GSoC 2016 - Finding code clones with clang

Raphael Isemann

Motivation

```
char *gUserAllow[k];
unsigned int gUserAlwLen[k];
char *gUserIgnore[k];
unsigned int gUserIgnLen[k];

if(strlen(gUserAllow[mth[jm]])
> gUserAllowLen[mth[jm]]-10)
{ … }

SPrintf(gUserAllow[mth[jm]],
        gUserIgnLen[mth[jm]],
        "%s %d",
        gUserAllow[mth[jm]],
        (int)pw->pw_uid);
```

Motivation

```
char *gUserAllow[k];
unsigned int gUserAlwLen[k];
char *gUserIgnore[k];
unsigned int gUserIgnLen[k];

if(strlen(gUserAllow[mth[jm]])
> gUserAllowLen[mth[jm]]-10)
{ … }

SPrintf(gUserAllow[mth[jm]],
        gUserIgnLen[mth[jm]],
        "%s %d",
        gUserAllow[mth[jm]],
        (int)pw->pw_uid);
```

Can we warn about this?

Motivation

```
char *gUserAllow[k];
unsigned int gUserAlwLen[k];
char *gUserIgnore[k];
unsigned int gUserIgnLen[k];

if(strlen(gUserAllow[mth[jm]])      if(strlen(gUserIgnore[mth[jm]])
> gUserAllowLen[mth[jm]]-10)        > gUserIgnLen[mth[jm]]-10)
{ … }                               { … }

SPrintf(gUserAllow[mth[jm]],        SPrintf(gUserIgnore[mth[jm]],
        gUserIgnLen[mth[jm]],               gUserIgnLen[mth[jm]],
        "%s %d",                            "%s %d",
        gUserAllow[mth[jm]],                gUserIgnore[mth[jm]],
        (int)pw->pw_uid);                   (int)pw->pw_uid);
```

←

copy-pasted

Overview

- Clone detection framework
  - Find similar AST nodes in an efficient way
  - Flexible through a modular constraint system (in review as D23418)
  - Available in clang/analysis/CloneDetection.h
- Checker for copy-paste errors
  - Uses the framework to find clones
  - Analyses them on variable pattern errors

Checking for copy-paste errors

- Enumerate variables
- Compare resulting integer vectors
- If the vectors are different, possible copy paste errors
  - Looking for hints such as only a single different variable
  - The bigger the clone, the more likely it was copy-pasted

Checking for copy-paste errors

```
    char *gUserAllow[k];
    unsigned int gUserAlwLen[k];
    char *gUserIgnore[k];
    unsigned int gUserIgnLen[k];

0   if(strlen(gUserAllow[mth[jm]])      if(strlen(gUserIgnore[mth[jm]])  0
1   > gUserAllowLen[mth[jm]]-10)        > gUserIgnLen[mth[jm]]-10)       1
    { … }                               { … }

0   SPrintf(gUserAllow[mth[jm]],        SPrintf(gUserIgnore[mth[jm]],    0
2           gUserIgnLen[mth[jm]],               gUserIgnLen[mth[jm]],    1
            "%s %d",                            "%s %d",
0           gUserAllow[mth[jm]],                gUserIgnore[mth[jm]],    0
3           (int)pw->pw_uid);                   (int)pw->pw_uid);        2
```

Problems to solve

- Some mathematical algorithms rely on switching variable names in only one place
- Order of arguments isn’t important for some functions,
- Erroneous clones with many pattern errors probably not detectable

Future work

- Get remaining framework patches merged
- Reducing the false-positive rate of the copy-paste checker
- You can help by running it over your project when it’s ready (in ~2 Months) and send any false-positives and positives to me.

Thanks!

Clone detection example

```
#include "clang/Analysis/CloneDetection.h"

CloneDetector Detector;
Detector.analyzeFunctionBody(Function1);

Detector.findClones(Result,
 // Search with hash codes
 HashConstraint(),
 // At least two Stmts define a clone
 MinGroupSizeConstraint(2),
 // Filter too small clones
 MinComplexityConstraint(140),
 // Make clones as big as possible
 OnlyLargestCloneConstraint());
```

Finding clones

- Need to be faster than $O(n^2)$
- Algorithm:
  1. Hash all statements: $O(n)$
  2. Sort hashes $O(n \log(n))$
  3. Find identical neighbors: $O(n)$
     a. n becomes much smaller here
  4. Check for hash collision $O(n)$
  5. Remove all clones that are children of bigger clones: $O(n^2)$

Hashing

- H(S)=H(data(S) + foreach child C: H(C))
- Like Stmt::Profile, but ...
  - Generated hash codes are identical across processes and TUs
  - Can hash all Stmts in the AST in linear time
  - But slower for single statements