Introduction to Coccinelle

Julia Lawall (Inria/LIP6)

http://coccinelle.lip6.fr

September 17, 2014

Common programming problems

- Programmers don't really understand how C works.
  - `!e1 & e2` does a bit-and with 0 or 1.
- A simpler API function exists, but not everyone uses it.
  - Mixing different functions for the same purpose is confusing.
- A function may fail, but the call site doesn't check for that.
  - A rare error case will cause an unexpected crash.
- Etc.

Need for pervasive code changes.

Example: Bad bit-and

```
if (!dma_cntrl & DMA_START_BIT) {
   BCMLOG(BCMLOG_DBG, "Already Stopped\n");
   return BC_STS_SUCCESS;
}
```

From drivers/staging/crystalhd/crystalhd_hw.c

Example: Inconsistent API usage

drivers/mtd/nand/r852.c:

```
if (!bounce) {
  dev->phys_dma_addr =
    pci_map_single(dev->pci_dev, (void *)buf, R852_DMA_LEN,
      (do_read ? PCI_DMA_FROMDEVICE : PCI_DMA_TODEVICE));

  if (pci_dma_mapping_error(dev->pci_dev, dev->phys_dma_addr))
    bounce = 1;
}
```

drivers/mtd/nand/denali.c:

```
denali->buf.dma_buf =
   dma_map_single(&dev->dev, denali->buf.buf, DENALI_BUF_SIZE,
                  DMA_BIDIRECTIONAL);
if (dma_mapping_error(&dev->dev, denali->buf.dma_buf))  ...
pci_set_master(dev);
...
ret = pci_request_regions(dev, DENALI_NAND_NAME);
```

Example: Missing error check

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                         MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

From arch/cris/arch-v32/mm/intmem.c

Our goals

- Automatically find code containing bugs or defects, or requiring collateral evolutions.
- Automatically fix bugs or defects, and perform collateral evolutions.
- Provide a system that is accessible to software developers.

Requirements for automation

The ability to abstract over irrelevant information:

- `if (!dma_cntrl & DMA_START_BIT) { ... }`: `dma_cntrl` is not important.

The ability to match scattered code fragments:

- `kmalloc` may be far from the first dereference.

The ability to transform code fragments:

- Replace `pci_map_single` by `dma_map_single`, or vice versa.

Coccinelle

Program matching and transformation for unpreprocessed C code.

Fits with the existing habits of C programmers.

- C-like, patch-like notation

Semantic patch language (SmPL):

- Metavariables for abstracting over subterms.
- "..." for abstracting over code sequences.
- Patch-like notation ($-$/$+$) for expressing transformations.

The !& problem

The problem: Combining a boolean (0/1) with a constant using & is usually meaningless:

```
if (!dma_cntrl & DMA_START_BIT) {
   BCMLOG(BCMLOG_DBG, "Already Stopped\n");
   return BC_STS_SUCCESS;
}
```

The solution: Add parentheses.

Our goal: Do so automatically for any expression E and constant C.

A semantic patch for the !& problem

```
@@
expression E;
constant C;
@@

- !E & C
+ !(E & C)
```

Two parts per rule:

- Metavariable declaration
- Transformation specification

A semantic patch can contain multiple rules.

Metavariable types

Surrounded by `@@` `@@`.

- expression, statement, type, constant, local idexpression
- A type from the source program
- iterator, declarer, iterator name, declarer name, typedef

Transformation specification

- `-` in the leftmost column for something to remove
- `+` in the leftmost column for something to add
- `*` in the leftmost column for something of interest
  - Cannot be used with + and -.
- Spaces, newlines irrelevant.

Exercise 1

1. Create a file ex1.cocci containing the following:

```
@@
expression E;
constant C;
@@

- !E & C
+ !(E & C)
```

2. Run spatch: `spatch --sp-file ex1.cocci --dir linux-3.2/drivers/staging/crystalhd`
3. Did your semantic patch do everything it should have?
4. Did it do something it should not have?

