Introduction to Semantic Patching of C programs with Coccinelle

Michele MARTONE

Leibniz Supercomputing Centre
Garching bei München, Germany

October 8, 2019

"What is Coccinelle?"

(from http://coccinelle.lip6.fr/)

> *Coccinelle is a program matching and transformation engine which provides the language SmPL (Semantic Patch Language) for specifying desired matches and transformations in C code. Coccinelle was initially targeted towards performing collateral evolutions in Linux. Such evolutions comprise the changes that are needed in client code in response to evolutions in library APIs, and may include modifications such as renaming a function, adding a function argument whose value is somehow context-dependent, and reorganizing a data structure. Beyond collateral evolutions, Coccinelle is successfully used (by us and others) for finding and fixing bugs in systems code.*

Coccinelle (http://coccinelle.lip6.fr)

Coccinelle *"...a program matching and transformation engine ... for specifying desired matches and transformations in C code"*

source to source translation

- arbitrary transformations of C code

refactoring

- making program structure easier to understand

spotting bugs

- detect bad code patterns (e.g. spot missing `free()`)

Contents overview

Description

Intro

Invocation and patching workflow

Semantic Patch Language (SmPL) elements overview

Example use cases

Outro

Description

Intro

Invocation and patching workflow

Semantic Patch Language (SmPL) elements overview

Example use cases

Outro

Table of Contents

Description
Intro
    example application field
Invocation and patching workflow
    spatch
    parse checks
    HPC experts branch integration
Semantic Patch Language (SmPL) elements overview
    minus
    quiz session: minus
    break till 11:00
    plus
    minus and plus
    quiz session: plusminus
    multiple insertions
    question time
    multiple rules
    depends
    sequences of rules
    rulekind
    break till 11:45
    ellipses: basic dots and dot variants
    metavariables
    constant
    identifier
    quiz session: identifier
    typedef
    type
    idexpression
    operator
    break till 14:00
    expression and isomorphisms
    field
    disjunction
    quiz session: disjunction
    conjunction
    quiz session: conjunction
    break till 15:00
    advanced ellipses
    ellipses for control flow
    format
    parameter
    parameter list
    inheritance
    declaration
    multidecls tricks
    position
    statement
    preprocessor
    scripting
    question time
    break till 16:15
Example use cases
    automating `printf`
    debugging
    cloning functions
    AoS to SoA
    generating co-routines
    Detect use and restructure C++
    inter-function relations
    data layout change
    insert pragma/specifier before loop
    custom comments
    insertion
Outro

Story of Coccinelle: a bugs' story

- a project from Inria (France)
- appeared in 2006
- originally for
  - *collateral evolutions* in Linux kernel drivers[1]
  - **smashing bugs** (hence the name)[2]

[1] https://git.kernel.org/pub/scm/linux/kernel/git/backports/backports.git/tree/patches

[2] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/coccinelle

A word of caution

Limitations

Coccinelle was born to serve the Linux kernel community.
It was not designed to cover **all** *possible C modification needs.*

But

...is **incredibly versatile**, and in active development!

Version used here:

spatch version 1.0.8 compiled with OCaml version 4.06.0

Coccinelle for HPC?

Possible workflow

- HPC expert gets a code branch / snapshot
- develops a series of semantic patches
- consults with code authors / community
- *backports* (propagates changes back to the original) at the very end of the optimization activity time frame

Possible *collateral evolutions* in HPC

- API change and necessary update
- introduce specific `pragma` directives
- Keyword add/remove
- introducing *intrinsics*
- simplify expressions
- AoS $\Longleftrightarrow$ SoA (Array of Structures $\Longleftrightarrow$ Structure of Arrays)
- parallelization: serial to OpenMP-parallel
- parallelization: serial to MPI-parallel
- serialization: remove OpenMP[3]
- serialization: remove MPI[4]

[3] Open Multi-Processing: standard for shared memory programming
[4] Message Passing Interface: standard for distributed memory programming

Further possible applications in HPC

- produce statistics and reports, analysis
  - e.g. of API misuse (bugs)
  - detecting notoriously inefficient code patterns
- instrument the code
- assist in a C $\Rightarrow$ C++ transition (e.g. cast after `malloc`,`calloc`)

Semantic patching invocation

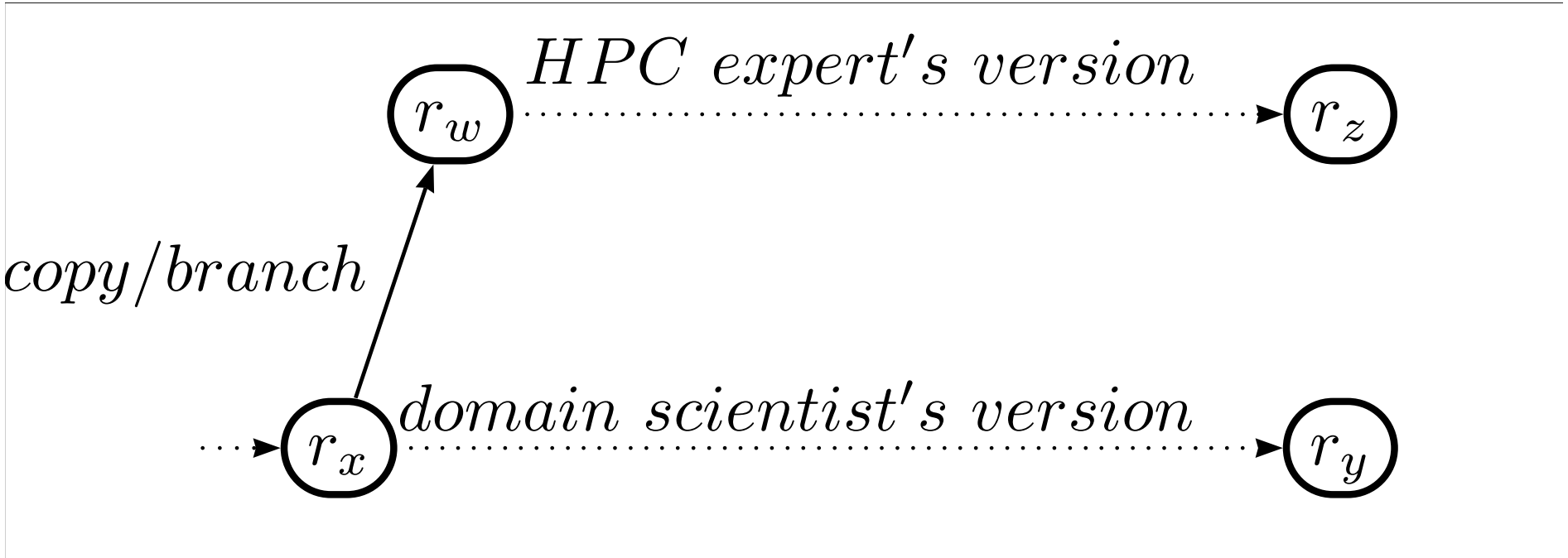
- identify a C file to be changed, say: `f.c`
- write a semantic patch representing the change: `$EDITOR sp.cocci`
- apply:

```
# produce patch:
spatch --sp-file sp.cocci f.c > sp.diff
# apply patch:
patch < sp.diff # this patches f.c
```

Important switches

```
spatch ...

  -j # threaded parallel

  --parse-cocci # parse rules

  --parse-c # parse C source

  --verbose-parsing

  --debug

  --local-includes # C headers

  --recursive-includes # C headers

  --iso-limit 0 # no isomorphisms
```

"Can you optimize my code?"



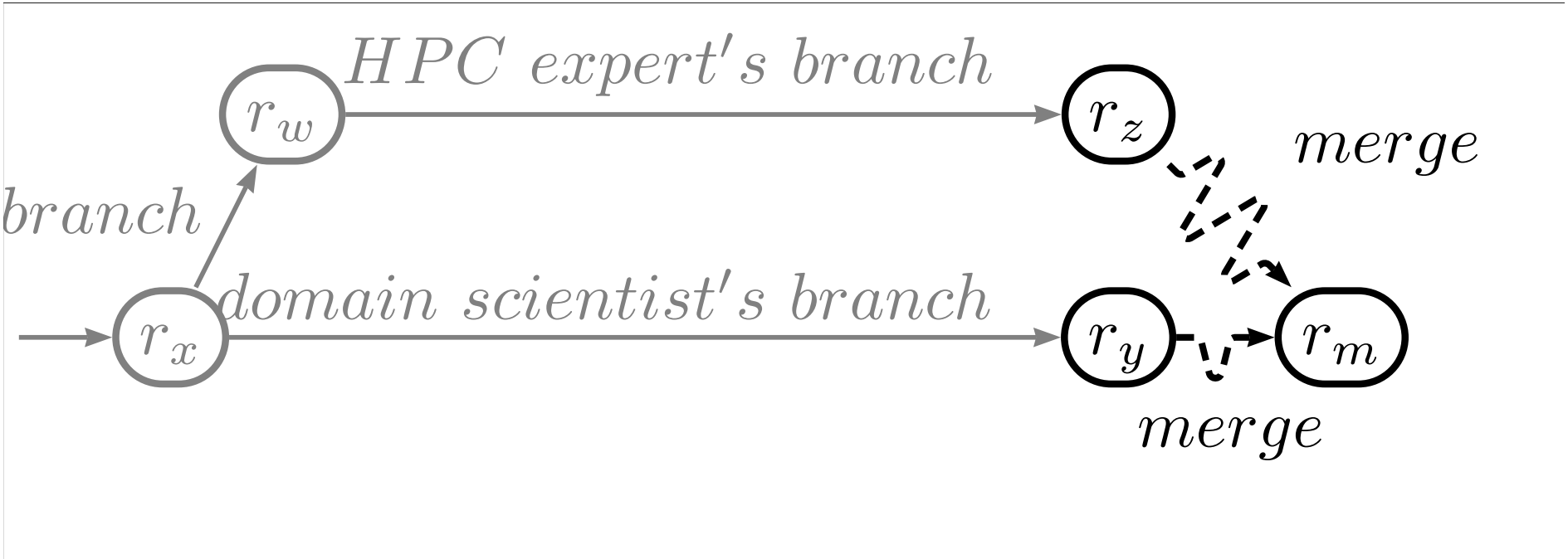
Possible workflow agreement

1. determine a "starting" relevant code snapshot

2.A. domain expert continues on usual development line

2.B. HPC expert works on another

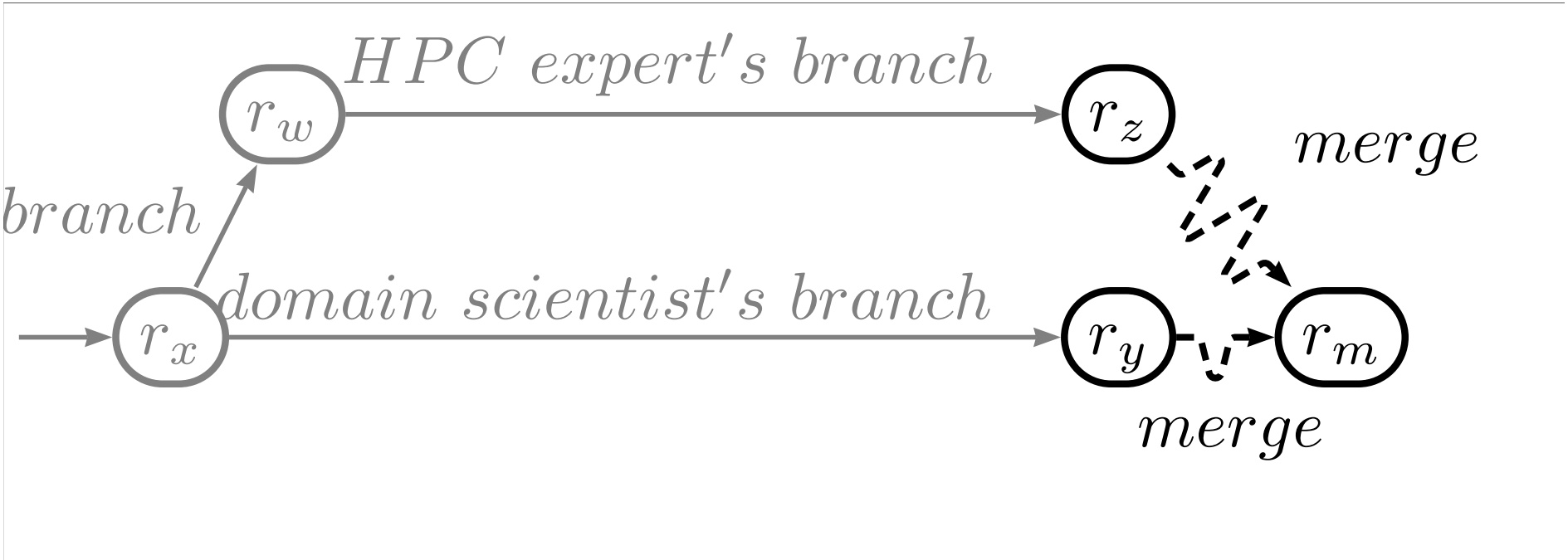
branch and *merge*



Possible workflow

- the two parties can work independently
- weeks to months pass
- at some point, performance-enhancing changes need *merge*

Backport / merge may be problematic



Merge? OK if branches did not diverge too much

- what if say, **every second line** changed?
- would you accept such a large "patch" to your code?

Possible *testing* workflow

Maintain `*.cocci` files meant as tests.

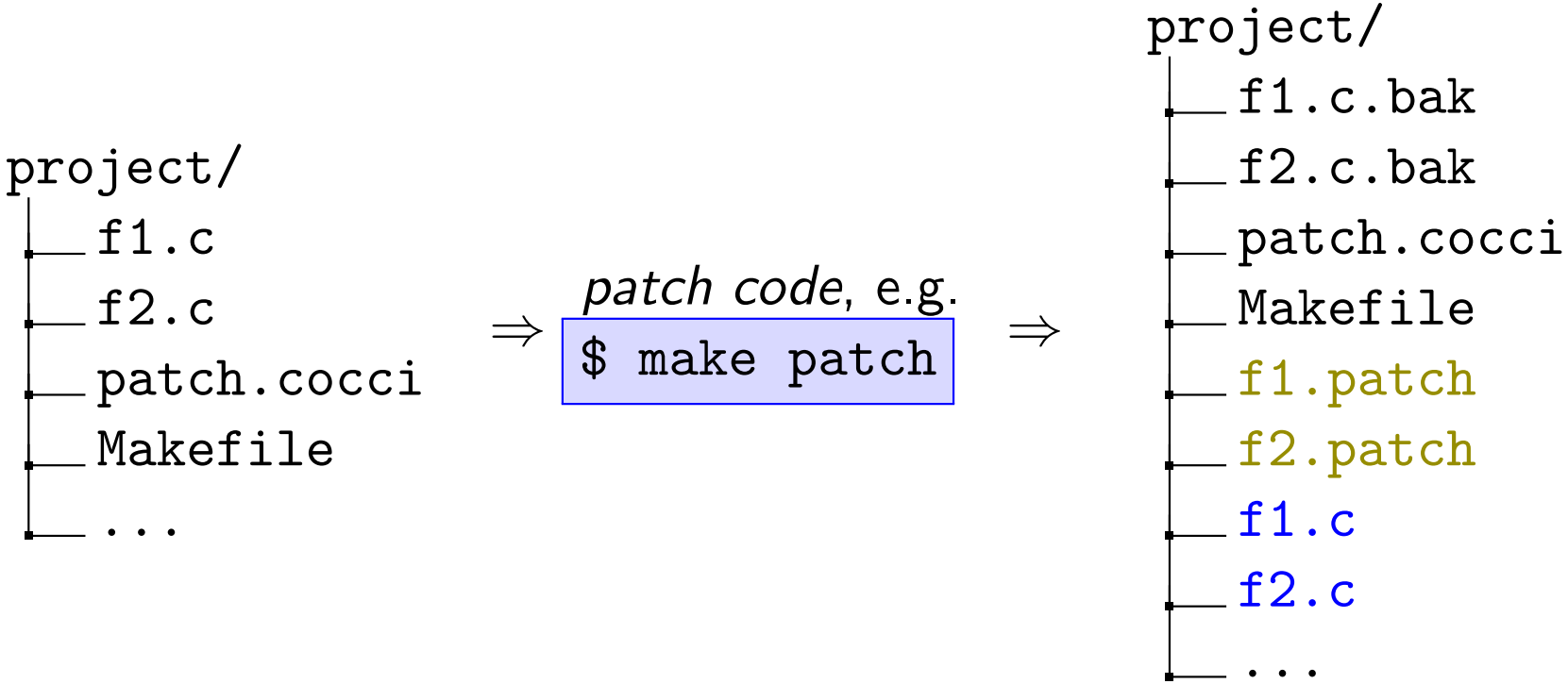
```
project/
├── src/
│   ├── f1.c
│   └── f2.c
├── tests/
│   ├── t1.cocci
│   └── t2.cocci
├── Makefile
└── ..
```

and have Makefile rules to trigger them, e.g.:

```
$ make check
$ ...
```

Possible *performance patch engineering* workflow

Develop e.g. a data layout change codified in *semantic patches*.
Maintain them together with sources.



Measure new code performance.
Change original sources if really needed.

...*semantic patching* with Coccinelle!

"...engine for specifying desired **matches** *and* **transformations** *in C code"*

...*semantic patching* with Coccinelle!

"...engine for specifying desired **matches** *and* **transformations** *in C code"*

Example AoS ⇒ SoA *conversion rules*

```
@@
identifier id,I;
type T;
@@
struct id { ...
- T  I;
+ T *I;
  ...
};
```

```
@@
expression E;
identifier AoS,J;
fresh identifier SoA=AoS##"_SoA";
@@
- AoS[E].J
+ SoA.J[E]
```

...*semantic patching* with Coccinelle!

