FUNCTIONAL PROGRAMMING WITHOUT A FUNCTIONAL LANGUAGE

Meredith L. Patterson

May 28, 2016

Upstanding Hackers, Inc.

https://github.com/UpstandingHackers/hammer

INTRODUCTION

WHOAMI

- Linguist, security dork

WHOAMI

- Linguist, security dork
- Language-theoretic security

WHOAMI

- Linguist, security dork
- Language-theoretic security
- Parsing, let me tell you about it

WHY WE PARSE

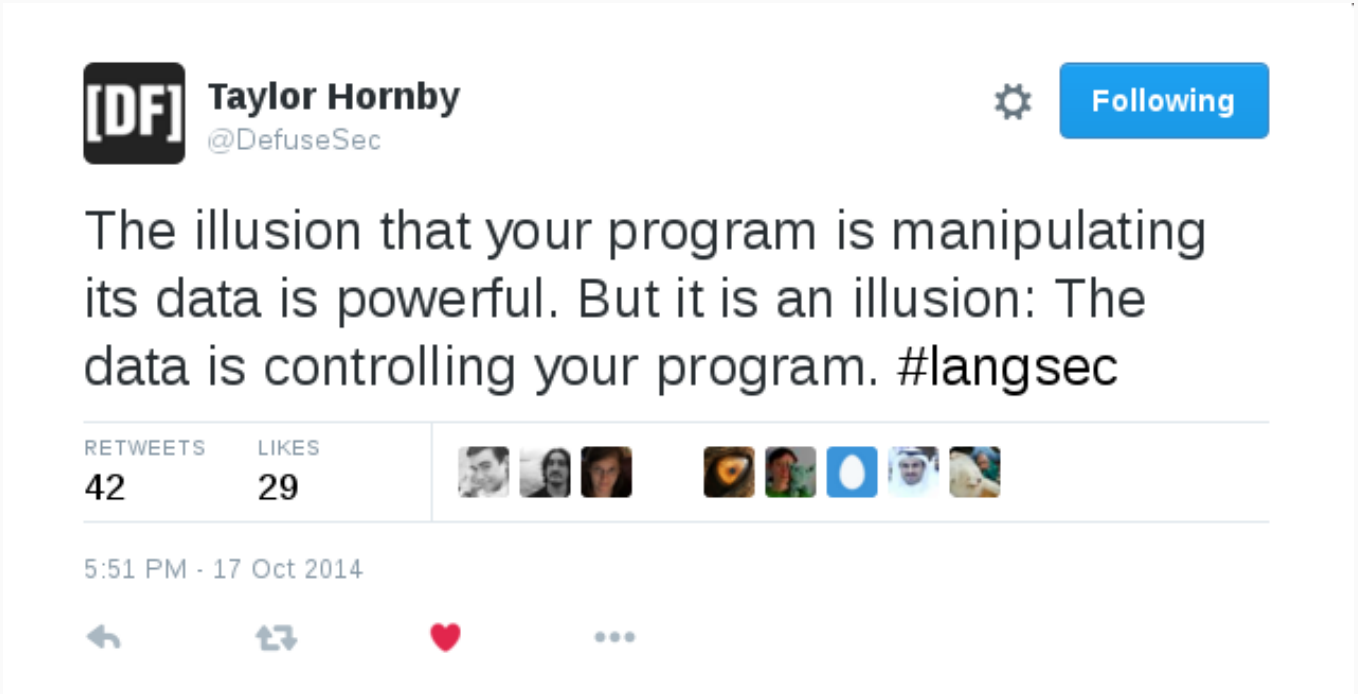
- *Parsing* turns raw bytes into well-typed objects.

WHY WE PARSE

- *Parsing* turns raw bytes into well-typed objects.
- *Serialization* turns well-typed objects into bytes from which they can be reconstructed.

WHY WE PARSE

- *Parsing* turns raw bytes into well-typed objects.
- *Serialization* turns well-typed objects into bytes from which they can be reconstructed.



https://github.com/UpstandingHackers/hammer

AMBIGUITY IS INSECURITY

- Mixing input handling with other logic is hazardous to your program. Full recognition before processing!

AMBIGUITY IS INSECURITY

- Mixing input handling with other logic is hazardous to your program. Full recognition before processing!
- If an input language *can* be defined by a grammar, but your implementation does not explicitly implement that grammar, your implementation is wrong. Computational equivalence for all protocol endpoints!

AMBIGUITY IS INSECURITY

- Mixing input handling with other logic is hazardous to your program. Full recognition before processing!
- If an input language *can* be defined by a grammar, but your implementation does not explicitly implement that grammar, your implementation is wrong. Computational equivalence for all protocol endpoints!

https://github.com/UpstandingHackers/hammer

AMBIGUITY IS INSECURITY

- Mixing input handling with other logic is hazardous to your program. Full recognition before processing!
- If an input language *can* be defined by a grammar, but your implementation does not explicitly implement that grammar, your implementation is wrong. Computational equivalence for all protocol endpoints!

https://github.com/UpstandingHackers/hammer

AMBIGUITY IS INSECURITY

- Mixing input handling with other logic is hazardous to your program. Full recognition before processing!
- If an input language *can* be defined by a grammar, but your implementation does not explicitly implement that grammar, your implementation is wrong. Computational equivalence for all protocol endpoints!



https://github.com/UpstandingHackers/hammer

AMBIGUITY IS INSECURITY

- Mixing input handling with other logic is hazardous to your program. Full recognition before processing!
- If an input language *can* be defined by a grammar, but your implementation does not explicitly implement that grammar, your implementation is wrong. Computational equivalence for all protocol endpoints!

https://github.com/UpstandingHackers/hammer

WHEN PRINCIPLES CONVERGE

- **Soundness:** Every returned answer is true.