Exercise 2

Some code contains a cast on the result of `kmalloc`. For example:

```
info->RegsBuf = (unsigned char *)
  kmalloc(sizeof(info->ATARegs), GFP_KERNEL);
```

If the destination of the returned value has pointer type, this cast is not needed.

1. Complete the following semantic patch to remove this unnecessary cast.

   ```
   @@ expression * e; expression arg1, arg2; type T; @@

   [fill it in]
   ```

2. Test your semantic patch on the code in `linux-3.2/drivers/isdn`
3. Are you satisfied with the appearance of the results? If not, try to improve it.

Practical issues

To check that your semantic patch is valid:

```
spatch --parse-cocci mysp.cocci
```

To run your semantic patch:

```
spatch --sp-file mysp.cocci file.c
spatch --sp-file mysp.cocci --dir directory
```

If you don't need to include header files:

```
spatch --sp-file mysp.cocci --dir directory
                    --no-includes --include-headers
```

To understand why your semantic patch didn't work:

```
spatch --sp-file mysp.cocci file.c --debug
```

More practical issues

Put the interesting output in a file:

```
spatch ... > output.patch
```

Omit the uninteresting output:

```
spatch --very-quiet ...
```

The source code:

```
/usr/src/linux-source-3.2/scripts/coccinelle/
```

These slides:

```
http://pagesperso-systeme.lip6.fr/Julia.Lawall/
suse_tutorial.pdf
```

Inconsistent API usage

Do we need this function?

```
static inline dma_addr_t
pci_map_single(struct pci_dev *hwdev, void *ptr, size_t size,
               int direction)
{
  return dma_map_single(hwdev == NULL ? NULL : &hwdev->dev, ptr,
                        size, (enum dma_data_direction)direction);
}
```

The use of pci_map_single

The code:

```
dev->phys_dma_addr =
    pci_map_single(dev->pci_dev, (void *)buf, R852_DMA_LEN,
      (do_read ? PCI_DMA_FROMDEVICE : PCI_DMA_TODEVICE));
```

would be more uniform as:

```
dev->phys_dma_addr =
    dma_map_single(&dev->pci_dev->dev, (void *)buf, R852_DMA_LEN,
      (do_read ? DMA_FROM_DEVICE : DMA_TO_DEVICE));
```

Issues:

- Change function name.
- Add field access to the first argument.
- Rename the fourth argument.

pci_map_single: Example and definitions

Commit b0eb57cb

```
- rbi->dma_addr = pci_map_single(adapter->pdev,
+ rbi->dma_addr = dma_map_single(
+     &adapter->pdev->dev,
      rbi->skb->data, rbi->len,
      PCI_DMA_FROMDEVICE);
```

PCI constants

```
/* This defines the direction arg
 to the DMA mapping routines. */
#define PCI_DMA_BIDIRECTIONAL   0
#define PCI_DMA_TODEVICE        1
#define PCI_DMA_FROMDEVICE      2
#define PCI_DMA_NONE            3
```

DMA constants

```
enum dma_data_direction {
        DMA_BIDIRECTIONAL = 0,
        DMA_TO_DEVICE = 1,
        DMA_FROM_DEVICE = 2,
        DMA_NONE = 3,
};
```

pci_map_single: First attempt

Outline of a semantic patch, including the patch example:

```
@@

@@

- rbi->dma_addr = pci_map_single(adapter->pdev,
+ rbi->dma_addr = dma_map_single(
+     &adapter->pdev->dev,
      rbi->skb->data, rbi->len,
      PCI_DMA_FROMDEVICE);
```

pci_map_single: First attempt

Eliminate irrelevant code:

```
@@

@@

- pci_map_single(adapter->pdev,
+ dma_map_single(
+     &adapter->pdev->dev,
      rbi->skb->data, rbi->len,
      PCI_DMA_FROMDEVICE)
```

pci_map_single: First attempt

Abstract over subterms:

```
@@
expression E1,E2,E3;
@@

- pci_map_single(E1,
+ dma_map_single(
+     &E1->dev,
      E2, E3,
      PCI_DMA_FROMDEVICE)
```