"...engine for specifying desired **matches** *and* **transformations** *in C code"*

Example AoS ⇒ SoA *conversion rules*

```
@@
identifier id,I;
type T;
@@
struct id { ...
- T  I;
+ T *I;
  ...
};
```

```
@@
expression E;
identifier AoS,J;
fresh identifier SoA=AoS##"_SoA";
@@
- AoS[E].J
+ SoA.J[E]

```

Strengths

- **Generality**: multiple code forks, if semantic structures match
- **Flexibility**: conversion can be partial
- **Consistency**: patch only if semantic model satisfied

SmPL: semantic patch language of Coccinelle

- optional header followed by *rules*
- operates mainly on C *functions*
- almost *free-form*

Rules

- each rule can:
  - have a `@rulename@`
  - `@ depend on anotherrulename@`
  - have a few other `@ ...specifiers@`

  is then followed by `metadeclarations@@`, and then either:
  - `transformations`, or a
  - `script` in Python or OCaml

Let's review SmPL's elements in detail

minus code

- begins with `-` on first column
- is followed by C code to be **matched** and **deleted**

mainly:

- *statements*
- *expressions* in context

not e.g.

- *group declarations*
- *comments*
- any *preprocessor directive*

```
1  @@
2  @@
3  //context (optional):
4   a = 0;
5
6  // valid:
7  -a = 0;
8
9  // invalid:
10 -//a = 0;
```

minus code and *context* make a *minus transformation*

minus code

```
1 @@
2 @@
3
4 -a = 0;
```

```
0 @@ -1,6 +1,5 @@
1  int main() {
2     int a = 0;
3     a = 1;
4 -   a = 0;
5
6  }
```

example name: `cex_minus1.c`

minus code

```
@@
@@

-a = 0;
```

```
@@ -1,6 +1,4 @@
 int main() {
    int a = 0;
    a = 1;
-   a = 0;
-   a = 0;
 }
```

example name: `cex_minus1b.c`

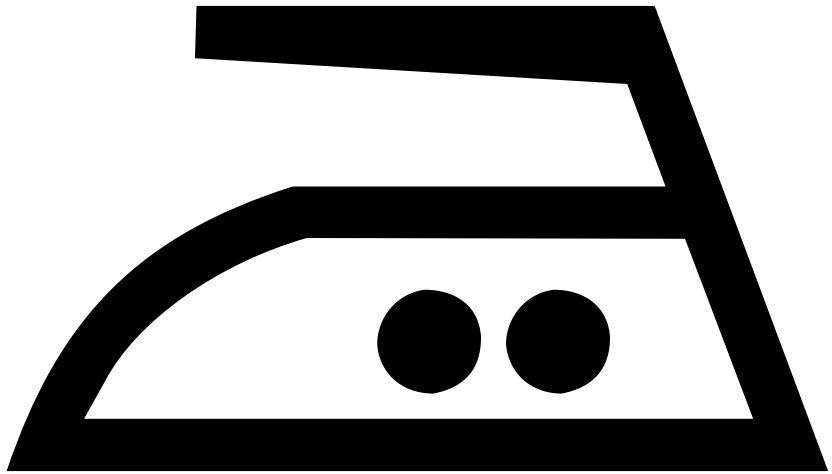
minus code and `--`

```
1 @@
2 @@
3
4 - --a;
```

```
0 @@ -1,4 +1,3 @@
1  int main() {
2     int a=0;
3 -   --a;
4  }
```

example name: `cex_minus2.c`

Quiz time!



minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - --a;
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

example name: `cex_quiz_minus1.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@

- --a;
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

example name: `cex_quiz_minus1.c`

ANSWER: Nothing: no `--a` *statement* matched in the code.

minus code: quiz

QUESTION: What will this delete?

```
@@
@@


- --a;
```

```
int main() {
  int A = 0;
  --A;
}
```

example name: `cex_quiz_minus1b.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - --a;
```

```
1 int main() {
2   int A = 0;
3   --A;
4 }
```

```
1 int main() {
2   int A = 0;
3   --A;
4 }
```

example name: `cex_quiz_minus1b.c`

ANSWER: Nothing: no `--a` *statement* matched in the code (C is case sensitive).

minus code: quiz

QUESTION: What will this delete?

```
@@
@@

- --
-   a;
```

```
int main() {
  int a = 0, b = 0;
  --a;
}
```

example name: `cex_quiz_minus1c.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@

- --
-   a;
```

```
@@ -1,4 +1,3 @@
 int main() {
    int a = 0, b = 0;
-   --a;
 }
```

```
int main() {
  int a = 0, b = 0;
  --a;
}
```

```
int main() {
  int a = 0, b = 0;
}
```

example name: `cex_quiz_minus1c.c`

ANSWER: Matches and deletes the `--a` statement (free-form of coccinelle rules).

minus code: quiz

QUESTION: What will this delete?

```
@@
@@

- -
- -
-    a;
```

```
int main() {
  int a = 0, b = 0;
  --a;
}
```

example name: `cex_quiz_minus1d.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4 -  -
5 -  -
6 -    a;
```

```
1 int main() {
2   int a = 0, b = 0;
3   --a;
4 }
```

```
1 int main() {
2   int a = 0, b = 0;
3   --a;
4 }
```

example name: `cex_quiz_minus1d.c`

ANSWER: Nothing: two `-` tokens do not make one `--`.

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5
6 - --a
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

example name: `cex_quiz_minus2.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@


- --a
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

example name: `cex_quiz_minus2.c`

ANSWER: Nothing: rule is invalid: cannot delete lone expression (see `--parse-cocci`).

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5
6 - --b
```

```
1 #define b a;
2 int main() {
3   int a = 0;
4   --b
5 }
```

example name: `cex_quiz_minus2b.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@


- --b
```

```
#define b a;
int main() {
  int a = 0;
  --b
}
```

```
#define b a;
int main() {
  int a = 0;
  --b
}
```

example name: `cex_quiz_minus2b.c`

ANSWER: Nothing: rule is invalid again. It won't delete lone expression here (and btw, no preprocessor expansion).

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - = --a
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

example name: `cex_quiz_minus3.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - = --a
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

example name: `cex_quiz_minus3.c`

ANSWER: Nothing. Rule is invalid (see `--parse-cocci`).

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - b=--a
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

example name: `cex_quiz_minus4.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@


- b=--a
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

example name: `cex_quiz_minus4.c`

ANSWER: Nothing: rule is invalid again: cannot delete lone expression here.

minus code: quiz

QUESTION: What will this delete?

```
@@
@@


- b=--a;
```

```
int main() {
  int a = 0, b = 0;
  b = --a;
}
```

example name: `cex_quiz_minus5.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - b=--a;
```

```
0 @@ -1,4 +1,3 @@
1  int main() {
2     int a = 0, b = 0;
3 -   b = --a;
4  }
```

```
1 int main() {
2   int a = 0, b = 0;
3   b = --a;
4 }
```

```
1 int main() {
2   int a = 0, b = 0;
3 }
```

example name: `cex_quiz_minus5.c`

ANSWER: This matches and deletes the matched statement. Fine.

minus code: quiz

QUESTION: What will this delete?

```
@@
@@

a -=
- b = --a
 +1;
```

```
int main() {
  int a = 0, b = 0;
  a -= b = --a +1;
}
```

example name: `cex_quiz_minus6b.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@

a -=
- b = --a
 +1;
```

```
@@ -1,4 +1,4 @@
 int main() {
    int a = 0, b = 0;
-   a -= b = --a +1;
+   a -= +1;
 }
```

```
int main() {
   int a = 0, b = 0;
   a -= b = --a +1;
}
```

```
int main() {
   int a = 0, b = 0;
   a -= +1;
}
```

example name: `cex_quiz_minus6b.c`

ANSWER: The relevant terms.

minus code: quiz

QUESTION: What will this delete?

```
1 @@
2 @@
3
4
5 - b=--a
```

```
1 int main() {
2   int a = 0, b = 0;
3   a -= b = --a +1;
4 }
```

example name: `cex_quiz_minus6.c`

ANSWER:

minus code: quiz

QUESTION: What will this delete?

```
@@
@@


- b=--a
```

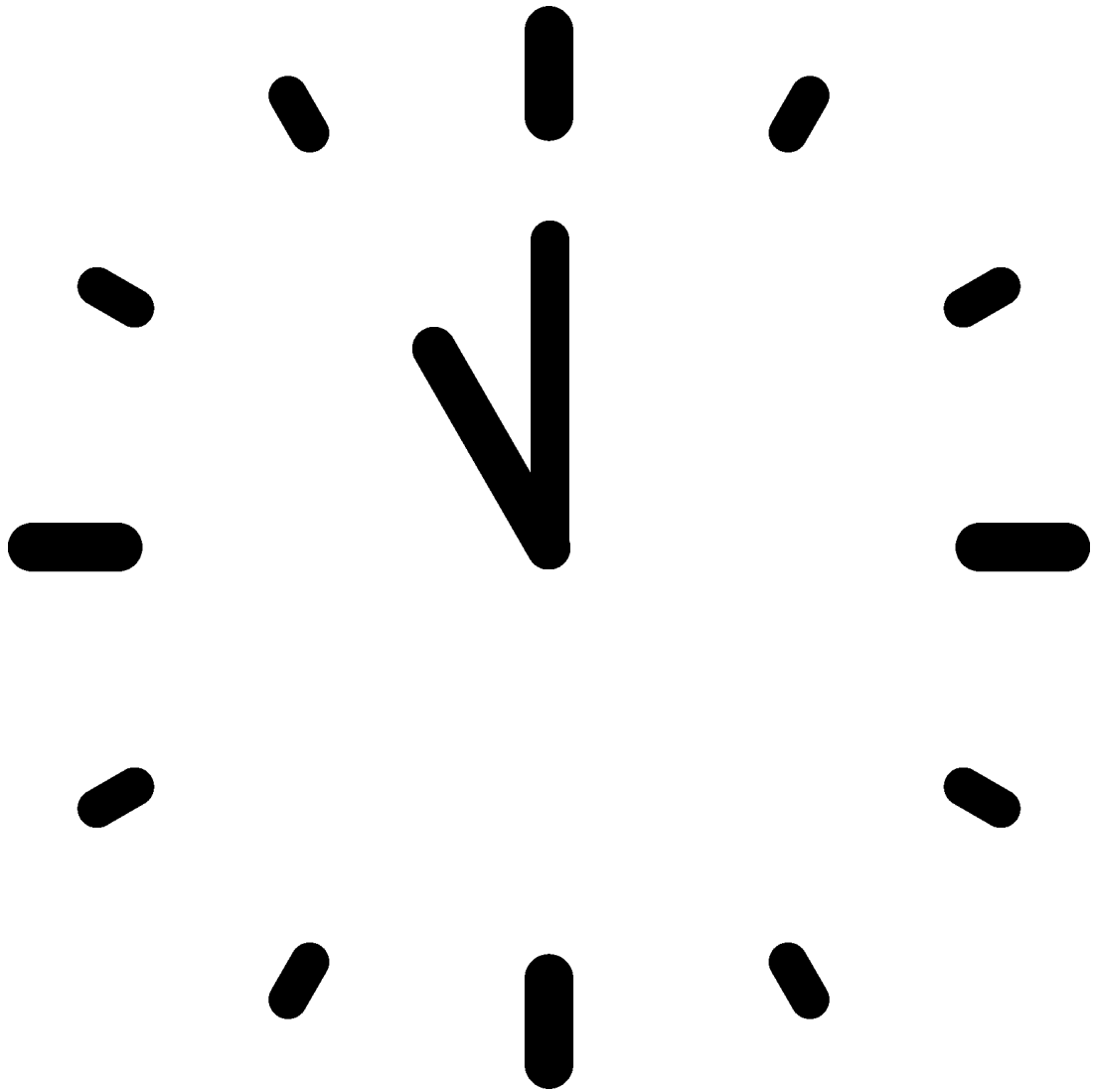
```
int main() {
  int a = 0, b = 0;
  a -= b = --a +1;
}
```

```
int main() {
  int a = 0, b = 0;
  a -= b = --a +1;
}
```

example name: `cex_quiz_minus6.c`

ANSWER: Nothing: invalid rule: does not parse a statement.

Break time! till 11:00



plus code

- begins with `+`, attaches to **context**
- followed by C code to be **inserted**
- can insert **anything** not breaking the code
  - (well almost... e.g. no statement after declaration)

```
1  @@
2  @@
3  //context (required):
4   a = 0;
5
6  //statement in plus code (valid):
7  +a = 0;
8
9  //comment in plus code  (valid):
10 +//a = 0;
```

plus code and *context* make a *plus transformation*

plus code: comment

```
@@
@@
 a = 0;
+// found a = 0 !
```

```
@@ -1,6 +1,8 @@
 int main() {
     int a = 0;
     a = 1;
     a = /* comment */ 0;
+    // found a = 0 !
     a = 0;
+    // found a = 0 !
 }
```

example name: `cex_plus1.c`

plus code: prefix `++`

```
1 @@
2 @@
3 + ++a;
4   return 0;
```

```
0 @@ -1,4 +1,5 @@
1  int main() {
2     int a=0;
3 +   ++a;
4     return 0;
5  }
```

example name: `cex_plus2.c`

plus code: variable declaration

```
@@
@@
 int a = 0;
+int b = 0;
```

```
@@ -1,3 +1,4 @@
 int main() {
    int a = 0;
+   int b = 0;
 }
```

example name: `cex_plus3.c`

plus code: mixed insert

insert declaration after declarations is ok

```
@@
@@
 int a = 0;
+int b = 0;
//+ ++a; // not allowed
// after C declarations
```

```
@@ -1,3 +1,4 @@
 int main() {
    int a = 0;
+   int b = 0;
 }
```

example name: cex_plus4.c

but explicit insert statement after declarations is not!

general transformations

- first **match** and **delete** code, then **add** code
- follow respective rules of *minus* and *plus* code
- context can be anywhere

```
1  @@
2  @@
3  //context:
4   a = 0;
5
6  // statement replacement code:
7  -a = 0;
8  +a = 1;
9
10 // only comment insertion is valid here:
11 -//a = 0;
12 +//a = 0;
```

minus and plus code combined

```
1 @@
2 @@
3
4 -a = 0;
5 +a = 1;
```

```
0 @@ -1,5 +1,5 @@
1  int main() {
2     int a = 0;
3     a = 1;
4 -   a = 0;
5 +   a = 1;
6  }
```

example name: `cex_plus_minus1.c`

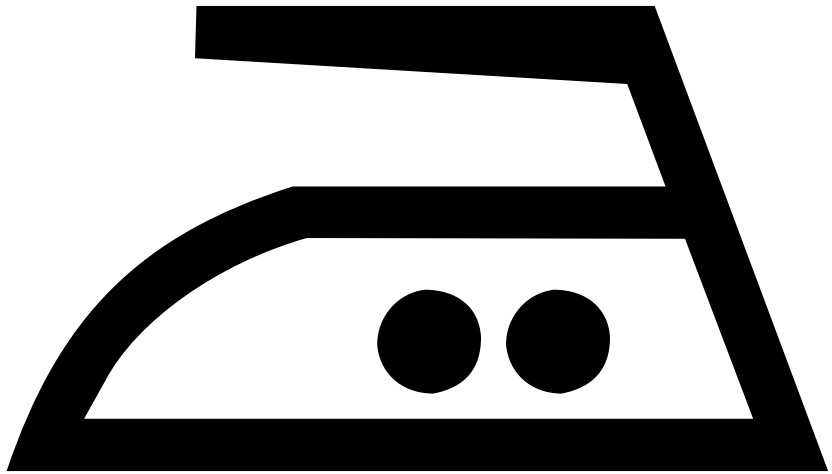
minus and plus code combined

```
@@
@@
-a = 1;
 a = 0;
+a = 1;
```

```
@@ -1,5 +1,5 @@
 int main() {
    int a = 0;
-   a = 1;
    a = 0;
+   a = 1;
 }
```

example name: `cex_plus_minus2.c`

Quiz time!



minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
+ ++b
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7b.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
+ ++b
```

```
@@ -1,5 +1,5 @@
 void f(int i) { }
 int main() {
   int a = 0, b = 0;
-  f(--a);
+  f(++b);
 }
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(++b);
}
```

example name: `cex_quiz_minus7b.c`

ANSWER: Expression is being recognized and substituted.

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
+
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7c.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
+
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7c.c`

ANSWER: Invalid rule. Expression not replaced by expression.

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7d.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7d.c`

ANSWER: Invalid rule. Expression not replaced by expression.

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
+ /* */
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7e.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


- --a
+ /* */
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7e.c`

ANSWER: Invalid rule: expression not replaced by an expression.

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@

 f(
- --a
+ /* */
 );
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7f.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@

 f(
- --a
+ /* */
 );
```

```
@@ -1,5 +1,5 @@
 void f(int i) { }
 int main() {
   int a = 0, b = 0;
-  f(--a);
+  f(/* */);
 }
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(/* */);
}
```

example name: `cex_quiz_minus7f.c`

ANSWER: It replaces a statement: OK.

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@

 f(
- --a
+ ++a
 );
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7g.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@

 f(
- --a
+ ++a
 );
```

```
@@ -1,5 +1,5 @@
 void f(int i) { }
 int main() {
   int a = 0, b = 0;
-  f(--a);
+  f(++a);
 }
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(++a);
}
```

example name: `cex_quiz_minus7g.c`

ANSWER: Replace expression to expression: OK.

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


-f(--a);
+f(/* */);
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

example name: `cex_quiz_minus7h.c`

ANSWER:

minus and plus code: quiz

QUESTION: What will this change?

```
@@
@@


-f(--a);
+f(/* */);
```

```
@@ -1,5 +1,5 @@
 void f(int i) { }
 int main() {
   int a = 0, b = 0;
-  f(--a);
+  f(/* */);
 }
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(--a);
}
```

```
void f(int i) { }
int main() {
  int a = 0, b = 0;
  f(/* */);
}
```

example name: `cex_quiz_minus7h.c`

ANSWER: It replaces a statement: OK.