WHEN PRINCIPLES CONVERGE

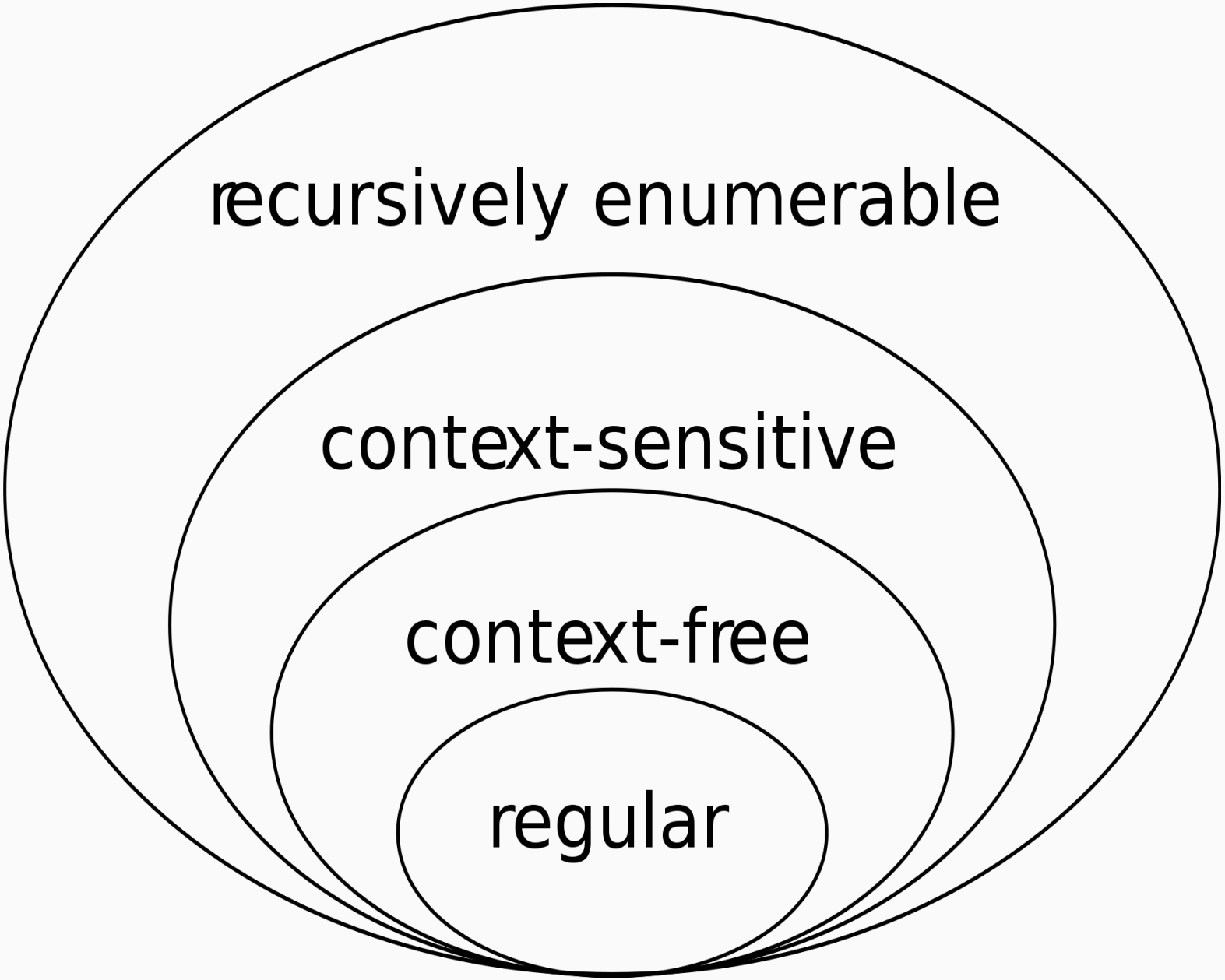
- **Soundness:** Every returned answer is true.
- **Completeness:** Every possible input yields an answer, if one exists.

WHEN PRINCIPLES CONVERGE

- **Soundness:** Every returned answer is true.
- **Completeness:** Every possible input yields an answer, if one exists.
- **Totality:** Every execution terminates yielding exactly one output.

