# A SWIFTLY TILTING PARSER

## (in memory of Madeleine L’Engle)

https://github.com/robrix/A-Swiftly-Tilting-Parser
rob.rix@github.com ❧ @rob_rix

# THE DERIVATIVE of PARSERS

in

# OBJECTIVE-C

&

# SWIFT

# PARSER COMBINATORS are NOT SCARY

We’ll use “parser” as a synonym

- Executable LEGOs for parsing text
  - Each one is a tiny program
  - Some parse input directly
  - Some combine other parsers
- Together, they match specific patterns

# THE DERIVATIVE of PARSERS is ALSO NOT SCARY

- Might, Darais, & Spiewak's 2011 paper *Parsing with Derivatives—a Functional Pearl*
- *Recognizes* and *parses* programming languages*
    - Recognizing: "is my input valid?"
    - Parsing: "how is the input structured?"
- Validity and structure are defined by the grammar, which is made of parser combinators

# BREAKING it DOWN

- Parsing
- Derivative
- Nullability
- Parse forest
- Compaction

# PARSING with DERIVATIVES: NOT SCARY

- Go through the input character by character
- At each step, compute the derivative of the parser
- Return the parsed input as a parse tree*

*Technically, parse *forest*

*parsing in Objective-C and Swift*

# DERIVATIVE is SLIGHTLY SCARY

- Returns the parser that would match *after* the current one
- Stores matched input in parse trees
- On failure, returns the empty parser
- Holds eye contact slightly longer than comfortable

*derivative in Objective-C and Swift*

# INTERLUDE: RECURSION is KIND OF SCARY

# CONTEXT-FREE LANGUAGES are RECURSIVE

- NB: Not just the types: the object graph is cyclic!
- The only difference from regular expressions
- Key to why you can't parse arbitrary HTML with a regexp
- Regexps can be matched with a list, but context-free languages need a stack
- Naïve implementations will infinite loop 💥

# PROTECTING your PARSERS from NONTERMINATION 😎

1. **Laziness** 😴

# LAZINESS 😴

## DO ONLY WHAT YOU MUST, ONLY WHEN YOU MUST

- Evaluate the parsers in alternations, concatenations, repetitions, & reductions at the last moment
- Avoids nontermination when constructing the derivative
- Necessary to even construct cyclic grammars!

*laziness in Objective-C and Swift*

# PROTECTING your PARSERS from NONTERMINATION 😎

1. Laziness 😴
2. **Memoization** 📎

# MEMOIZATION 📎

## WHEN YOU DO IT RIGHT, YOU ONLY DO IT ONCE

- Memoize ≅ cache
- The first time you call a memoized function with a set of arguments, it stores the results
- The next time, it just looks them up
- Can store results in a dictionary or an ivar
- Allows the derivative to “tie the knot” when building a cyclic grammar *from* a cyclic grammar

*memoization in Objective-C and Swift*

# NULLABILITY is NOT SCARY AT ALL

- “Can it match the empty string?”
- Equivalent: “Can it match at the end of the input?”
- Equivalent: “Can it be skipped?”

*nullability in Objective-C and Swift*

# BUT SUDDENLY: NONTERMINATION

# NULLABILITY is ~~NOT~~ **ACTUALLY QUITE SCARY** ~~AT ALL~~

- Nullability walks the grammar *eagerly*, defeating laziness 😴
- Nullability computes pass/fail, not a structure; e.g.:

$$\delta(L) = \delta(L)\ \alpha \mid \epsilon$$

  - It can't finish $\delta(L)$ before recurring: nontermination 💥
  - Thus defeating memoization 📎

# PROTECTING your PARSERS from NONTERMINATION 😎

1. Laziness 😴
2. Memoization 📎
3. ~~***Math***~~ **Fixed points** 🔨☝️

# ~~*MATH*~~ FIXED POINTS 🔨☝🏼

NOW *THIS* is SCARY ✅

# FIXED POINTS at a GLANCE

- If $f(x) = x$, then $x$ is a fixpoint of $f$; e.g. $x^2$ is fixed at 0 and 1
- $\delta(L)$ is null if its argument is nullable, empty otherwise
- A fixpoint of δ is therefore either null or empty (true/false)
- Define $\delta(L) = \delta(L)\ \alpha \mid \epsilon$ as the *least* fixed point of δ
- Iterate $\delta^n(L)$ from $\delta^0(L) = \texttt{false}$ until $\delta^n(L) = \delta^{n-1}(L)$ (Kleene fixpoint theorem)

# CONSEQUENCES of KEEPING it KLEENE ⚠️

- Computing $\delta(L)$ is doing work
- Computing $\delta(\delta(L))$ is doing *more* work
- $\delta$ is worst-case $O(G)$ where *G* is the size of the grammar
- If this is measurable in time, we lose performance
- If visiting any parser causes side-effects (💥), they'll be performed twice → potentially wrong results
  - ("So don't do that")

