SuperC: Parsing All of C by Taming the Preprocessor

Paul Gazzillo and Robert Grimm

New York University

We Need Better C Tools

- Linux and other critical systems written in C
  - Need source code browsers
    - 7,500+ compilation units, 5.5 million lines
  - Need bug finders
    - 1,000 found by static checkers [Chou et al., SOSP ’01]
  - Need refactoring tools
    - 150+ errors due to interface changes [Padioleau et al., EuroSys ’08]

We Need Better C Tools

- Linux and other critical systems written in C
  - Need source code browsers
    - 7,500 nes

Need to Parse C First!

  - Need bug finders
    - 1,000 found by static checkers [Chou et al., SOSP ’01]
  - Need refactoring tools
    - 150+ errors due to interface changes [Padioleau et al., EuroSys ’08]

C Source Code Written in Two Languages

- C proper and the preprocessor
- Preprocessor is a simple, text processing language
  - Static conditionals: configure source code
  - Macros: abbreviate C constructs
  - Headers: break source code into separate files
- Preprocessor makes parsing source code tricky
  - Hides C source code in macros and headers
  - Breaks C syntax

Preprocess First?

- Linux x86 contains many programs

```
#ifdef CONFIG_USB_DEVICEFS
 extern int usbfs_init(void);
#else
 static inline int usbfs_init(void){return 0;}
#endif
```

- 6,000 configuration variables ➜ $2^{6,000}$ programs
- Turning on all configuration variables yields only 80% of code [Tartler et al., OSR ’11]

Add Preprocessor to C Grammar?

- Macros expand to arbitrary C fragments

```
#define for_each_class(c) \
  for (c = highest_class; c; c = c->next)
```

Add Preprocessor to C Grammar?

- Macros expand to arbitrary C fragments

```
#define for_each_class(c) \
  for (c = highest_class; c; c = c->next)
```

- Directives appear between arbitrary C fragments

```
#ifdef CONFIG_INPUT_MOUSEDEV_PSAUX
  if (imajor(inode) == 10)
    i = 31;
  else
#endif
    i = iminor(inode) – 32;
```

SuperC to the Rescue!

- Processes source in two steps, like a compiler
  - Preprocessor
    - Expands macros and includes headers
    - But preserves conditionals!
  - Parser creates an AST for *all* configurations

SuperC to the Rescue!

- Processes source in two steps, like a compiler
  - Preprocessor
    - Expands macros and includes headers
    - But preserves conditionals!
  - Parser creates an AST for *all* configurations
- Evaluation

Conditionals Invade the Preprocessor!

Object-like macros

Function-like macros

Macro definitions

Static conditionals

Conditional expressions

Includes

Stringification

Token-pasting

de the
r!

```
#ifdef CONFIG_64BIT
#  define BITS_PER_LONG 64
#else
#  define BITS_PER_LONG 32
#endif
```

Function-like macros

Macro definitions

Static conditionals

Conditional expressions

Includes

Stringification

Token-pasting

de the
r!

```
#ifdef CONFIG_64BIT
#  define BITS_PER_LONG 64
#else
#  define BITS_PER_LONG 32
#endif
```

Function-like macros

Macro definitions

Static

```
__le ## 32
 ➜ __le32
```

Conditional expressions

Includes

Stringification

Token-pasting

Conditionals Need Hoisting

```
__le ## BITS_PER_LONG
```

Conditio Hoisting

Macro expands to conditional

```
__le ## BITS_PER_LONG
```

```
__le ##
#ifdef CONFIG_64BIT
   64
#else
   32
#endif
```

Conditio Hoisting

Macro expands to conditional

```
le ## BITS_PER_LONG
```

One operator:
Two operations

```
__le ##
#ifdef CONFIG_64BIT
   64
#else
   32
#endif
```

Conditio Hoisting

Macro expands to conditional

```
le ## BITS_PER_LONG
```

One operator: Two operations

Hoist conditional around token-paste

```
__le ##
#ifdef CONFIG_64BIT
   64
#else
   32
#endif
```

```
#ifdef CONFIG_64BIT
   __le ## 64
#else
   __le ## 32
#endif
```

