SuperC: Parsing All of C by Taming the Preprocessor

Paul Gazzillo and Robert Grimm
New York University

http://cs.nyu.edu/xtc

Problem: Parsing All of C

We need better C tools

- Linux x86 is large and complex
  - 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]
- These tools all need to parse C first

C source code written in both C and the preprocessor

- Source code contains many programs

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

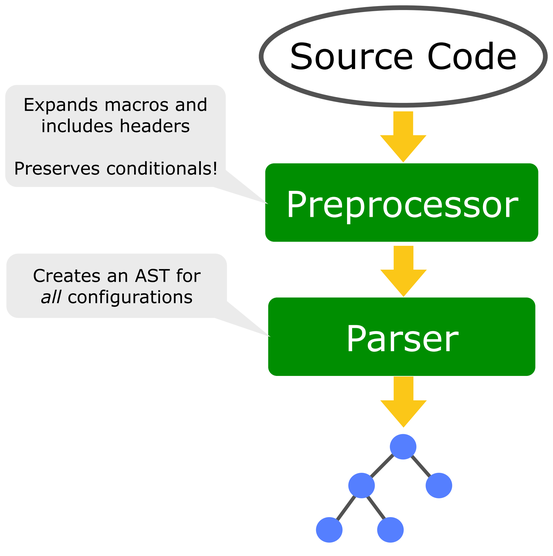
- Linux x86 has 6,000 configuration variables, $2^{6,000}$ combinations
  - Turning on all configuration variables yields only 80% of code [Tartler et al., OSR '11]
- Macros expand to arbitrary C fragments

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

- Directives appear between arbitrary C fragments

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

Solution Approach



How SuperC Works

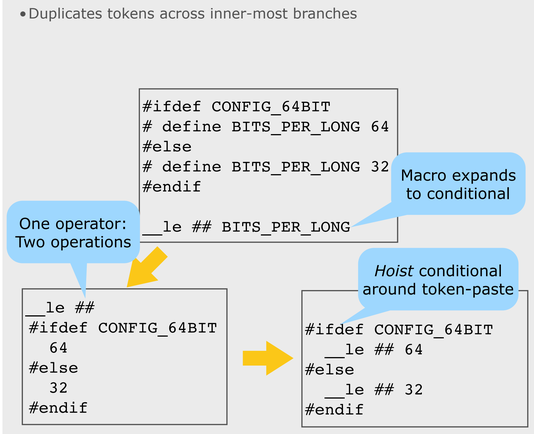
The Preprocessor

Conditionals Invade the Preprocessor

- The preprocessor leaves condtiionals in place
- Conditionals then compose with most preprocessor operations
  - Many operations require *hoisting*

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



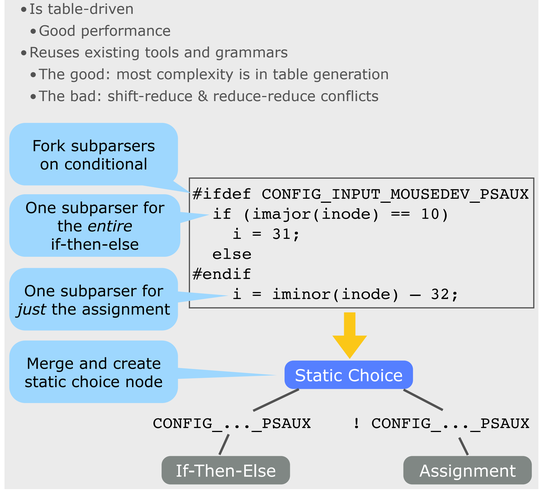
The Parser

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

History Repeats Itself: LR Subparsers

- 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

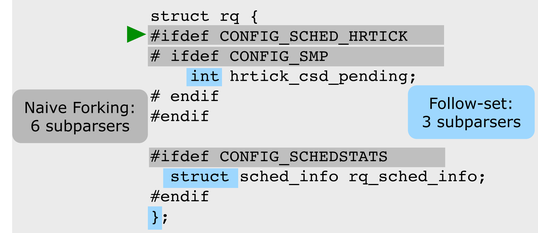


Parsing Real-World C

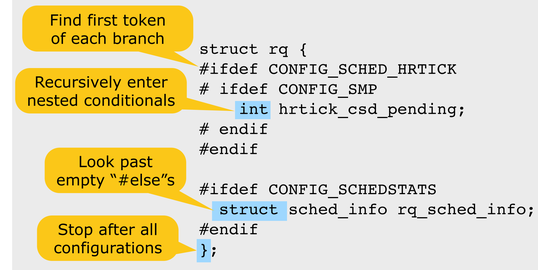
When to Fork Subparsers?

- 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

Follow-set forks fewer subparsers



The follow-set algorithm in action

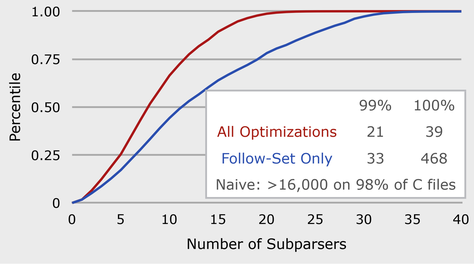


Follow-set supports further optimization

- *Shared reductions*
  - Reduce one stack for many follow-set tokens before forking
  - Limits redundant work by subparsers
- *Lazy shifts*
  - Only fork tokens in the nearest condtiional
  - Limits number of subparsers needed
- *Early Reduces*
  - Pick reducing subparser before a shifting one
  - Improves chances of merging

Evaluation

Number of Subparsers Used at Any Given Point while Parsing Linux x86



Performance Across Compilation Units of Linux x86