A *Bind-free* Experience Report

Proving a type of inequality with bind-free guided rewriting

Hanbing Liu

May 11, 2009

Background: Verifying FP Algorithms

At AMD,

- We verify our floating point DIV/SQRT algorithms
- A typical algorithm may look like this:

```
y  := lookup-reciprocal(b)  ;;       1/b *(1+e0)
e  := rnd(1 - b*y,64)       ;;   (1-b*y)*(1+e1)
y1 := rnd(y + y*e,64)       ;;   (y+y*e)*(1+e2)
y2 := rnd(y + y1*e,64)      ;;  (y+y1*e)*(1+e3)
q  := rnd(a*y2,23)
r  := a - q*b
Q  := rn(q + r*y2,23)
```

- Such algorithms have two stages:
  - Approximation stage
  - Rounding stage
- One task is to show that the *relative error* between `a*y2` and the true value `a/b` is bounded by a small constant

Background: Verifying FP Algorithms

At AMD,

- We verify our floating point DIV/SQRT algorithms
- A typical algorithm may look like this:

```
y  := lookup-reciprocal(b)  ;;       1/b *(1+e0)
e  := rnd(1 - b*y,64)       ;;   (1-b*y)*(1+e1)
y1:= rnd(y + y*e,64)        ;;   (y+y*e)*(1+e2)
y2:= rnd(y + y1*e,64)       ;; (y+y1*e)*(1+e3)
q  := rnd(a*y2,23)
r  := a - q*b
Q  := rn(q + r*y2,23)
```

- Such algorithms have two stages:
  - Approximation stage
  - Rounding stage
- One task is to show that the *relative error* between `a*y2` and the true value `a/b` is bounded by a small constant

In short, we often need to prove $|P(\vec{e})| \leq C$ type theorems

Problem: ACL2 Needs Better Guidence

A simple but illuminating example:

- Prove p2 theorem — easy

```
(defthm p2
  (implies (and (<= (abs e1) 1)
                (<= (abs e2) 1))
           (<= (abs (+ e1 e2)) 2)))
```

- Prove p10 theorem — hard

```
(defthm p10
  (implies (and (<= (abs e1) 1)
                (<= (abs e2) 1)
                ...
                (<= (abs e10) 1))
           (<= (abs (+ e1 e2 ... e10)) 10)))
```

- Prove p100 theorem — not practical

Problem: ACL2 Needs Better Guidence

A simple but illuminating example:

- Prove p2 theorem — easy

```
(defthm p2
  (implies (and (<= (abs e1) 1)
                (<= (abs e2) 1))
           (<= (abs (+ e1 e2)) 2)))
```

- Prove p10 theorem — hard

```
(defthm p10
  (implies (and (<= (abs e1) 1)
                (<= (abs e2) 1)
                ...
                (<= (abs e10) 1))
           (<= (abs (+ e1 e2 ... e10)) 10)))
```

- Prove p100 theorem — not practical

When $P(\vec{e})$ is complex, the ACL2 built-in linear procedures and strategies (as embodied in the its arithmetic library) are too general to be effective. ACL2 needs better guidence.

Solution: A Simple Strategy

To prove a p100 theorem:

```
(implies (and (<= (abs e1) 1)
              (<= (abs e2) 1)
              ...
              (<= (abs e100) 1))
         (<= (abs (+ e1 e2 ... e100)) 100))
```

A simple strategy does exist

- Prove the following rule

```
    abs(term) <= d1
    abs(poly) <= d2
    d1+d2 <= C
=>
    abs(term + poly) <= C
```

- Apply this rule and backchain to relieve the second hypothesis `abs(poly)<= d2`

Solution: A Simple Strategy

To prove a p100 theorem:

```
(implies (and (<= (abs e1) 1)
              (<= (abs e2) 1)
              ...
              (<= (abs e100) 1))
         (<= (abs (+ e1 e2 ... e100)) 100))
```

A simple strategy does exist

- Prove the following rule

```
    abs(term) <= d1
    abs(poly) <= d2
    d1+d2 <= C
=>
    abs(term + poly) <= C
```

- Apply this rule and backchain to relieve the second hypothesis `abs(poly)<= d2`

The key is that the ACL2 theorem prover does not know how to find suitable bindings for *free variable* in the rule: d1 and d2

Solution: Two Tasks And *Bind-free* Trick

To help the ACL2 theorem prover to mimic what one would do:

Two Tasks

- Introduce rewrite rules that codify the general (backchain) strategy. They have *free variables* in their hypothesises. They are "templates" for what kind of proof obligations to create.
- Define an algorithm that examines the conjecture and finds suitable bindings for the "parameters" (free variables) in the "templates".