pci_map_single: First attempt

Rename the fourth argument:

```
@@
expression E1,E2,E3;
@@

- pci_map_single(E1,
+ dma_map_single(
+     &E1->dev,
      E2, E3,
-     PCI_DMA_FROMDEVICE)
+     DMA_FROM_DEVICE)
```

pci_map_single: Second attempt

Need to consider all direction constants.

```
@@ expression E1,E2,E3; @@
- pci_map_single(E1,
+ dma_map_single(&E1->dev,
      E2, E3,
-     PCI_DMA_FROMDEVICE)
+     DMA_FROM_DEVICE)

@@ expression E1,E2,E3; @@
- pci_map_single(E1,
+ dma_map_single(&E1->dev,
      E2, E3,
-     PCI_DMA_TODEVICE)
+     DMA_TO_DEVICE)
```

Etc. Four rules in all.

pci_map_single: Third attempt

Avoid code duplication: Use a disjunction.

```
@@ expression E1,E2,E3; @@
- pci_map_single(E1,
+ dma_map_single(&E1->dev,
      E2, E3,
(
-     PCI_DMA_BIDIRECTIONAL
+     DMA_BIDIRECTIONAL
|
-     PCI_DMA_TODEVICE
+     DMA_TO_DEVICE
|
-     PCI_DMA_FROMDEVICE
+     DMA_FROM_DEVICE
|
-     PCI_DMA_NONE
+     DMA_NONE_DEVICE
)
  )
```

pci_map_single: Fourth attempt

```
@@ expression E1,E2,E3,E4; @@
- pci_map_single(E1,
+ dma_map_single(&E1->dev,
      E2, E3, E4)

@@ expression E1,E2,E3; @@
dma_map_single(E1, E2, E3,
(
-     PCI_DMA_BIDIRECTIONAL
+     DMA_BIDIRECTIONAL
|
-     PCI_DMA_TODEVICE
+     DMA_TO_DEVICE
|
-     PCI_DMA_FROMDEVICE
+     DMA_FROM_DEVICE
|
-     PCI_DMA_NONE
+     DMA_NONE_DEVICE
)
  )
```

Exercise 3

1. Implement some version of the semantic patch for converting calls to `pci_map_single` to calls to `dma_map_single`.
2. Test your implementation on the directory `linux-3.2/drivers/net/ethernet`.
3. Implement both the third version and the fourth version. Compare the results.
4. Other PCI functions replicate DMA behavior, *e.g.*, `pci_unmap_single`. For example, commit b0eb57cb contains:

```
-   pci_unmap_single(pdev, tbi->dma_addr, tbi->len,
+   dma_unmap_single(&pdev->dev, tbi->dma_addr, tbi->len,
                     PCI_DMA_TODEVICE);
```

   Extend your semantic patch to implement this transformation. Try to minimize the number of rules.

Getter and setter functions

Some functions from `include/linux/ide.h`:

```
static inline void *
ide_get_hwifdata (ide_hwif_t * hwif)
{
        return hwif->hwif_data;
}

static inline void
ide_set_hwifdata (ide_hwif_t * hwif, void *data)
{
        hwif->hwif_data = data;
}
```

Goal: Replace uses of `hwif->hwif_data` by calls to these function.

Getter and setter functions: First attempt

```
@@
expression hwif;
@@
- hwif->hwif_data
+ ide_get_hwifdata(hwif)

@@
expression hwif, data;
@@
- hwif->hwif_data = data
+ ide_set_hwifdata(hwif, data)
```

Problems

```
@@ expression hwif; @@
- hwif->hwif_data
+ ide_get_hwifdata(hwif)
```

- The rule applies to

  ```
  unsigned long base = (unsigned long)hwif->hwif_data;
  ```

  but also to

  ```
  hwif->hwif_data = NULL;
  ```

  The rule transforms all `hwif_data` field references.

Second attempt: Rule order

```
@@
expression hwif, data;
@@
- hwif->hwif_data = data
+ ide_set_hwifdata(hwif, data)

@@
expression hwif;
@@
- hwif->hwif_data
+ ide_get_hwifdata(hwif)
```

Applies to 9 code sites, in 2 files.