A WHIRLWIND TOUR OF PARSING

THE CHOMSKY HIERARCHY



https://github.com/UpstandingHackers/hammer

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers

https://github.com/UpstandingHackers/hammer

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can’t do left recursion
  - Can have exponential runtime

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can’t do left recursion
  - Can have exponential runtime
- Parsing expression grammars

https://github.com/UpstandingHackers/hammer

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can't do left recursion
  - Can have exponential runtime
- Parsing expression grammars
  - Look a lot like CFGs
  - Always deterministic (unambiguous)
  - Provide lookahead

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can't do left recursion
  - Can have exponential runtime
- Parsing expression grammars
  - Look a lot like CFGs
  - Always deterministic (unambiguous)
  - Provide lookahead
  - Still can't do left recursion

https://github.com/UpstandingHackers/hammer

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can't do left recursion
  - Can have exponential runtime
- Parsing expression grammars
  - Look a lot like CFGs
  - Always deterministic (unambiguous)
  - Provide lookahead
  - Still can't do left recursion
- Packrat parsers

https://github.com/UpstandingHackers/hammer

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can't do left recursion
  - Can have exponential runtime
- Parsing expression grammars
  - Look a lot like CFGs
  - Always deterministic (unambiguous)
  - Provide lookahead
  - Still can't do left recursion
- Packrat parsers
  - They're PEGs, but memoized
  - Can handle left-recursion! (mostly)

https://github.com/UpstandingHackers/hammer

THE RECURSIVE-DESCENT FAMILY

- Recursive descent parsers
  - Parsing like mom used to do it (if your mom is Jack Crenshaw)
  - Conceptually really simple
  - Can't do left recursion
  - Can have exponential runtime
- Parsing expression grammars
  - Look a lot like CFGs
  - Always deterministic (unambiguous)
  - Provide lookahead
  - Still can't do left recursion
- Packrat parsers
  - They're PEGs, but memoized
  - Can handle left-recursion! (mostly)
  - But nobody can quite agree on how

https://github.com/UpstandingHackers/hammer

Context-free and regular

- LL(k)
  - Context-free
  - Left-recursive

Context-free and regular

- LL(k)
  - Context-free
  - Left-recursive
- LALR
  - Also context-free
  - Right-recursive

CONTEXT-FREE AND REGULAR

- LL(k)
  - Context-free
  - Left-recursive
- LALR
  - Also context-free
  - Right-recursive
- GLR
  - Also context-free and right-recursive
  - Like LALR, but yields parse *forests*

CONTEXT-FREE AND REGULAR

- LL(k)
  - Context-free
  - Left-recursive
- LALR
  - Also context-free
  - Right-recursive
- GLR
  - Also context-free and right-recursive
  - Like LALR, but yields parse *forests*
- Thompson regex VM
  - Regular (duh)
  - Doesn't have pcre's pathological runtime
  - Also known as `nfa2dfa`

WHY BINARY PARSING?

- None of the existing tools do it well
  - Limited to character streams
  - Endianness is a pain
  - So are bit-fields

WHY BINARY PARSING?

- None of the existing tools do it well
  - Limited to character streams
  - Endianness is a pain
  - So are bit-fields
- Except bison, which nobody likes
  - Interface is awful for everything except compilers/interpreters
  - Shift-reduce conflicts are confusing
  - Bit-fields still hard unless everything's nicely byte-aligned

THE DESIGN SPACE

REQUIREMENTS

- Thread-safe and reentrant
- Simple API
- POSIX compliant
- Fast
- Correct

PROBLEM: WEAK TYPE SYSTEM

PROBLEM: WEAK TYPE SYSTEM

Solution: make a better one. Even if the compiler can't help.

https://github.com/UpstandingHackers/hammer

Problem: functions aren't first-order data

Problem: functions aren't first-order data

Solution: function *pointers* are

PROBLEM: STRICT EVALUATION

PROBLEM: STRICT EVALUATION

Solution: forward declaration

PROBLEM: NULL POINTERS EXIST

PROBLEM: NULL POINTERS EXIST

- Half-assed solution: make them mean something

PROBLEM: NULL POINTERS EXIST

- Half-assed solution: make them mean something
- Actual solution: that, plus lots of null checks ☹

PROBLEM: NO GARBAGE COLLECTION

https://github.com/UpstandingHackers/hammer

PROBLEM: NO GARBAGE COLLECTION

Solution: custom allocator (and pluggable interface)

https://github.com/UpstandingHackers/hammer

Problem: method dispatch (what methods?)

Problem: method dispatch (what methods?)

Solution: vtables! They're not just for C++ anymore.

