Defining and Proving Interval Bounds for Arithmetic Functions

J Strother Moore

ForrestHunt, Inc.

January, 2013
(source: /u/moore/work/x86-y86/devel/talk-on-bounders.tex)

Defining and Proving Interval Bounds for Arithmetic Functions

and

Getting Tau to Use Them

J Strother Moore

ForrestHunt, Inc.

January, 2013
(source: /u/moore/work/x86-y86/devel/talk-on-bounders.tex)

The Tau Project

Ideal Goal: Allow any monadic predicate to be a “type” and build a “type checker.”

The Tau Project

Ideal Goal: Allow any monadic predicate to be a “type” and build a “type checker”

Realistic Goal: Do it well enough to allow “most” Common Lisp programs to be “type checked” without inconveniencing the user “too much.”

Basic Ideas

A *tau* ("type") is a succinct description of the objects satisfying a conjunction of monadic predicates. E.g., "naturals less than 16" is
`(and (natp x) (< x 16))`.

Tau maintains a *data base* of inclusion relations between predicates ("types") and "signatures" of function symbols.

Currently, the data base is derived entirely from *theorems posed by the user*.

Tau Predicates: Primitive “Types”

The primitive tau are the predicates of the form $(\lambda(v)\beta)$, where β:

- `(`p v`)`
- `(EQUAL` v `'`$const$`)`
- `(<` v `'`$const$`)`, `(<=` v `'`$const$`)`, `(<` `'`$const$ v`)`, `(<=` `'`$const$ v`)`
- negations and obvious variants of the forms above (e.g., `(EQ` v `'`$const$`)` or `(>=` v `'`$const$`)`)

Tau Predicate Examples

- `(natp x)`
- `(not (integerp x))`
- `(eq x 'ARRAY)`
- `(not (equal x 'SKIP))`
- `(<= x 15)`
- `(not (= x 7))`

Tau Rules

Tau builds its data base from theorems proved by the user.

Tau automatically (and silently) tries to extract tau rules from *every kind of* `:rule-class`.

The two most basic shapes tau recognizes are:

```
Simple:  Used to transitively close data base
(implies (p x) (q x))

Signature: Used to compute tau of fn appl
(implies (and (p x) (q y))
         (r (fn x y)))
```

Where `p`, `q`, and `r` are tau predicates (possibly negated).

Representation of Tau Objects

If x has tau τ, we say x is the *subject* of τ.

Internally, a *tau* data object with subject x explicitly includes:

- all the objects x is equal to,

- certain objects x is not equal to,

- an arithmetic interval containing x,

- all the recognizers x satisfies, and

- all the recognizers x dis-satisfies.

Example

Consider the goal:

```
(implies (and (natp x) (not (equal x 5)) (< x 17))
         (p x))
```

then x has τ

```
((NIL (5)) (INTEGERP (NIL . 0) NIL . 16)
  ((144 . FILE-CLOCK-P)
   (135 . 32-BIT-INTEGERP) (19 . O-FINP) (17 . NATP) (9 . EQLABLEP)
   (5 . RATIONALP) (4 . INTEGERP) (0 . ACL2-NUMBERP)) .
  ((167 . BAD-ATOM) (154 . WRITABLE-FILE-LISTP1) (151 . READ-FILE-LISTP1)
   (148 . WRITTEN-FILE) (145 . READABLE-FILE) (141 . OPEN-CHANNEL1)
   (121 . KEYWORDP) (114 . ALPHA-CHAR-P) (105 . IMPROPER-CONSP)
   (104 . PROPER-CONSP) (100 . BOOLEANP) (60 . WEAK-CURRENT-LITERAL-P)
   (31 . WEAK-IO-RECORD-P) (26 . MINUSP) (7 . SYMBOLP) (6 . STRINGP)
   (3 . CONSP) (2 . COMPLEX-RATIONALP) (1 . CHARACTERP)))
```

Representation

If x has tau τ, we say x is the *subject* of τ.

Internally, a *tau* data object with subject x explicitly includes:

- all the objects x is equal to,
- certain objects x is not equal to,
- an arithmetic interval containing x,
- all the recognizers x satisfies, and
- all the recognizers x dis-satisfies.

Intervals

Internally, every tau includes an *interval* known to contain the subject.

```
(and (integerp x)
     (<= 3 x)
     (<= x 17))
```

is represented in the corresponding tau by:

```
(INTEGERP (nil . 3) . (nil . 17))
```

Intervals

$(domain\ \ (lo-rel\ \ .\ \ lo)\ \ .\ \ (hi-rel\ \ .\ \ hi))$

$domain \in \{$`INTEGERP`, `RATIONALP`, `ACL2-NUMBERP`, `NIL`$\}$

$lo-rel \in \{$`T`, `NIL`$\}$ meaning $<$or $\leq$

$lo =$ `NIL` or a rational (`NIL` means $-\infty$)

$hi-rel \in \{$`T`, `NIL`$\}$ meaning $<$or $\leq$

$hi =$ `NIL` or a rational (`NIL` means $+\infty$)

Thus, there are 64 different configurations.

<<go to `*shell*`>>

Computing the Interval for Addition

Suppose `x` and `y` are rationals and

$0 < \texttt{x} < 15$ and $0 < \texttt{y} < 10$.

Then, clearly,

$0 < \texttt{(+ x y)} < 25$

Computing the Interval for Addition

Suppose `x` and `y` are rationals and

$lo_x < \texttt{x} < hi_x$ and $lo_y < \texttt{y} < hi_y$.

Then, clearly,

$lo_x + lo_y <$ `(+ x y)` $< hi_x + hi_y$

with appropriate interpretations of infinities and strengths of the relations.

