How to not write parsers

Wow, that’s a lot of grammars, too bad I’m not reading them

Curtis Chin Jen Sem

Scan for these slides

It started with wanting `.cabal` files to look nice

- syntax-highlighting is nice.
- started with wanting it for `.cabal` files in my editor helix.
  - pre-existing `.cabal` tree-sitter grammar by Magnus Therning

It started with wanting `.cabal` files to look nice

- syntax-highlighting is nice.
- started with wanting it for `.cabal` files in my editor helix.
  - pre-existing `.cabal` tree-sitter grammar by Magnus Therning

It started with wanting `.cabal` files to look nice

- syntax-highlighting is nice.
- started with wanting it for `.cabal` files in my editor helix.
  - pre-existing `.cabal` tree-sitter grammar by Magnus Therning

- a PR by Ananda Umamil fixed the segfaults
- but then I wanted `cabal.project`, which look like `.cabal` files
- Grammar?

Is this a valid `.cabal` file?

```
Library {
    build-depends: base ⩾ 4.0 && < 4.2,
                   syb  ⩾ 0.1 && < 0.2

    if impl(ghc < 6.9) {
        buildable: False
    }
}
```

Is this a valid `.cabal` file?

```
Library {
    build-depends: base ⩾ 4.0 && < 4.2,
                   syb  ⩾ 0.1 && < 0.2

    if impl(ghc < 6.9) {
        buildable: False
    }
}
```

`cabal` says **valid, no warnings**

- lots of legacy syntax
- there is a whole brace-delimited syntax

How about this one?

```
library
    build-depends: base

-- Duplicating inclusion plugin conditions
    if flag(eval)
        cpp-options: -Dhls_eval
```

How about this one?

```
library
    build-depends: base

-- Duplicating inclusion plugin conditions
    if flag(eval)
        cpp-options: -Dhls_eval
```

`cabal` says **valid, no warnings**

- a comment at column zero does not end the stanza
- the `if` still belongs to the `library` above it
- in `haskell-language-server.cabal`

And these? Which of these is right?

```
library
  if os(windows)
    build-depends: Win32
  elif os(linux)
    build-depends: unix
```

```
library
  if os(windows)
    build-depends: Win32
  elseif os(linux)
    build-depends: unix
```

And these? Which of these is right?

```
library
  if os(windows)
    build-depends: Win32
  elif os(linux)
    build-depends: unix
```

`cabal` **valid, since `cabal-version: 2.2`**

```
library
  if os(windows)
    build-depends: Win32
  elseif os(linux)
    build-depends: unix
```

`cabal` ***invalid subsection*, ignored**

- `elif` is the real one.
- `elseif` still parses, warns, and drops the branch
- both successfully `cabal build`!

That’s a lot of legacy

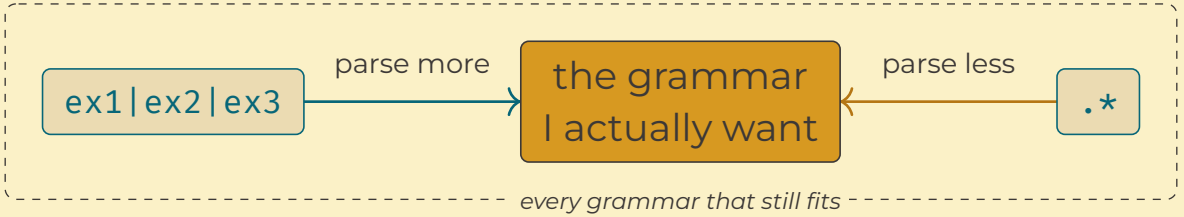
- twenty years of accumulated legacy
- writing parsers is already hard enough
- coverage is a problem

That’s a lot of legacy

- twenty years of accumulated legacy
- writing parsers is already hard enough
- coverage is a problem
- so what if I don’t?
- instead of a parsing problem,

That’s a lot of legacy

- twenty years of accumulated legacy
- writing parsers is already hard enough
- coverage is a problem
- so what if I don’t?
- instead of a parsing problem,
- … how about tackling it like a constraint solving problem?