MEET HAMMER

THE ALL-SINGING, ALL-DANCING PARSER GENERATOR LIBRARY

- Parser combinator frontend

https://github.com/UpstandingHackers/hammer

THE ALL-SINGING, ALL-DANCING PARSER GENERATOR LIBRARY

- Parser combinator frontend
- Selectable backends
  - Packrat
  - LL(k), LALR, GLR, LR(0)
  - Thompson regex VM

https://github.com/UpstandingHackers/hammer

THE ALL-SINGING, ALL-DANCING PARSER GENERATOR LIBRARY

- Parser combinator frontend
- Selectable backends
  - Packrat
  - LL(k), LALR, GLR, LR(0)
  - Thompson regex VM
- Many, many language bindings

https://github.com/UpstandingHackers/hammer

THE ALL-SINGING, ALL-DANCING PARSER GENERATOR LIBRARY

- Parser combinator frontend
- Selectable backends
  - Packrat
  - LL(k), LALR, GLR, LR(0)
  - Thompson regex VM
- Many, many language bindings
- Linux, OS X, Windows (f yeah POSIX)

https://github.com/UpstandingHackers/hammer

NAMING CONVENTIONS

- Types
  - Start with **H** and are CamelCased
  - `HParser`, `HParsedToken`, etc

NAMING CONVENTIONS

- Types
  - Start with **H** and are CamelCased
  - `HParser`, `HParsedToken`, etc
- Functions
  - Start with **h_** and use underscores
  - `h_parse()`, `h_length_value()`, etc

BASIC USAGE

```c
#include <hammer.h>

const HParsedToken* build_my_struct(const HParseResult *p) {
  // ...
}
int main(int argc, char** argv) {
  // obtain data, and its length, from somewhere
  // Create a parser
  HParser *parser = h_action(..., build_my_struct);

  // Parse the data
  HParseResult *result = h_parse(parser, data, length);
  // Get your struct back from the result token and use it
  do_something(result->ast->user);
  return 0;
}
```

https://github.com/UpstandingHackers/hammer

PARSER TYPES

```
typedef struct HParser_ {
  const HParserVtable *vtable;
  HParserBackend backend;
  void *backend_data;
  void *env;
  HCFChoice *desugared;
} HParser;
```

https://github.com/UpstandingHackers/hammer

FUNCTION TYPES

```
// An action to apply to an AST, used in h_action().
typedef HParsedToken* (*HAction)(const HParseResult *p, void
↪  *user_data);

// A boolean attribute-checking function, used in
↪  h_attr_bool().
typedef bool (*HPredicate)(HParseResult *p, void *user_data);

// A parser that depends on the result of a previous parser,
↪  used in h_bind().
typedef HParser* (*HContinuation)(HAllocator *mm__, const
↪  HParsedToken *x, void *env);
```

https://github.com/UpstandingHackers/hammer

RESULT TYPES

- HParseResult
  - A tree of parsed tokens
  - The total number of bits parsed
  - A reference to the memory context for this parse

RESULT TYPES

- HParseResult
  - A tree of parsed tokens
  - The total number of bits parsed
  - A reference to the memory context for this parse
- HParsedToken
  - Token type: bytes, signed/unsigned int, sequence, user-defined
  - Token (a tagged union)
  - Byte index and bit offset

PRIMITIVES

- Character and token parsers
  - `h_ch(const uint8_t c)`
  - `h_token(const uint8_t *str, size_t len)`
  - `h_ch_range(const uint8_t lower, const uint8_t upper)`
  - `h_not_in(const uint8_t charset, size_t length)`

PRIMITIVES

- Character and token parsers
  - `h_ch(const uint8_t c)`
  - `h_token(const uint8_t *str, size_t len)`
  - `h_ch_range(const uint8_t lower, const uint8_t upper)`
  - `h_not_in(const uint8_t charset, size_t length)`
- Integral parsers
  - `h_uint8()`, `h_int64()`
  - `h_bits(size_t len, bool sign)`
  - `h_int_range(const HParser *p, const int64_t lower, const int64_t upper)`

PRIMITIVES

- Character and token parsers
  - `h_ch(const uint8_t c)`
  - `h_token(const uint8_t *str, size_t len)`
  - `h_ch_range(const uint8_t lower, const uint8_t upper)`
  - `h_not_in(const uint8_t charset, size_t length)`
- Integral parsers
  - `h_uint8()`, `h_int64()`
  - `h_bits(size_t len, bool sign)`
  - `h_int_range(const HParser *p, const int64_t lower, const int64_t upper)`
- End-of-input
  - `h_end_p()`

COMBINING PRIMITIVES

- Sequential and alternative
  - `h_sequence(const HParser *p, ...)`
  - `h_choice(const HParser *p, ...)`

COMBINING PRIMITIVES

- Sequential and alternative
  - `h_sequence(const HParser *p, ...)`
  - `h_choice(const HParser *p, ...)`
- Repetition
  - `h_many(const HParser *p)`
  - `h_many1(const HParser *p)`
  - `h_repeat_n(const HParser *p, size_t len)`

COMBINING PRIMITIVES

- Sequential and alternative
  - `h_sequence(const HParser *p, ...)`
  - `h_choice(const HParser *p, ...)`
- Repetition
  - `h_many(const HParser *p)`
  - `h_many1(const HParser *p)`
  - `h_repeat_n(const HParser *p, size_t len)`
- Optional
  - `h_optional(const HParser *p)`

COMBINING PRIMITIVES

- Not Actually Appearing In This Parse Tree
  - `h_ignore(const HParser *p)`

COMBINING PRIMITIVES

- Not Actually Appearing In This Parse Tree
  - `h_ignore(const HParser *p)`
- Higher-order
  - `h_length_value(const HParser *length, const HParser *value)`
  - `h_and(const HParser *p)`, `h_not(const HParser *p)`
  - `h_indirect(const HParser *p)`, `h_bind_indirect`

