cHash: Detection of Redundant Compilations via AST Hashing

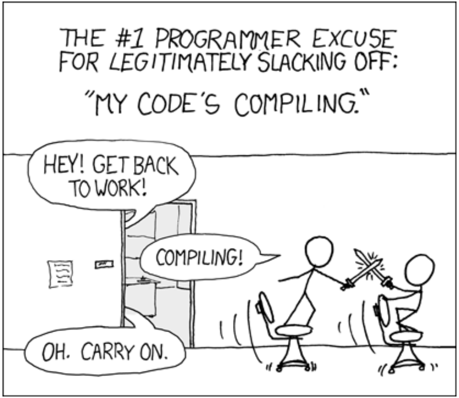
Christian Dietrich[‡], Valentin Rothberg[‡], Ludwig Füracker[*], Andreas Ziegler[*], Daniel Lohmann[‡]

[‡] Leibniz Universität Hannover
[*] Friedrich-Alexander-Universität Erlangen-Nürnberg

13. July 2017

supported by DFG

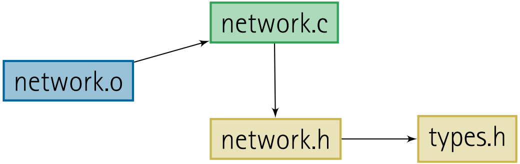
Compilation and Recompilation



Compile Time is not the Problem.

The Problem is **Recompile Time**

C/C++ Projects and Makefiles

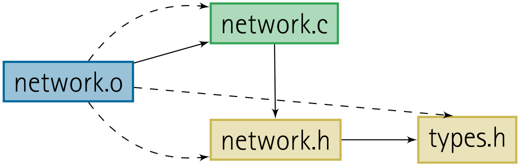


Makefile Fragment for network module

```
network.o: network.c network.h types.h
        cc -o network.o -c network.c
```

- In C projects, modular decomposition is done on file granularity
  - Headers export an interface, `#include` includes an interface
  - Source files (`.c`) are module implementations
- Recompilation decided upon timestamp comparison (e.g. `make`)
  - Dependencies of module are encoded in Makefile
  - Compare all dependent timestamps against last build artifact

C/C++ Projects and Makefiles

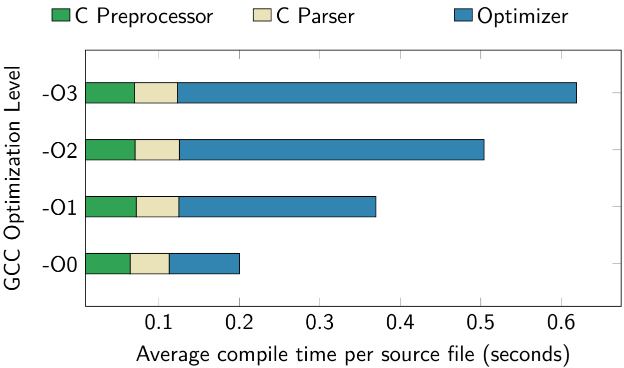


Makefile Fragment for network module

```
network.o: network.c network.h types.h
        cc -o network.o -c network.c
```

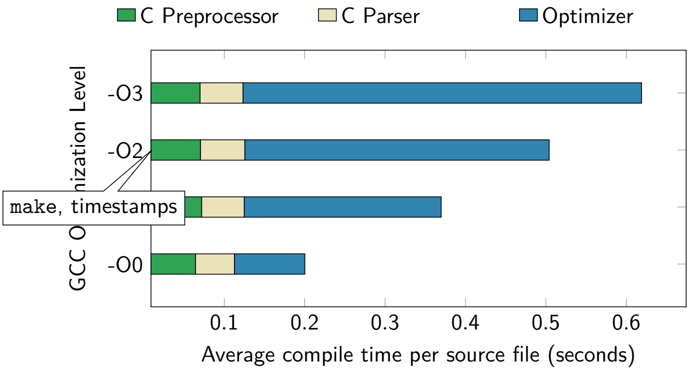
- In C projects, modular decomposition is done on file granularity
  - Headers export an interface, `#include` includes an interface
  - Source files (`.c`) are module implementations
- Recompilation decided upon timestamp comparison (e.g. `make`)
  - Dependencies of module are encoded in Makefile
  - Compare all dependent timestamps against last build artifact

When to detect change and stop compilation?



