#31: Paper Discussion

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

Regae

[Zhang, Lowmanstone, Wang, Glassman, UIST'20]

Better UI for a regex synthesizer

Regae: contributions

- Novel way to express intent: semantics augmentation
- Novel way to explain synthesis results to user: data augmentation
- Automata-theoretic algorithms to generate explanatory examples
    - familiar examples with different output, corner cases, distinguishing examples
- Usability confirmed by user study
    - Completion rate: 12/12 vs 4/12; twice more confident; less cognitive load

Regae: limitations

Limited to regexes

- Which parts are generalizable and which not?

Marking as general affects completeness

Not tolerant to user mistakes

User study participants might not be representative

Regae: questions

- Behavioral constraints? Structural constraints? Search strategy?
    - IO examples
    - Built-in DSL
    - Top-down enumerative search

Regae: questions

- Does semantic augmentation contribute to behavioral or structural constraints, or something else?
    - Structural because it affects the search space
- What about data augmentation?
    - Directly contributes only to result comprehension
    - Indirectly to behavioral because users can use those examples as input

Regae: questions

- When can we soundly reject the sketch concat(<num>, e)?
  - If e.g. <num> is marked excluded [that’s not what I meant]
  - When there is a positive example that doesn’t start with a number
  - More generally, replace e with star(<any>) and check whether all positives can be parsed!
    - if under not, then replace with an empty-language regex
  - Another idea is define equivalence on regexes, e.g. optional(star(e)) is equivalent to star(e)

Regae: questions

- Why is it important to randomize the order of control vs treatment?

RbSyn

[Guria, Foster, Van Horn, PLDI'21]

Program synthesis from side effects

RbSyn: contributions

- Using side-effects to guide search
- Rule based merging to create if-then-else branches
- Automatic side effect inference from test failures
- Evaluated on programs from real-world benchmarks

RbSyn: limitations

- Cannot synthesize loops/lambdas/etc.
- Effect annotation burden may require domain insight
- Limited to typed subset of Ruby

RbSyn: questions

- Branch merging strategy wrt Synquid and EUSolver?
    - Synquid uses liquid abduction
    - EUSolver uses decision tree learning (information gain heuristic)
    - RbSyn uses rules-based approach as no counterexample possible

RbSyn: questions

- Scaling of branch merging with changing no. of tests (N)
    - N! ways of merges to be checked
    - It is M! if there are only M distinct programs from tests
    - The bottleneck is the number of distinct solutions to individual tests

RbSyn: questions

- Order of search for effect annotations:
    - `User.name`
    - `User`
    - `*`

    - Methods to be considered in this order: `name=`, `save`, `delete`

RbSyn: questions

- Why does RbSyn stay sound?
  - Wrong effect inferred: Adds program to the work list, no sound program is eliminated from the work list
  - Wrong method substituted: All correct effect annotated method choices are enumerated