DOING THINGS TO COMBINATIONS OF PRIMITIVES

- Validate the result against a predicate function
  - `h_attr_bool(const HParser *p, const HPredicate pred)`

DOING THINGS TO COMBINATIONS OF PRIMITIVES

- Validate the result against a predicate function
  - `h_attr_bool(const HParser *p, const HPredicate pred)`
- Semantic actions
  - `h_action(const HParser *p, const HAction a)`

DOING THINGS TO COMBINATIONS OF PRIMITIVES

- Validate the result against a predicate function
  - `h_attr_bool(const HParser *p, const HPredicate pred)`
- Semantic actions
  - `h_action(const HParser *p, const HAction a)`
- Monadic bind
  - `h_bind(const HParser *p, HContinuation k, void *env)`

OTHERWISE UNCATEGORISED COMBINATORS

- Changing endianness
  - `h_with_endianness(const HParser *p, char endianness)`

OTHERWISE UNCATEGORISED COMBINATORS

- Changing endianness
  - `h_with_endianness(const HParser *p, char endianness)`
- Long-distance dependencies
  - `h_put_value(const HParser *p, const char* name)`
  - `h_get_value(const char* name)`
  - Please don't use these if you can possibly avoid them

A PRACTICAL EXAMPLE

TOP-LEVEL DNS

```
const HParser *dns_message =
    h_action(h_attr_bool(h_sequence(dns_header,
                                    h_many(dns_question),
                                    h_many(dns_rr),
                                    h_end_p(),
                                    NULL),
                         validate_dns),
             pack_dns_struct);
```

https://github.com/UpstandingHackers/hammer

DNS QUESTIONS

```
const HParser *dns_question =
    h_sequence(
        h_sequence(
            h_many1(
                h_length_value(h_int_range(h_uint8(), 1, 255),
                               h_uint8())),
            h_ch('\x00'),
            NULL),  // QNAME
        qtype,
        qclass,
        NULL);
```

https://github.com/UpstandingHackers/hammer

DNS RRS

```
const HParser *dns_rr =
    h_sequence(domain,
               type,
               class,
               h_uint32(),
               h_length_value(h_uint16(), h_uint8()),
               NULL);
```

https://github.com/UpstandingHackers/hammer

VALIDATING A DNS PACKET

```
bool validate_dns(HParseResult *p) {
  if (TT_SEQUENCE != p->ast->token_type)
    return false;
  HParsedToken **elems = p->ast->seq->elements[0]->seq->elements;
  size_t qd = elems[8]->uint;
  size_t an = elems[9]->uint;
  size_t ns = elems[10]->uint;
  size_t ar = elems[11]->uint;
  HParsedToken *questions = p->ast->seq->elements[1];
  if (questions->seq->used != qd)
    return false;
  HParsedToken *rrs = p->ast->seq->elements[2];
  if (an+as+ar != rrs->seq->used)
    return false;
  return true;
}
```

https://github.com/UpstandingHackers/hammer

UNDER THE HOOD

LET THE COMPILER WRITE CODE FOR YOU

```
 #define HAMMER_FN_DECL_NOARG(rtype_t, name) \
   rtype_t name(void); \
   rtype_t name##__m(HAllocator* mm__)

#define HAMMER_FN_DECL(rtype_t, name, ...) \
  rtype_t name(__VA_ARGS__); \
  rtype_t name##__m(HAllocator* mm__, __VA_ARGS__)

#define HAMMER_FN_DECL_VARARGS(rtype_t, name, ...) \
  rtype_t name(__VA_ARGS__, ...); \
  rtype_t name##__m(HAllocator* mm__, __VA_ARGS__, ...); \
  rtype_t name##__mv(HAllocator* mm__, __VA_ARGS__, va_list ap); \
  rtype_t name##__v(__VA_ARGS__, va_list ap); \
  rtype_t name##__a(void *args[]); \
  rtype_t name##__ma(HAllocator *mm__, void *args[])
```

https://github.com/UpstandingHackers/hammer

IN THE BEGINNING THERE WERE OCTETS

```
typedef struct HInputStream_ {
  // This should be considered to be a really big value type.
  const uint8_t *input;
  size_t pos;  // position of this chunk in a multi-chunk stream
  size_t index;
  size_t length;
  char bit_offset;
  char margin; // The number of bits on the end that is being read
               // towards that should be ignored.
  char endianness;
  bool overrun;
  bool last_chunk;
} HInputStream;
```

https://github.com/UpstandingHackers/hammer

TO THE VTABLE!

```
HParseResult* h_parse(const HParser* parser, const uint8_t*
↪  input, size_t length) {
  return h_parse__m(&system_allocator, parser, input, length);
}
HParseResult* h_parse__m(HAllocator* mm__, const HParser*
↪  parser, const uint8_t* input, size_t length) {
  HInputStream input_stream = {
    .pos = 0, .index = 0,
    .bit_offset = 0, .overrun = 0,
    .endianness = DEFAULT_ENDIANNESS,
    .length = length, .input = input,
    .last_chunk = true
  };
  return backends[parser->backend]->parse(mm__, parser,
↪  &input_stream);
}
```