Third attempt: Metavariable type constraints

```
@@ ide_hwif_t *hwif; expression data; @@
- hwif->hwif_data = data
+ ide_set_hwifdata(hwif, data)

@@ ide_hwif_t *hwif; @@
- hwif->hwif_data
+ ide_get_hwifdata(hwif)
```

Can optionally add `typedef ide_hwif_t;` in the first rule.

- Typedef is needed when the type appears only in a cast.
- Typedef appears only once, where earliest needed.

Exercise 4

1. Implement all three variants of the semantic patch for introducing the `ide_get_hwifdata` and `ide_set_hwifdata` getter and setter functions.
2. Test each variant on the directory `linux-3.2/drivers/ide`, and compare the results.
3. Reimplement each variant using a disjunction.
4. Compare the results of the disjunction variants to the original implementations.

Exercise 5

In the case of `pci_map_single` and `dma_map_single`, we could also prefer to convert PCI occurrences of `dma_map_single` to calls to `pci_map_single` (i.e., the reverse transformation).

1. Implement a semantic patch to do this transformation.
2. Test your semantic patch on the directory `linux-3.2/drivers/net/ethernet`.
3. Given the definition of `pci_map_single` in `include/asm-generic/pci-dma-compat.h`, why should the file `ethernet/cadence/macb.c` not be transformed?
4. Check that your semantic patch makes no modification in this file.

Summary

SmPL features seen so far:

- Metavariables for abstracting over arbitrary terms.
- Metavariables restricted to particular types.
- Disjunctions.
- Multiple rules.
- Rule ordering.

Coccinelle Features

Julia Lawall (Inria/LIP6)

http://coccinelle.lip6.fr

September 17, 2014

Coccinelle features

- Isomorphisms.
- Dots.
- Positions.
- Python.

Isomorphisms

Issue:

- Coccinelle matches code exactly as it appears.
- `x == NULL` does not match `!x`.

Goal:

- Transparently treat similar code patterns in a similar way.

Example: DIV_ROUND_UP

The following code is fairly hard to understand:

```
return (time_ns * 1000 + tick_ps - 1) / tick_ps;
```

kernel.h provides the following macro:

```
#define DIV_ROUND_UP(n,d) (((n) + (d) - 1) / (d))
```

This is used, but not everywhere it could be.

We can write a semantic patch to introduce new uses.

DIV_ROUND_UP semantic patch

One option:

```
@@ expression n,d; @@

- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Another option:

```
@@ expression n,d; @@

- (n + d - 1) / d
+ DIV_ROUND_UP(n,d)
```

Problem: How many parentheses to put, to capture all occurrences?

Isomorphisms

An isomorphism relates code patterns that are considered to be similar:

```
Expression
@ drop_cast @ expression E; pure type T; @@

 (T)E => E

Expression
@ paren @ expression E; @@

 (E) => E

Expression
@ is_null @ expression X; @@

 X == NULL <=> NULL == X => !X
```

Isomorphisms, contd.

Isomorphisms are handled by rewriting.

```
(((n) + (d) - 1) / (d))
```

becomes:

```
(
 (((n) + (d) - 1) / (d))
|
 (((n) + (d) - 1) / d)
|
 (((n) + d - 1) / (d))
|
 (((n) + d - 1) / d)
|
 ((n + (d) - 1) / (d))
|
 ((n + (d) - 1) / d)
|
 ((n + d - 1) / (d))
|
 ((n + d - 1) / d)
|
 etc.
)
```

Results

```
@@
expression n,d;
@@

- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Changes 281 occurrences in Linux 3.2.