QUESTION: guess, how many rules in this patch?

```
@@
@@
-a = 0;
+a = 1;
```

```
-c = 0;
+c = 1;
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 0;
  b = 0;
  c = 0;
}
```

example name: `cex_multiple_plusminus1.c`

ANSWER:

QUESTION: guess, how many rules in this patch?

```
@@
@@
-a = 0;
+a = 1;



-c = 0;
+c = 1;
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 0;
  b = 0;
  c = 0;
}
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 0;
  b = 0;
  c = 0;
}
```

example name: `cex_multiple_plusminus1.c`

ANSWER: one, with two plus/minus transforms

one rule can have several `+-` lines, remember?

```
@one_rule@
@@
-a = 0;
+a = 1;




-b = 0;
+b = 1;
```

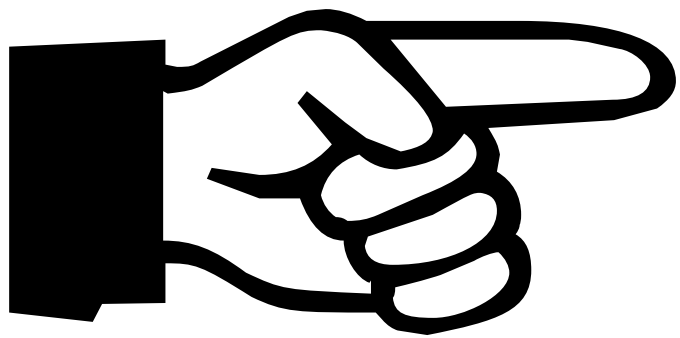
```
@@ -1,6 +1,6 @@
 int main() {
   int a = 0, b = 0, c = 0;
-  a = 0;
-  b = 0;
+  a = 1;
+  b = 1;
   c = 0;
 }
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 0;
  b = 0;
  c = 0;
}
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 1;
  b = 1;
  c = 0;
}
```

example name: `cex_multiple_plusminus2.c`

Question time!



multiple rules

- a file can have many *rules*
- rules apply in sequence, by default *independently*
- one can `#include "a_rules_file.cocci"`

```
#include "further_rules.cocci"
@rule_one@
@@
// first rule transformations...

@rule_two@
@@
// second rule transformations...

@rule_three@
@@
// ...
```

```

@only_one_rule@
@@
- delete1;
+ insert1;

- delete2;
+ insert2;

- delete3;
+ insert3;
// ...
```

multiple rules

```
@rule_a@
@@
-a = 0;
+a = 1;



@rule_c@
@@
-c = 0;
+c = 1;
```

```
@@ -1,6 +1,6 @@
 int main() {
    int a = 0, b = 0, c = 0;
-   a = 0;
+   a = 1;
    b = 0;
-   c = 0;
+   c = 1;
 }
```

example name: `cex_multiple_rules1.c`

`@rule_a@`: a=0; ⇒ a=1; `@rule_c@`: c=0; ⇒ c=1;

multiple independent rules

```
1  @rule_a@
2  @@
3  -a = 2;
4  +a = 1;
5
6
7
8  @rule_c@
9  @@
10 -c = 0;
11 +c = 1;
```

```
0  @@ -1,6 +1,6 @@
1   int main() {
2      int a = 0, b = 0, c = 0;
3      a = 0;
4      b = 0;
5  -   c = 0;
6  +   c = 1;
7   }
```

example name: `cex_multiple_rules_dep1.c`

`@rule_a@`: `a=2;` ⇒ `a=1;` `@rule_c@`: `c=0;` ⇒ `c=1;`

multiple rules with dependencies: failure

```
@rule_a@
@@
-a = 2;
+a = 1;



@rule_c depends on rule_a@
@@
-c = 0;
+c = 1;
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 0;
  b = 0;
  c = 0;
}
```

```
int main() {
  int a = 0, b = 0, c = 0;
  a = 0;
  b = 0;
  c = 0;
}
```

example name: `cex_multiple_rules_dep2.c`

`@rule_a@`: `a=2;` ⇒ `a=1;` `@rule_c@`: `c=0;` ⇒ `c=1;`

dependency can be `virtual` (with `-D`)

rules in a sequence

```
@@ @@
- a
+ T

@@ @@
- c
+ a

@@ @@
- T
+ c
```

```
@@ -1,6 +1,6 @@
 int main() {
    int a = 0, b = 0, c = 0;
-   a = 1;
+   c = 1;
    b = 2;
-   c = 3;
+   a = 3;
 }
```

example name: `cex_multiple_rules_seq1.c`

swap of `a` and `c` identifiers

rules in a sequence, again

```
@@ @@
- a
+ T

@@ @@
- b
+ A

@@ @@
- T
+ B
```

```
@@ -1,5 +1,5 @@
 int main() {
    int a = 0, b = 0;
-   a = 0;
-   b = 0;
+   B = 0;
+   A = 0;
 }
```

example name: `cex_multiple_rules_seq2.c`

declarations' identifiers unaffected!

declarations unaffected!

```
@@
@@
- a
+ c
```

```
@@ -1,6 +1,6 @@
 int main() {
     int a = 0;
     int b = 0;
-    a = 0;
+    c = 0;
     b = 0;
 }
```

example name: `cex_minus_idrk1.c`

default matched entity is `expression` (see p. 107)

implicit *rulekind* made explicit

```
1 @a2ce expression@
2 @@
3 - a
4 + c
```

```
0 @@ -1,6 +1,6 @@
1  int main() {
2     int a = 0;
3     int b = 0;
4 -   a = 0;
5 +   c = 0;
6     b = 0;
7  }
```

example name: `cex_minus_idrk2.c`

default *rulekind* is `expression` (also see p. 107)

using an `identifier` *rulekind*

```
@a2ci identifier@
@@
- a
+ c
```

```
@@ -1,6 +1,6 @@
 int main() {
-  int a = 0;
+  int c = 0;
   int b = 0;
-  a = 0;
+  c = 0;
   b = 0;
 }
```

example name: `cex_minus_idrk3.c`

forces matching of *all* identifiers

rules in a sequence, with `identifier` *rulekind*

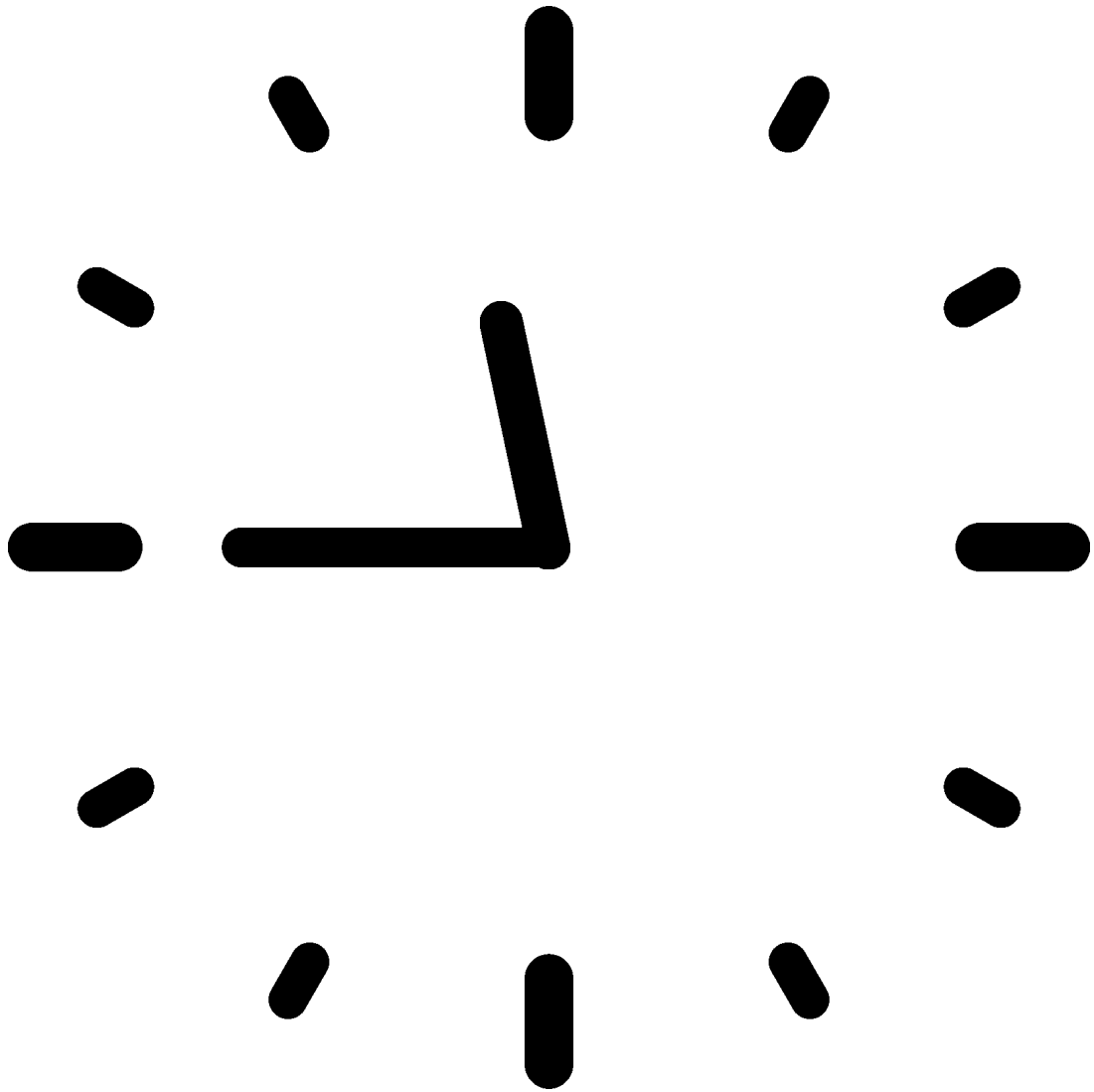
```
@a2T identifier@ @@
- a
+ T

@c2a identifier@ @@
- c
+ a

@T2c identifier@ @@
- T
+ c
```

```
@@ -1,6 +1,6 @@
 int main() {
-  int a = 0, b = 0, c = 0;
-  a = 1;
+  int c = 0, b = 0, a = 0;
+  c = 1;
   b = 2;
-  c = 3;
+  a = 3;
 }
```

example name: `cex_multiple_rules_seq3.c`

`@a2T@` : a ⇒ T `@c2a@` : c ⇒ a `@T2c@` : T ⇒ c

Break time! till 11:45



ellipses a.k.a. basic dots

- context-aware *shortest path* match of *almost anything*
- for **context** or **minus code**
- no metavariable: once matched, **cannot reuse**

```
1 @@
2 @@
3 - 0 + ...;
4 + // ...
5 -...
6 +// ...
7 -f( ... );
8 +f(/*...*/);
9  return ...;
```

- tokens
- expressions
  (more on p. 146)
- sequences of statements
  (*control flow* aspects on p. 155)
- *dot variants*:
  - *optional* match (control flow only)
    `<... ...>`
  - *required* match
    `<+... ...+>`

control-flow block match with ellipses (`...`)

```
@@
@@
{
+/* a block: */
  ...
}
```

```
@@ -1,9 +1,12 @@
 void a() {
+        /* a block: */
 }
 int main() {
+  /* a block: */
   if(1) {
+    /* a block: */
     a();
   }
   if(1)
     a();
 }
```

example name: `cex_ellipses.c`

ellipses for removal (`-...`)

```
@@
@@
{
-...
}
```

```
@@ -1,6 +1,3 @@
 void a() { }
 int main() {
-   int i;
-   if(1) { a(); }
-   if(1)   a();
 }
```

example name: `cex_ellipses2.c`

ellipses as context

```
@@
@@
-return
+exit(
  ...
+)
;
```

```
@@ -1,7 +1,7 @@
 int f() {
-  return 1;
+  exit( 1);
 }
 int main() {
   f();
-  return 0;
+  exit( 0);
 }
```

example name: `cex_basic_dots_1.c`

ellipses (`...`) for expression list match

```
@@
@@
-f(...);
```

```
@@ -1,6 +1,3 @@
 void f() { }
 int main() {
-   f( );
-   f(1);
-   f(1,2);
 }
```

example name: `cex_ellipses_parameter1.c`

ellipses (`...`) for expression list match

```
@@
@@
-f(...,2);
+f(-2);

@@
@@
// NOTE:
// removes f(1), too
-f( 1,...);
+f(-1);
```

```
@@ -1,7 +1,7 @@
 void f() { }
 int main() {
    f( );
-   f(1);
-   f(2,2);
-   f(1,1);
+   f(-1);
+   f(-2);
+   f(-1);
 }
```

example name: `cex_ellipses_parameter2.c`

Note

ellipses can match an *empty list*

ellipses (`...`) for array indices

```
@@

@@
- a[...]
+ a[0+0]
```

```
@@ -1,8 +1,8 @@
 int two() { return 2; }
 int main() {
   int a[3];
   int b[3];
- a[0]=1;
- a[b[0]]=0;
- a[two()]=2;
+ a[0 + 0]=1;
+ a[0 + 0]=0;
+ a[0 + 0]=2;
 }
```

example name: `cex_ellipses_squarebrackets1.c`

Note: for no auto-spacing, use `--smpl-spacing`.

ellipses for *optional transform* (`<... ...>`)

```
@@
@@
{
 <...
- int a = 1;
+ int a = 0;
  ...>
- ++a;
}
```

```
@@ -1,6 +1,6 @@
 int a;
-void f() { int a = 0; a--; ++a; }
-void g() { int a = 1; a--; ++a; }
-void h() {            a--; ++a; }
+void f() { int a = 0; a--; }
+void g() { int a = 0; a--; }
+void h() {            a--; }
 int main() {
 }
```

example name: `cex_ellipses3.c`

rule with `<... ...>` block matches even if block does not match

(more on p. 155)

ellipses for *required transform* (`<+...  ...+>`)

```
@@
@@
{
 <+...
- int a = 1;
+ int a = 0;
  ...+>
- ++a;
}
```

```
@@ -1,6 +1,6 @@
 int a;
 void f() { int a = 0; a--; ++a; }
-void g() { int a = 1; a--; ++a; }
+void g() { int a = 0; a--; }
 void h() {                 a--; ++a; }
 int main() {
 }
```

example name: `cex_ellipses4.c`

`<+...  ...+>` block matches *iff* match occurs

(more on p. 155)

metavariables

SmPL variables to **match** and **remove** / **manipulate**:

- tokens as: `symbol`, `constant`, `identifier`, `operator`, `type`, ...
- `expressions` and `statements`
- portions of other, structured C entities as `struct`s or `union`s ...
- `positions` in the code, a `format` string, ...

```
1 @@
2 identifier I =~ "i|j";
3 binary operator o;
4 type T = {int,double};
5 @@
6 -T I;
7  ...
8 -I o I;
```

- instantiate when parsed C entity *matches*
- no match ⇒ no instance
- certain metavariables' values can be **whitelisted** or **blacklisted**

constant

```
@@
constant K;
@@
- K
+ 0
```

```
@@ -1,7 +1,7 @@
 int main()
 {
-   int a = 10;
-   float f = 1.0f;
-   double d = 3.14;
-   const char * c = "string";
+   int a = 0;
+   float f = 0;
+   double d = 0;
+   const char * c = 0;
 }
```

example name: `cex_constant.c`

Numerical constants and string literals

identifier

- variables, functions, preprocessor symbols
- not: C keywords, type names (unless `struct`, `enum` names)
- coin `fresh` identifiers off matched ones
- different `identifier`s can match the same identifier
- usable with `declaration`s (see p. 172)

```
1  @@
2  identifier I;
3  @@
4  - I
5  + I+I;
6
7  @@
8  identifier J,K;
9  @@
10 //  ...
```

- filter with a `"regexp"`
- select from a `= {list}`
- blacklist from a `!= {list}`

identifier

```
@@
identifier I;
@@
- int I;
+ double I;
```

```
@@ -1,6 +1,6 @@
 int main()
 {
-   int a;
-   int b;
+   double a;
+   double b;
    double c;
 }
```

example name: `cex_identifier.c`

fresh identifier

```
@@
identifier I;
fresh identifier J = "_" ## I;
@@
- int I;
+ int I,J;
```

```
@@ -1,6 +1,8 @@
 int main()
 {
     int a;
+    int _a;
     int b;
+    int _b;
     double c;
 }
```

example name: `cex_identifier_fresh.c`

`identifier` *regexp*-based filtering

```
@@
identifier I =~ "^d_[13]$";
@@
- int I;
+ double I;
```

```
@@ -1,8 +1,8 @@
 int main()
 {
    int  a_1;
-   int  d_1;
+   double d_1;
    int  d_11;
    int dd_1;
    int  d_2;
 }
```

example name: `cex_identifier_regexp.c`

`identifier` list-based filtering

```
1 @@
2 identifier I = {a,c};
3 @@
4 - int I;
5 + double I;
```

```
0 @@ -1,6 +1,6 @@
1  int main()
2  {
3 -   int a;
4 +   double a;
5     int b;
6     double c;
7  }
```

example name: `cex_identifier_flt.c`

Whitelisting of identifiers `a`, `c`

`identifier` list-based filtering

```
1 @@
2 identifier I != {b,c};
3 @@
4 - int I;
5 + double I;
```

```
0 @@ -1,6 +1,6 @@
1  int main()
2  {
3 -   int a;
4 +   double a;
5     int b;
6     double c;
7  }
```

example name: `cex_identifier_excl.c`

Blacklisting of identifiers `b,c`

Full power when used with inheritance — see p. 168, p. 170.

