Spreadsheets, Formal Methods, and Winning the XPRIZE

Byron Davies, Ph.D.

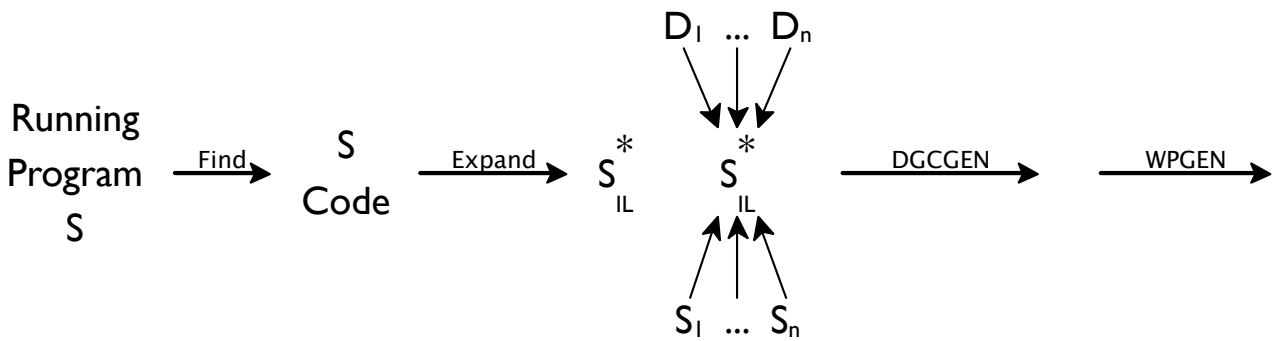
ByronDavies@starshine.us

StarShine Academy

StarShinePlanet.com

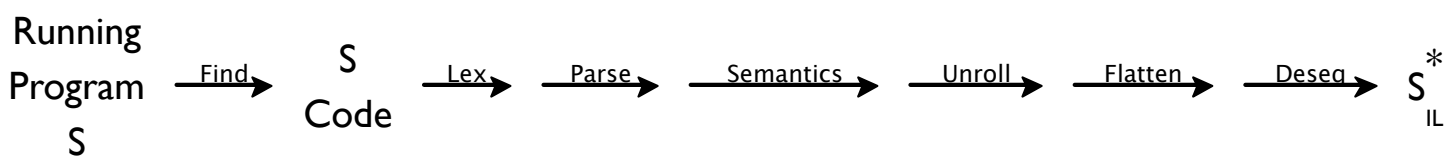
RacketCon 2015

Tool Development
for
System Integrity Analysis

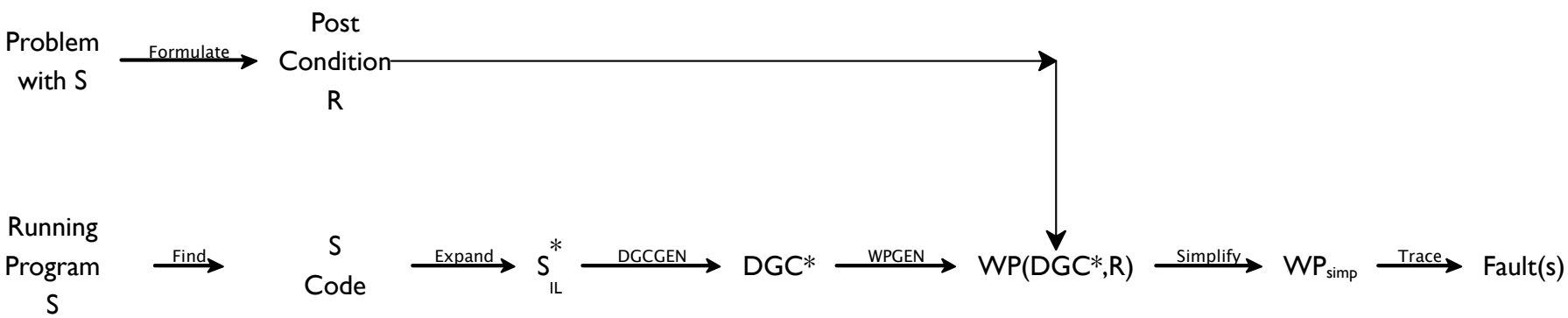


Byron Davies, Ph.D. http://ontopilot.com

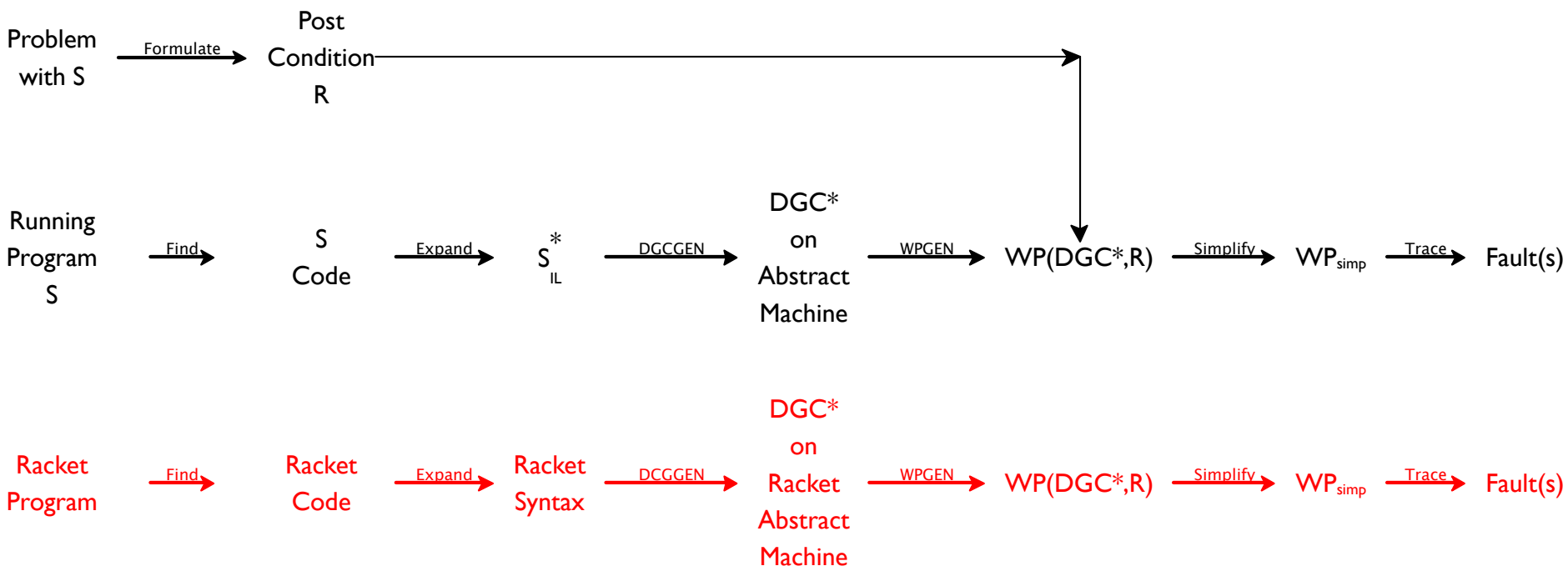
Tool Development
for
System Integrity Analysis



Tool Development
for
System Integrity Analysis



Tool Development for System Integrity Analysis



Byron Davies, Ph.D. http://ontopilot.com

identifier=? <nothing> Clear Properties

Expanded Syntax Unroll

```
(define-values (result4)
  (#%app
   (#%top . hysteresis)
   (#%top . a)
   (#%top . b)
   (#%top . c)
   (#%top . d)
   (#%top . e)
   (#%top . f)))
```

Term Syntax Object

No syntax selected

identifier=? <nothing> Clear Properties

Unrolled Syntax Flatten Globals

```
(begin
  (define-values (result4) '*unassigned*)
  (begin
    (define-values (x) '*unassigned*)
    (define-values (dx) '*unassigned*)
    (define-values (x1) '*unassigned*)
    (define-values (xh) '*unassigned*)
    (define-values (y1) '*unassigned*)
    (define-values (yh) '*unassigned*)
    (set! x (#%top . a))
    (set! dx (#%top . b))
    (set! x1 (#%top . c))
    (set! xh (#%top . d))
    (set! y1 (#%top . e))
    (set! yh (#%top . f))
    (begin
      (define-values (y) '*unassigned*)
      (begin
        (define-values (g1) '*unassigned*)
        (set! g1 (#%app < x xh))
        (if g1
          (begin
            (define-values (g2) '*unassigned*)
            (set! g2 (#%app > x x1))
            (if g2
              (begin
                (define-values (g3) '*unassigned*)
```

