College of Engineering and Computer Science
UNIVERSITY OF CENTRAL FLORIDA

Conditional Compilation Is Dead, Long Live Conditional Compilation!

Paul Gazzillo and Shiyi Wei

ICSE-NIER 2019

@paul_gazzillo

pgazz.com

The C Preprocessor Creates a Dilemma

- Conditional compilation implemented with the C preprocessor
- Really great for performance
- Really bad for software tools

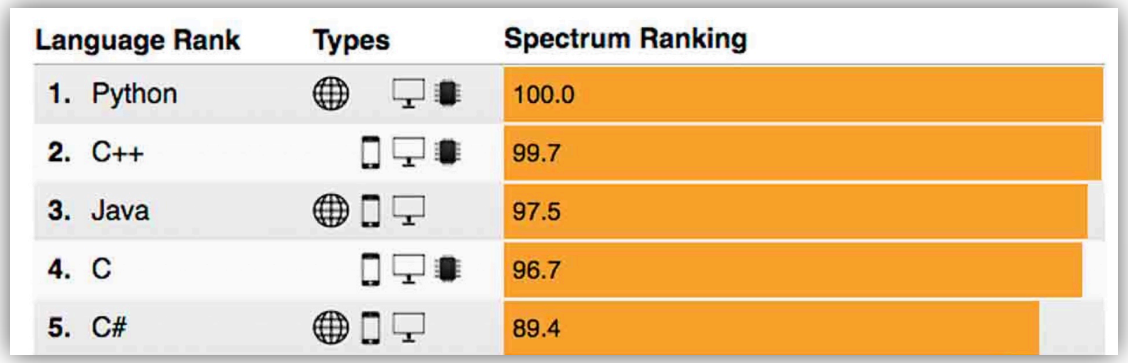
UCF

Goal: Replace Preprocessor Usage with C Itself

- Easier for software tools
- Preserve existing C software
- How?
  - New C language constructs
  - Automate conditional compilation with compiler optimizations

UCF

C Use Has Grown!



IEEE Spectrum Popularity Rankings, May 2019

May 2019	May 2018	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	4		Python

TIOBE Popularity Rankings, May 2019

Top programming languages

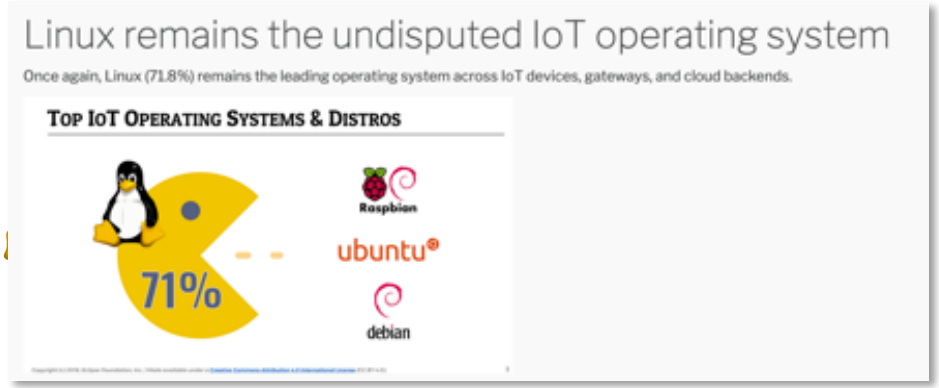
Constrained devices	Gateways and edge nodes	IoT Cloud
C	Java	Java
C++	Python	Javascript
Java	C++	Python
Javascript	C	PHP

IoT Developer Survey, Eclipse Foundation 2019

UCF

Conditional Compilation Makes Reuse Possible

- Linux configurable to many devices
- No extra programming needed



UCF

C Preprocessor Used Extensively

- Macros used about 1 in 4 SLoC, in general [Ernst et al 1999]
- Linux v4.19 (late 2018)

Source lines of code	about 12 million
Preprocessor macros defined	about 1 million
Preprocessor directives used	about 2 million
Preprocessor conditional blocks	about 60,000

- Developers use it to hand-optimize object file size
  - Compiling all Linux features would make an enormous binary

UCF

Conditional Compilation Is Implemented with the Preprocessor

Configuration options tested at build-time

```
#ifdef CONFIG_OF_IRQ_DOMAIN
void irq_add(int *ops) {
  int irq = *ops;
}
#endif

int *ops = NULL;
#ifdef CONFIG_OF_IRQ
ops = &irq_ops;
#endif
irq_add(ops);
```


UCF

Variability Bugs: Existence Depends on Configuration Settings

3. *Null pointer error* in some configurations

1. Initialize "ops" pointer

2. Only set in *some* configurations

```
#ifdef CONFIG_OF_IRQ_DOMAIN
void irq_add(int *ops) {
    int irq = *ops;
}
#endif

int *ops = NULL;
#ifdef CONFIG_OF_IRQ
ops = &irq_ops;
#endif
irq_add(ops);
```

Only *certain* configurations have bugs

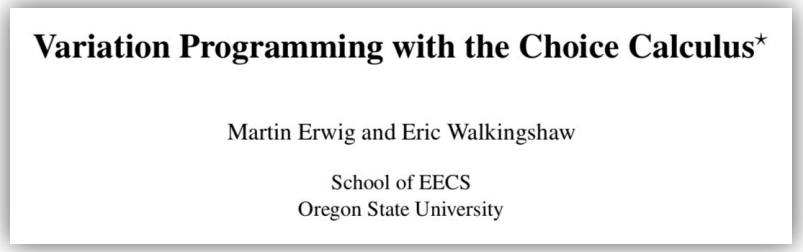
UCF

Why Don’t We Just Use a “Better” Language?