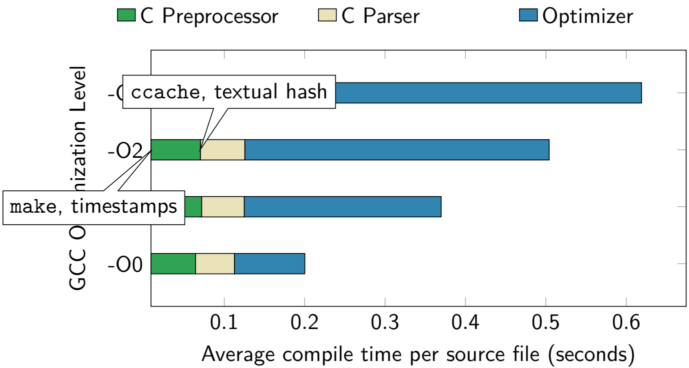
- Detect that a compilation will result in the same output
- The later we apply detection mechanism, the more precise it becomes

When to detect change and stop compilation?



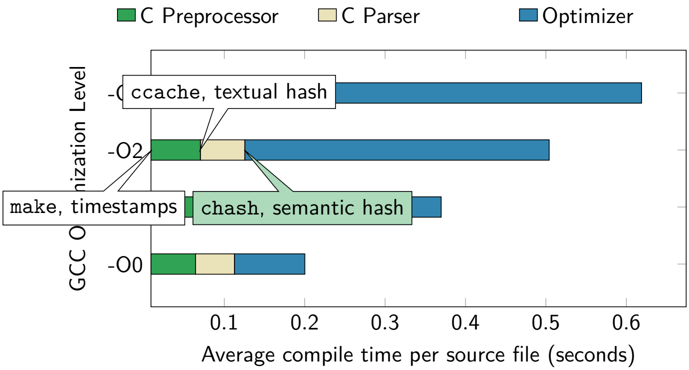
- Detect that a compilation will result in the same output
- The later we apply detection mechanism, the more precise it becomes

When to detect change and stop compilation?



- Detect that a compilation will result in the same output
- The later we apply detection mechanism, the more precise it becomes

When to detect change and stop compilation?

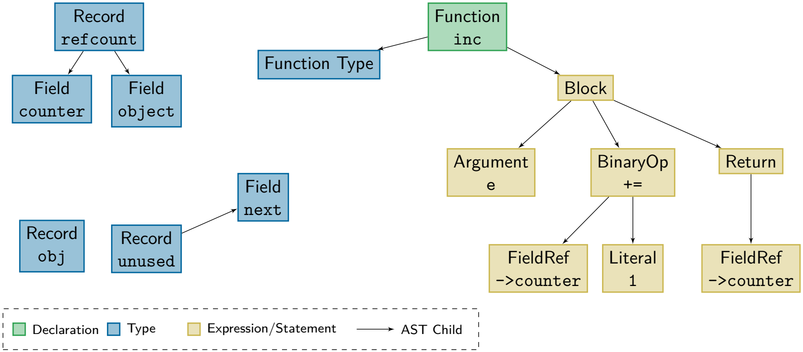


- Detect that a compilation will result in the same output
- The later we apply detection mechanism, the more precise it becomes
- In a nutshell: cHash calculates an hash after the parser

Outline

- Motivation and Introduction
- **cHash: Hash the abstract-syntax tree**
- Evaluation
  - ...with incremental (minimal) modifications
  - ...with commit-sized modifications
- Conclusion