`identifier list` to delete a macro

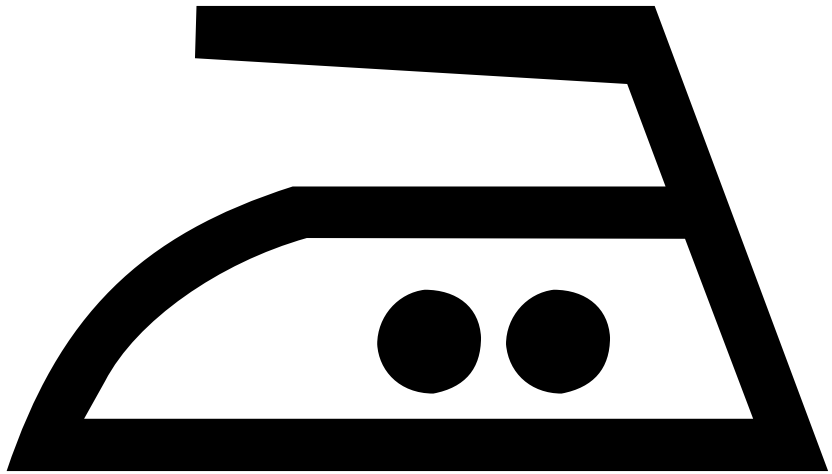
```
@@
identifier M;
identifier list IL;
expression E1,E2;
@@
- #define M(IL) E1+E2
```

```
@@ -1,2 +1,2 @@
-#define sumof(A,B) (A)+(B)
+
 int main() { }
```

example name: `cex_identifier_list1.c`

Caution: macro manipulation is a pretty new functionality

Quiz time!



identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier I;
@@



- I = I (I);
```

```
int f(int i) {
  return i;
}
int main() {
  int i;
  i = f(i);
}
```

example name: `cex_quiz_identifier2.c`

ANSWER:

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier I;
@@



- I = I (I);
```

```
int f(int i) {
  return i;
}
int main() {
  int i;
  i = f(i);
}
```

```
int f(int i) {
  return i;
}
int main() {
  int i;
  i = f(i);
}
```

example name: `cex_quiz_identifier2.c`

ANSWER: Nothing: no expression with identifier occurring *thrice* exists

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier A,B,C;
@@



- A = B (C);
```

```
int f(int i) {
  return i;
}
int main() {
  int i;
  i = f(i);
}
```

example name: `cex_quiz_identifier3.c`

ANSWER:

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier A,B,C;
@@



- A = B (C);
```

```
@@ -1,7 +1,6 @@
 int f(int i) {
   return i;
 }
 int main() {
   int i;
-  i = f(i);
 }
```

```
int f(int i) {
  return i;
}
int main() {
  int i;
  i = f(i);
}
```

```
int f(int i) {
  return i;
}
int main() {
  int i;
}
```

example name: `cex_quiz_identifier3.c`

ANSWER: Both `A` and `C` match `i`; `B` matches `f`.

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier A,B;
identifier C != A;
@@



- A = B (C);
```

```
int f(int i) {
  return i;
}
int main() {
  int i,j;
  i = f(i);
}
```

example name: `cex_quiz_identifier4.c`

ANSWER:

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier A,B;
identifier C != A;
@@



- A = B (C);
```

```
int f(int i) {
  return i;
}
int main() {
  int i,j;
  i = f(i);
}
```

```
int f(int i) {
  return i;
}
int main() {
  int i,j;
  i = f(i);
}
```

example name: `cex_quiz_identifier4.c`

ANSWER: Nothing (parse error). Unfortunately `identifier !=` is not supported *this way* – need *inheritance* (see later p. 168) for that.

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier A,B,C;
@@


- A = (B) C;
```

```
int main() {
  int i;
  double d;
  d = (double) i;
}
```

example name: `cex_quiz_identifier1.c`

ANSWER:

identifier code: quiz

QUESTION: What will this delete?

```
@@
identifier A,B,C;
@@


- A = (B) C;
```

```
int main() {
  int i;
  double d;
  d = (double) i;
}
```

```
int main() {
  int i;
  double d;
  d = (double) i;
}
```

example name: `cex_quiz_identifier1.c`

ANSWER: Nothing: `identifier` won't match type `double`.

typedef

- coccinelle heuristics infer `typedef`s from declarations...
- ...but in-rule `typedef`s are needed for in-rule casts

```
1 @@
2 typedef dbl;
3 @@
4 -d = (dbl) i;
```

typedef

```
@@

@@
//remove assignment with cast:
- d = (double) i;
```

```
@@ -1,7 +1,6 @@
 typedef double dbl;
 int main() {
    dbl d;
    int i;
    d = (dbl) i;
-   d = (double) i;
 }
```

example name: `cex_typedef_cast0.c`

typedef

```
@@

@@
// unparsable rule
- d = (dbl) i;
```

```
typedef double dbl;
int main() {
  dbl d;
  int i;
  d = (dbl) i;
  d = (double) i;
}
```

```
typedef double dbl;
int main() {
  dbl d;
  int i;
  d = (dbl) i;
  d = (double) i;
}
```

example name: cex_typedef_cast1.c

typedef

```
@@
typedef dbl;
@@
// rule now parsable:
- d = (dbl) i;
```

```
@@ -1,7 +1,6 @@
 typedef double dbl;
 int main() {
   dbl d;
   int i;
-  d = (dbl) i;
   d = (double) i;
 }
```

example name: `cex_typedef_cast2.c`

type

- matches C *types*
- can: **whitelist, blacklist, transform** types
- unlike `fresh identifier` (see p. 80), cannot create "`fresh` types"

```
@@
type T;
@@
-T    i;
+int  i;
```

```
@@
type T = {int, float**};
identifier I;
@@
-T *I;
+T  I;
```

`type`: change a declaration

```
@@
type T;
identifier I;
@@
-T I;
+double I;
```

```
@@ -1,6 +1,6 @@
 int main()
 {
-   int i;
-   float f;
+   double i;
+   double f;
    double d;
 }
```

example name: `cex_type.c`

`type` filtering

```
@@
type T = {int, float};
identifier I;
@@
-T I;
+short I;
```

```
@@ -1,6 +1,6 @@
 int main()
 {
-   int i;
-   float  f;
+   short i;
+   short f;
    double d;
 }
```

example name: `cex_type_flt.c`

pointer `type`s

```
@@
type T = {int, float**};
identifier I;
@@
-T  I;
+T *I;
```

```
@@ -1,9 +1,9 @@
 int main()
 {
-   int i;
+   int *i;
    const int c=0;
    double d;
    float f;
    float  *fp;
-   float **fpp;
+   float ***fpp;
 }
```

example name: `cex_type_flt_ptr.c`

`idexpression T I`

- match `identifier`s of a certain `type`
- have to be expressions (so, e.g. no declarations)

```
1 @@
2 type T;
3 local idexpression T I,E;
4 @@
5 -E = (T) I;
6 +E = I;
```

can filter:

- on definition scope: `local` or `global`
- on identifier

idexpression T I

```
1 @@
2 type T;
3 idexpression T I,E;
4 @@
5 -E = (T) I;
6 +E = I;
```

```
0  @@ -1,9 +1,9 @@
1      int g;
2   int main()
3   {
4      int i,j;
5      double d;
6  -   i = (int) j;
7  -   i = (int) g;
8  +   i = j;
9  +   i = g;
10     i = (int) d;
11  }
```

example name: cex_idexpression.c

Example: remove useless cast

idexpression T I

```
@@
type T;
idexpression T I={i,k};
@@
-I
+j
```

```
@@ -1,8 +1,8 @@

 int main()
 {
    int i=0;
    int j=0;
-   i++;
+   j++;
    j++;
 }
```

example name: cex_idexpression2.c

Can filter as identifier

local idexpression

```
1 @@
2 type T;
3 local idexpression T I,E;
4 @@
5 -E = (T) I;
6 +E = I;
```

```
0 @@ -1,7 +1,7 @@
1  int i,j; // globals
2  int main()
3  {
4     int k,l; // locals
5     i = (int) j;
6 -   k = (int) l;
7 +   k = l;
8  }
```

example name: `cex_idexpression_local.c`

Can filter on definition scope

global idexpression

```
1 @@
2 type T;
3 global idexpression T I,E;
4 @@
5 -E = (T) I;
6 +E = I;
```

```
0 @@ -1,7 +1,7 @@
1  int i,j; // globals
2  int main()
3  {
4     int k,l; // locals
5 -   i = (int) j;
6 +   i = j;
7     k = (int) l;
8  }
```

example name: `cex_idexpression_global.c`

Can filter on definition scope

operator

- matches, whitelists, blacklists operators
- either `binary` or `assignment`

```
@@
identifier A,B;
binary operator o;
assignment operator a;
@@
(
  A o B;
+// binary
|
  A a B;
+// assignment
)
```

operator

```
@@
identifier A,B;
binary operator o;
assignment operator a;
@@
(
  A o B;
+// binary
|
  A a B;
+// assignment
)
```

```
@@ -1,8 +1,12 @@
 int main()
 {
    int a = 0, b = 0;
    a + b;
+   // binary
    b << a;
+   // binary
    a = b;
+   // assignment
    b <<= a;
+   // assignment
 }
```

example name: `cex_operator_1.cpp`

`operator` blacklisting

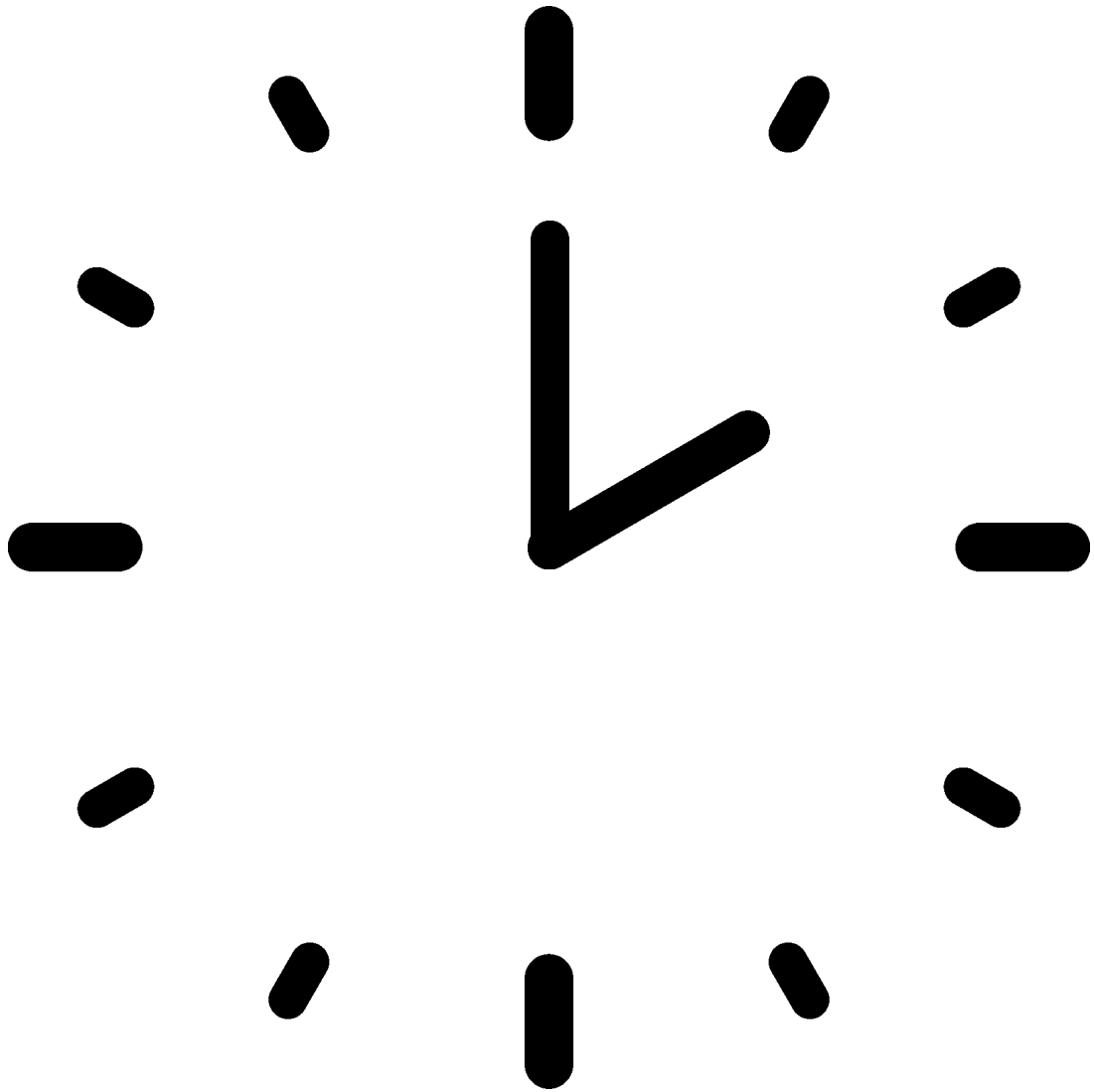
```
@@
identifier A,B;
binary operator o != {+,-};
@@
- A o B;
```

```
@@ -1,8 +1,7 @@
 int main()
 {
    int a = 0, b = 0;
    a + b;
-   b << a;
    a = b;
    b <<= a;
 }
```

example name: `cex_operator_2.cpp`

and so on for *whitelisting*

Break time! till 14:00



expression

- matches C *expressions* (*terms with values*)
- from lone identifiers to nested ones with *side effects*
- note that `if`, `for`, `while` constructs are not expressions

```
@@
expression E1,E2;
@@
-a = E;
 ...
 f(E);
+a = E;
```

```
@@
expression E1,E2;
@@
-oldfunc(E1,E2);
+newfunc(E1,E2);
```

`expression`: a broad category

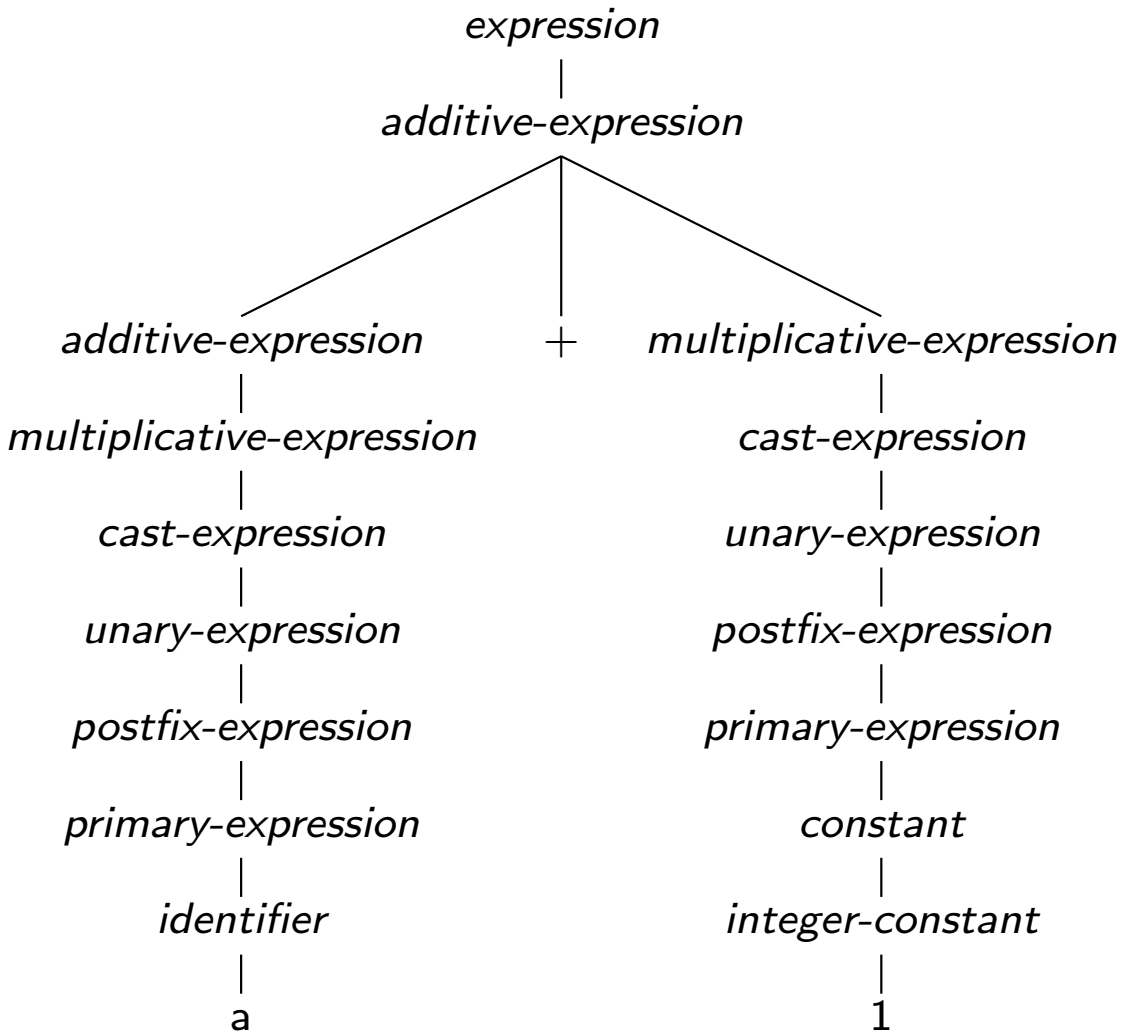
```
@@
expression E;
@@
- E;
```

