Type Inference Using Meta-Extract for Smtlink and Beyond

Yan Peng and Mark R. Greenstreet

CpSc 418 – May 28, 2020

Types: ACL2 vs. Z3

ACL2

```
(equal (rev (rev (rev x)))
       (rev x))
```


Z3

```
?
```

```
(implies
 (and (posp a)(posp b)
      (posp c)(posp n)(< 2 n))
 (not (equal (+ (expt a n)
                (expt b n))
             (expt c n))))
```



```
a,b,c,n = Ints('a b c n')
S = Solver()
s.add(And(0 < a, 0 < b,
          0 < c, 2 < n,
             expt(a, n) + expt(b,n)
          == expt(c, n))
if(s.check() == unsat):
   print('qed')
else: print('cex: ' + str(s.model()))
```

- ACL2 is based on untyped, first-order logic with induction
- SMT solvers (e.g. Z3) are based on many-sorted, first-order logic without induction
- How do we bridge the two?

Types in Smtlink

- Use type recognizers
  - ACL2 has type recognizers: booleanp, integerp, rationalp, symbolp etc.
  - Users can define their own recognizers: integer-list-p, cow-p, cow-pig-alist-p, etc.
  - If all free-variables in a theorem statement have hypotheses that assert their types
    - ★ A verified clause processor can show that the claim holds trivially for any model where the value of a variable does not satisfy the type recognizer.
    - ★ A SMT solver can show that there are no models where all variables satisfy their type recognizers.
    - ★ QED
- Need to handle "polymorphic" functions:
  - `cons`, `car`, `cdr`, . . . , and possibly user-defined functions.
  - Smtlink 2.0 requires the user to annotate terms with fixing functions
    - ★ e.g. (integer-list-fix nil)
    - ★ while it's straightforward, it is annoying "clutter".
- Contributions of current work:
  - automated type-inference
  - cleaner soundness arguments for user-defined types

A running example

```
(implies (and (rational-list-p x) ...)
         (...  (equal (list 1 2 3 4 5)
                      x) ...))
```

- What is the type of `(list 1 2 3 4 5)`?
- In ACL we get:

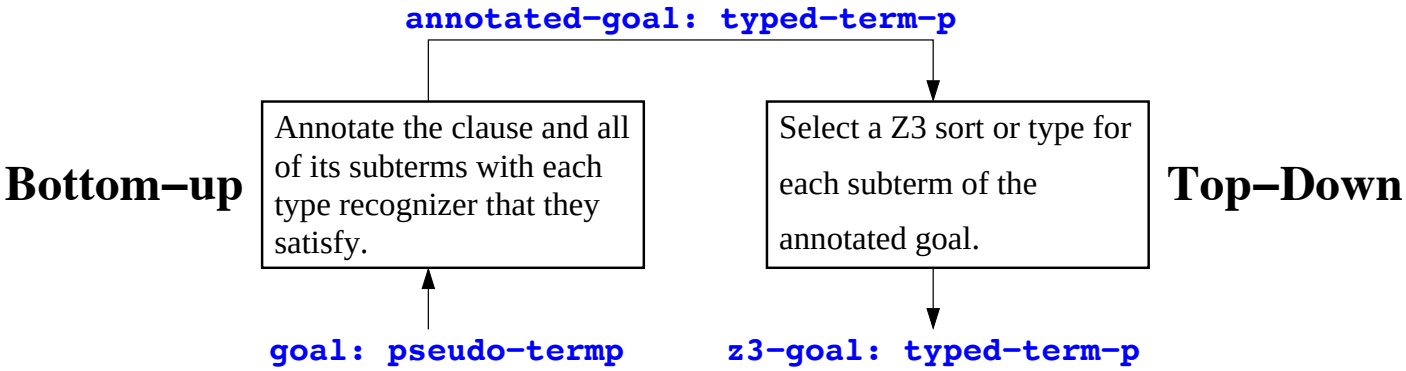
```
(and (true-listp (list 1 2 3 4 5))
     (integer-listp (list 1 2 3 4 5))
     (rational-listp (list 1 2 3 4 5)))
```

- Z3 supports user-defined datatypes, e.g.

```
IntegerList = Datatype('IntegerList')
IntegerList.declare('cons', ('car', IntSort()),
                            ('cdr', IntegerList))
IntegerList.declare('nil')
IntegerList = IntegerList.create()
```

  - But, `IntegerList` is not a subtype of `RealList`
  - If `i_list` is an `IntegerList`, `r_list` is a `RealList`, and `x` is a `Real`,
  - `i_list == r_list` is an illegal operation.
  - And more issues with `cons`, `nil`, ...

