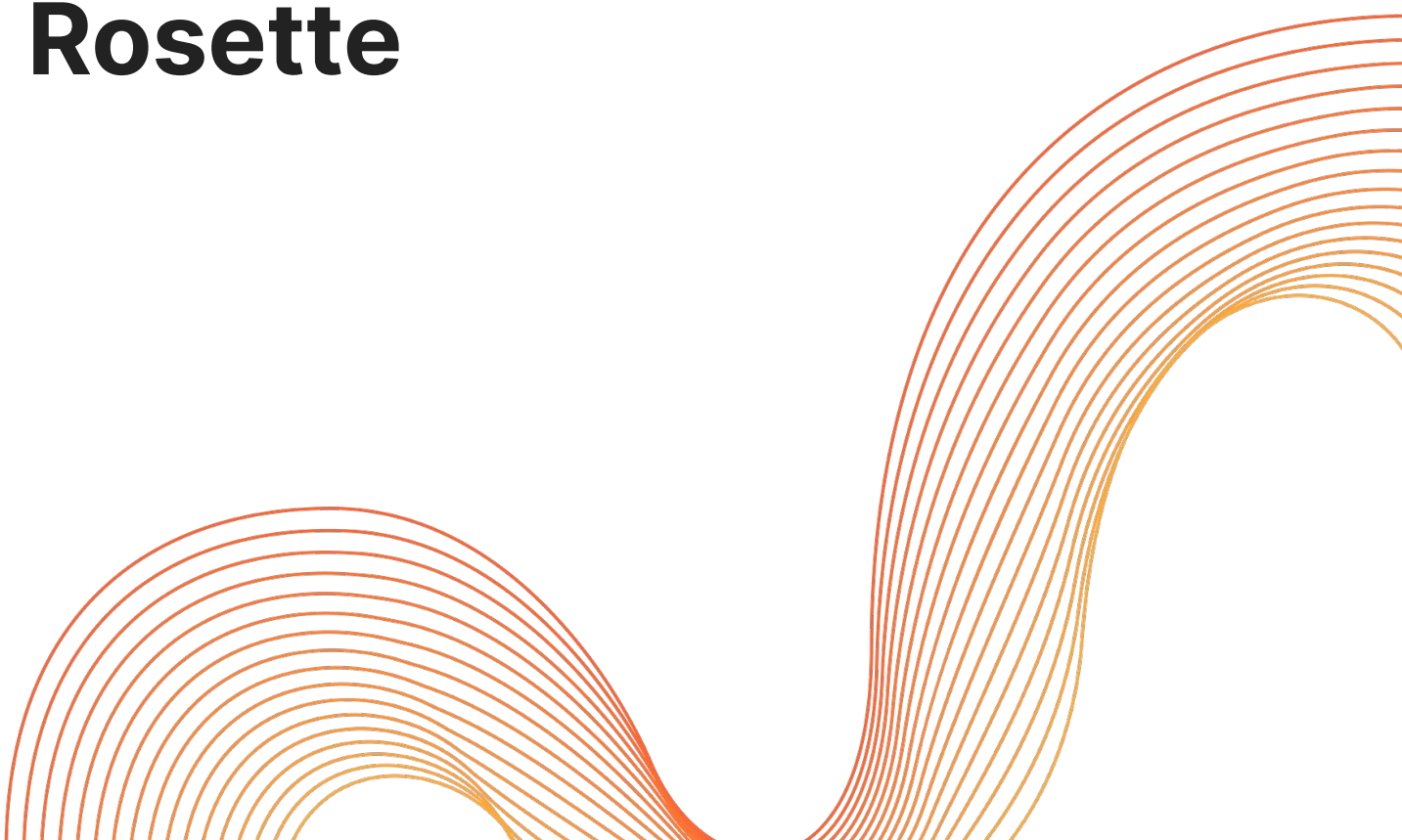
CLOUDFLARE

Fun With Rosette

Bob Halley
May 24, 2023



CLOUDFLARE

Background

- Long career as an engineer, working primarily on carrier-grade DNS servers
- No training in formal methods
- The original author and one of the maintainers of dnspython

The fullcompare() method from dnspython

Compare two domain names determining their ancestry relationship, order (in the DNSSEC ordering) and number of significant labels in common. Names may be fully qualified, or relative to an unspecified origin.