```
@@ -1,8 +1,3 @@
 int main() {
   int i;
- // ANSI-C's
- // "expression-statement"s:
- sqrt(1.0);
- "a string";
- 1 ? 1 : 0;
 }
```

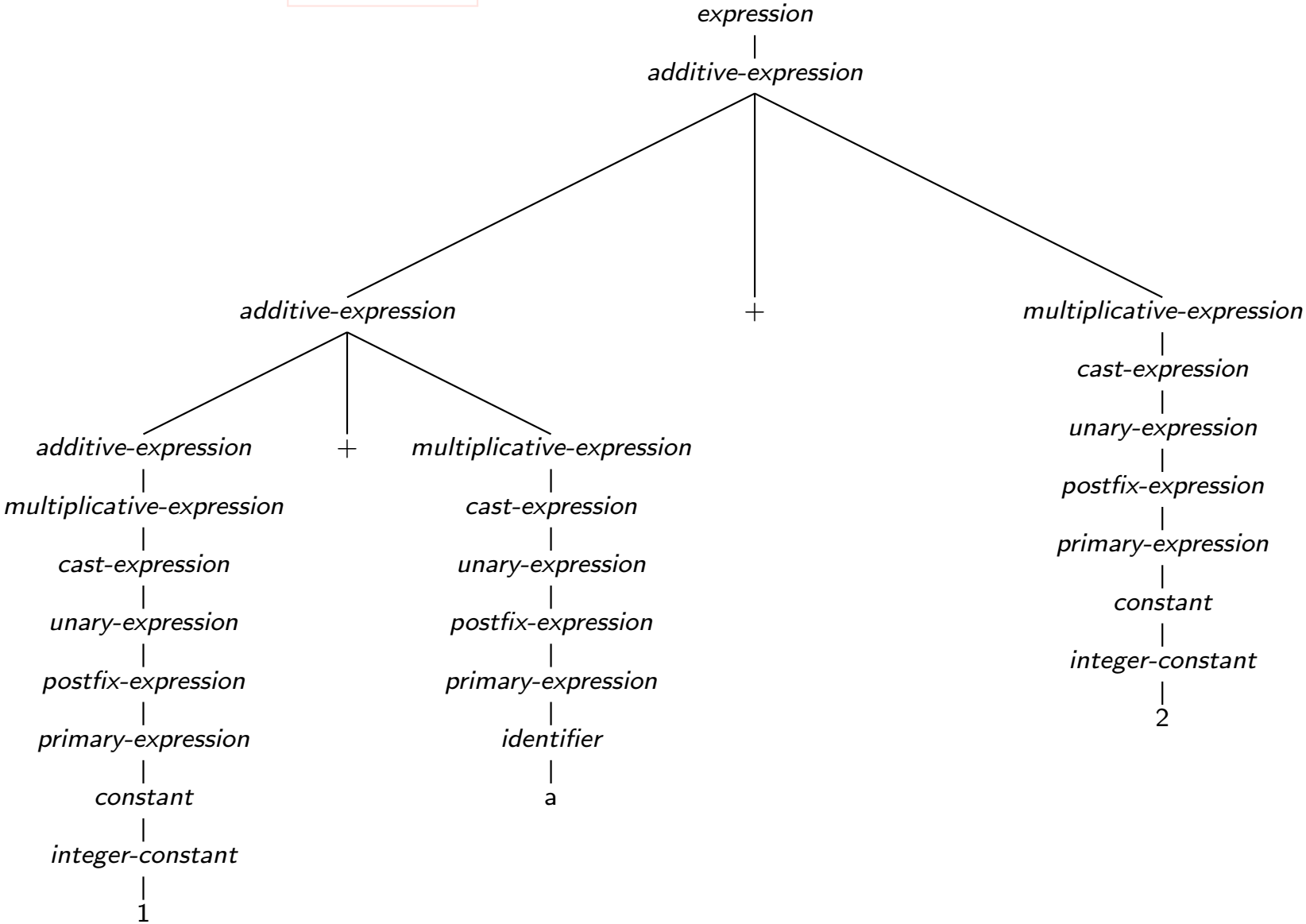
example name: `cex_expression_0.cpp`

Your code has **plenty** of expressions.

parse tree for `a+1`, ANSI-C-wise

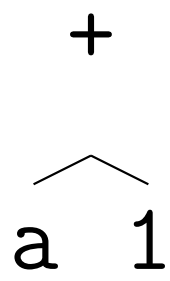


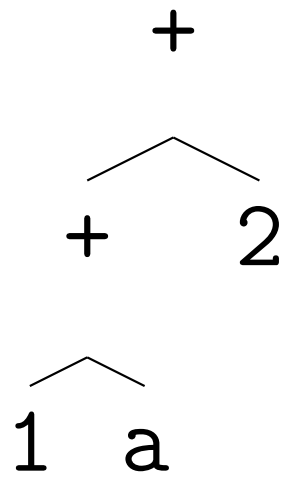
parse tree for `1+a+2`, ANSI-C-wise



Note: this is similar to `(1+a)+2`.

simplified parse trees



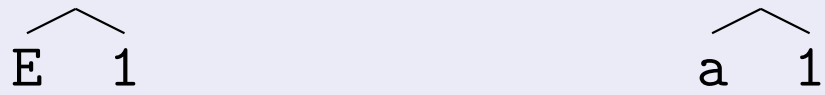



```
$ cat cex_1a2.c
int main() {
 int a;
 return 1 + a + 2;
}
$ clang -cc1 -ast-dump cex_1a2.c | sed 's/0x[^ ]*//g'
 ...
      `-BinaryOperator  <col:9, col:17> 'int' '+'
        |-BinaryOperator  <col:9, col:13> 'int' '+'
        | |-IntegerLiteral  <col:9> 'int' 1
        | `-ImplicitCastExpr  <col:13> 'int' <LValueToRValue>
        |   `-DeclRefExpr  <col:13> 'int' lvalue Var  'a' 'int'
        `-IntegerLiteral  <col:17> 'int' 2
```

`expression` matching

according to the parse tree...

`E+1` `+` matches `a+1` `+`




```
1 @@
2 expression E;
3 @@
4 - E + 1;
```

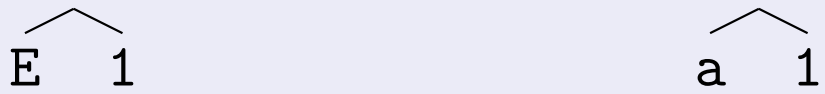
```
0 @@ -1,4 +1,3 @@
1  int main() {
2    int a;
3 - a + 1;
4  }
```

example name: `cex_expression_2.cpp`

`expression` matching

according to the parse tree...

`E+1` + matches `a+1` +




```
1 @@
2 expression E;
3 @@
4 - E + 1;
```

```
0 @@ -1,4 +1,3 @@
1  int main() {
2    int a;
3 - a + 1;
4  }
```

example name: `cex_expression_2.cpp`

`+` is non-commutative:

shall `E+1` + also match `1+a` + ?

`expression` matching and Coccinelle's *isomorphisms*

isomorphisms can make `+` commutative

rewrite rules applied to the semantic patch allowing other commonly useful variants to match

```
@@
expression E;
@@
- E + 1;
```

```
@@ -1,4 +1,3 @@
 int main() {
   int a;
- 1 + a;
 }
```

example name: `cex_expression_1.cpp`

`expression` matching

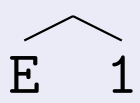
```
1 @@
2 expression E;
3 @@
4 - E + 1;
```

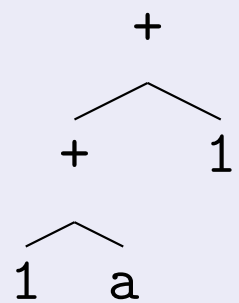
```
0 @@ -1,4 +1,3 @@
1  int main() {
2    int a;
3 - 1 + a + 1;
4  }
```

example name: `cex_expression_3.cpp`

`+` is left-associative:

`E+1` `+` `(1+a)+1` matches





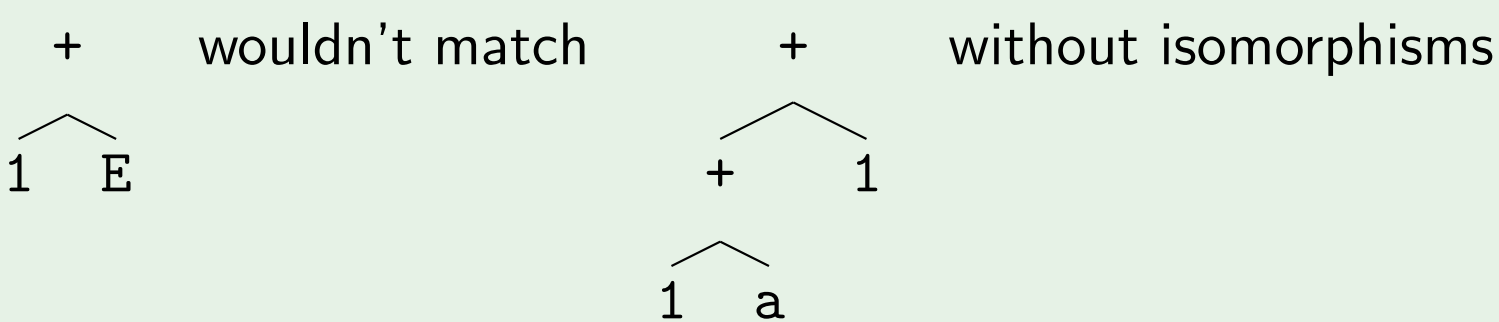
`expression` and associativity

```
1 @@
2 expression E;
3 @@
4 - 1 + E;
```

```
0 @@ -1,4 +1,3 @@
1  int main() {
2    int a;
3 - 1 + a + 1;
4  }
```

example name: `cex_expression_4.cpp`

`+` is left-associative:



Turn off isomorphisms with `--iso-limit 0`.

`expression` and associativity

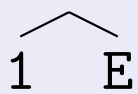
```
@@
expression E;
@@
- 1 + E;
```

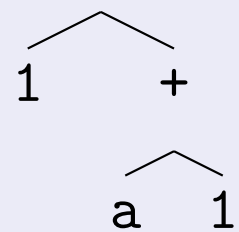
```
@@ -1,4 +1,3 @@
 int main() {
-  int a;
- 1 +(a + 1);
 }
```

example name: `cex_expression_5.cpp`

explicit parenthesisation

`1+E` `+` matches `1+(a+1)` `+`





expression: if/for constructs

```
1 @@
2 expression E;
3 @@
4 - E;
```

```
0 @@ -1,6 +1,6 @@
1  int main() {
2   if(1)
3 -    sqrt(1.0);
4 +    {}
5   for(i=0;i<10;++i)
6 -    sqrt(1.0);
7 +    {}
8  }
```

example name: `cex_expression_if.cpp`

'failsafe' mechanism when deleting 'only' statement

expression: if/for constructs

```
@@
expression E;
@@
 sqrt(  E);
+sqrt(2*E);
```

```
@@ -1,6 +1,8 @@
 int main() {
   int i;
-  if(1)
+  if(1) {
     sqrt(1.0);
+    sqrt(2 * 1.0);
+  }
   // and similarly for 'for'
 }
```

example name: `cex_expression_for.cpp`

'failsafe' mechanism when adding statement after 'only' one

expression

```
@@
expression E;
@@
-E + E
+E * 2
```

```
@@ -1,7 +1,7 @@
 int main()
 {
-   int i = 3 + 3;
-   int j = i + i;
+   int i = 3 * 2;
+   int j = i * 2;
    int k =  i + k  +  i + k ;
-   int k = (i + k) + (i + k);
+   int k = (i + k) * 2;
 }
```

example name: cex_expression.c

expression list

```
1 @@
2 identifier F;
3 expression list EL;
4 @@
5 -F(EL);
```

```
0 @@ -1,8 +1,6 @@
1  int f(int i){}
2  int g(int i, int j){}
3  int main()
4  {
5     double a,b;
6 -   f(a);
7 -   g(a,b);
8  }
```

example name: `cex_expression_list.c`

expression list [n]

```
1 @@
2 identifier F;
3 expression list [n={1...2}]EL;
4 @@
5 -F(EL);
```

```
0 @@ -1,9 +1,7 @@
1  int f(int i){}
2  int g(int i, int j){}
3  int h(int i, int j, int k){}
4  int main() {
5     double a,b,c;
6 -   f(a);
7 -   g(a,b);
8     h(a,b,c);
9  }
```

example name: `cex_expression_listnr.c`

expression list [n]

```
@@
identifier F;
expression list [n={1,3}]
    EL;
@@
-F(EL);
```

```
@@ -1,9 +1,7 @@
 int f(int i){}
 int g(int i, int j){}
 int h(int i, int j, int k)
     {}
 int main() {
   double a,b,c;
-  f(a);
   g(a,b);
-  h(a,b,c);
 }
```

example name: `cex_expression_listn.c`

`field`

- match for *fields* in `structs`
- allow
  - restructure existing `structs`,
  - create ad-hoc ones

```
1  @@
2  field lfld;
3  field list [n={2}] f2fld;
4  @@
5  struct str_t {
6  - f2fld
7    ...
8  - lfld
9  };
10 + struct l_t { f2fld lfld };
```

e.g. match and move selected

- `field` s, or
- `field list` s

field

```
@@
field fld1,fld2;
identifier I;
@@
struct I {
  fld1
  fld2
};
+//above is a two-field
    struct
```

```
@@ -1,10 +1,11 @@
 struct t_t {
   int i;
   int j;
 };
+//above is a two-field
    struct
 struct o_t {
   int i;
 };
 int main()
 {
 }
```

example name: cex_field.c

field list

```
@@
field list [n={2}] fldl;
identifier I;
@@
struct I {
  fldl
};
+//above is a two-field
    struct
```

```
@@ -1,10 +1,11 @@
 struct t_t {
   int i;
   int j;
 };
+//above is a two-field
    struct
 struct o_t {
   int i;
 };
 int main()
 {
 }
```

example name: cex_field_list.c

disjunction

- **matches** on first matching branch
- matching metavariables are usable in `-` and `+` code
- `-` and `+` code has to be *per-branch*

```
1 @@
2 @@
3 (
4 - a++;
5 |
6 - b++;
7 )
```

- long and short forms
- long: clearer
- short: need *escaping* but can be useful

```
1 @@
2 @@
3 -\( a++ \| b++ \);
```

disjunctions

```
@@
type T;
symbol a,b;
@@
(
- T a;
|
- T b;
)
```

```
@@ -1,6 +1,4 @@
 int main()
 {
-   int a;
-   int b;
    int c;
 }
```

example name: `cex_disjunction.c`

`disjunctions`: beware!

```
@@
type T;
symbol a,b;
@@
// disjunctions match
// from top to bottom
// once per declaration
(
- T a; // matches first
|
- T b; // matches second
)
```

```
@@ -1,7 +1,5 @@
 int main()
 {
-   int a,b,c;
-   int a;
-   int b;
+   int b,c;
    int c;
 }
```

example name: `cex_disjunction2.c`

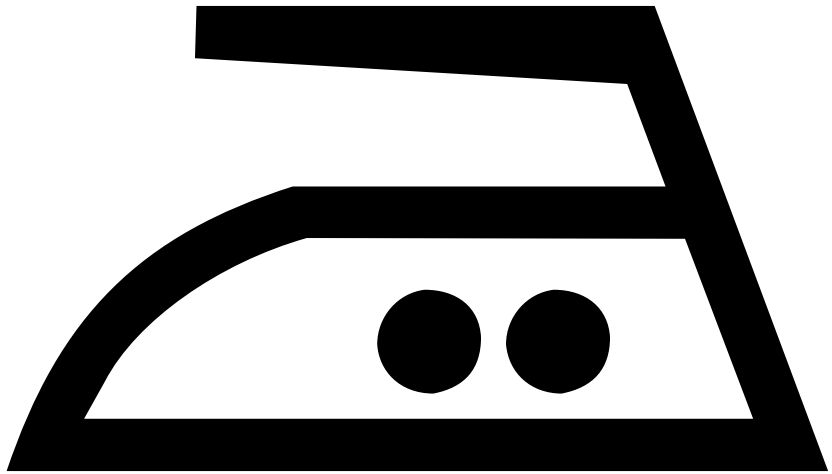
disjunctions

```
@@
type T1, T2 = {double};
symbol a,b;
@@
(
// only if this fails...
- T1 a; // removes all a's
|
// ... will try this one:
- T2 b; // removes 'double b;'
)
```

```
@@ -1,8 +1,5 @@
 int main() {
-   int a,b,c;
-   int a;
+   int b,c;
    int b;
-   double a,b,c;
-   double a;
-   double b;
+   double b,c;
 }
```

example name: `cex_disjunction3.c`

Quiz time!



disjunction : quiz

QUESTION: Which branch will match?

```
@@
expression E;
identifier I;
@@
(
- E;
|
- I++;
)
  I = 0;
+ E;
```

```
int main()
{
  int a;
  a++;
  a = 0;
}
```

example name: `cex_quiz_disjunction1.c`

ANSWER:

disjunction : quiz

QUESTION: Which branch will match?

```
@@
expression E;
identifier I;
@@
(
- E;
|
- I++;
)
  I = 0;
+ E;
```

```
@@ -1,6 +1,6 @@
 int main()
 {
   int a;
-  a++;
   a = 0;
+  a++;
 }
```

```
int main()
{
  int a;
  a++;
  a = 0;
}
```

```
int main()
{
  int a;
  a = 0;
  a++;
}
```

example name: `cex_quiz_disjunction1.c`

ANSWER: First branch matches, so everything fine. But ... **sure** ?

disjunction: quiz

QUESTION: What will this delete?

```
@@
expression E;
@@
(
- E + 1;
|
- 1 - E;
)
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

example name: `cex_quiz_disjunction2.c`

ANSWER:

disjunction: quiz

QUESTION: What will this delete?

```
@@
expression E;
@@
(
- E + 1;
|
- 1 - E;
)
```

```
@@ -1,7 +1,4 @@
 int main()
 {
    int a;
-   a + 1;
-   1 - a;
-   1 - a + 1;
 }
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

```
int main()
{
  int a;
}
```

example name: `cex_quiz_disjunction2.c`

ANSWER: Each of these expression-statements matches.

disjunction: quiz

QUESTION: What will this change?

```
@@
expression E;
@@
(
  1 - E;
+ E; // right of 1
|
  E + 1;
+ E; // left of 1
)
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

example name: cex_quiz_disjunction3b.c

disjunction: quiz

QUESTION: What will this change?

```
@@
expression E;
@@
(
  1 - E;
+ E; // right of 1
|
  E + 1;
+ E; // left of 1
)
```

```
@@ -1,7 +1,10 @@
 int main()
 {
   int a;
   a + 1;
+  a; // left of 1
   1 - a;
+  a; // right of 1
   1 - a + 1;
+  1 - a; // left of 1
 }
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

```
int main()
{
  int a;
  a + 1;
  a; // left of 1
  1 - a;
  a; // right of 1
  1 - a + 1;
  1 - a; // left of 1
}
```

example name: cex_quiz_disjunction3b.c

disjunction: quiz

QUESTION: What will this change?