Term Syntax Object

No syntax selected

identifier=? <nothing> Clear Properties

Flattened Syntax DGC Path Slice

```
(set! g1 (#%app < x xh))
(if g1
  (begin
    (define-values (g2) '*unassigned*)
    (set! g2 (#%app > x x1))
    (if g2
      (begin
        (define-values (g3) '*unassigned*)
        (set! g3 (#%app > dx (quote 0.0)))
        (if g3 (set! y y1) (set! y yh)))
      (set! y y1)))
  (begin
    (define-values (g4) '*unassigned*)
    (set! g4 (#%app > x xh))
    (if g4
      (set! y yh)
      (begin
        (define-values (g5) '*unassigned*)
        (define-values (g6) '*unassigned*)
        (set! g6 (#%app > x x1))
        (if g6
          (set! g5 (#%app > dx (quote 0.0)))
          (set! g5 '#f))
        (if g5 (set! y y1) (set! y yh)))))
(set! result4 y)
(unbind-vars (y)))
```

Term Syntax Object

Apparent identifier binding
Not applicable

Binding if used for #%app
in phase 0:
defined in: "kw.rkt" <= "pre-base.rkt" <= "private/base.rk
as: new-app
imported from: mischief
in phase 1 (transformer phase):
defined in: "kw.rkt" <= "pre-base.rkt" <= "private/base.rk

identifier=?

DGC Pa

```
(extend
(set-va
(dgc-if
(dgc-s
(dgc-s
(not
(exte
(exte
(set-
(dgc-
(dgc
g6
(se
g5
(#
*g
(dgc
(no
(se
(dgc-
(dgc
(dgc
(no
(se
(set-vari
(unbind-v
```

No syntax

SIA Browser

identifier=? <nothing> Clear Properties

Flattened Syntax DGC Path Slice

```
(set! g1 (#%app < x xh))
(if g1
  (begin
    (define-values (g2) '*unassigned*)
    (set! g2 (#%app > x xl))
    (if g2
      (begin
        (define-values (g3) '*unassigned*)
        (set! g3 (#%app > dx (quote 0.0)))
        (if g3 (set! y yl) (set! y yh)))
      (set! y yl)))
  (begin
    (define-values (g4) '*unassigned*)
    (set! g4 (#%app > x xh))
    (if g4
      (set! y yh)
      (begin
        (define-values (g5) '*unassigned*)
        (define-values (g6) '*unassigned*)
        (set! g6 (#%app > x xl))
        (if g6
          (set! g5 (#%app > dx (quote 0.0)))
          (set! g5 '#f))
        (if g5 (set! y yl) (set! y yh)))))
(set! result4 y)
(unbind-vars (y)))
```

identifier=? <nothing> Clear Properties

DGC Path Slice

```
(extend-environment g4 *global-env*)
(set-variable g4 (#%app > x xh) *global-env*)
(dgc-if
 (dgc-seq g4 (set-variable y yh *global-env*))
 (dgc-seq
  (not g4)
  (extend-environment g5 *global-env*)
  (extend-environment g6 *global-env*)
  (set-variable g6 (#%app > x xl) *global-env*)
  (dgc-if
   (dgc-seq
    g6
    (set-variable
     g5
     (#%app > dx (quote 0.0))
     *global-env*))
   (dgc-seq
    (not g6)
    (set-variable g5 '#f *global-env*)))
  (dgc-if
   (dgc-seq g5 (set-variable y yl *global-env*))
   (dgc-seq
    (not g5)
    (set-variable y yh *global-env*)))))))
(set-variable result4 y *global-env*)
(unbind-vars (y)))
```