Name 1	Name 2	relation	order	common-labels
www.example.	www.example.	equal	0	3
www.example.	example.	subdomain	> 0	2
example.	www.example.	superdomain	< 0	2
example1.com.	example2.com.	common-ancestor	< 0	2
example1	example2.	none	< 0	0
example1.	example2	none	> 0	0

CLOUDFLARE

Why did I look at this method as part of learning about formal methods?

- I understand the subject area
- It's important
- It's complicated enough to be interesting
- It has very good test coverage, but the unit tests are not exhaustive

Rosette Explorations

- Simplify the problem
  - Model DNS labels as integers and not bytestrings; label comparison is just integer comparison
  - A domain name is a list of between 0 and 3 labels
- Convert fullcompare() into a tail-recursive function in Scheme/Racket/Rosette that always makes progress.
- Check for consistency
  - A is a subdomain of B ⇔ B is a superdomain of A
  - A = B ⇔ B = A
  - etc.

The Check

```
(define-symbolic l1 l2 integer?)

(define (check-one n1 n2)
  (let* ([res1 (full-compare n1 n2)]
         [res2 (full-compare n2 n1)]
         [nr1 (full-comparison-relation res1)]
         [nr2 (full-comparison-relation res2)]
         [o1 (full-comparison-order res1)]
         [o2 (full-comparison-order res2)]
         [nl1 (full-comparison-common-labels res1)]
         [nl2 (full-comparison-common-labels res2)])
    ; Check that (full-compare n1 n2) and (full-compare n2 n1) have appropriately matching results
    (cond
      [(and (= nr1 nr-none) (= nr2 nr-none))
       ; No relation; orders should be opposite and common labels 0
       (and (or (and (= o1 nr-gt) (= o2 nr-lt)) (and (= o1 nr-lt) (= o2 nr-gt))) (= nl1 0) (= nl2 0))]
      [(and (= nr1 nr-proper-subdomain) (= nr2 nr-proper-superdomain))
       ; Subdomain and superdomain; orders should be opposite and common labels equal and >= 0
       (and (= o1 nr-gt) (= o2 nr-lt) (= nl1 nl2) (>= nl1 0))]
      [(and (= nr1 nr-proper-superdomain) (= nr2 nr-proper-subdomain))
       ; Superdomain and subdomain; orders should be opposite and common labels equal and >= 0
       (and (= o1 nr-lt) (= o2 nr-gt) (= nl1 nl2) (>= nl1 0))]
      ; Equal; everything should be equal.
      [(and (= nr1 nr-equal) (= nr2 nr-equal)) (and (= o1 nr-eq) (= o2 nr-eq) (= nl1 nl2))]
      [(and (= nr1 nr-common-ancestor) (= nr2 nr-common-ancestor))
       ; Common ancestor; orders should be opposite, common labels should be equal and > 0
       (and (or (and (= o1 nr-gt) (= o2 nr-lt)) (and (= o1 nr-lt) (= o2 nr-gt)))
            (= nl1 nl2)
            (> nl1 0))]
      [else #f])))
```

The Fun Part

```
(define (check)
  (assume (and (>= l1 0) (<= l1 3)))
  (assume (and (>= l2 0) (<= l2 3)))
  (let* ([n1 (new-name l1)] [n2 (new-name l2)]) (check-one n1 n2)))

(verify (assert (check)))
```

; → (unsat)

This is strong, as we now now there is no way to make *check* false; a much stronger result than we could get from unit testing.

CLOUDFLARE

Take Aways

- Although the problem was simplified, what we learned about the restricted case carries over to full domain names, with a few extra conditions on what a valid name is.
- We have stronger assurances that fullcompare() is correct.
- Even limited proving of only critical fragments of codebases could lead to greatly improved correctness and security.
- Debugging was improved; the checker gives counterexamples when it finds them.