```
@@
expression E;
@@
(
  E + 1;
+ E; // left of 1
|
  1 - E;
+ E; // right of 1
)
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

example name: `cex_quiz_disjunction3c.c`

disjunction: quiz

QUESTION: What will this change?

```
@@
expression E;
@@
(
  E + 1;
+ E; // left of 1
|
  1 - E;
+ E; // right of 1
)
```

```
@@ -1,7 +1,10 @@
 int main()
 {
   int a;
   a + 1;
+  a; // left of 1
   1 - a;
+  a; // right of 1
   1 - a + 1;
+  1 - a; // left of 1
 }
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

```
int main()
{
  int a;
  a + 1;
  a; // left of 1
  1 - a;
  a; // right of 1
  1 - a + 1;
  1 - a; // left of 1
}
```

example name: cex_quiz_disjunction3c.c

disjunction: quiz

QUESTION: What will this change?

```
@@
expression A,B;
@@
(
- (A + B);
+((A + B));
|
- (A + A);
+ (2 * A);
)
```

```
int main()
{
  int a,b;
  (a + b);
  (a + a);
}
```

example name: `cex_quiz_disjunction4.c`

ANSWER:

disjunction: quiz

QUESTION: What will this change?

```
1  @@
2  expression A,B;
3  @@
4  (
5  - (A + B);
6  +((A + B));
7  |
8  - (A + A);
9  + (2 * A);
10 )
```

```
0  @@ -1,6 +1,6 @@
1   int main()
2   {
3      int a,b;
4  -   (a + b);
5  -   (a + a);
6  +   ((a + b));
7  +   ((a + a));
8   }
```

```
1  int main()
2  {
3    int a,b;
4    (a + b);
5    (a + a);
6  }
```

```
1  int main()
2  {
3    int a,b;
4    ((a + b));
5    ((a + a));
6  }
```

example name: `cex_quiz_disjunction4.c`

ANSWER: First pattern matches both statements.

disjunction: quiz

QUESTION: What will this change?

```
@@
expression A,B;
@@
(
- (A + A);
+ (2 * A);
|
- (A + B);
+((A + B));
)
```

```
int main()
{
  int a,b;
  (a + b);
  (a + a);
}
```

example name: `cex_quiz_disjunction5.c`

ANSWER:

disjunction: quiz

QUESTION: What will this change?

```
1  @@
2  expression A,B;
3  @@
4  (
5  - (A + A);
6  + (2 * A);
7  |
8  - (A + B);
9  +((A + B));
10 )
```

```
0  @@ -1,6 +1,6 @@
1   int main()
2   {
3      int a,b;
4  -   (a + b);
5  -   (a + a);
6  +   ((a + b));
7  +   (2 * a);
8   }
```

```
1  int main()
2  {
3    int a,b;
4    (a + b);
5    (a + a);
6  }
```

```
1  int main()
2  {
3    int a,b;
4    ((a + b));
5    (2 * a);
6  }
```

example name: cex_quiz_disjunction5.c

ANSWER: First first branch, then second.

conjunction

- **match** parsed entity conforming to many forms
- optionally **delete** or **substitute** via *plus code*
- allows *less verbose* substitutions

```
1  @@
2  expression E;
3  identifier I;
4  @@
5  (
6  - E;
7  &
8     I++;
9  )
10    I = 0;
11 + E;
```

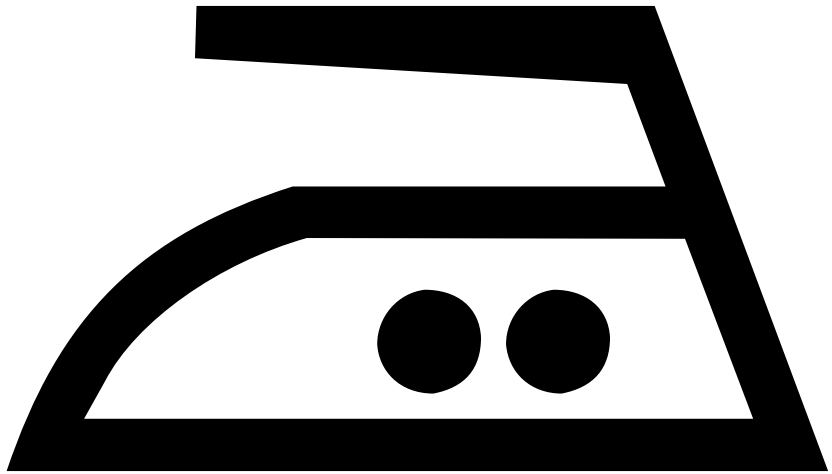
conjunction

```
@@
type T;
identifier I =~ "a";
@@
(
- T I;
&
- T b;
)
```

```
@@ -1,7 +1,7 @@
 int main()
 {
-   int a, b, c;
+   int c;
    int a;
    int b;
    int c;
 }
```

example name: cex_conjunction2.c

Quiz time!



conjunction: quiz

QUESTION: What will this change?

```
@@
expression E;
identifier I;
@@
(
- E;
&
- I++;
)
  I = 0;
+ E;
```

```
int main()
{
  int a;
  a++;
  a = 0;
}
```

example name: `cex_quiz_conjunction1.c`

ANSWER:

conjunction: quiz

QUESTION: What will this change?

```
@@
expression E;
identifier I;
@@
(
- E;
&
- I++;
)
  I = 0;
+ E;
```

```
@@ -1,6 +1,6 @@
 int main()
 {
   int a;
-  a++;
   a = 0;
+  a++;
 }
```

```
int main()
{
  int a;
  a++;
  a = 0;
}
```

```
int main()
{
  int a;
  a = 0;
  a++;
}
```

example name: `cex_quiz_conjunction1.c`

ANSWER: Will move increment after assignment.

conjunction: quiz

QUESTION: What will this delete?

```
@@
expression P,M;
@@

(
- P + 1;
&
- 1 - M;
)
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

example name: cex_quiz_conjunction3.c

ANSWER:

conjunction: quiz

QUESTION: What will this delete?

```
@@
expression P,M;
@@

(
- P + 1;
&
- 1 - M;
)
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

```
int main()
{
  int a;
  a + 1;
  1 - a;
  1 - a + 1;
}
```

example name: `cex_quiz_conjunction3.c`

ANSWER: `1-a+1` associates as `(1-a)+1`, so differently than `1-(a+1)`

conjunction: quiz

QUESTION: What will this delete?

```
@@
expression A,B;
@@

(
- (A + A);
&
- (B + B);
)
```

```
int main()
{
  int a, b;
  (a + a);
  (b + b);
}
```

example name: cex_quiz_conjunction4.c

ANSWER:

conjunction : quiz

QUESTION: What will this delete?

```
@@
expression A,B;
@@

(
- (A + A);
&
- (B + B);
)
```

```
@@ -1,6 +1,4 @@
 int main()
 {
    int a, b;
-   (a + a);
-   (b + b);
 }
```

```
int main()
{
  int a, b;
  (a + a);
  (b + b);
}
```

```
int main()
{
  int a, b;
}
```

example name: cex_quiz_conjunction4.c

ANSWER: Both patterns match.

conjunction : quiz

QUESTION: What will this delete?

```
@@
expression A,B;
@@

(
- (A + B);
&
- (B + A);
)
```

```
int main()
{
  int a, b;
  (a + a);
  (b + b);
}
```

example name: cex_quiz_conjunction5.c

ANSWER:

conjunction : quiz

QUESTION: What will this delete?

```
@@
expression A,B;
@@

(
- (A + B);
&
- (B + A);
)
```

```
@@ -1,6 +1,4 @@
 int main()
 {
    int a, b;
-   (a + a);
-   (b + b);
 }
```

```
int main()
{
  int a, b;
  (a + a);
  (b + b);
}
```

```
int main()
{
  int a, b;
}
```

example name: `cex_quiz_conjunction5.c`

ANSWER: Both patterns match, again.

conjunction: quiz

QUESTION: What will this delete?

```
@@
expression A,B;
@@

(
- (A + B);
&
- (B + A);
)
```

```
int main()
{
  int a, b;
  (a + b);
  (b + a);
}
```

example name: cex_quiz_conjunction6.c

ANSWER:

`conjunction`: quiz

QUESTION: What will this delete?

```
@@
expression A,B;
@@

(
- (A + B);
&
- (B + A);
)
```

```
@@ -1,6 +1,4 @@
 int main()
 {
    int a, b;
-   (a + b);
-   (b + a);
 }
```

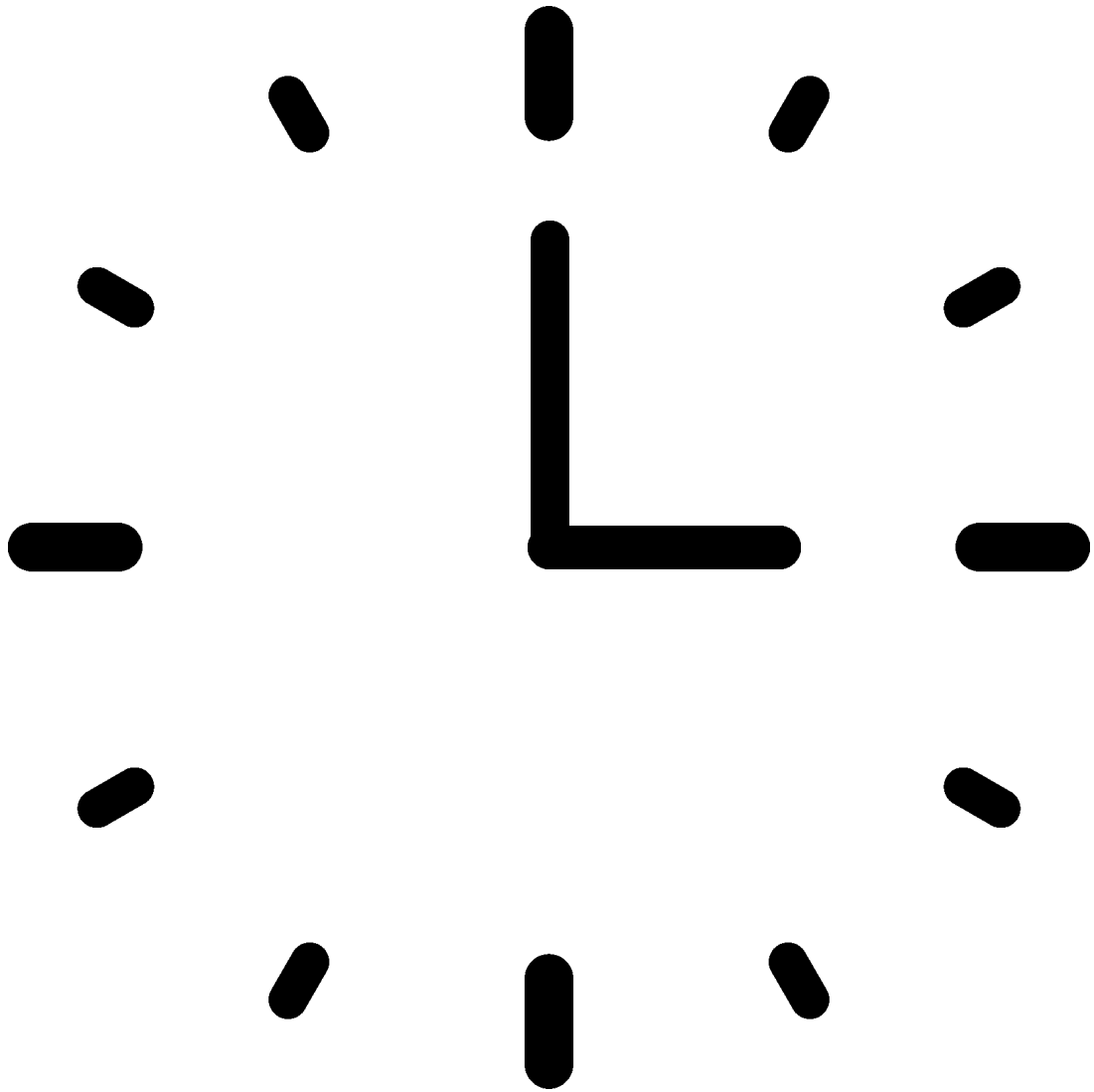
```
int main()
{
  int a, b;
  (a + b);
  (b + a);
}
```

```
int main()
{
  int a, b;
}
```

example name: `cex_quiz_conjunction6.c`

ANSWER: It matches: isomorphisms turn `+` commutative (recall p. 113).

Break time! till 15:00



Refresher on *ellipses*: `...`

Match an arbitrary term or sequence thereof, e.g.:

- expressions: `return ...;`
- blocks: `if { ... };`
- function call arguments: `f( ... );`
- function parameters:
  `int f( ... ){}`
  `int f( arg,... ){}`
  `int f( ...,arg ){}`
- field lists: `struct s{ ... };`
- array indices: `a[ ... ];`

recall:

- basics at p. 67
- *control flow* later at p. 155

Note:

dot variants: `<...  ...>` for control flow only; `<+...  ...+>` for expressions, too.

ellipses (`...`) for arguments match

```
@@
type T;
identifier F;
@@
+/* match: */
 T F (...) {
    ...
 }
```

```
@@ -1,8 +1,12 @@
+/* match: */
 void g(int i, int j) { }
+/* match: */
 void f(int i) { }
+/* match: */
 void e() { }
+/* match: */
 int main() {
    g(1,2);
    f(1);
    e();
 }
```

example name: `cex_ellipses_argument.c`

ellipses (`...`) for `struct` fields match

```
@@
identifier I;
@@
 struct I {
-  ...
 };
```

```
@@ -1,13 +1,10 @@
 struct s0 { };
 struct s1 {
-  int i;
 };
 struct s2 {
-  int i,j;
 };
 struct s3 {
-  int i; int j;
 };
 int main() {
   return 0;
 }
```

example name: `cex_ellipses_fields1.c`

ellipses (`...`) for `struct` fields match

```
@@
identifier I;
@@
 struct I {
 int i;
- ...
 };
```

```
@@ -1,10 +1,9 @@
 struct s0 { };
 struct s1 { int i; };
 struct s2 { int i,j; };
 struct s3 {
   int i;
-  int j;
 };
 int main() {
   return 0;
 }
```

example name: `cex_ellipses_fields2.c`

ellipses (`...`) for `struct` fields match

Dots in a structure's name won't parse.

```
@@

@@
// deliberately broken rule
 struct ... { // BAD
- ...          // GOOD
 };
```

```
struct s0 { };
struct s1 { int i; };
struct s2 { int i,j; };
struct s3 { int i;int j; };
int main() { }
```

```
struct s0 { };
struct s1 { int i; };
struct s2 { int i,j; };
struct s3 { int i;int j; };
int main() { }
```

example name: `cex_ellipses_fields4.c`

ellipses (`...`) for `struct` fields match

```
@@
identifier I;
@@
 struct I {
   ...
- int i;
   ...
   int j;
 };
```

```
@@ -1,8 +1,7 @@
 struct s0 { };
 struct s1 { int i; };
 struct s2 { int i,j; };
 struct s3 {
-   int i;
    int j;
 };
 int main() { return 0; }
```

example name: `cex_ellipses_fields3.c`

match between `if` branch...

```
@@
@@

   i = 0;

- i--;
```

```
int i;
void f3() {
  if(1) i = 1; else i = 1;
  i--;
}
void f2() {
  if(1) i = 0; else i = 1;
  i--;
}
void f1() {
  if(1) i = 0; else i = 0;
  i--;
}
int main(){f1();f2();f3();}
```

example name: `cex_cf_0.c`

...and out of it?

ellipses (`...`) matches with `forall` semantics

```
@@
@@
   ...
   i = 0;
   ...
- i--;
```

```
@@ -1,14 +1,13 @@
 int i;
 void f3() {
   if(1) i = 1; else i = 1;
   i--;
 }
 void f2() {
   if(1) i = 0; else i = 1;
   i--;
 }
 void f1() {
   if(1) i = 0; else i = 0;
-  i--;
 }
 int main(){f1();f2();f3();}
```

example name: `cex_cf_1.c`

transform if *all control-flow paths* match

ellipses (`...`) defaults overridden

```
@ forall@
@@
   ...
   i = 0;
   ...
- i--;
```

```
@@ -1,14 +1,13 @@
 int i;
 void f3() {
   if(1) i = 1; else i = 1;
   i--;
 }
 void f2() {
   if(1) i = 0; else i = 1;
   i--;
 }
 void f1() {
   if(1) i = 0; else i = 0;
-  i--;
 }
 int main(){f1();f2();f3();}
```

example name: `cex_cf_2.c`

explicit request of `forall` semantics

Matching control flow and `exists` semantics

Does a path through a matching branch **exist** ?

```
@ exists@
@@

 ...
 f(a);
 ...