The Power of Hoisting

- Works on: token-pasting, stringification, includes, conditional expressions, macros
- Iterates over conditional branches
- Recurses into nested conditionals
- Duplicates tokens across inner-most branches

Parsing All Configurations

- *Forks* subparsers at conditionals
- *Merges* subparsers in the same state after conditionals
  - Joins AST subtrees with *static choice nodes*
  - Preserves mutually exclusive configurations

Fork-Merge Parsing in Action

```
#ifdef CONFIG_INPUT_MOUSEDEV_PSAUX
  if (imajor(inode) == 10)
    i = 31;
  else
#endif
    i = iminor(inode) – 32;
```

Fork-Mer g in Action

(1) Fork subparsers on conditional

```
#ifdef CONFIG_INPUT_MOUSEDEV_PSAUX
  if (imajor(inode) == 10)
    i = 31;
  else
#endif
    i = iminor(inode) – 32;
```

Fork-Mer g in Action

(1) Fork subparsers on conditional

```
#ifdef CONFIG_INPUT_MOUSEDEV_PSAUX
  if (imajor(inode) == 10)
    i = 31;
  else
#endif
    i = iminor(inode) - 32;
```

(2) Parse the *entire* if-then-else

If-Then-Else

Fork-Mer g in Action

(1) Fork subparsers on conditional

```
#ifdef CONFIG_INPUT_MOUSEDEV_PSAUX
  if (imajor(inode) == 10)
    i = 31;
  else
#endif
    i = iminor(inode) - 32;
```

(2) Parse the *entire* if-then-else

(3) Parse *just* the assignment

If-Then-Else

Assignment

Fork-Mer[illegible] in Action

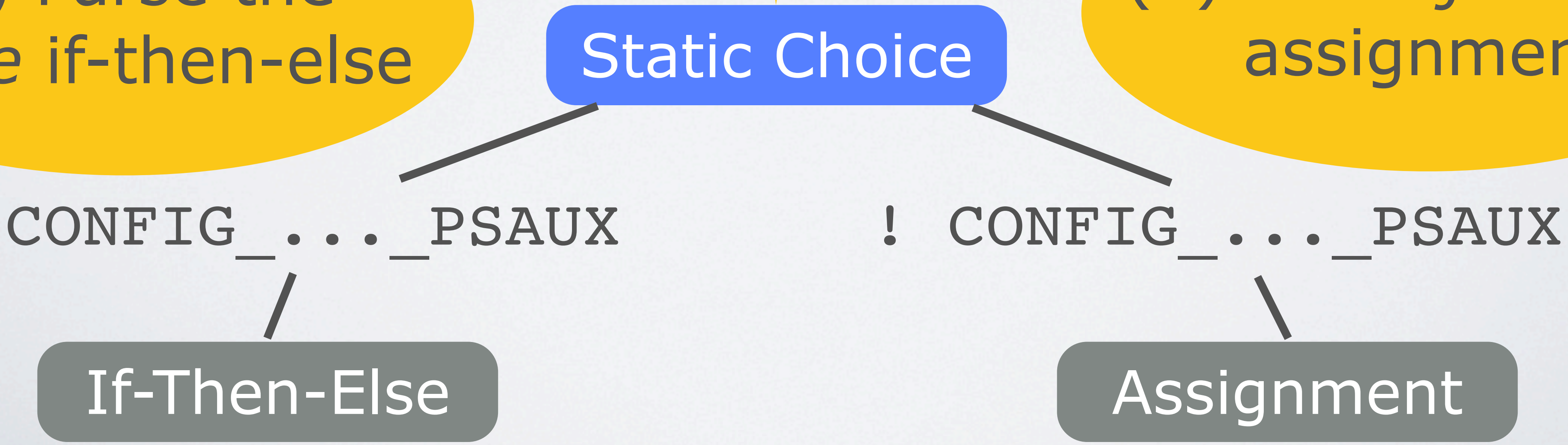
(1) Fork subparsers on conditional

```
#ifdef CONFIG_INPUT_MOUSEDEV_PSAUX
  if (imajor(inode) == 10)
    i = 31
  else
#endif
    i = iminor(inode) - 32;
```

(4) Merge and create the static choice node

(2) Parse the *entire* if-then-else

(3) Parse *just* the assignment



History Repeats Itself: LR Subparser

- Organizes state in stacks
  - Easy forking and merging with DAG
- Is table-driven
  - Good performance
- Reuses existing tools and grammars
  - The good: most complexity is in table generation
  - The bad: shift-reduce & reduce-reduce conflicts

When to Fork Subparsers?