identifier=? <nothing> Clear Properties

Source Expand

```
(define arr (vector x y z))
(define (array_example a j)
  (define i 1)
  (vector-set! a i (+ 1 (vector-ref a i)))
  (vector-set! a j (+ 1 (vector-ref a 1))))
(define result3 (array_example arr 2))
(define (hysteresis x dx xl xh yl yh)
  (let [(y
         (if (< x xh)
           (if (> x xl) (if (> dx 0.0) yl yh) yl)
           (if (> x xh)
             yh
             (if (and (> x xl) (> dx 0.0))
               yl
               yh))))]
    y))
(define-values (a b c d e f)
  (values 1 2 3 4 5 6))
(define result4 (hysteresis a b c d e f))
(define zero 0.0)
(define one (+ zero 1.0))
(define (hysteresis-z x dx xl xh yl yh)
  (let [(y
         (if (< x xh)
           (if (> x xl)
             (if (> dx zero) yl yh)
             yl)
```

Weakes

```
(and
```

Term Syntax Object

Apparent identifier binding
Not applicable

Binding if used for #%app
in phase 0:
defined in: "kw.rkt" <= "pre-base.rkt" <= "private/base.rk
as: new-app
imported from: mischief
in phase 1 (transformer phase):
defined in: "kw.rkt" <= "pre-base.rkt" <= "private/base.rk

Term Syntax Object

No syntax selected

Term Syntax Object

Apparent identifier binding
Not applicable

Binding if used for #%app
none

No s

Byron Davies, Ph.D. http://ontopilot.com

Properties | identifier=? <nothing> | Clear | Properties

Source | Expand

```
(define arr (vector x y z))
(define (array_example a j)
  (define i 1)
  (vector-set! a i (+ 1 (vector-ref a i)))
  (vector-set! a j (+ 1 (vector-ref a 1))))
(define result3 (array_example arr 2))
(define (hysteresis x dx xl xh yl yh)
  (let [(y
         (if (< x xh)
           (if (> x xl) (if (> dx 0.0) yl yh) yl)
           (if (> x xh)
             yh
             (if (and (> x xl) (> dx 0.0))
               yl
               yh))))]
    y))
(define-values (a b c d e f)
  (values 1 2 3 4 5 6))
(define result4 (hysteresis a b c d e f))
(define zero 0.0)
(define one (+ zero 1.0))
(define (hysteresis-z x dx xl xh yl yh)
  (let [(y
         (if (< x xh)
           (if (> x xl)
             (if (> dx zero) yl yh)
             yl)
```

Term | Syntax Object

Apparent identifier binding
Not applicable

Binding if used for #%app
none

Properties | identifier=? <nothing> | Clear | Properties

Weakest Precondition

```
(and (=>
      (< a d)
      (=> (> a c) (=> (> b 0.0) (= e a0))))
     (=>
      (< a d)
      (=> (> a c) (=> (not (> b 0.0)) (= f a0))))
     (=> (< a d) (=> (not (> a c)) (= e a0)))
     (=> (not (< a d)) (=> (> a d) (= f a0)))
     (=>
      (not (< a d))
      (=>
       (not (> a d))
       (=> (> a c) (=> (> b 0.0) (= e a0)))))
     (=>
      (not (< a d))
      (=>
       (not (> a d))
       (=> (> a c) (=> (not (> b 0.0)) (= f a0)))))
     (=>
      (not (< a d))
      (=>
       (not (> a d))
       (=> (> a c)) (=> '#f (= e a0)))))
     (=>
      (not (< a d))
      (=>
       (not (> a d))
       (=>
```

Term | Syntax Object

No syntax selected

Properties | identifier=? <nothing> | Clear | Properties

Syntax Path | Slice

```
(fi
 ((set! g5 (#%app > dx (quote 0.0))) (guard g6))
 ((set! g5 '#f) (guard (not g6))))
(set! g6 (#%app > x xl))
(define-values (g6) '*unassigned*)
(define-values (g5) '*unassigned*)
(guard (not g4))
(set! g4 (#%app > x xh))
(define-values (g4) '*unassigned*)
(guard (not g1))
(set! g1 (#%app < x xh))
(define-values (g1) '*unassigned*)
(define-values (y) '*unassigned*)
(set! yh (#%top . f))
(set! yl (#%top . e))
(set! xh (#%top . d))
(set! xl (#%top . c))
(set! dx (#%top . b))
(set! x (#%top . a))
(define-values (yh) '*unassigned*)
(define-values (yl) '*unassigned*)
(define-values (xh) '*unassigned*)
(define-values (xl) '*unassigned*)
(define-values (dx) '*unassigned*)
(define-values (x) '*unassigned*)
(define-values (result4) '*unassigned*))
```

