

A SWIFTLY TILTING PARSER

(in memory of Madeleine L'Engle)

<https://github.com/robrix/A-Swiftly-Tilting-Parser>

rob.rix@github.com 🐙 @rob_rix

PARSER COMBINATORS

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

- Might, Darais, & Spiewak's 2011 paper *Parsing with Derivatives—a Functional Pearl*
- *Recognizes and parses* context-free 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

OPERATIONS

- Parsing
- Derivative
- Nullability
- Parse forest
- Compaction

PARSING

- Go through the input character by character
- At each step, compute the derivative of the parser
- Compact it
- Use it for the next step
- Return the parsed input as a parse forest


parsing in Objective-C and Swift

DERIVATIVE

- Returns the parser that would match *after* the current one
- Stores matched input in parse trees
- On failure, returns the empty parser
- Different definition for each kind of parser

derivative in Objective-C and Swift

RECURSION & NONTERMINATION

- Context-free languages & grammars are recursive
- NB: Not just the *types*: the object graph is cyclic!
- Key to why you can't parse arbitrary HTML with a regexp
- Regexp 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, WHEN YOU MUST

- Alternations, concatenations, repetitions, & reductions use closures to delay evaluation
- 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

- The first time you call a memoized function with a set of arguments, it stores the results
- The next time, it looks them up; memoize \cong cache
- Can store results in a dictionary, ivar, etc.
- Allows the derivative to “tie the knot” when building a cyclic grammar *from* a cyclic grammar


memoization in Objective-C and Swift

NULLABILITY

- “Is this grammar nullable?” = “Will it match an empty string?”
- Equivalent: “Can it match at the end of the input?”
- Equivalent: “Can it be skipped?”

nullability in Objective-C and Swift

NULLABILITY and NONTERMINATION

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

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

Can't finish & memoize before recurring, thus defeating memoization 

PROTECTING your PARSERS from NONTERMINATION 😎

1. Laziness 🤪

2. Memoization 📎

3. **~~Math~~** Fixed points 🛠️👉

~~MATH~~ FIXED POINTS 🛠️👉

- If $f(x) = x$, f is fixed at x ; x^2 is fixed at 0 and 1
- If L is nullable, $\delta(L)$ is null, otherwise empty
- Any fixpoints of δ are likewise either null or empty
- Interpret $\delta(L) = \delta(L) \ \alpha \mid \epsilon$ as a fixpoint of δ
- Iterate $\delta^n(L)$ from $\delta^0(L) = \text{false}$ until $\delta^n(L) = \delta^{n-1}(L)$ (Kleene fixpoint theorem)

fixpoints in Objective-C and Swift


PARSE FOREST

- Construct and return any matched parse trees
- Apply reductions
 - This is how you construct *your* objects
- If > 1 parser matched the input, > 1 parse tree in the parse forest
 - This means there's ambiguity in the grammar 🤔

*parse forest in Objective-C and
Swift*

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: $O(2^{2^n}G^2)$: G = grammar size, n = input length 

COMPACTION

- Replace complex parsers with simpler equivalents
- Enables better performance
 - Worst case still terrible
 - Expected case (unambiguous grammars) is $O(nG)$
 - Quite reasonable in practice; *no* algorithm is fast under ambiguity

compaction in Objective-C and Swift

COMPACTION in the FUTURE

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

CHALLENGES in OBJC & SWIFT

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

CHALLENGES UNIQUE to OBJC

- Language/algorithm impedance mismatch
- Verbose; dense; splits functions across many files
- Pattern matching cyclic grammars is *really* tricky
 - The language doesn't have pattern matching 😭
 - 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 yet for some features
- Crash-happy 😂💥 (as of Xcode 6b2)
- Bad error reporting (ProTip™: extract nested expressions into constants to isolate issues)
- Some language design/prioritization choices need workarounds
- Making it up as I go 🛩️🪑👖

BENEFITS of SWIFT vs. OBJC

- Much better tool:job match
 - enum is a better fit than classes for parsers 👍
 - Pattern matching 😍
 - Operator overloading for constructing parsers ✨
- Stronger typing → safer, better program 💪
- I solve *my* problems more; incidental ones less 🙌
- Enables me to make mistakes faster, & with greater confidence 🎢

BENEFITS of OBJC vs. SWIFT

- ObjC is stable
- clang is stable
- Familiarity
- Unlikely to break my code on the day of the talk 😏

SUBTLETIES: OBJC > SWIFT...?

- It was initially hard describing parse trees' type in Swift
 - ObjC: sets, pairs, input, & AST are all `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: SWIFT > OBJC...?

- *Much* more readable with enum/pattern matching
 - Wasn't sure this approach would work 1w ago 🥵
 - If not, same solution as ObjC, with beta tools 😡
- @auto_closure & operators are ✨ for grammars
 - Potentially masks refcycles
 - Hard to break cycles automatically *or* manually

ADVANTAGE: SWIFT

- Objective-C is the wrong tool for the job
- Swift can access Apple's frameworks & customers
- Much more sound theoretically
- Much more sound practically
 - Safer
 - More productive
 - Increasingly, *fast fast fast*

¿Q&A!

THANK YOU! 🙌

David Darais, Matt Might, Kelly Rix,
David Smith, Daniel Spiewak,
the Swift team, & especially you 💖

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