- Naive strategy: fork on every conditional branch
  - Blows up on Linux x86
  - Conditionals are 40 levels deep, 10 in a row
- Our forking strategy: *token follow-set*
  - All tokens reachable from current position
  - Across all configurations

Token Follow-Set in Action

```
   struct rq {
▶  #ifdef CONFIG_SCHED_HRTICK
   # ifdef CONFIG_SMP
       int hrtick_csd_pending;
   # endif
   #endif

   #ifdef CONFIG_SCHEDSTATS
     struct sched_info rq_sched_info;
   #endif
   };
```

6 subparsers

Follow-Set in Action

```
    struct rq {
  ▶ #ifdef CONFIG_SCHED_HRTICK
    # ifdef CONFIG_SMP
        int hrtick_csd_pending;
    # endif
    #endif

    #ifdef CONFIG_SCHEDSTATS
      struct sched_info rq_sched_info;
    #endif
    };
```

6 subparsers

ollow-Set i

3 subparsers

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
        int hrtick_csd_pending;
  # endif
  #endif

  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

Token Follow-Set in Action

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
        int hrtick_csd_pending;
  # endif
  #endif

  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

Token Follow-Set in Action

SCHED_HRTICK
&& SMP

```
   struct rq {
▶  #ifdef CONFIG_SCHED_HRTICK
   # ifdef CONFIG_SMP
       int hrtick_csd_pending;
   # endif
   #endif

   #ifdef CONFIG_SCHEDSTATS
     struct sched_info rq_sched_info;
   #endif
   };
```

Token Follow-Set in Action

SCHED_HRTICK
&& SMP

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
      int hrtick_csd_pending;
  # endif
  #endif
```

! (SCHED_HRTICK && SMP)
&& SCHEDSTATS

```
  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

Token Follow-Set in Action

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
      int hrtick_csd_pending;
  # endif
  #endif

  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

SCHED_HRTICK
&& SMP

! (SCHED_HRTICK && SMP)
&& SCHEDSTATS

! (SCHED_HRTICK && SMP)
&& ! SCHEDSTATS

How Does the Follow-Set Algorithm Work?

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
      int hrtick_csd_pending;
  # endif
  #endif

  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

How Does the Follow-Set Algorithm Work?

Find first token of each branch

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
        int hrtick_csd_pending;
  # endif
  #endif

  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

How Does the Follow-Set Algorithm Work?

Find first token of each branch

Recursively look in nested conditionals

```
    struct rq {
▶   #ifdef CONFIG_SCHED_HRTICK
    # ifdef CONFIG_SMP
        int hrtick_csd_pending;
    # endif
    #endif

    #ifdef CONFIG_SCHEDSTATS
      struct sched_info rq_sched_info;
    #endif
    };
```

How Does the Follow-Set Algorithm Work?

Find first token of each branch

Recursively look in nested conditionals

Keep looking past empty “#else”s

```
  struct rq {
▶ #ifdef CONFIG_SCHED_HRTICK
  # ifdef CONFIG_SMP
        int hrtick_csd_pending;
  # endif
  #endif

  #ifdef CONFIG_SCHEDSTATS
    struct sched_info rq_sched_info;
  #endif
  };
```

How Does the Follow-Set Algorithm Work?

```
struct rq {
#ifdef CONFIG_SCHED_HRTICK
# ifdef CONFIG_SMP
    int hrtick_csd_pending;
# endif
#endif

#ifdef CONFIG_SCHEDSTATS
  struct sched_info rq_sched_info;
#endif
};
```