Tree-sitter

- a parser generator DSL for incremental parsers
- grammar in javascript, generates parsers in C
- out comes a concrete syntax tree

```
library
    build-depends: base ≥ 4.9
```

```
(library
  type: (section_type)
  properties: (property_or_conditional_block
    (field
      name: (field_name)
      value: (field_value
        (identifier)
        (constraint_op)
        (version)))))
```

Tree-sitter

- always returns a tree even if syntax is invalid

```
library
    build-depends base ⩾ 4.9
                  ↑ no colon
```

```
(library
  type: (section_type))
(ERROR
  (field_name)
  (identifier)
  (constraint_op)
  (version))
```

Tree-sitter

- always returns a tree even if syntax is invalid

```
library
    build-depends base ⩾ 4.9
                  ↑ no colon
```

```
(library
  type: (section_type))
(ERROR
  (field_name)
  (identifier)
  (constraint_op)
  (version))
```

- most common used for syntax highlighting
- editors have to highlight actively edited files
- also used for tags, tree-based navigation, code-folding

Pattern matching parse trees

```
library core
    build-depends: base
```

its parse tree

```
(cabal
  (sections
    (library
      type: (section_type)
      name: (section_name)
      properties: (property_or_conditional_block
        (field
          name: (field_name)
          value: (field_value
            (identifier)))))))
```

query

```
(library
  name: (section_name) @name)
```

Pattern matching parse trees

```
library core
    build-depends: base
```

its parse tree

```
(cabal
  (sections
    (library
      type: (section_type)
      name: (section_name)
      properties: (property_or_conditional_block
        (field
          name: (field_name)
          value: (field_value
            (identifier)))))))
```

query

```
(library
  name: (section_name) @name)
```

`@name` captures **`core`**

- change the capture to `type: (section_type)` and you get `library`

Using captures

- going from captures to behavior

`pkg.cabal`, three stanzas:

```
library core
    build-depends: base
executable mypkg
    ghc-options: -O2
flag fast
    default: False
```

`tags.scm`, one pattern per symbol kind

```
(library     ...  @definition.module
(executable  ...  @definition.function
(flag        ...  @definition.constant
... is  name: (section_name) @name)
```

what a tags reader gets

```
→  core  module
→  mypkg  function
→  fast  constant
```

- the outer node decides which stanza matches
- `@name` gives the symbol's text, `definition.<kind>` gives its kind
- so `tags.scm` turns node names into definition variants

Features as queries

- pull nodes out of the tree, tag each one with a capture

query file	pulls out of the tree	which Helix exposes as	lines
`highlights.scm`	every literal, name, operator	colour	101
`textobjects.scm`	sections and their bodies	select or jump a stanza	45
`indents.scm`	section and `if` bodies	where the cursor goes on enter	32
`tags.scm`	section names	the symbol picker	15
`locals.scm`	`flag` defs and their uses	go-to-definition on a flag	11
`rainbows.scm`	parens in conditions	bracket pair colours	7

- query one single tree six different ways easily
- only have to give named nodes

Features as queries

- pull nodes out of the tree, tag each one with a capture

query file	pulls out of the tree	which Helix exposes as	lines
`highlights.scm`	every literal, name, operator	colour	101
`textobjects.scm`	sections and their bodies	select or jump a stanza	45
`indents.scm`	section and `if` bodies	where the cursor goes on enter	32
`tags.scm`	section names	the symbol picker	15
`locals.scm`	`flag` defs and their uses	go-to-definition on a flag	11
`rainbows.scm`	parens in conditions	bracket pair colours	7

- query one single tree six different ways easily
- only have to give named nodes

grammar and queries define the solution space

Can the queries even compile?

- every pattern in every query file
- resolved against the generated parser

the pattern	what `check-queries` says
`(library name: (section_name) @name)`	captures `core`
`(librari name: (section_name) @name)`	`Invalid node type "librari"`
`(library nmae: (section_name) @name)`	`Invalid field name "nmae"`

- node types and field names are checked
- the grammar is now forced to support every query

Harvesting corpora

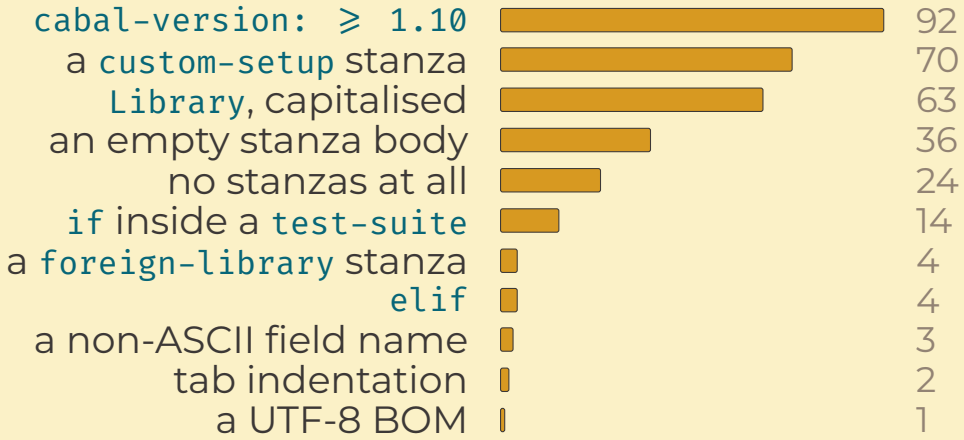
- no formal cabal grammar
- …but, there's an elaborate testsuite
- `find` over two testsuites, cabal and HLS

harvested from	files
`cabal-testsuite/PackageTests`	760
`Cabal-tests/…/ParserTests`	84
`haskell-language-server/test/testdata`	64
`cabal-install/tests`	35
the two projects' `.cabal` files	24

- 943/967 test-cases, median 14 lines

Does every real file parse?

