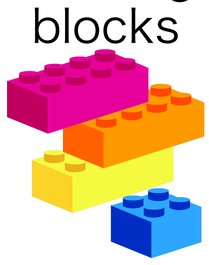
Synthesis from Partial Refinements

UC San Diego
Michael James (m3james@ucsd.edu)
Advisor: Nadia Polikarpova

what is synthesis?

specification + program building blocks = synthesized code



```
reconstructTopLevel :: MonadHorn s => Goal -> Explorer
reconstructTopLevel (Goal funName env (ForallT a sch)
  reconstructTopLevel (Goal funName (addTypeVar a env)
reconstructTopLevel (Goal funName env (ForallP sig sch
  -- reconstructTopLevel
    -- (Goal funName (addBoundPredicate sig env) sch )
reconstructTopLevel (Goal funName env (Monotype typ@(F
  = local (set (_1 . auxDepth) depth) (reconstructFix
reconstructTopLevel g@(Goal funName env (Monotype typ
  writeLog 1 $ text "reconstructTopLevel"
  if isFunctionType 1
    -- Must add the whole intersection to the environm
    then local (set (_1 . auxDepth) depth) (reconstruc
```

Ease of Use ← Styles of Specification → Precision

examples

```
stutter [] => []
      [1] => [1,1]
    [2,1] => [2,2,1,1]
```

middle ground ?

my work

refinements

```
stutter :: xs: List a → {List a | len ν == 2 * len xs}
```

(xs: List a: Named argument; List a: Base Type; len ν == 2 * len xs: Refinement; {List a | len ν == 2 * len xs}: Refinement Type Postcondition)

what is the problem?

```
filter :: (a → Bool) → [a] → [a]
```

examples

```
filter isEven [] => []          filter isOdd [] => []
            [1] => []                       [1] => [1]
          [2,1] => [2]                    [2,1] => [1]
        [3,2,1] => [2]                  [3,2,1] => [3,1]
      [4,3,2,1] => [4,2]              [4,3,2,1] => [3,1]
```

Verbose!

refinements

```
filter :: pred: (x:a → {Bool|pred x}) →
```

```
xs: [a] → {[a | pred ν]}
filter pred xs = Nil
```

```
xs: [a|pred ν] → {[a] | ν == xs]}
filter pred xs = xs
```

Not expressive enough!

partial refinements

```
filter :: pred:(x:a → {Bool|pred x}) →
```

```
(xs: [a] → {[a|pred ν]})
```
Predicate positive values

∩

```
(xs: [a|pred ν] →
{[a]|ν==xs})
```
Trace-complete example

∩

```
[x:{a|¬pred ν},
 y:{a| pred ν}] → {[a]|ν==y}
```
Polymorphic Example

how, efficiently?

Worlds[1]

Split worlds per intersection

```
filter pred xs =
  match xs of
    Nil → ??
    Cons y ys → (??)::τ₁∩τ₂
```

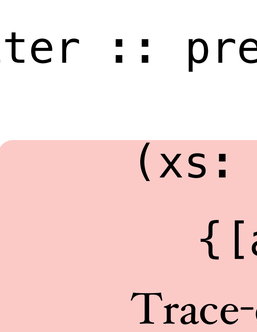
Attempt	Terms for τ_1	Terms for τ_1
1.	var ?? =	var ??
2.	match ?? ≠	?? ??
3. ite ?? ?? ?? =	ite ?? ??	??

Rule selected must match **per step**

Round trip type checking[2]

Searching for an app

$\Gamma \vdash$?? ?? $\downarrow \tau \uparrow \tau'$



Top down then back up!

Gather subtype constraints as Horn Clauses

```
filter pred xs =
  match xs of
    Nil → Nil
    Cons y ys → if (pred y)
      then (Cons y (filter pred ys))
      else (filter pred ys)
```

what's next?

Type Negation

```
filter ::
  pred:(x:a → {Bool | pred x}) →
  [a| pred ν] → {[a]|ν == xs} ∩
  ¬[a| pred ν] → [a| pred ν}
```

Only 2 partial specs!

Mixed Abstract / Concrete

Condition Abduction from intersections

Synthesize outside of trace-complete examples

[1] J. Frankle, P.-M. Osera, D. Walker, and S. Zdancewic, "Example-directed Synthesis: A Type-theoretic Interpretation," in Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New York, NY, USA, 2016, pp. 802–815, doi: 10.1145/2837614.2837629.
[2] N. Polikarpova, I. Kuraj, and A. Solar-Lezama, "Program Synthesis from Polymorphic Refinement Types," p. 17.
[3]P.-M. Osera and S. Zdancewic, "Type-and-example-directed Program Synthesis," in Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, New York, NY, USA, 2015, pp. 619–630, doi: 10.1145/2737924.2738007.

Printed for POPL2020, Student Research Competition, in New Orleans, LA