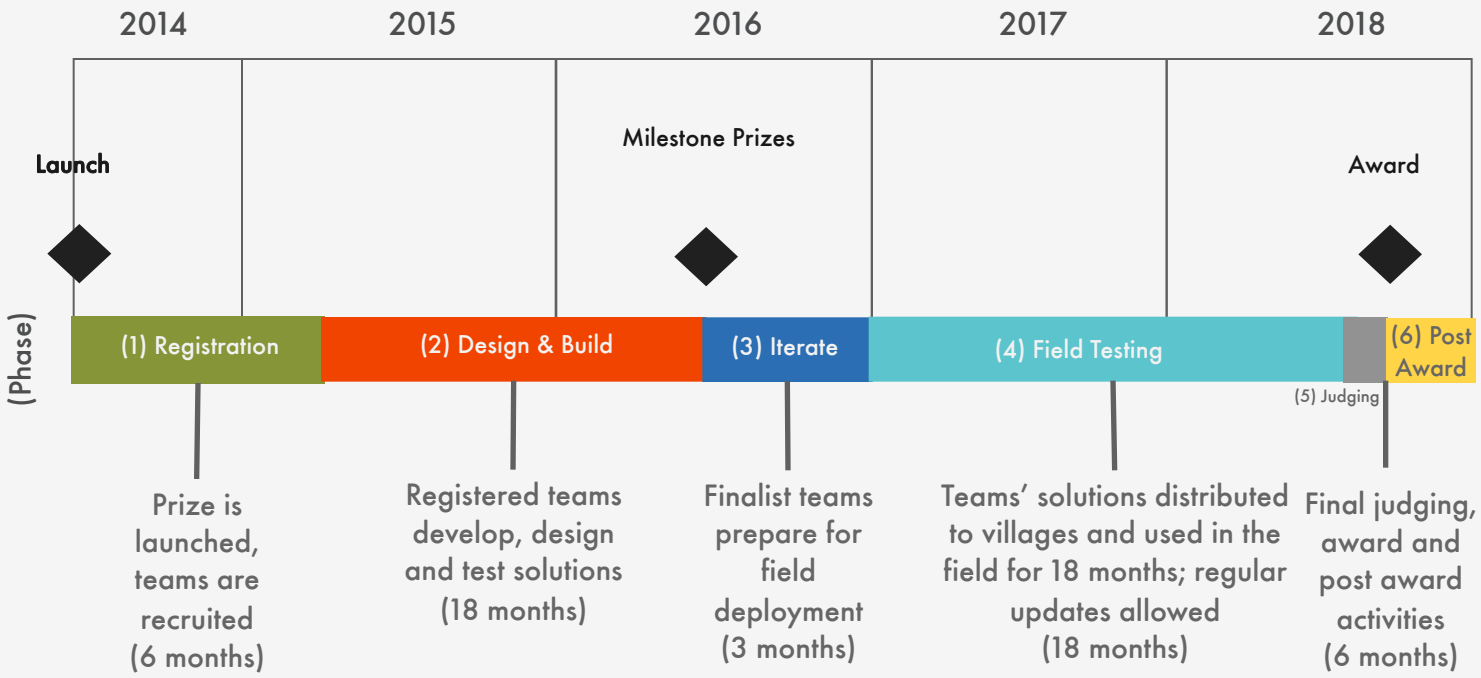
Term | Syntax Object

No syntax selected

UNLOCKING MINDS.
EMPOWERING FUTURES.
GLOBAL LEARNING XPRIZE
Empowering Young Minds Everywhere