+ // possibly after f(a)
 return a;
```

```
#include <stdlib.h>
int g(int i) { return i-1; }
int f(int i) { return i+1; }
int main() {
  int a = 1;

  if(rand(0)%2==0) // if even
    g(a); // sometimes g(a)
  else
    f(a); // sometimes f(a)

  return a;
}
```

Dot variants have `exists` semantics by default

- `<...  ...>` for *optional match*
- `<+...  ...+>` for *required match*
- `...` (p. 67) can be either `exists` or `forall`

ellipses (`...`) defaults overridden

```
@ exists@
@@
   ...
   i = 0;
   ...
- i--;
```

```
@@ -1,14 +1,12 @@
 int i;
 void f3() {
   if(1) i = 1; else i = 1;
   i--;
 }
 void f2() {
   if(1) i = 0; else i = 1;
-  i--;
 }
 void f1() {
   if(1) i = 0; else i = 0;
-  i--;
 }
 int main(){f1();f2();f3();}
```

example name: `cex_cf_3.c`

`exists` semantics: any *existing* matching path suffices

`<... ...>`: optional match

```
@@
@@
 <...
  i = 1;
  ...>
- i--;
```

```
@@ -1,14 +1,11 @@
 int i;
 void f3() {
   if(1) i = 1; else i = 1;
-  i--;
 }
 void f2() {
   if(1) i = 0; else i = 1;
-  i--;
 }
 void f1() {
   if(1) i = 0; else i = 0;
-  i--;
 }
 int main(){f1();f2();f3();}
```

example name: `cex_cf_4.c`

Any path suffices.
Implied `exists` semantics.

<+... ...+>: required match

```
@@
@@
<+...
   i = 1;
   ...+>
- i--;
```

```
@@ -1,14 +1,12 @@
 int i;
 void f3() {
    if(1) i = 1; else i = 1;
-   i--;
 }
 void f2() {
    if(1) i = 0; else i = 1;
-   i--;
 }
 void f1() {
    if(1) i = 0; else i = 0;
    i--;
 }
 int main(){f1();f2();f3();}
```

example name: `cex_cf_5.c`

At least one branch is required to match.
Still, implied `exists` semantics.

`printf` & co. `format` strings

a major source of bugs are:

- wrong C string formats
- wrong arguments associated to *vararg* functions with string specifier
- `format` metavariable ease matching `printf`-ish functions *expandos*

format match

```
1 @@
2 format F =~ "d";
3 expression list EL;
4 @@
5 -printf("%@F@\n",EL);
6 +fprintf(stdout,"%@F@\n",EL);
```

```
0 @@ -1,6 +1,6 @@
1  int main()
2  {
3 -   printf("%d\n",1   );
4 +   fprintf(stdout, "%d\n", 1);
5     printf("%f\n",1.0);
6     printf("%c\n",'c');
7  }
```

example name: `cex_format.c`

parameter

- **match** declarations of function parameter
- useful in identifying/cloning functions
- create `PL@EL` from `expression list EL` and `parameter list PL`

```
1 @@
2 type T;
3 identifier POW =~ "pow";
4 parameter X,Y;
5 @@
6
7  T POW(X,Y) {
8    ...
9  }
```

- single `parameter`
- `parameter list`
- `parameter list [2]`
- `parameter list [n]`
- `parameter list [n]={1..2}`

`parameter` of a function

```
@@
identifier F;
parameter p;
@@
F(p)
{
+/* 1-par func */
  ...
}
```

```
@@ -1,13 +1,14 @@
 int a() {
   return 0;
 }

 int b(int i) {
+  /* 1-par func */
   return 0;
 }

 int c(int i,int j) {
   return 0;
 }

 int main() { }
```

example name: `cex_parameter.c`

`parameter` of a function

```
@@
type T;
identifier POW =~ "pow";
parameter X,Y;
@@
+/*
+    looks like a power
+    function definition:
+    */
 T POW(X,Y) {
   ...
 }
```

```
@@ -1,9 +1,13 @@
 double pow(double x,
            double y);

+/*
+    looks like a power
+    function definition:
+    */
 int ipow(int x, int y)
 {
     return pow(x,y);
 }

 int main() { }
```

example name: `cex_parameter1.cpp`

parameter list modify

```
@@
identifier F;
parameter list[2] PL;
@@
F(
- PL
+ double i, double j
 )
{
+ /* 2-args func */
  ...
}
```

```
@@ -1,13 +1,14 @@
 int a() {
   return 0;
 }
 int b(int i) {
   return 0;
 }
-int c(int i,int j) {
+int c(double i, double j) {
+  /* 2-args func */
   return 0;
 }
 int d(int i,int j,int k) {
   return 0;
 }
 int main() { }
```

example name: cex_parameter_list_n.c

parameter list

```
@@
identifier F;
parameter list[n={1...2}] PL;
@@
 F(PL)
 {
+ /* 1 to 2 arg func */
   ...
 }
```

```
@@ -1,13 +1,15 @@
 int a() {
   return 0;
 }
 int b(int i) {
+  /* 1 to 2 arg func */
   return 0;
 }
 int c(int i,int j) {
+  /* 1 to 2 arg func */
   return 0;
 }
 int d(int i,int j,int k
     ) {
   return 0;
 }
 int main() { }
```

example name: cex_parameter_list.c

parameter list in context

```
@@
identifier F;
parameter list[n={1...2}] PL;
@@
 F(int i, PL)
 {
+ /* arg i and ...*/
   ...
 }
```

```
@@ -1,12 +1,14 @@
 int a() {
 }
 int b(int i) {
   return 0;
 }
 int c(int i,int j) {
+  /* arg i and ...*/
   return 0;
 }
 int d(int i,int j,int k
     ) {
+  /* arg i and ...*/
   return 0;
 }
 int main() { }
```

example name: cex_parameter_list2.c

expression list from parameter list

```
@@
identifier F != main;
parameter list[n] PL;
fresh identifier
    FF = F##"_new";
expression list EL;
@@
+ void FF(PL){}
 F(PL@EL)
 {
+ FF(EL);
  ...
 }
```

```
@@ -1,5 +1,15 @@
+void a_new()
+{
+}
+
 int a() {
+        a_new();
+}
+void c_new(int i, int j)
+{
 }
+
 int c(int i,int j) {
+        c_new(i, j);
 }
 int main() { }
```

example name: `cex_parameter_list3.c`

inheritance:

- a rule can use another, already matched rule's *bound* metavariables
- implicit dependency (recall `depends` on p. 59)

```
1  @r1@
2  identifier I;
3  @@
4   I = 0;
5
6  @r2@
7  identifier r1.I;
8  @@
9  - I--;
10
11 @r3@
12 identifier r1.I;
13 @@
14     a = b + c;
15 ++ I++;
```

- inherited `identifier`s can be blacklisted with `!=`
- depending rules may trigger for different inherited instances...
- ...such multiple insertions require `++`

inheritance across rules

```
1  @first_rule@
2  identifier I;
3  @@
4   I = 0;
5
6  @@
7  identifier first_rule.I;
8
9  @@
10 - I--;
```

```
0  @@ -1,10 +1,6 @@
1   int main() {
2      int a,b,c;
3
4      a = 0;
5  -   a--;
6      b = 0;
7  -   b--;
8  -   a--;
9  -   b--;
10  }
```

example name: `cex_inheritance1.c`

Reuse `identifier`s across rules!

inheritance across rules: `!=` constraint

```
@first_rule@
identifier I;
@@
 I = 0;

@@
identifier J != I;
identifier first_rule.I;
@@
- I--;
  J--;
```

```
@@ -1,10 +1,8 @@
 int main() {
    int a,b,c;

    a = 0;
    a--;
    b = 0;
    b--;
-   a--;
-   b--;
 }
```

example name: `cex_inheritance2.c`

Avoid previously matched `identifier`s! (recall p. 83?)

Multiple insertion and `++`

```
@first_rule@
identifier I;
@@
  I = 0;
  ...
- I--;

@@
identifier first_rule.I;
@@
++ I++;
   b=0;
```

```
@@ -1,10 +1,10 @@
 int main() {
    int a,b,c;

    a = 0;
-   a--;
+   b++;
+   a++;
    b = 0;
-   b--;
    a--;
    b--;
 }
```

example name: `cex_inheritance3.c`

Repeated insertion needs explicit `++` !

declaration

- match and manipulate variables declaration
- `@` shortcut syntax

```
1  @g@
2  declaration d;
3  identifier I = {b,d};
4  @@
5  -int b@d;
6
7  @@
8  declaration g.d;
9  @@
10  int a;
11 +d
```

declaration

```
@g@
declaration d;
@@
-int b@d;

@@
declaration g.d;
@@
 int a;
+d
```

```
@@ -1,4 +1,5 @@
-int b;
+
 int main() {
     int a;
+    int b;
 }
```

example name: cex_declaration1.c

declaration quiz

QUESTION: what will be changed?

```
@@
declaration D1,D2;
@@
+// before decls
 D1
 D2
+// after decls
```

```
int main() {
 int i;
 int j;
 int k;
}
```

example name: `cex_cmt_in_between_decl.c`

ANSWER:

declaration quiz

QUESTION: what will be changed?

```
@@
declaration D1,D2;
@@
+// before decls
 D1
 D2
+// after decls
```

```
@@ -1,5 +1,9 @@
 int main() {
+ // before decls
   int i;
+ // before decls
   int j;
+ // after decls
   int k;
+ // after decls
 }
```

```
int main() {
 int i;
 int j;
 int k;
}
```

```
int main() {
 // before decls
 int i;
 // before decls
 int j;
 // after decls
 int k;
 // after decls
}
```

example name: `cex_cmt_in_between_decl.c`

ANSWER: `int i;int j;` as `D1` and `int j;int k;` as `D2`

declarations can be tricky 1

```
@ra@
identifier A = a;
@@
- int A;
+ int d;

@rb@
identifier B = {b,z};
@@
- int B;
+ int e;
```

```
@@ -1,4 +1,4 @@
 int main() {
-   int a;
+   int d;
    int b,c;
 }
```

example name: `cex_declaration_tricky1.c`

b ignored

declarations can be tricky 2

```
@@
identifier A = a;
@@
 int
- A
+ d
 ;

@@
identifier B = {b,z};
@@
 int
- B
+ e
 ;
```

```
@@ -1,4 +1,4 @@
 int main() {
-   int a;
-   int b,c;
+   int d;
+   int e,c;
 }
```

example name: `cex_declaration_tricky1b.c`

b transformed, too

`position`

- *inherited* `position`s (see p. 168) simplify context *rematching*
- the primary use of `position` is for scripting
- with `script`ing, recover *file, line numbers* information

position

```
1  @F@
2  position p;
3  identifier F;
4  statement S;
5  parameter list PL;
6  @@
7   F(PL)
8   {
9    S@p
10 + /* first stmt! */;
11   ...
12  }
```

```
0  @@ -1,7 +1,9 @@
1   int  f(int i) {
2      i++;
3  +   /* first stmt! */;
4      return i;
5   }
6   int main() {
7      f(0);
8  +   /* first stmt! */;
9   }
```

example name: `cex_position.c`

see `script`ing (p. 184) for full usage of `p`

`statement`: match and manipulate statements

- help matching `if` / `for` constructs
- recover statements from fragments
- usage syntax remniscent of `position` using `@` (see p. 177)

```
@r1@
expression E;
identifier I;
@@
 \( I + I \& E\)

@r2@
expression r1.E;
statement S;
@@
 E@S

@@
statement r2.S;
@@
   S
+ S
```

```
@@
type T;
identifier P;
statement S1,S2;
@@

 T *P;

 ...

 if (
-     P == NULL
+    !P
  )
      S1
 else
      S2
```

statement

```
1  @r1@
2  expression E;
3  identifier I;
4  @@
5   \( I + I \& E\)
6
7  @r2@
8  expression r1.E;
9  statement S;
10 @@
11  E@S
12
13 @@
14 statement r2.S;
15 @@
16   S
17 + S
```

```
0  @@ -1,5 +1,6 @@
1   int main() {
2      int i,k;
3      k += (k + k) + i;
4  +   k += (k + k) + i;
5      i += i;
6   }
```

example name: `cex_snip_statement.cpp`

no *preprocessing* (`cpp`) step

but...

- partial (internal) macros interpretation
- ignore `ifdef`s by default
- no match on expanded
- *almost* no matching
- but `ifdefs` stay attached around `field`s :-)

no implicit macro expansion

```
@@
@@
+ ++a;
  return 0;
```

```
#define zero 0
int main() {
  int a=0;
  return zero;
}
```

```
#define zero 0
int main() {
  int a=0;
  return zero;
}
```

example name: `cex_preprocessor1.c`

rudimentary preprocessor support: ignore `#if 0 ...`

```
@@
constant K;
@@
- return K;
+ exit(K);
```

```
@@ -1,6 +1,6 @@
 int main() {
-    return 0;
+    exit(0);
 #if 0
     return 1;
 #endif
 }
```

example name: `cex_preprocessor3.c`

`--noif0-passing` would unignore it.

Scripting

- many *internals* are accessible
- via `script: python` or `script: ocaml`

```
@r@
// metadecls
@@
// normal rule ...

@script:python p@
// variables binding
I << r.I;
N; // new variables
@@
// python code using I and N

@@
identifier r.I;
identifier p.N;
@@
// normal rule ...
```

```
@initialize:python@
@@
// python code ...

@script:python@
I << r.I;
// ...
@@
// python code using  ...

@finalize:python@
@@
// python code ...
```

manipulate `identifier`s via `script`

```
@r@
identifier I;
@@
 int I;

@script:python p@
I << r.I;
J;
@@
coccinelle.J=I.upper();

@ identifier@
identifier r.I;
identifier p.J;
@@
- I
+ J
```

```
@@ -1,5 +1,5 @@
 void main()
 {
-        int i;
-        i = 0;
+        int I;
+        I = 0;
 }
```

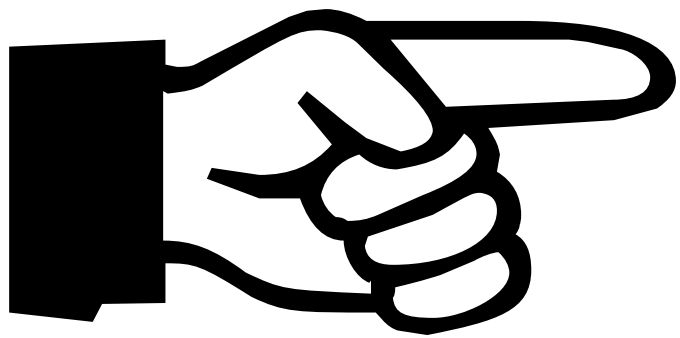
example name: `cex_script1.c`

identify and delimit code

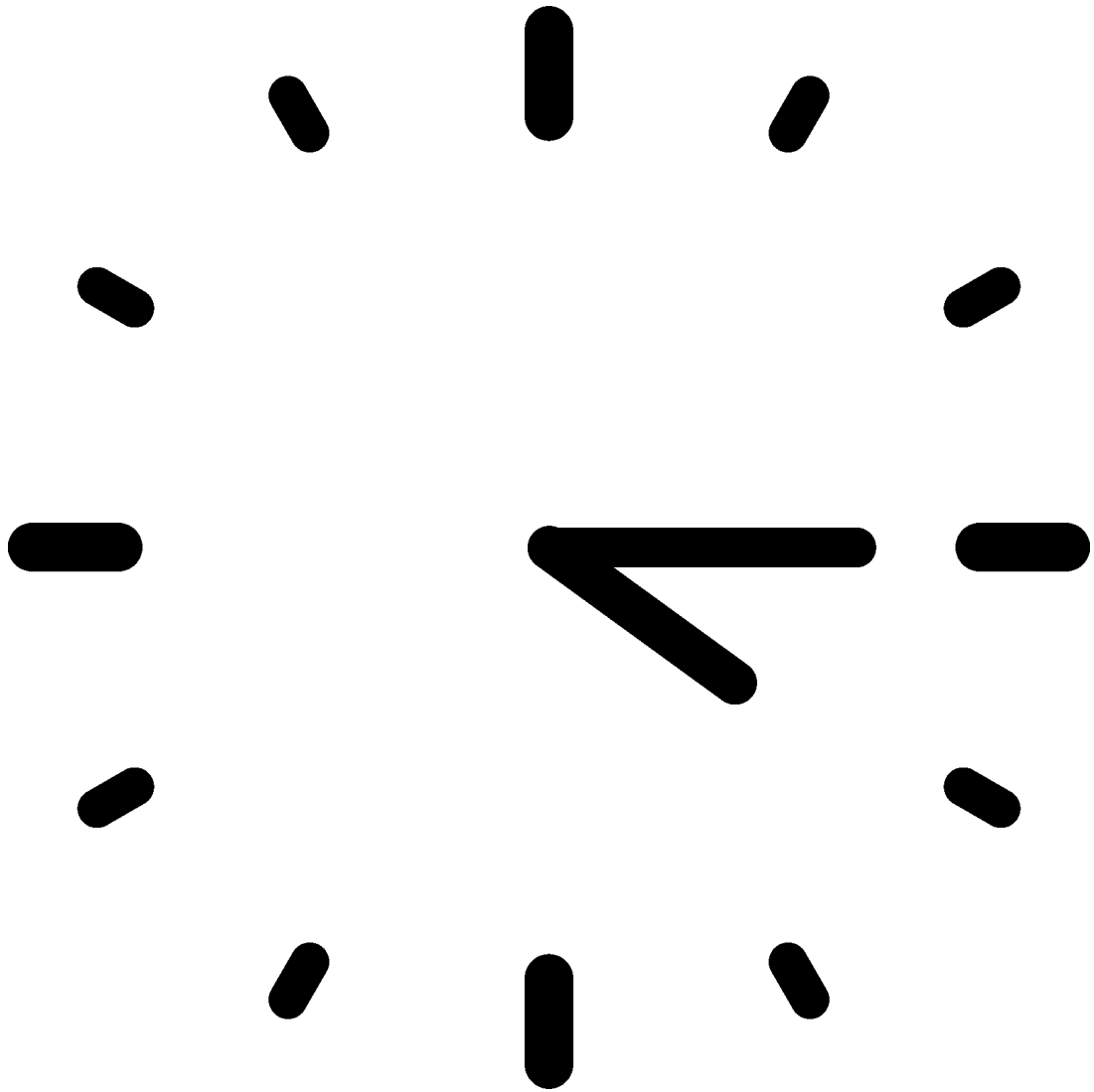
```
@r@
identifier I;
position p;
@@
* I=0;
* ...
* I++@p;
@script:python@
p << r.p;
@@
cocci.print_secs("identified context:",p)
print "fl:",     p[0].file
print "line:",   p[0].line,   "...", p[0].line_end
print "column:", p[0].column, "...", p[0].column_end
```

example name: `cex_exists1.c`

Question time!



Break time! till 16:15



Insert statement after local variable declarations

```
@@
declaration D;
statement S;
@@
 D
+printf("in %s\n",
     __FUNCTION__);
 S
```

```
@@ -1,14 +1,17 @@
 void v() {
   return;
 }

 int f(int i) { int j;
+  printf("in %s\n", __FUNCTION__);
   return i+j; }

 int f(int i) { int j,k;
+  printf("in %s\n", __FUNCTION__);
   return i+j+k; }

 int main() {
   int i; int j;
+  printf("in %s\n", __FUNCTION__);
   i=0; j=i; v( ); f(j);
 }
```

example name: cex_stmt_after_decl.c

E.g. poor man's tracing

Insert first statement in function