# CONJECTURE: NULLABILITY must CONVERGE in a SINGLE ITERATION

- If δ returns Boolean, we start with $\delta^0(L)$ `= false`
- $\delta^1(L)$ must be either `true` or `false`
  - If `false`, we're done
  - Otherwise, $\delta^2(L)$ is `true` (we're done), or `false` (implying non-monotone, invalidating use of Kleene fixpoint theorem)
  - $\therefore$ We never have to compute $\delta^2(L)$

*fixpoints in Objective-C and Swift*

# PARSE FOREST is KINDLY and ATTENTIVE

- Constructs and returns the matched parse trees
- Applies reductions
  - This is how you construct *your* objects

*parse forest in Objective-C and Swift*

# PARSING ~~with DERIVATIVES~~ without COMPACTION

*The implementation is brief. The code is pure. The theory is elegant. So, how does this perform in practice? In brief, it is awful.*

- Derivative of concatenation doubles grammar size
- Worst case: $O(2^{2n}G^2)$ : $G$ = grammar size, $n$ = input length 💥

# COMPACTION is QUICK

- Replace complex parsers with simpler equivalents
- Enables better performance
  - Worst case unchanged
  - Expected case (unambiguous grammars) is O($nG$)
  - (Competitive with other general solutions)

*compaction in Objective-C and Swift*

# COMPACTION is AMBITIOUS

- Generally must compact after derivative, or else cyclic → 💥
  - Can we avoid complex parsers altogether in some cases?
- Enables better features
  - Incremental results: 🔢 vs. 1️⃣…2️⃣…3️⃣…4️⃣…
  - (Good) error reporting?
  - Disambiguation? ✨

# CHALLENGES common to OBJC & SWIFT

- Understanding the paper is hard 😨
- ObjC & Swift are reference counted
  - Cyclic grammars = refcycles (unless handled specially)
  - Possible solution: a refcycle-breaking combinator
- Pattern matching cyclic grammars is tricky

# CHALLENGES UNIQUE to OBJC

- *Huge* impedance mismatch between the language & algorithm
- Verbose; dense; splits functions across many files
- Pattern matching against cyclic grammars is *really* tricky
  - The language doesn't have pattern matching at all 😭
  - Implemented pattern matching for parsers *using* parsers 💫💥
- Nontermination is much harder to solve, e.g. `-isEqual:` for equal cyclic grammars

# CHALLENGES UNIQUE to SWIFT

- Beta (& evolving!) compiler & IDE 😱
  - No codegen for recursive enums/`structs`, classes with non-fixed layouts, & enums with multiple non-fixed layouts
  - Crash-happy 😂💥 (as of Xcode 6b2)
  - Unbelievably broken error reporting (ProTip™: extract nested expressions into constants to isolate issues)
- Some language design/prioritization choices need workarounds
  - No best practices, so making those up as I go ✈️💺👖

# BENEFITS of SWIFT vs. OBJC

- Much better tool:job match
  - enums make better parsers than inheritance does
  - Pattern matching ❤️
  - Operator overloading for constructing parsers ✨
- Stronger typing → safer program 💪🏼
- Lets me solve *my* problem, not incidental ones 🙌🏼
- Enables me to make mistakes faster & with greater confidence 🎢

# BENEFITS of OBJC vs. SWIFT

- ObjC is stable
- `clang` is stable
- Familiarity
- Unlikely to break the code on the day of the talk 😆

# SUBTLETIES favouring OBJC?

- Tougher defining parse trees' type in Swift
  - ObjC: sets, pairs, input characters, & AST, it's all just `id`
  - However: easy ≠ *good*
- Can use macros & dynamic proxies in ObjC
  - No real equivalents in Swift
  - *Had* to use macros & dynamic proxies in ObjC

# SUBTLETIES favouring SWIFT?

- Much more readable because of enum/pattern matching
  - Didn’t actually know if this approach would work < 1w ago 😰
  - Would’ve required the ObjC solution, with a buggy compiler 😡
- `@auto_closure` & operator overloading cleans up grammar construction
  - Potentially masks refcycles
  - Hard to break cycles automatically; very hard to do manually

# ADVANTAGE: SWIFT

# EPILOGUE: AMBIGUITY is *TERRIFYING*

- Eats RAM, souls
- Fastest, least productive way to use 10 GB of RAM
- Easy to introduce, hard to locate in the grammar, harder to solve without rewriting the grammar → breaking assumptions about parse tree structure
- The literature on disambiguation is appropriately vast
- Disambiguation via compaction (& reductions?) is going to be fun to explore

¿Q&A!

# THANKS

- The Swift team at Apple
- Matt Might
- Kelly Rix
- David Smith
- You ❤️

https://github.com/robrix/A-Swiftly-Tilting-Parser
rob.rix@github.com ❧ @rob_rix