Find first token of each branch

Recursively look in nested conditionals

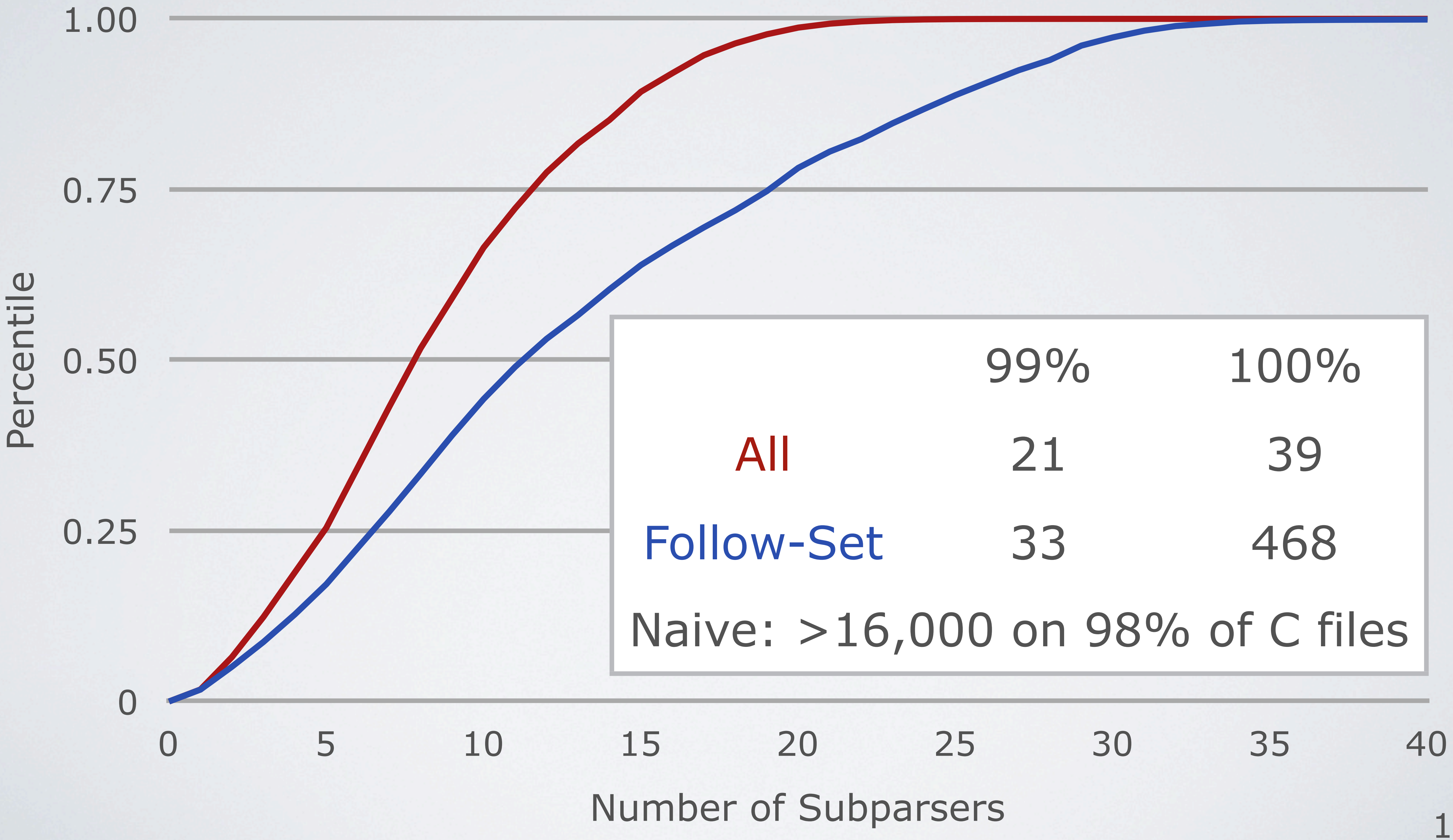
Keep looking past empty "#else"s

Stop when all reachable tokens found

Evaluation

- Feasibility: number of subparsers
  - Compare to naive strategy
- Performance: running time on compilation units
  - Compare to TypeChef [Kaestner et al, OOPSLA '11]
    - Preprocessor: ad-hoc hoisting for expressions
    - Parser: LL combinator library
    - No automatic merging: 7 combinators