- Millions (billions?) of SLoC in active, widely-used projects
- Rust and Go will (hopefully) supplant C, but…
  - Rust has configuration macros
    - #[cfg] attributes
  - Go has build constraints

UCF

Great Research Efforts Tackling Conditional Compilation



Variation Programming with the Choice Calculus*

Martin Erwig and Eric Walkingshaw

School of EECS
Oregon State University

Variability-Aware Static Analysis at Scale: An Empirical Study

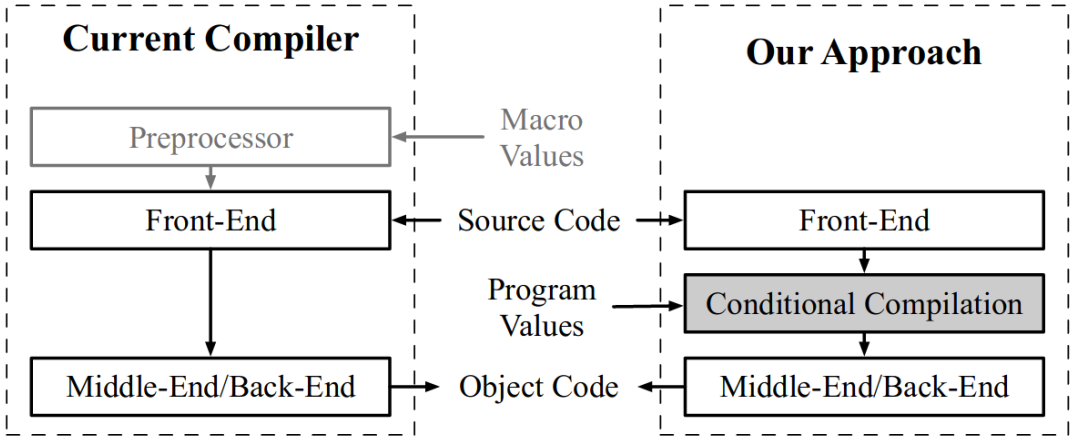
ALEXANDER VON RHEIN, CQSE GmbH, Germany
JÖRG LIEBIG, 4Soft GmbH, Germany
ANDREAS JANKER, Capgemini Deutschland GmbH, Germany
CHRISTIAN KÄSTNER, Carnegie Mellon University, USA
SVEN APEL, University of Passau, Germany

- New language
- Capture conditional compilation as variability
- Similar challenges for analysis tools

- "Lift" analyses to all configurations
- State-of-the-art is intraprocedural data flow
- Much left to match pure C tools, e.g.,
  - Points-to analysis
  - Abstract interpretation
  - Model checking
  - Separation logic
  - Symbolic execution

UCF

Best of Both Worlds: Keep C and Automate Conditional Compilation



- Replace preprocessor with a new compiler phase
- Configuration macros -> program values
- Conditional compilation becomes compiler optimization
  - Constant prop + dead code elimination = #ifdef

UCF

What are the Constructs of the Combined Language?

- Formal semantics typically relies on well-defined abstract syntax
- The combined C/preprocessor language has wonky syntax
  - Some usage should probably be restricted

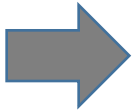
```
#define LBRACE {
int main() LBRACE
}
```

- What are the semantics of the combined language?
  - CMod formally defined #include usage [Srivastava et al., TSE 2008]

UCF

Map Preprocessor Usage to C

```
int *ops = NULL;
#ifdef CONFIG_OF_IRQ
ops = &irq_ops;
#endif
irq_add(ops);
```



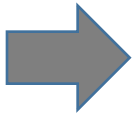
```
bool CONFIG_OF_IRQ;
int *ops = NULL;
if (CONFIG_OF_IRQ) {
ops = &irq_ops;
}
irq_add(ops);
```

- Macro -> program variable
- #ifdef -> C conditional
- Conditional compilation -> dead code elimination
- Transformation has been done before [Iosif-Lazar et al., Sci. Prog. 2017]

UCF

Some Constructs Are Questionable

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



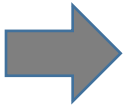
```
bool CONFIG_PSAUX;
if (CONFIG_PSAUX) {
  if (imajor == 10)
    i = 31;
  else
    i = iminor – 32;
} else {
  i = iminor – 32;
}
```

- Code duplication is awkward
  - Could alter conditions to have only two branches
- Should such cases be prohibited?

UCF

#ifdefs Can Appear Around Declarations

```
struct {
  u16 size;
#ifdef CONFIG_QUOTA
  int quota;
#endif
}
```



```
bool CONFIG_QUOTA;
struct {
  u16 size;

  int __attribute__((config (CONFIG_SMP))) quota;

}
```

- #ifdefs frequently surround declarations and definitions
- Akin to a dependent type
  - Type and existence of "quota" depends on program variable
- Similarity observed before in [Chen et al., TOPLAS 2014]

UCF

Conclusion

- Preprocessor dilemma
  - Great for performance
  - Bad for tools
- Goal: Replace preprocessor usage with C itself
  - Automate conditional compilation
  - Extensions for some preprocessor use cases
- Future work
  - Language definition: What are the right constructs? What should be illegal?
  - Empirical evaluation of how often translation is possible
  - New compiler phase and optimizations

@paul_gazzillo

pgazz.com


Fin