https://github.com/UpstandingHackers/hammer

ARE WE CONTEXT-SENSITIVE?

```
HParserBackendVTable h__packrat_backend_vtable = {
  .compile = h_packrat_compile, // no-op
  .parse = h_packrat_parse,
  .free = h_packrat_free
};
```

https://github.com/UpstandingHackers/hammer

COVERING ALL POSSIBLE OUTCOMES

```
HParseResult *h_packrat_parse(HAllocator* mm__, const HParser*
↪  parser, HInputStream *input_stream) {
  HArena * arena = h_new_arena(mm__, 0);
  // out-of-memory handling
  jmp_buf except;
  h_arena_set_except(arena, &except);
  if (setjmp(except)) {
    h_delete_arena(arena);
    return NULL;
  }
  HParseState *parse_state = a_new_(arena, HParseState, 1);
  // setup setup setup
  HParseResult *res = h_do_parse(parser, parse_state);
  // teardown teardown teardown
  if (!res)
    h_delete_arena(parse_state->arena);
  return res;
}
```

https://github.com/UpstandingHackers/hammer

OR ARE WE CONTEXT-FREE?

```
HParserBackendVTable h__llk_backend_vtable = {
  .compile = h_llk_compile,
  .parse = h_llk_parse,
  .free = h_llk_free,
  // ...
};

int h_llk_compile(HAllocator* mm__, HParser* parser, const
↪  void* params) {
  size_t kmax = params? (uintptr_t)params : DEFAULT_KMAX;
  assert(kmax>0);

  // Convert parser to a CFG. This can fail as indicated by a
↪  NULL return.
  HCFGrammar *grammar = h_cfgrammar(mm__, parser);
  // ...
}
```

https://github.com/UpstandingHackers/hammer

DESUGARING

- LL(k) needs first and follow sets, LR family needs parse tables

DESUGARING

- LL(k) needs first and follow sets, LR family needs parse tables
- We can obtain both from a sum-of-products representation

DESUGARING

- LL(k) needs first and follow sets, LR family needs parse tables
- We can obtain both from a sum-of-products representation
- From root HParser*, walk combinator tree recursively, transforming as you go

DESUGARING

- LL(k) needs first and follow sets, LR family needs parse tables
- We can obtain both from a sum-of-products representation
- From root HParser*, walk combinator tree recursively, transforming as you go
- Generate sets/tables according to backend

DESUGARING

- LL(k) needs first and follow sets, LR family needs parse tables
- We can obtain both from a sum-of-products representation
- From root HParser*, walk combinator tree recursively, transforming as you go
- Generate sets/tables according to backend
- Store in `parser->backend_data`

OR ARE WE REGULAR?

- Walk combinator tree, translating combinators to RVM instructions

OR ARE WE REGULAR?

- Walk combinator tree, translating combinators to RVM instructions
- Store in `parser->backend_data`

OR ARE WE REGULAR?

- Walk combinator tree, translating combinators to RVM instructions
- Store in `parser->backend_data`
- If this sounds like a compiler, that's because it is

OR ARE WE REGULAR?

- Walk combinator tree, translating combinators to RVM instructions
- Store in `parser->backend_data`
- If this sounds like a compiler, that's because it is
- Wait. How do we know a combinatoric parser is regular? Or CF?

https://github.com/UpstandingHackers/hammer

MOAR VTABLES

```
static const HParserVtable ch_vt = {
  .parse = parse_ch,
  .isValidRegular = h_true,
  .isValidCF = h_true,
  .desugar = desugar_ch,
  .compile_to_rvm = ch_ctrvm,
  .higher = false,
};
```

https://github.com/UpstandingHackers/hammer

VOLUNTEER WHAT YOU CAN'T INTROSPECT

```
static const HParserVtable many_vt = {
  .parse = parse_many,
  .isValidRegular = many_isValidRegular,
  .isValidCF = many_isValidCF,
  .desugar = desugar_many,
  .compile_to_rvm = many_ctrvm,
  .higher = true,
};
```

