.2 Natural Numbers Program

$$\texttt{Zero}(x.\ \epsilon) \tag{22}$$
$$\texttt{Succ}(x.\ x.p :: \texttt{Nat}) \tag{23}$$
$$\forall x.\ x :: \texttt{Zero} \Rightarrow x :: \texttt{Nat} \tag{24}$$
$$\forall x.\ x :: \texttt{Succ}, x.p :: \texttt{Nat} \Rightarrow x :: \texttt{Nat} \tag{25}$$
$$\texttt{prev}(x.\ x :: \texttt{Nat}) : [y.\ y :: \texttt{Nat}] \tag{26}$$
$$\texttt{prev}(x.\ x :: \texttt{Zero}) : [y.\ y :: \texttt{Nat}] := \textbf{new}\ \texttt{Zero}() \tag{27}$$
$$\texttt{prev}(x.\ x :: \texttt{Succ}, x.p :: \texttt{Nat}) : [y.\ y :: \texttt{Nat}] := x.p \tag{28}$$

1.3 C-Prog Rules for Natural Numbers Program

$$\forall a{:}\mathcal{B}, p{:}\texttt{Path}. \tag{29}$$
$$(a \rightarrow p :: \texttt{Zero}) \rightarrow \tag{30}$$
$$(a \rightarrow p :: \texttt{Nat}) \tag{31}$$
$$\forall a{:}\mathcal{B}, p{:}\texttt{Path}. \tag{32}$$
$$(a \rightarrow p :: \texttt{Succ} \wedge x.p_{\{\!\{x \mapsto p\}\!\}} :: \texttt{Nat}) \rightarrow \tag{33}$$
$$(a \rightarrow p :: \texttt{Nat}) \tag{34}$$

2 Example Entailments

2.1 Working valid entailment

$p.\mathbf{cls} \equiv \texttt{Zero} \vdash p :: \texttt{Nat}$

- Checks unsatisfiable
- Unsat Core: C-Class, C-Prog-Zero

2.2 Working invalid entailment

$\vdash x \equiv y$

- Checks satisfiable
- Model: $p \equiv q \doteq p = q$

$a \equiv b \vdash a \equiv c$
Checks satisfiable

Model:

$$
\begin{array}{l}
\mathit{helper}(q\,{:}\,\texttt{Path}) := \\
\quad q = a \ ?\ a \ : \\
\quad\quad q = c \ ?\ c \ : \\
\quad\quad\quad q = x \ ?\ x \ : \\
\quad\quad\quad\quad q = b \ ?\ b \ : \\
\quad\quad\quad\quad\quad q = b.p \ ?\ b.p \ : \\
\quad\quad\quad\quad\quad\quad q = a.b \ ?\ a.b \ : \ c.p \\
p\,{:}\,\texttt{Path} \equiv q\,{:}\,\texttt{Path} := \\
\quad \textbf{let}\ a_1 := \mathit{helper}(p) = a.p \wedge \mathit{helper}(q) = b.p \\
\quad\quad\quad a_2 := \mathit{helper}(p) = a.p \wedge \mathit{helper}(q) = a.p \\
\quad\quad\quad a_3 := \mathit{helper}(p) = c.p \wedge \mathit{helper}(q) = c.p \\
\quad\quad\quad a_4 := \mathit{helper}(p) = b.p \wedge \mathit{helper}(q) = b.p \\
\quad\quad\quad a_5 := \mathit{helper}(p) = b.p \wedge \mathit{helper}(q) = a.p \\
\quad \textbf{in} \displaystyle\bigvee_{i \in \{j \mid 1 \le j \le 5\}} a_i \\
\quad \vee\ \mathit{helper}(p) = a \wedge \mathit{helper}(q) = a \\
\quad \vee\ \mathit{helper}(p) = b \wedge \mathit{helper}(q) = b \\
\quad \vee\ \mathit{helper}(p) = b \wedge \mathit{helper}(q) = a \\
\quad \vee\ \mathit{helper}(p) = c \wedge \mathit{helper}(q) = c \\
\quad \vee\ \mathit{helper}(p) = x \wedge \mathit{helper}(q) = x \\
\quad \vee\ \mathit{helper}(p) = a \wedge \mathit{helper}(q) = b
\end{array}
$$

2.3 Non-working invalid entailment

$x :: \texttt{Succ}, x.p :: \texttt{Zero} \vdash x :: \texttt{Zero}$

- Location: /paper/dep-classes/smt/semantic_entailment/fieldAccessTimeout.smt
- Solver has “infinite” runtime.
- Wanted behavior: Checks satisfiable.
- Current Solution: Impose a timeout on the solver.
  - Assume an entailment to be invalid if the solver times out (to retain soundness).
  - Procedure isn’t complete anymore.
  - Balance the timeout between the amount of checkable valid entailments and tolerable waiting time. (Set it such that the majority of valid entailments can still be checked.)

3 Undefinedness

3.1 Add expert knowledge to successfully check 2.3

The solver is able to check the example satisfiable if we either add

- $\neg x \equiv x.p$ to the context, resulting in $x :: \texttt{Succ} \wedge x.p :: \texttt{Zero} \wedge \neg x \equiv x.p \rightarrow x :: \texttt{Zero}$ or
- additionally assert $\neg x \equiv x.p$.

TODO: investigate steps performed by the solver

3.2 Asserting two mutually exclusive classes to the same path

Asserting both $x :: \texttt{Zero}$ and $x :: \texttt{Succ}$ in the solver yields SAT.

This, at first, seems to be unintended behaviour. In reality, this is not the kind of problem we want to solve with the encoding. What we ask of the solver is if there exists a conflict between the entailment to check (context implies conclusion) and the asserted calculus rules.

In other words: If the entailment context allows this to be true, the problem is a wrong annotation made by the programmer.

4 Faithful encoding

5 Change entailment translation

Instead $\bigwedge c_i \rightarrow \mathit{conclusion}$, use multiple asserts. for c_i in $\mathit{context}$. assert c_i assert $\neg \mathit{conclusion}$