GLOBAL LEARNING XPRIZE®

EMPOWERING YOUNG MINDS EVERYWHERE


2014
2015
2016
2017
2018
Launch
Milestone Prizes
Award
(Phase)
(1) Registration
(2) Design & Build
(3) Iterate
(4) Field Testing
(5) Judging
(6) Post Award
Prize is launched, teams are recruited (6 months)
Registered teams develop, design and test solutions (18 months)
Finalist teams prepare for field deployment (3 months)
Teams' solutions distributed to villages and used in the field for 18 months; regular updates allowed (18 months)
Final judging, award and post award activities (6 months)

forum.xprize.org

STARSHINE GLEE

Racket as the Platform for Winning the Global Learning XPRIZE

September 27, 2015

STARSHINE GLEE

Why Are We Competing?

- Education is broken, globally.
- We don't want the XPRIZE to be exclusively about reading, writing, and arithmetic (and test scores).
- We have a great team of systems thinkers, computer scientists, arts integrators, and educators.
- We have our own laboratory for testing solutions.
- The solution will be as valuable for our students as for preliterate and illiterate children around the world.

STARSHINE GLEE

How Are We Competing?

- Systemic solution: it's not just about technology, but about learning, children, their peers, their community.
- The order of things:
  - A theory of learning
  - Content and internationalization
  - Experience design
  - System design
  - Programming

STARSHINE GLEE

Progress

- We researched “Black Swans of Learning.”
- We found very promising non-technology solutions in math and reading, with high potential for automation
- Central Theme: “Guided Discovery”
- We have developed relationships with key collaborators, on both coasts and in Canada.
- We have begun testing the pre-technology solutions at StarShine Academy.

Why Racket

- Why use anything but the second-best programming language?
- Powerful, cohesive, solid, reliable, interactive, graphical, parenthesized programming and execution environment — batteries included
- E.g., picts! (in the REPL)
- E.g., Pollen

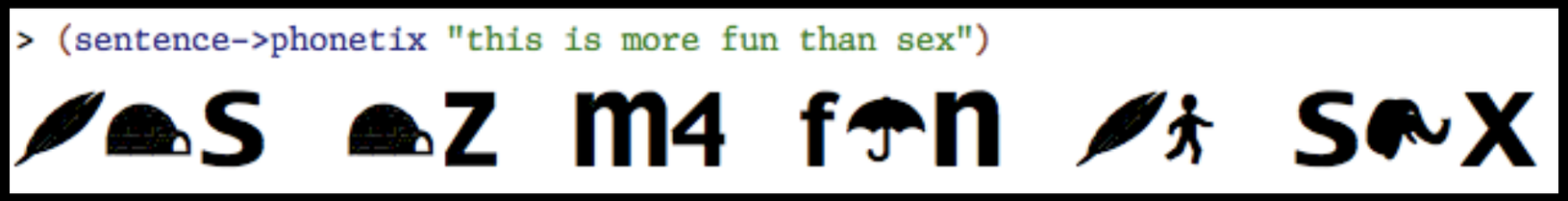
Why Racket

STARSHINE GLEE

Why Racket

STARSHINE GLEE

```
> (sentence->phonetix "this is more fun than sex")
```

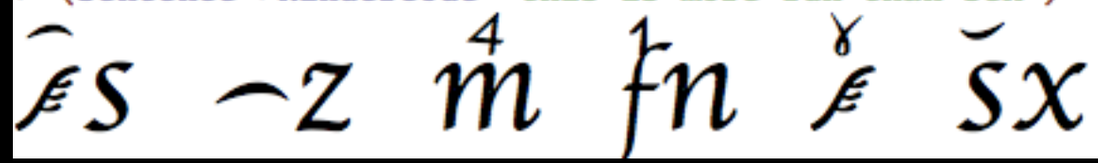


STARSHINE GLEE

Why Racket

```
> (sentence->phonetix "this is more fun than sex")
```

```
> (sentence->kindercode "this is more fun than sex")
```