Parse Tree and Semantic Analysis



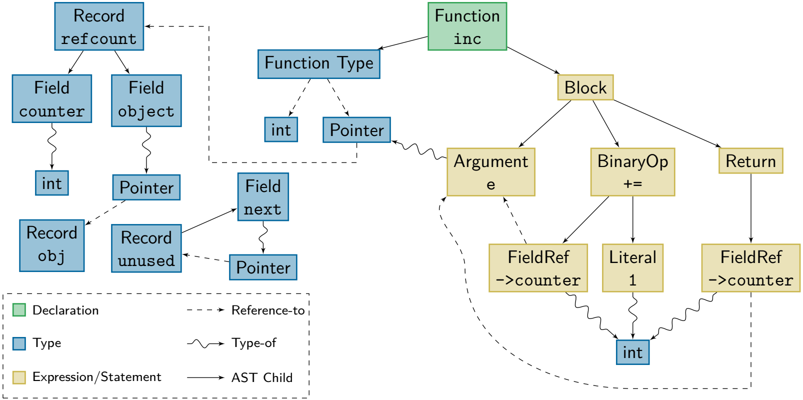

```c
struct unused {
    struct unused *next;
};

struct obj {};
```

```c
struct refcount
{
    int counter;
    struct obj * ptr;
};
```

```c
int
inc(struct refcount *e)
{
    e->counter += 1;
    return e->counter;
}
```

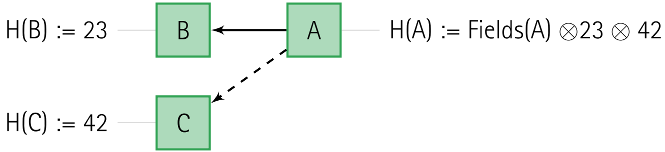
Parse Tree and Semantic Analysis



- Semantic analysis type checks and interconnects the AST
  - Nodes are annotated with their type
  - AST becomes a directed graph, it can include cycles

AST Hash for each Compilation Unit

- Calculate semantic fingerprint with a depth-first search
  - Hash relevant node properties (node class, operation,...)
  - Include hashes of all referenced nodes



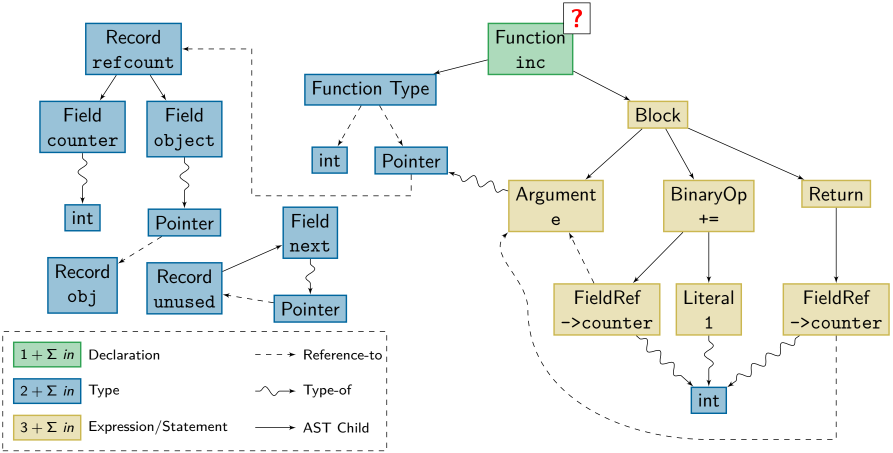
- Cycles in the semantically-enriched AST (recursive data structures)
  - Cache and reuse hash values for type definitions and declarations
  - Break cyclic dependencies by using a surrogate hash value

```
H(struct unused* next) := H("next") ⊗ H("struct unused*")
```