```
@@
identifier F;
statement S1,S2;
@@
F(...) {
 ... when != S1
+printf("in %s\n",
     __FUNCTION__);
 S2
 ... when any
}
```

```
@@ -1,14 +1,18 @@
 void v() {
+  printf("in %s\n", __FUNCTION__);
   return;
 }

 int f(int i) { int j;
+  printf("in %s\n", __FUNCTION__);
   return i+j; }

 int f(int i) { int j,k;
+  printf("in %s\n", __FUNCTION__);
   return i+j+k; }

 int main() {
   int i; int j;
+  printf("in %s\n", __FUNCTION__);
   i=0; j=i; v( ); f(j);
 }
```

example name: cex_stmt_after_decl2.c

Intercept first non-declaration

Transfer function contents

```
@r1@
statement list sl;
@@
 int main() {
- sl
+ sub_main( );
}

@r2@
statement list r1.sl;
@@
int main (...) {...}
+ void sub_main( ) { sl }
```

```
@@ -1,5 +1,11 @@
 int main() {
+
+  sub_main();
+}
+
+void sub_main()
+{
    int a = 1;
-   if(2)
+   if (2)
      a += 1;
 }
```

example name: `cex_stmt_f2f.c`

Split / clone specialized versions of function

Barebone AoS-to-SoA[5]: variables selection

```
@@
identifier M = {X,Y};
fresh identifier G="g_"##M;
type T;
@@
struct ptcl_t {
  ...
- T M;
  ...
 };
++ T G[N];
```

```
@@ -1,11 +1,13 @@
 #define N 3
 struct ptcl_t {
   int x,y,z;
-  double X,Y,Z;
+  double Z;
 };
+double g_X[N];
+double g_Y[N];


 int main() {
   struct ptcl_t aos[N];
   // ...
 }
```

example name: `cex_aos_to_soa1.c`

First step: rules to create data structure

[5] *Array of Structures* to *Structures of Arrays*

Barebone AoS-to-SoA: declarations and use

```
@r@
identifier M = {X,Y};
fresh identifier G="g_"##M;
symbol N;
type T;
@@
 struct ptcl_t {
- T M;
 };
++ T G[N];

@@
identifier r.M,P,r.G;
typedef ptcl_t;
expression E;
constant N;
@@
struct ptcl_t P[N];
  ...
- P[E].M
+ G[E]
```

```
@@ -1,11 +1,13 @@
 #define N 3
 struct ptcl_t {
-  double X,Y,Z;
+  double Z;
 };
+double g_X[N];
+double g_Y[N];


 int main() {
   struct ptcl_t aos[N];
-  aos[0].X = aos[0].Y
+  g_X[0] = g_Y[0]
             + aos[0].Z;
 }
```

example name: `cex_aos_to_soa2.c`

Second step: update expressions accordingly

Iterative method and *recovery*

```
1  @@
2  identifier X,A,Y;
3  fresh identifier Z=X##"_rec";
4  @@
5   v_t X;
6  +v_t Z; // CG recovery vector
7   m_t A;
8  ...
9   X= A* X;
10 +//post-mult CG recovery code
11 ...
12  Y= norm(X);
13 +//post-norm CG recovery code
```

```
0  @@ -1,11 +1,14 @@
1   // extract from a iterative method
2   typedef int m_t;
3   typedef int v_t;
4   int norm(v_t v) { return 0; }
5   int main() {
6    v_t v,p;
7  + v_t p_rec; // CG recovery vector
8    m_t A;
9    p= A*p;
10 + //post-mult CG recovery code
11   v= A*p;
12   v=norm(p);
13 + //post-norm CG recovery code
14  }
```

example name: `cex_cg1.c`

Instead of comments, specific functions calls here

(see e.g. Jaulmes et al., 2015)

Detect variable use and change its type

```
@vr@
identifier V;
type NT={double};
@@
 NT *V;

@br@
identifier vr.V;
identifier I,J,N,M;
identifier ins_fun=~"insert";
@@
 ins_fun(M, N, V, I, J)

@dr depends on br@
identifier vr.V;
type vr.NT;
@@
-NT    *V;
+float *V;
```

```
@@ -1,16 +1,16 @@
 #include <blas_sparse.h>
 int main() { // ...
   int nnz;
   int *IA, *JA;
   float  *FV;
   double *DV;
-  double *NV;
+  float *NV;
   // ...
   BLAS__uscr_insert_entries(A, nnz, FV,
     IA, JA);
   BLAS__usgt_entries       (A, nnz, DV,
     IA, JA);
   BLAS__uscr_insert_entries(A, nnz, NV,
     IA, JA);
   // ...
 }
```

example name: cex_var_type_change.c

Precision increase/decrease

http://www.netlib.org/blas/blast-forum/

C++ support

You can limitedly patch C++ code (mostly, the C subset..).

```
1 @@
2 @@
3 - std::cout
4 + std::cerr
```

```
0 @@ -1,4 +1,4 @@
1  #include <iostream>;
2  int main(){
3 -     std::cout << std::endl;
4 +     std::cerr << std::endl;
5  }
```

example name: `cex_cplusplus1.cpp`

reference types, `namespace`, `new`, `delete` keywords supported.

C++ support

Classes, namespaces, member fuctions are not parsed.

```
@@
@@
- return 0;
+ return 1;
```

```
@@ -1,9 +1,9 @@
 #include <iostream>;
-  int f() { return 0; }
+  int f() { return 1; }
 class foo () {
   int f() { return 0; } // no
   int g();
 };
 namespace ns { int f() { return 0; } }; // no
-int foo::g() { return 0; }
+int foo::g() { return 1; }
 int main(){ return foo f+f.f()+f.g()+f()+ns::f(); }
```

example name: cex_cplusplus2.cpp

Functions modifying variable

```
@@
identifier F;
type R,T;
parameter list p;
global idexpression T I = {a};
expression E;
assignment operator ao;
@@
+ // modifies a:
 R F(p)
 {
   <+...
   I ao E
   ...+>
 }
```

```
@@ -1,7 +1,8 @@
 int a,b;
 int g() { b=a; }
+// modifies a:
 int f() { a=b; }
 int h() { f( ); g( ); }
 int l() { h( ); g( ); }
 int i() { h( ); l( ); }
 int main() { i( ); }
```

example name: `cex_func_mod_var_1.c`

Debugging, documentation

Functions modifying variable, again

```
@mf@
identifier F;
type R,T;
parameter list p;
global idexpression T I = {a};
expression E;
assignment operator ao;
@@
 R F(p)
 {
   <+...
   I ao E
   ...+>
 }

@@
identifier mf.F,F1;
type R;
@@
+ // calls a function modifying a:
 R F1(...)
 {
   <+...
   F(...);
   ...+>
 }
```

```
@@ -1,7 +1,8 @@
 int a,b;
 int g() { b=a; }
 int f() { a=b; }
+// calls a function modifying a:
 int h() { f( ); g( ); }
 int l() { h( ); g( ); }
 int i() { h( ); l( ); }
 int main() { i( ); }
```

example name: `cex_func_mod_var_2.c`

Investigate tricky missing synchronization

Identifying recursive functions

```
@m0@
identifier F0;
type R;
parameter list p;
@@
+ // recursive:
 R F0(p) {
   ...
   F0(...)
   ...
 }
```

```
@@ -1,6 +1,8 @@
+// recursive:
 int f(int i) { f(i-1); }
 int h(int i);
 int g(int i) { h(i-1); }
 int h(int i) { return g(i-1); }
+// recursive:
 int l(int i) { return l(i-1); }
 int main() { f(1); g(1); h(1); }
```

example name: `cex_func_recursive_1.c`

Spot tricky interactions

Identifying mutually recursive functions

```
1  @ar@
2  identifier F0;
3  type R;
4  @@
5   R F0(...) { ...  }
6
7  @rf@
8  identifier ar.F0;
9  type ar.R;
10 @@
11  R F0(...) { ...  F0(...)  ...  }
12
13 @nr depends on !rf@
14 identifier F1;
15 identifier ar.F0;
16 type ar.R;
17 @@
18  R F0(...) { ...  F1(...)  ...  }
19
20 @@
21 identifier ar.F0,nr.F1;
22 type S;
23 @@
24 + // mutual recursion detected:
25  S F1(...) { ...  F0(...)  ...  }
```

```
0 @@ -1,6 +1,9 @@
1   int f(int i) { f(i-1); }
2 +// mutual recursion detected:
3   int h(int i);
4 +// mutual recursion detected:
5   int g(int i) { h(i-1); }
6 +// mutual recursion detected:
7   int h(int i) { return g(i-1); }
8   int l(int i) { return l(i-1); }
9   int main() { f(1); g(1); h(1); }
```

example name: `cex_func_recursive_4.c`

Spot trickier interactions

Array of Arrays of Arrays ⇒ Array

```
@@ @@
  double ***a3;
+ double *a1;
+ #define A3D(X,Y,Z) ((X)*(M*N)+(Y)*(N)
      +(M))

@@ @@
- a3 = calloc (...);
+ a1 = calloc (L*M*N,sizeof(*a1));

@@
expression E1,E2,E3;
@@
- a3[E1][E2][E3]
+ a1[A3D(E1,E2,E3)]
```

```
@@ -1,18 +1,20 @@
 #include <stdlib.h>
 double ***a3;
+double *a1;
+#define A3D(X,Y,Z) ((X) * (M * N) + (Y)
      * (N) + (M))
 int main() {
   int i,j,k;
   const int L=2,M=3,N=4;

-  a3 = calloc(L,sizeof(*a3));
+  a1 = calloc(L * M * N, sizeof(*a1));
   for (i=0;i<L;++i)
   {
    a3[i]= calloc(M,sizeof(**a3));
    for (j=0;j<M;++j)
      a3[i][j]= calloc(N,sizeof(***a3));
   }
   for (i=0;i<L;++i)
    for (j=0;j<M;++j)
     for (k=0;k<N;++k)
-      a3[i][j][k]=i+j+k;
+      a1[A3D(i, j, k)]=i+j+k;
 }
```

example name: `cex_arrays3Dto1D_1.c`

How to restructure code full of indirect accesses?

Thanks to Dr. Andre Kurzmann (LRZ) for suggesting this problem!

Array of Arrays of Arrays ⇒ Array (refinements)

```
@@ @@
- double ***a3;
+ double *a1;
+ #define A3D(X,Y,Z) ((X)*(M*N)+(Y)*(N)
      +(M))

@@ @@
- a3 = calloc (...);
+ a1 = calloc (L*M*N,sizeof(*a1));

@@ expression E1,E2,E3; @@
- a3[E1][E2][E3]
+ a1[A3D(E1,E2,E3)]

@@ statement S; @@
(
- a3@S = calloc ( ... );
|
- a3[...]@S = calloc ( ... );
|
- a3[...][...]@S = calloc( ... );
)

@@ @@
- for(...;...;...) { }
@@ @@
- for(...;...;...) { }
@ identifier@ @@
- a1
+ a3
```

```
@@ -1,18 +1,13 @@
 #include <stdlib.h>
-double ***a3;
+double *a3;
+#define A3D(X,Y,Z) ((X) * (M * N) + (Y)
      * (N) + (M))
 int main() {
   int i,j,k;
   const int L=2,M=3,N=4;

-  a3 = calloc(L,sizeof(*a3));
-  for (i=0;i<L;++i)
-  {
-   a3[i]= calloc(M,sizeof(**a3));
-   for (j=0;j<M;++j)
-     a3[i][j]= calloc(N,sizeof(***a3));
-  }
+  a3 = calloc(L * M * N, sizeof(*a3));
   for (i=0;i<L;++i)
    for (j=0;j<M;++j)
     for (k=0;k<N;++k)
-     a3[i][j][k]=i+j+k;
+     a3[A3D(i, j, k)]=i+j+k;
 }
```

example name: `cex_arrays3Dto1D_4.c`

`#pragma omp parallel` insertion

```
1  @sr@
2  identifier A={A};
3  statement S;
4  @@
5   \( S \& A \)
6
7  @fr@
8  identifier I;
9  statement sr.S;
10 position P;
11 @@
12  for( I=0; I<n; ++I) S@P
13
14 @ depends on fr@
15 statement sr.S;
16 position fr.P;
17 @@
18 + #pragma omp parallel
19   for( ...; ...; ...) S@P
```

```
0  @@ -1,10 +1,11 @@
1   int main() {
2     const n=10;
3     double A[n];
4     double B[3];
5     int i;
6  +  #pragma omp parallel
7     for(i=0;i<n;++i) A[i]++;
8     for(i=0;i<3;++i) A[i]++;
9     for(i=0;i<3;++i) B[i]++;
10    for(i=0;i<3;++i) A[i]--;
11  }
```

example name: `cex_wishlist_insert_omp_1.c`

Apply to *selected* loops

Scripting for custom comments insertion

```
@nr exists@
identifier CALLED;
identifier CALLER;
type R;
parameter list p;
@@
 R CALLER(p) { ... when any
   CALLED(...)
   ... when any
 }

@script:python pr@
CALLER << nr.CALLER;
CALLED << nr.CALLED;
K;
@@
coccinelle.K=cocci.make_ident("/* %s()
     invoked by %s() */" % (CALLED,
     CALLER));

@nri@
identifier pr.K;
identifier nr.CALLED;
type nr.R;
parameter list p;
@@
  R CALLED(p) {
++K;
  ...
}
```

```
@@ -1,8 +1,12 @@
-void f() { }
-void g() { f() ; }
-void h() { f() ; }
+void f() {
+       /* f() invoked by h() */;
+       /* f() invoked by g() */; }
+void g() {
+       /* g() invoked by i() */; f() ;
     }
+void h() {
+       /* h() invoked by i() */; f() ;
     }
 void i() { g() ; h() ; }
 int main() {
   f();
   g();
 }
```

Please note this is a workaround!

Call tree analysis

```
1  @initialize:python@
2  @@
3  KL=[]
4
5  @nr@
6  identifier CALLED;
7  identifier CALLER;
8  type R;
9  parameter list p;
10 @@
11  R CALLER(p) { ... CALLED(...) ... }
12
13 @script:python@
14 CALLER << nr.CALLER;
15 CALLED << nr.CALLED;
16 @@
17 KL.append("%s -> %s" % (CALLER,CALLED));
18
19 @finalize:python@
20 @@
21 print "// " + str(len(KL)) + " relations:"
22 for kl in KL:
23         print "//",kl
```

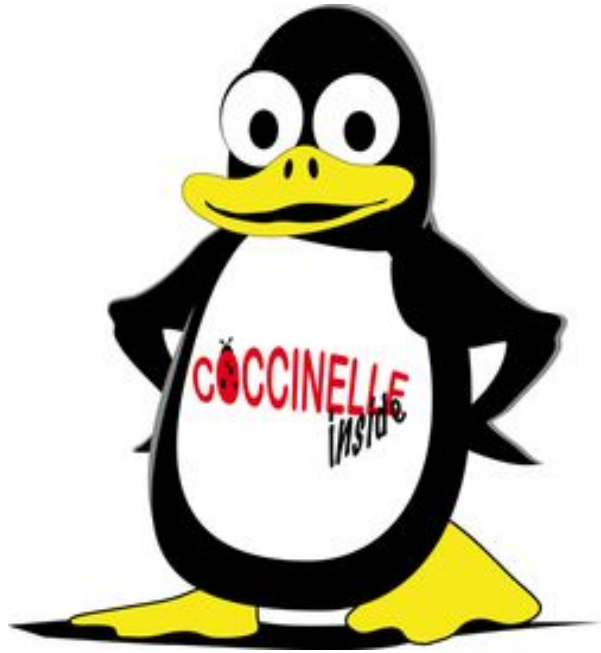
example name: `cex_call_tree_1.c`

Can arrange for other, specific analyses

Summing up

Coccinelle

- powerful *open source* tool
- unique in its kind
- expressible almost as C itself
- **let's check it out for HPC code restructuring!**

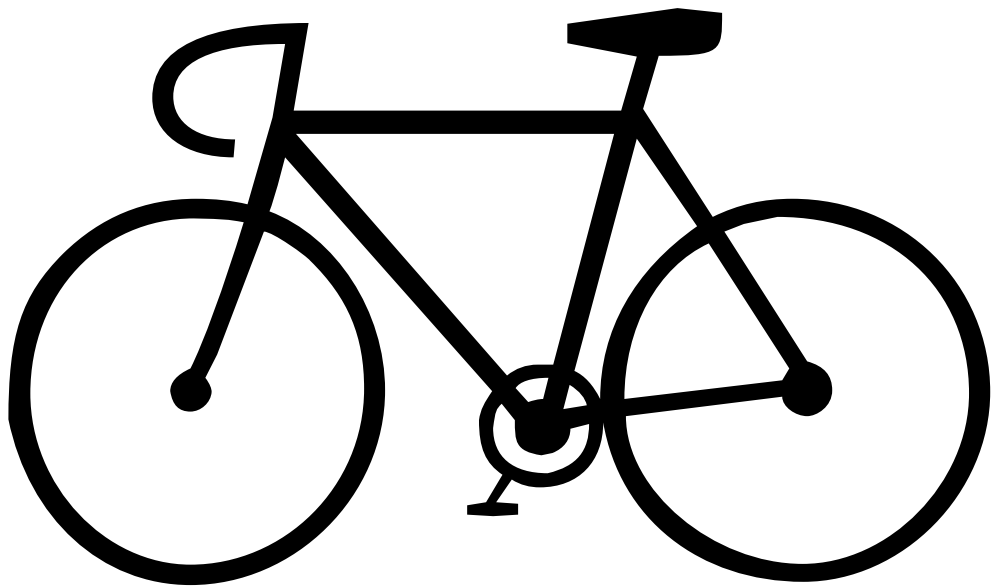


Thanks to:

Dr. Julia Lawall, Inria
Dr. Ruben Garcia Hernandez, LRZ

http://coccinelle.lip6.fr

Let's go home ! It's 17:00



License terms of this presentation: **CC BY 4.0**

Creative Commons Attribution 4.0 International Public License

https://creativecommons.org/licenses/by/4.0/deed.en