https://github.com/UpstandingHackers/hammer

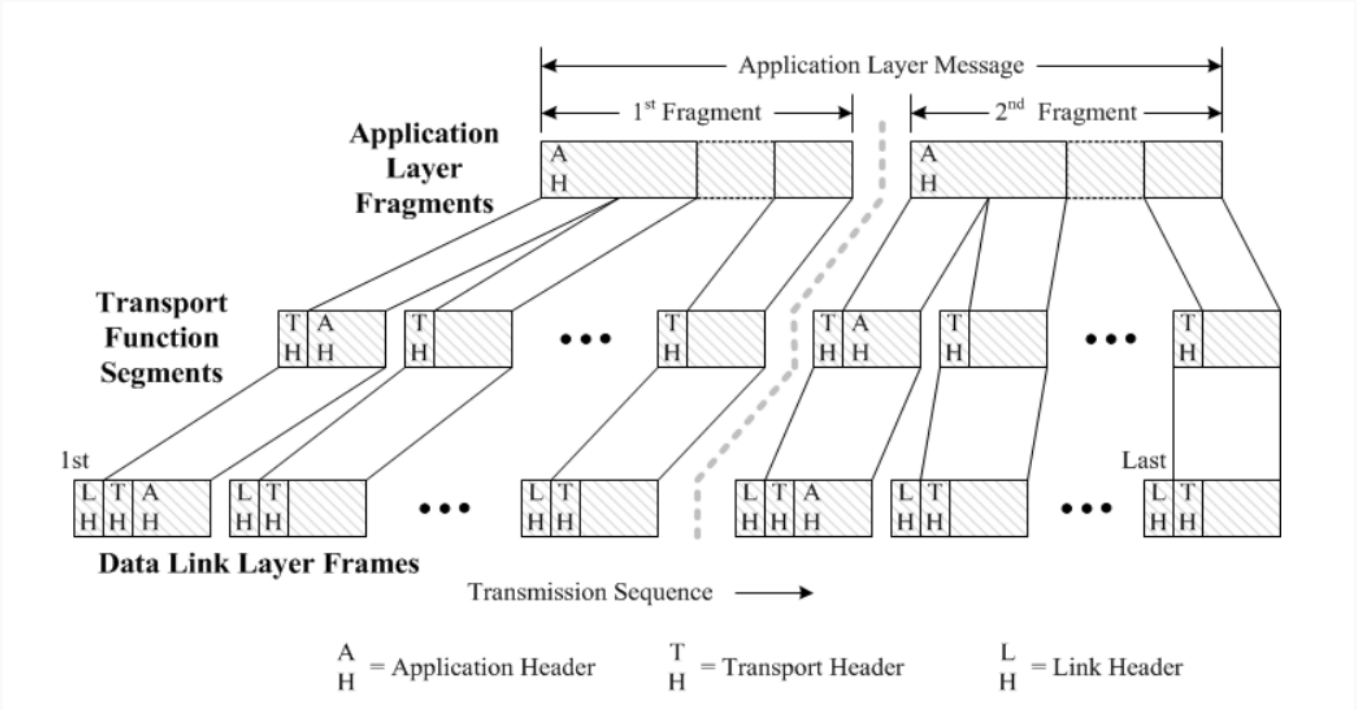
VOLUNTEER WHAT YOU CAN'T INTROSPECT

```
static const HParserVtable length_value_vt = {
  .parse = parse_length_value,
  .isValidRegular = h_false,
  .isValidCF = h_false,
};
```

https://github.com/UpstandingHackers/hammer

IN PRACTICE

DNP3 PROTOCOL LAYERS



https://github.com/UpstandingHackers/hammer

```
AppHdr (ObjHdr Object*)*
```

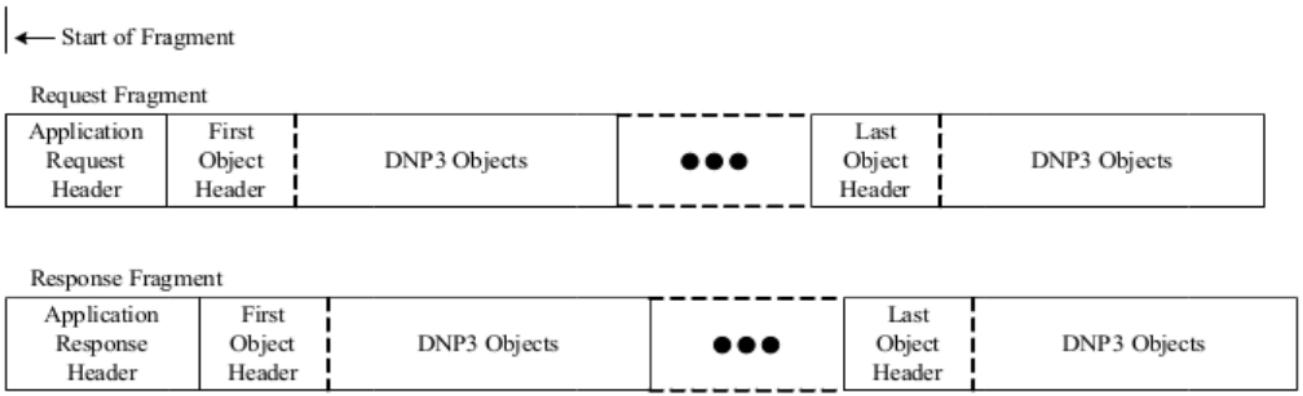


Figure 4-4—Fragment structure

```
AppHdr = SeqNo Flags FunctionCode
ObjHdr = Group Variation PC RSC Range
Object = Prefix? ObjectData
```

https://github.com/UpstandingHackers/hammer

PROBLEM?

The previous fails to express dependencies between

- `Flags` and `FunctionCode`
- `FunctionCode` and (`Group`, `Variation`)
- `RSC`, `Range`, `PC`
- (`Group`, `Variation`) and `Objects`

PROBLEM?