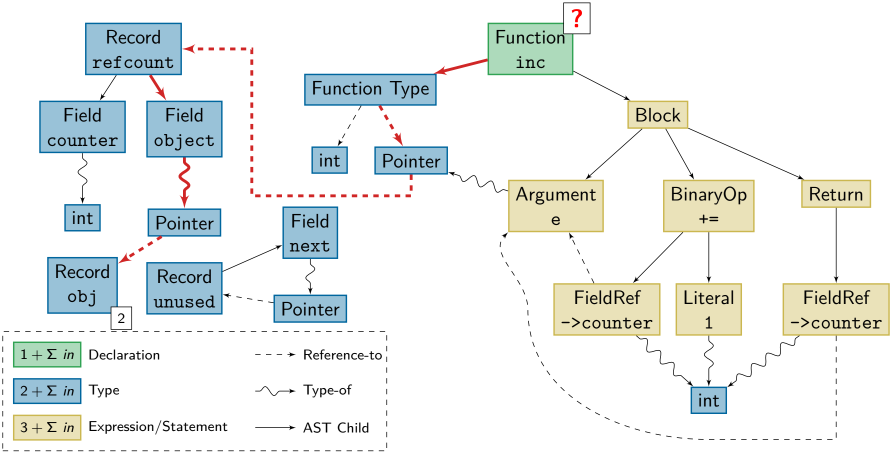
ISE SRA

AST Hashing with Depth-First Search

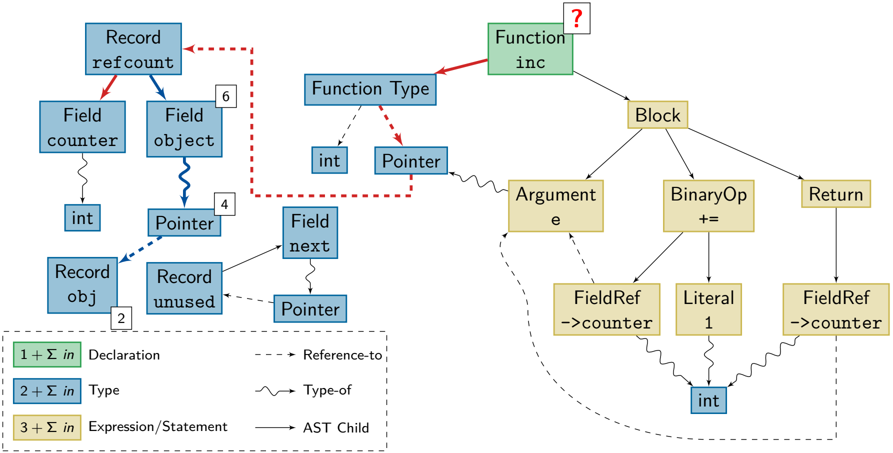
Leibniz Universität Hannover



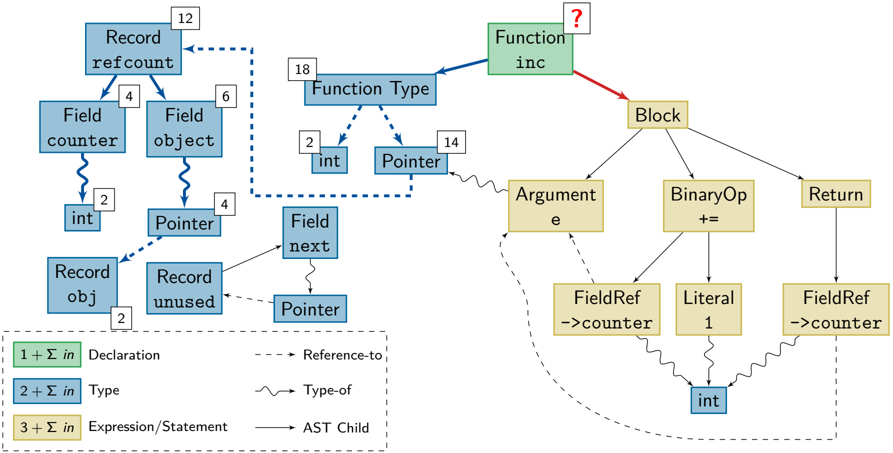
AST Hashing with Depth-First Search



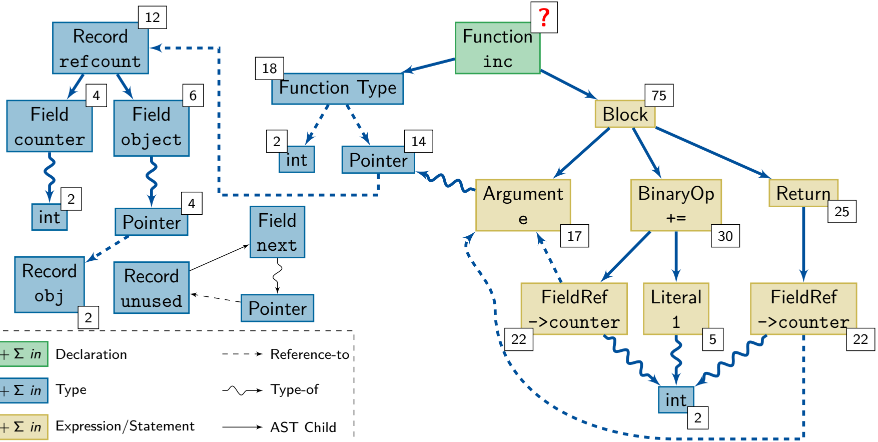
AST Hashing with Depth-First Search



AST Hashing with Depth-First Search

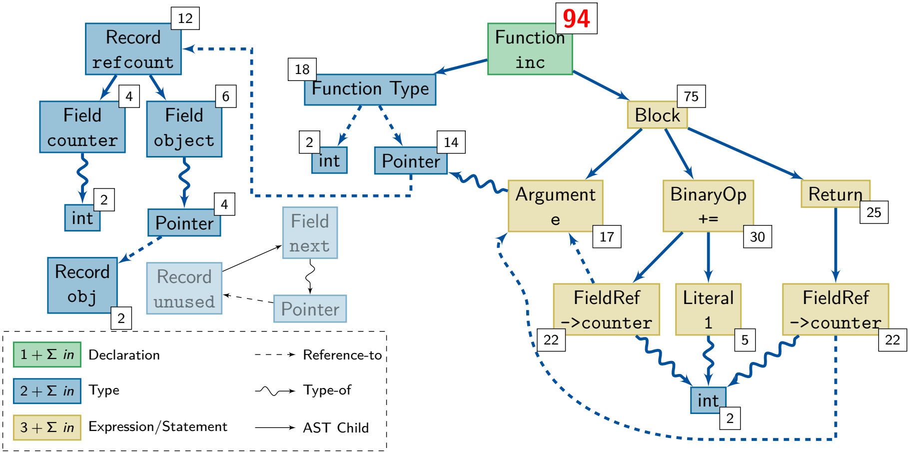


AST Hashing with Depth-First Search



$1 + \Sigma\ in$	Declaration	- - - →	Reference-to
$2 + \Sigma\ in$	Type	∿→	Type-of
$3 + \Sigma\ in$	Expression/Statement	——→	AST Child

AST Hashing with Depth-First Search



Integrate cHash into the Compiler

- We implemented cHash as a CLang plugin for C (GCC: in progress)
  1. Calculate hash over the semantically-enriched AST
  2. Read in hash for already existing object file
  3. Compare old hash and new hash
  4. Abort compilation on equality and update timestamp of object file

- Caching schemes for object files
  - CCache: A fixed size cache directory with the hash as index
  - cHash: Compare hash only with the last compilation result
  - Caching strategy is orthogonal to fingerprint mechanism

Outline

- Motivation and Introduction
- cHash: Hash the abstract-syntax tree
- **Evaluation**
  - **...with incremental (minimal) modifications**
  - ...with commit-sized modifications
- Conclusion

Incremental Modifications and Recompilation

Setting in the Reality

A developer works continuously on a source base. After a small modification to the source code, she recompiles the project to update the executables.