- `parse-corpus` reads all 967 harvested files
- asserts zero `ERROR`, zero `MISSING`
- `tree-sitter-cabal` failed 237



What did I capture?

- `extract-golden` records capture, span and text over fixtures
- and diffs that against a committed set

```
library core

the capture I asked for   name  3,8-3,12  core
one node to the left      name  3,0-3,7   library
```

- the capture slid to the wrong node
- span and text both wrong

What actually reaches the screen?

- `highlight-golden` records the winning capture
- one row per token, from rendered output

the same `.`, in two fields

```
hs-source-dirs: .   ln 26  string.special.path
 description: … .   ln  8  string
```

What actually reaches the screen?

- `highlight-golden` records the winning capture
- one row per token, from rendered output

the same `.`, in two fields

```
hs-source-dirs: .   ln 26  string.special.path
 description: … .   ln  8  string
```

- one node type interpreted two different ways
- swap those patterns in the file
- and the result changes

`parse` never fails

```
readFields :: ByteString → Either ParseError [Field]   -- Cabal
parse      :: Text       → Tree                        -- tree-sitter
```

```
library
    build-depends base ≥ 4.9
                  ↑ no colon
```

parses to →

```
(library
  type: (section_type))
(ERROR
  (field_name)
  (identifier)
  (constraint_op)
  (version))
```

did it parse? always yes

`parse-corpus` constrains

`match` never chooses

```
match :: Query → Tree → [Match]
```

```
hs-source-dirs: .
                → string
                → string.special.path
```

`match` returns everything

`extract-golden` and `highlight-golden` constrain

What's my variable?

each constraint is a fold

constraints $C_1, \ldots, C_n$, each form a fold from `Tree` into some M_i,
with expected values $e_i \in M_i$ over fixtures X

$$C_i\left(\text{parse}_{\boldsymbol{g}}\ x\right) = e_i(x), \quad \forall x \in X$$

solve for $\boldsymbol{g}$, the grammar and the queries

- the constraints are the spec
- every entry is a sample
- the grammar is whatever fits

Tree constraints can be phrased as `foldMap`

```
foldMap :: Monoid m ⇒ (a → m) → [a] → m
```