The clause processors



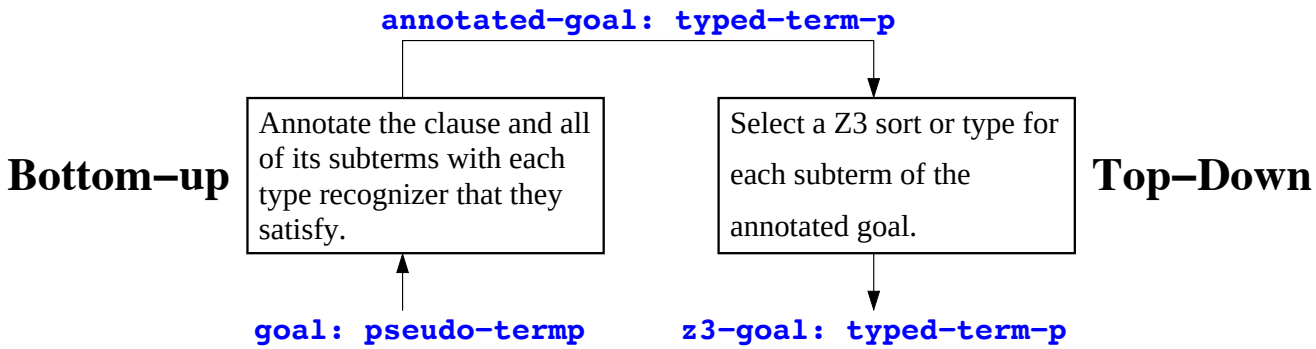
- Depth-first traversal of clause, maintaining the path-condition during traversal.
- Label each subterm with a conjunction of all type-recognizers that it satisfies
- A function can have one or more “returns” theorems, use these (see `meta-extract`) to infer type recognizers satisfied by return result.
- When done, we have an annotated term, where we know all type recognizers satisfied by each subterm.

type-term's

```
(defprod typed-term
  ((term pseudo-termp)          ;; the term
   (path psesudo-termp)         ;; the path condition for this term
   (judge psesudo-termp)))      ;; type judgements for this term
                                ;; and its subterms
```

- Accessor functions let code traverse a typed-term analogously to traversing a pseudo-term, but now we have type-info.
- Correctness conditions:
  - `(implies (typed-term->path tt)`
    `(typed-term->judge tt))`
  - A (recursive) structural property that the shape of the judgements matches the shape of the term.

The top-down clause processor



- For each subterm, choose a SMT-compatible type, or report an error in no such type exists.
  - For any choice of a type assignment of a term, there is a consistent assignment of types for the subterms.
- Selecting a type is just discarding conjuncts from `(typed-term->judge tt)`.
  Therefore,

```
(implies (typed-term->path tt)
         (typed-term->judge tt))
```

  is preserved.

The Back-End

Now that Smtlink has Z3 types for each subterm of the clause:

- Replace polymorphic ACL2 functions such as `car` with typed-equivalents, e.g. `integer-list-car`.
  - Note that in Z3 `IntegerList.car(x)` is an integer for any `x`.
  - Thus, the ACL2 counterpart, `integer-list-car` must "fix" `nil` to an integer value.
  - If Smtlink's clause processor can establish that `x` is non-nil, this replacement is sound.
  - Otherwise, the clause processor produces a subgoal for ACL2 to show that modified goal implies the original.
- Transliterate the resulting ACL2 term to Z3.py, and trust Z3.
  - This final transliteration and the execution of Z3.py are the only trusted parts of Smtlink.
  - All other transformations are performed by verified clause processors.

Is this sound?

- We have a sketch of a proof that a model of the logic of ACL2 that has a counter example implies that there is a model of the translated, negated goal in Z3.
- We would love to find a formal description of the logic of Z3?

```
>>> n = Int('n')
>>> x = Real('x')
>>> prove(Implies(n > 1, 1/n == 0))
proved                  # here / denotes integer division
>>> prove(Implies(n > 1, 1.0/n == 0))
proved # Z3 casts 1.0 to an integer. (Thanks, Andrew Walter ☺)
>>> prove(Implies(And(n > 1, x == 1.0), x/n == 0))
counterexample      # Z3 promotes n to RealSort()
[n = 2, x = 1]
>>> prove(Implies(n == 1, n == 1.5))
proved                  # !!!
>>> prove(Implies(And(n == 1, x == 1.5), n == x))
counterexample
[x = 3/2, n = 1]
```

Summary

- We are developing a type-inference engine for Smtlink.
- This should relieve the user of most of the type-annotation work needed when using Smtlink.
- Current status:
  - The bottom-up and top-down clause processors have been written and tested.
  - Proofs of the property that the syntactic shape of the typed-terms produced corresponds to the shape of the clause are in progress.
  - The final translation steps are in progress: converting ACL2 `alist`s to Z3 `array`s has presented some fun puzzles.
- The clause processors should be useful for automating other type-like reasoning.

Summary

- We are developing a type-inference engine for Smtlink.
- This should relieve the user of most of the type-annotation work needed when using Smtlink.
- Current status:
  - The bottom-up and top-down clause processors have been written and tested.
  - Proofs of the property that the syntactic shape of the typed-terms produced corresponds to the shape of the clause are in progress.
  - The final translation steps are in progress: converting ACL2 `alist`s to Z3 `array`s has presented some fun puzzles.
- The clause processors should be useful for automating other type-like reasoning.

Thank You!