- Six C open source projects, 18k–742k SLOC, 3 build systems
- Start with a fully built source base, all object files are up-to-date
- Timestamp-based dependency checking of build system still in place
- Comparison between: Baseline, CCache, cHash

For each source/header file:

1. Modify file: (a) update timestamp or (b) useless textual change
2. Start build system to update all build artifacts (with `-j48`)
3. Get one rebuild duration for each source file

Incremental Modifications and Recompilation

Setting in the Reality

A developer works continuously on a source base. After a small modification to the source code, she recompiles the project to update the executables.

- Six C open source projects, 18k–742k SLOC, 3 build systems
- Start with a fully built source base, all object files are up-to-date
- Timestamp-based dependency checking of build system still in place
- Comparison between: Baseline, CCache, cHash

For each source/header file:

1. Modify file: (a) update timestamp or (b) useless textual change
2. Start build system to update all build artifacts (with `-j48`)
3. Get one rebuild duration for each source file

Best-case scenario for cHash

Incremental Modification: Textual Change

Project	Baseline	CCache	cHash
LUA	1.10 s	16.4 %	−59.6 %
mbedTLS	1.33 s	18.9 %	−4.3 %
musl	0.86 s	17.6 %	−4.7 %
bash	1.48 s	−9.2 %	−65.3 %
CPython	8.22 s	−24.7 %	−64.1 %
PostgreSQL	3.12 s	8.6 %	−41.8 %

Table: Average rebuild duration after a textual change.

- CCache cannot identify redundant build (hash on preprocessed code)
- cHash ignores purely syntactical changes

Outline

- Motivation and Introduction
- cHash: Hash the abstract-syntax tree
- **Evaluation**
  - ...with incremental (minimal) modifications
  - **...with commit-sized modifications**
- Conclusion

Commit-sized Changes and Recompilation

Setting in the Reality

A build server in a continuous integration system builds one uploaded change/commit after the other. Only the increment introduced by the change should lead to recompilations

- Build the last 500 non-merge commits from our six projects
- Prepare the source tree by fully building the parent commit
- Comparison between: CCache, cHash, CCache+cHash

For each commit file:

1. Apply the commit on the source
2. Start build system to update all build artifacts (with `-j48`)
3. Record the rebuild duration

Commit-sized Changes: Results

	Commits	Baseline	CCache	cHash	CCache+cHash
LUA	479	2.14 s	-38.8 %	**-49.3 %**	-46.7 %
mbedTLS	498	2.13 s	-20.7 %	-7.3 %	**-21.6 %**
musl	500	1.25 s	**-3.8 %**	0.7 %	-3.2 %
bash	108	2.88 s	-11 %	**-22.7 %**	-16 %
CPython	500	8.27 s	-46.4 %	-51.4 %	**-53.7 %**
PostgreSQL	498	5.63 s	-11 %	**-31.6 %**	-25.3 %

Table: Rebuild time for the last 500 non-merge changes.

- Some commits were broken, bash had only 128 commits
- Avg. compiler abortions: CCache (61 %), cHash (79.75 %)
- Avg. recompilation speedup: CCache (-23.63 %), cHash (-29.63 %)

Outline

- Motivation and Introduction
- cHash: Hash the abstract-syntax tree
- Evaluation
  - ...with incremental (minimal) modifications
  - ...with commit-sized modifications
- **Conclusion**

Summary and Conclusion

- cHash: AST hash is used to detect redundant build operation
  - ...excludes purely syntactic changes
  - ...excludes unreferenced types and declarations
- cHash improves recompilation times for developers and build farms
  - Build system agnostic, since compiler extension
  - Combinable with other detection schemes (timestamps, CCache)
- Future work for cHash and AST hashing
  - Integration into mainline compilers (at least the hashing)
  - Partial recompilation (e.g. a single function)
  - More complex languages with more emphasis on headers (C++)