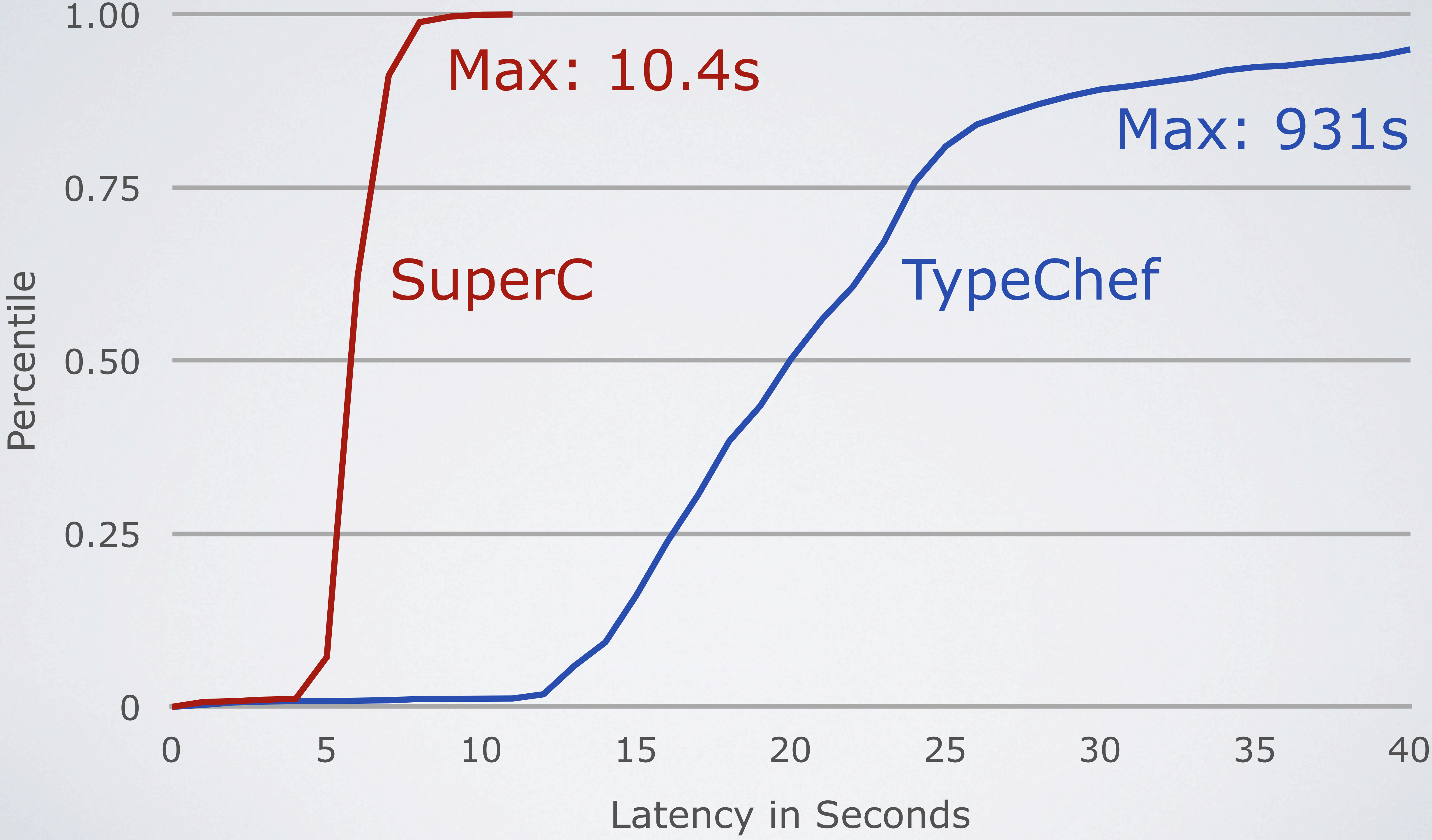
Number of Subparsers Used at Any Given Point



	99%	100%
All	21	39
Follow-Set	33	468

Naive: >16,000 on 98% of C files

Performance Across Compilation Units



In Conclusion

- SuperC is the first solution to parsing *all* of C
  - Preprocessor preserves all conditionals
    - Hoisting enables token-level operations
  - Parser forks and merges subparsers
    - Reuses existing parser generator and grammar
  - Token follow-set makes parsing feasible
    - Further optimized for fewer subparsers
- SuperC scales well across Linux x86

http://cs.nyu.edu/xtc/