Computing the Interval for Product

Suppose `x` and `y` are rationals and

$0 <$ `x` < 15 and $0 <$ `y` < 10.

Then, clearly,

$0 <$`(* x y)`< 150

Computing the Interval for Product

Suppose `x` and `y` are rationals and

$0 < \texttt{x} < 15$ and $-10 < \texttt{y} < 10$.

Then, clearly,

$-150 < \texttt{(* x y)} < 150$

Questions

How do we compute the bounds for product?

How do we know our bounds are correct?

How do we bound logical operations like `LOGAND` and `LOGXOR`?

How accurate are they?

How can I make tau exploit user-defined bounds functions?

Computing the Interval for Product

Suppose `x` and `y` are rationals and

$0 < \texttt{x} < 15$ and $-10 < \texttt{y} < 10$.

Then, clearly,

$-150 <$ `(* x y)` < 150

<<Go to `*shell*`>>

Correctness of tau-bounder-*

```
(implies
 (and (tau-intervalp int-x)
      (tau-intervalp int-y)
      (or (equal (tau-interval-dom int-x) 'INTEGERP)
          (equal (tau-interval-dom int-x) 'RATIONALP))
      (or (equal (tau-interval-dom int-y) 'INTEGERP)
          (equal (tau-interval-dom int-y) 'RATIONALP))
      (in-tau-intervalp x int-x)
      (in-tau-intervalp y int-y))
 (and (tau-intervalp (tau-bounder-* int-x int-y))
      (in-tau-intervalp (* x y)
                        (tau-bounder-* int-x int-y)))))
; Subgoals: 40,661; Time: 581s; Steps: 42M
```

Acknowledgement:

Robert Krug defined the first version of `tau-bounder-*` and verified its correctness. Thank you Robert!

Computing the Interval for LOGAND

Suppose `x` and `y` are integers and

$lo_x < \texttt{x} < hi_x$ and $lo_y < \texttt{y} < hi_y$.

Then,

$??? <$`(logand x y)`$< ???$

<<go to `*shell*`>>

On the Accuracy of LOGAND Bound

[threshhold = 100 instead of 1,048,576]

k	A	E	(C)
5	65	100	(99)
10	60	87	(80)
15	61	87	(58)
20	57	77	(43)
25	58	75	(32)
30	60	73	(25)
35	57	68	(20)
40	56	65	(17)
45	56	65	(14)
50	57	65	(12)

```
k: radius of population
A: percentage bounded
    perfectly by purely
    analytical method
E: percentage bounded
    perfectly when
    empirical method incl'd
C: percentage of cases
    covered by empirical
    method
```

Correctness of tau-bounder-logand

```
(IMPLIES
 (AND (TAU-INTERVALP INT1)
      (TAU-INTERVALP INT2)
      (EQUAL (TAU-INTERVAL-DOM INT1) 'INTEGERP)
      (EQUAL (TAU-INTERVAL-DOM INT2) 'INTEGERP)
      (IN-TAU-INTERVALP X INT1)
      (IN-TAU-INTERVALP Y INT2))
 (AND (TAU-INTERVALP (TAU-BOUNDER-LOGAND INT1 INT2))
      (IN-TAU-INTERVALP (LOGAND X Y)
                        (TAU-BOUNDER-LOGAND INT1 INT2)))
```

Computing the Interval for LOGEQV

```
(logeqv x y) = (logand (logorc1 x y) (logorc1 y x))

(defun tau-bounder-logeqv (int1 int2)
  (tau-bounder-logand
   (tau-bounder-logorc1 int1 int2)
   (tau-bounder-logorc1 int2 int1)))
```

Using Bounders in the ACL2 Code

I defined bounders for `+`, `*`, `/`, `FLOOR`, `MOD`, `LOGAND`, `LOGNOT`, `LOGIOR`, `LOGORC1`, `LOGEQV`, `LOGXOR`, `ASH`.

I was worried whether they were correct so I verified them.

Then I was faced with either

(a) including them in the trusted source code and calling all 12 appropriately from the tau code, or

(b) supporting user-defined bounders.

General Form of a Tau Bounder Rule

```
(implies (and (tau-intervalp $i_1$) ... (tau-intervalp $i_k$)
              (or (equal (tau-interval-dom $i_1$) '$dom_{1,1}$)
                  ...)
              ...
              (or (equal (tau-interval-dom $i_k$) '$dom_{k,1}$)
                  ...)
              ...
              (in-tau-intervalp $v_1$ $i_1$)
              ...
              (in-tau-intervalp $v_k$ $i_k$))
         (and
           (tau-intervalp ($bounder$ $i_1$ ... $i_k$))
           (in-tau-interval ($fn$ $v_1$ ... $v_k$ $x_1$ ...)
                            ($bounder$ $i_1$ ... $i_k$))))
```

Making Tau Use Bounder Rules

The lemma shapes recognized by the tau system have been expanded to include tau bounder rules.

When the system is asked to compute the tau for a function application it

- applies all bounder rules and intersects the intervals, and then

- applies signature rules

<<go to `*shell*`>>

Further Work

Add special handling in tau for `(<` α β`)`; right now that is just Boolean unless α or β are constants

What other functions should have bounders defined in the bounders book? Currently supported: `+`, `*`, `/`, `FLOOR`, `MOD`, `LOGAND`, `LOGNOT`, `LOGIOR`, `LOGORC1`, `LOGEQV`, `LOGXOR`, `ASH`.

Perhaps the bounders book should be included as part of `arithmetic-5/top`? Some other arithmetic library?