Bind-free trick

- Allow the ACL2 theorem prover to invoke the algorithm during rewriting to find the right way to backchain
- Details on this later

Rewrite Rules For Our $|P(\vec{e})| \leq C$ Type problem

Match how a polynomial may be constructed.

- One rule for each type

```
(defthmd over-estimate-rule-var-leaf
  (implies (and (syntaxp (symbolp x))
                (<= (abs x) d1)
                (<= d1 d2))
           (<= (abs x) d2)))
...
(defthmd over-estimate-rule-add
  (implies (and (<= (abs x) d1)
                (<= (abs y) (+ (- d1) d2)))
           (<= (abs (+ x y)) d2)))
```

We note that, in their current forms, the ACL2 theorem prover could not make use these rules properly.

One Workable Algorithm For Picking Bindings

Essentially a simple upper bound finding algorithm

- Two inputs:
  - A polynomial: `'(+ (* e1 e2) (* e2 (+ e3 e3)) ...)`
  - A list of upper bounds on the absolute value of variables: `'((e1 . 1/16) (e2 . 1) (e3 . 1) ...)`
- Output: upper bound of the polynomial under the assumption
- Operations:
  - For *atomic* polynomial such as a simple variable, looking up the upper bound in the input list
  - For *compound* polynomial, find the upper bounds for subcomponent recurisively; combine the upper bounds found in a conservative way

Bind-free Trick

```
(defthmd over-estimate-rule-add    ;; old
  (implies (and (<= (abs x) d1)
                (<= (abs y) (+ (- d1) d2)))
           (<= (abs (+ x y)) d2)))

(defthmd over-estimate-rule-add    ;; new
 (implies
  (and (bind-free (bind-d1-with-hints x hints) (d1))
       (less_equal_than_with_hints (abs x) d1 hints)
       (less_equal_than_with_hints (abs y)
                                   (+ (- d1) d2) hints))
  (less_equal_than_with_hints (abs (+ x y)) d2 hints)))
```

- Adding the *bind-free* hypothesis to the rewrite rule
- Replacing $\leq$ with *less_equal_than_with_hints*
- Coming up with a suitable *hints*

Example

Suppose we want to prove the follow:

```
(defthmd numeric-fact-old
  (implies
   (and (<= (abs e) (expt 2 -14))
        (<= (abs rne2) (expt 2 -64))
        (<= (abs rne3) (expt 2 -64))
        (rationalp e)
        (rationalp rne2)
        (rationalp rne3))
   (<= (abs (+ 1 (* -1 e)
                 (* rne3 rne3)
                 (* rne2 rne3 (+ e e))))
       2)))
```

Example

```
(defthmd numeric-fact-new
  (implies
    (and (less_equal_than (abs e) (expt 2 -14))
         (less_equal_than (abs rne2) (expt 2 -64))
         (less_equal_than (abs rne3) (expt 2 -64))
         (rationalp e)
         (rationalp rne2)
         (rationalp rne3))
     (less_equal_than_with_hints
          (abs (+ 1 (* -1 e) (* rne3 rne3)
                    (* rne2 rne3 (+ e e))))
           2
           '((e . 1/16384)
             (rne2 . 1/18446744073709551616)
             (rne3 . 1/18446744073709551616))))
  :hints (("Goal" :in-theory
                   (e/d (over-estimate-rule-add
                          over-estimate-rule-prod
                          over-estimate-rule-var-leaf
```

Conclusion

Our type of $|P(\vec{e})| \leq C$ inequality is both easy and difficult

- $P(\vec{e})$ has an explicit structure
- C does not have such an explicit strucure

Our technique is simple and effective

- Write an algorithm to analyze the structure of $P(\vec{e})$
- Introduce bind-free hypothesis into a few rewrite rules
- Extract the hypothesises into a "hints" constant

This is a good showcase of how one might use bind-free