The previous fails to express dependencies between

- `Flags` and `FunctionCode`
- `FunctionCode` and (`Group`, `Variation`)
- `RSC`, `Range`, `PC`
- (`Group`, `Variation`) and `Objects`
- That's why we had to add the symbol table ☹

BUT DOES IT WORK?

- Enter *American Fuzzy Lop*, http://lcamtuf.coredump.cx/afl/

BUT DOES IT WORK?

- Enter *American Fuzzy Lop*, http://lcamtuf.coredump.cx/afl/
- Compile-time instrumentation + genetic algorithms = rocks fall, everything dies

BUT DOES IT WORK?

- Enter *American Fuzzy Lop*, http://lcamtuf.coredump.cx/afl/
- Compile-time instrumentation + genetic algorithms = rocks fall, everything dies
- CVEs in over 100 projects you've heard of, from Firefox to ext4 to wireshark

BUT DOES IT WORK?

- Enter *American Fuzzy Lop*, http://lcamtuf.coredump.cx/afl/
- Compile-time instrumentation + genetic algorithms = rocks fall, everything dies
- CVEs in over 100 projects you've heard of, from Firefox to ext4 to wireshark
- So how many bugs did it find in hammer/DNP3?

- Remember that setjmp we saw earlier?

ONE.

- Remember that setjmp we saw earlier?
- “Out of memory” is an exception, which introduces partiality

ONE.

- Remember that setjmp we saw earlier?
- “Out of memory” is an exception, which introduces partiality
- Handle the exception by cleaning up the wasted resources, and we’ve reclaimed totality

BEYOND HAMMER

NOM NOM NOM

- One problem: Hammer is basically interpreted

https://github.com/Geal/nom

NOM NOM NOM

- One problem: Hammer is basically interpreted
- Break out the hygienic macros!

https://github.com/Geal/nom

NOM NOM NOM

- One problem: Hammer is basically interpreted
- Break out the hygienic macros!
- Rust, pure recursive descent, and fast enough not to matter

https://github.com/Geal/nom

ITERATEES IN C

- Oleg Kiselyov's composable stream processors

http://code.khjk.org/citer/

https://github.com/UpstandingHackers/hammer

ITERATEES IN C

- Oleg Kiselyov's composable stream processors
- Enumeratee presents chunk of data, or "no data," or "done"

http://code.khjk.org/citer/

ITERATEES IN C

- Oleg Kiselyov's composable stream processors
- Enumeratee presents chunk of data, or "no data," or "done"
- Returns STOP (+ return value), CONTINUE (+ how), or ERROR

http://code.khjk.org/citer/

https://github.com/UpstandingHackers/hammer

ITERATEES IN C

- Oleg Kiselyov's composable stream processors
- Enumeratee presents chunk of data, or "no data," or "done"
- Returns STOP (+ return value), CONTINUE (+ how), or ERROR
- Need to handle multiple encodings (eg XML)? Let initial iteratee decide subsequent iteratee to dispatch to

http://code.khjk.org/citer/

https://github.com/UpstandingHackers/hammer

ITERATEES IN C

- Oleg Kiselyov's composable stream processors
- Enumeratee presents chunk of data, or "no data," or "done"
- Returns STOP (+ return value), CONTINUE (+ how), or ERROR
- Need to handle multiple encodings (eg XML)? Let initial iteratee decide subsequent iteratee to dispatch to
- `wc` implemented with iteratees is faster than stock `wc`

http://code.khjk.org/citer/

https://github.com/UpstandingHackers/hammer

CONCLUSION

TAKEAWAYS

- When the world gives you low standards, set better ones

TAKEAWAYS

- When the world gives you low standards, set better ones
- Your compiler works harder than you do, but you are smarter

WHAT'S NEXT?

- IMPLEMENT ALL THE FORMATS

WHAT'S NEXT?

- IMPLEMENT ALL THE FORMATS
- Better input streams: `HTokenInputStream`, `HIncrementalInputStream`

what's next?

- IMPLEMENT ALL THE FORMATS
- Better input streams: `HTokenInputStream`, `HIncrementalInputStream`
- Better error reporting ☹

https://github.com/UpstandingHackers/hammer

what's next?

- IMPLEMENT ALL THE FORMATS
- Better input streams: `HTokenInputStream`, `HIncrementalInputStream`
- Better error reporting ☹
- Context-free derivatives, maybe ALL*

https://github.com/UpstandingHackers/hammer

WHAT'S NEXT?

- IMPLEMENT ALL THE FORMATS
- Better input streams: `HTokenInputStream`, `HIncrementalInputStream`
- Better error reporting ☹
- Context-free derivatives, maybe ALL*
- Slightly-ahead-of-time assembly (compilerception!)

WHAT'S NEXT?

- IMPLEMENT ALL THE FORMATS
- Better input streams: `HTokenInputStream`, `HIncrementalInputStream`
- Better error reporting ☹
- Context-free derivatives, maybe ALL*
- Slightly-ahead-of-time assembly (compilerception!)
- Retire C

MORE TO COME!

- Watch langsec-discuss@lists.langsec.org for further announcements
- https://github.com/UpstandingHackers/hammer

mlp@upstandinghackers.com · @maradydd

Thank you!

Made with love and Beamer.