constrain queries by grammar

C_1	`check-queries`	`queries ⊆ grammar`	assertion

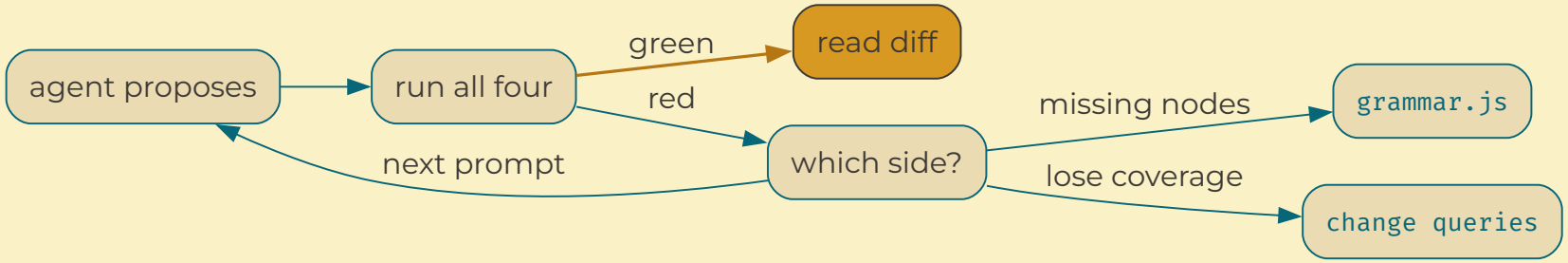
---

constrain g over files

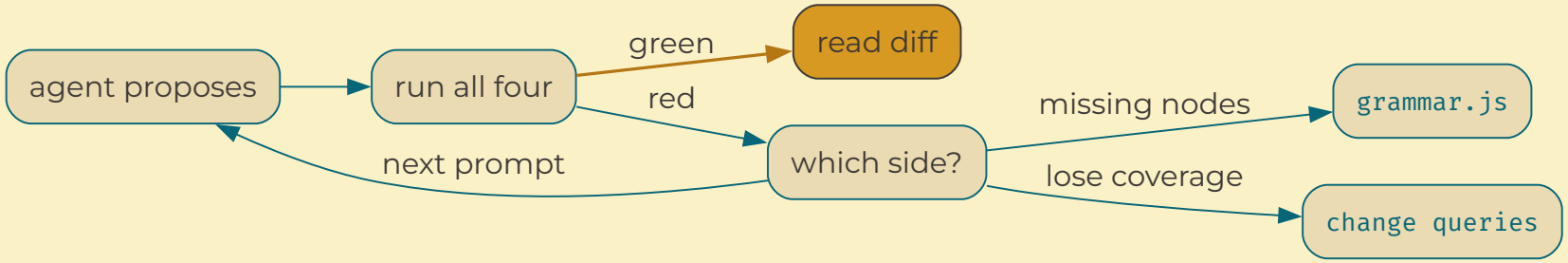
C_2	`parse-corpus`	`foldMap all . parse`	assertion	
C_3	`extract-golden`	`foldMap id . match . parse`	`value.path 25,18-25,19 '.'`	per capture
C_4	`highlight-golden`	`foldMap last . match . parse`	`26 string.special.path '.'`	per token

turn these `monoid` values into tests by rendering

Both paths are the same job for the agent



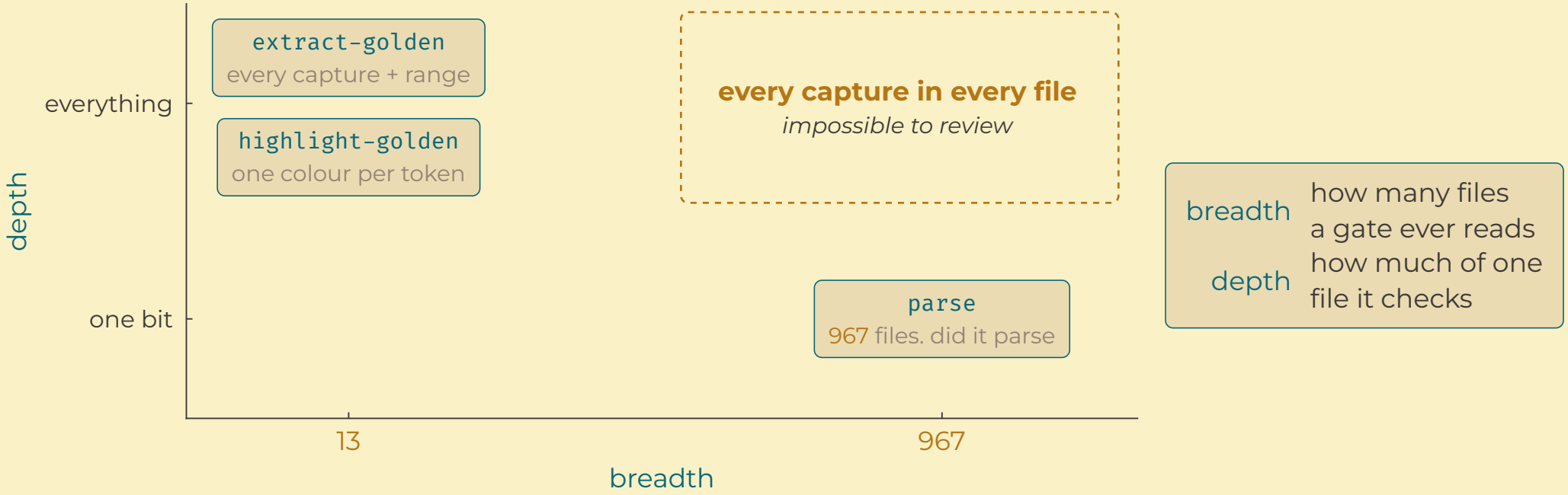
Both paths are the same job for the agent



measure, edit, re-run

- only the grammar and queries change
- manual step when everything is correct
- check is this the change I wanted?

Constraints require different sets



breadth — how many files a gate ever reads

depth — how much of one file it checks

stay with what's reviewable

I didn’t write the parser

just read everything else

---

the agent wrote
7 grammars, 4 C scanners

from
4 constraints, and 2 corpora

967 cabal files, zero `ERROR` *nodes*

same four gates built GHC’s `Core`*. nothing to fork, nothing to harvest*

Questions?

github.com/crtschin/tree-sitter-haskell-contrib

grammars for `.cabal`, `cabal.project`, and GHC's Core / STG / Cmm dumps.