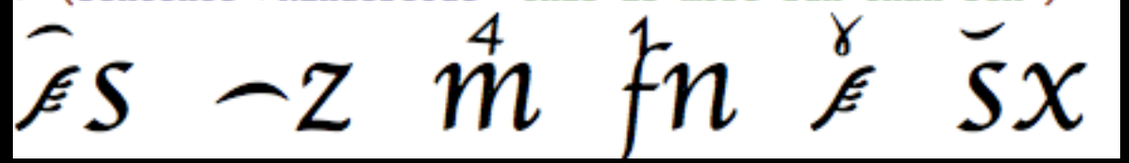


Why Racket

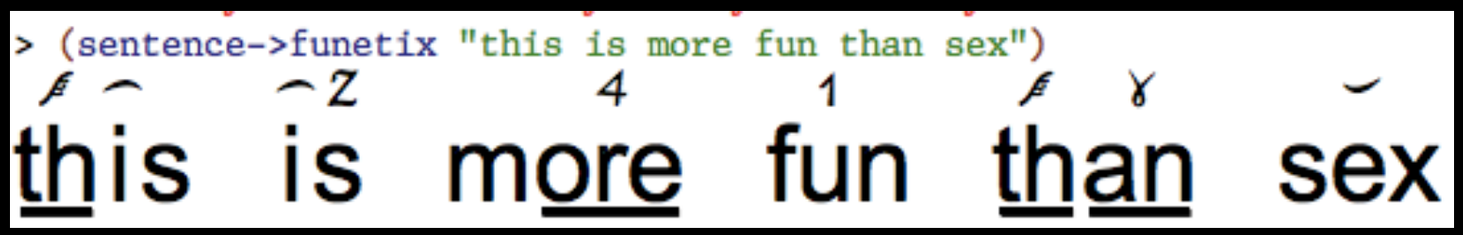
STARSHINE GLEE

```
> (sentence->phonetix "this is more fun than sex")
```

```
> (sentence->kindercode "this is more fun than sex")
```



```
> (sentence->funetix "this is more fun than sex")
```

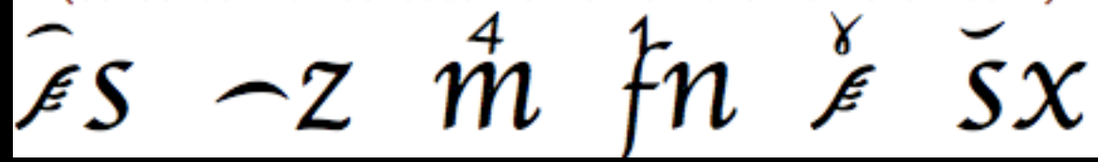


Why Racket

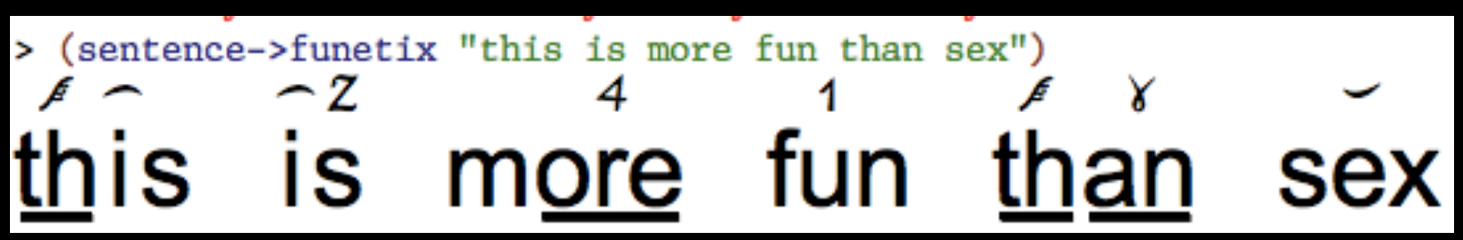
STARSHINE GLEE

```
> (sentence->phonetix "this is more fun than sex")
```

```
> (sentence->kindercode "this is more fun than sex")
```




```
> (sentence->funetix "this is more fun than sex")
```



STARSHINE GLEE

Projects

- Porting Racket to Android
- Implementing Guided Discovery for math, early reading, and systems thinking
- Architecture for managing a top-level learning loop: present, prompt, problem solve, assess.
- Math exercise generator to generate a tablet-based experience from a database of thousands of printed exercises.
- Publishing phonetically annotated versions of existing books.

STARSHINE GLEE

StarShinePlanet.com