Practical issues

Default isomorphisms are defined in standard.iso

To use a different set of default isomorphisms:

```
spatch --sp-file mysp.cocci --dir linux-x.y.z --iso-file empty.iso
```

To drop specific isomorpshisms:

```
@disable paren@ expression n,d; @@
- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

To add rule-specific isomorphisms:

```
@using "myparen.iso" disable paren@
expression n,d;
@@
- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Exercise 6

Some Linux code combines an assignment with a test, as illustrated by the following:

```
if (!(p = kmalloc(sz, GFP_KERNEL)))
  break;
```

The following semantic patch moves the assignment out of the conditional:

```
@@ identifier e1; expression e2; statement S1, S2; @@
+ e1 = e2;
  if (
-     (e1 = e2)
+     e1
      == NULL) S1 else S2
```

1. Test this semantic patch on linux-3.2/sound/pci/au88x0
2. How were isomorphisms used in these matches?

Exercise 7

Run

```
spatch --parse-cocci sp.cocci
```

For some semantic patch `sp.cocci` that you have developed.

Explain the result.

Dots

Issue:

- Sometimes it is necessary to search for multiple related code fragments.

Goals:

- Specify patterns consisting of fragments of code separated by arbitrary execution paths.
- Specify constraints on the contents of those execution paths.

Example: Inadequate error checking of kmalloc

`kmalloc` returns NULL on insufficient memory.

Good code:

```
block = kmalloc(WL12XX_HW_BLOCK_SIZE, GFP_KERNEL);
if (!block)
  return;
```

Bad code:

```
g = kmalloc (sizeof (*g), GFP_KERNEL);
g->next = chains[r_sym].next;
```

More bad code

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                         MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

The `kmalloc` and the dereference are not necessarily contiguous.

Using dots

Start with a typical example of code

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                         MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Using dots

Highlight what is wanted

```
* alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                           MEM_INTMEM_SIZE - RESERVED_SIZE);
  initiated = 1;
* alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Using dots

Replace the irrelevant statements by ...

```
* alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  ...
```

```
* alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Using dots

Abstract over irrelevant subterms.

- May use `...`.

```
@@ expression e; identifier f; @@
* e      = kmalloc(...);
  ...



* e->f
```

Using dots

Check properties of the matched statement sequence

```
@@ expression e; identifier f; @@
* e      = kmalloc(...);
  ... when != e == NULL
      when != e != NULL


* e->f
```

Using dots

Sanity check

```
@@ expression e, e1; identifier f; @@
* e      = kmalloc(...);
  ... when != e == NULL
      when != e != NULL
      when != e = e1

* e->f
```

Results: 18 kmallocs in 12 files

Real bug: linux-3.2/arch/cris/arch-v32/mm/intmem.c

```
- alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                           MEM_INTMEM_SIZE - RESERVED_SIZE);
  initiated = 1;
- alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Results: 18 kmallocs in 12 files

Real bug: linux-3.2/arch/cris/arch-v32/mm/intmem.c

```
- alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                           MEM_INTMEM_SIZE - RESERVED_SIZE);
  initiated = 1;
- alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

False positive! linux-3.2/net/ipv4/syncookies.c

```
- ireq->opt = kmalloc(opt_size, GFP_ATOMIC);
- if (ireq->opt != NULL && ip_options_echo(&ireq->opt->opt, skb)) {
    kfree(ireq->opt);
    ireq->opt = NULL;
  }
```

False positives

```
ireq->opt != NULL && ip_options_echo(&ireq->opt->opt, skb)
```

"`...`" matches complete statements.

- `ireq->opt != NULL` is not seen as being before `&ireq->opt->opt`.

Solution: stop at NULL tests or bad dereference (disjunction).

- `e == NULL`: OK
- `e != NULL`: OK
- `e->f`: Bug

Revised version

```
@@ expression e,e1; identifier f; @@
  e = kmalloc(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
* e->f
)
```

Shortest path property:

- "..." matches everything except what is on either side.

Matches 11 files, eliminating the false positive.

Exercise 8

The following code allocates a region of memory and then clears it:

```
state = kmalloc(sizeof(struct drxd_state), GFP_KERNEL);
if (!state)
  return NULL;
memset(state, 0, sizeof(*state));
```

The function `kzalloc` does both, *i.e.*, we could write:

```
state = kzalloc(sizeof(struct drxd_state), GFP_KERNEL);
if (!state)
  return NULL;
```

1. Write a semantic patch to make this transformation.
2. Test your semantic patch on `linux-3.2/drivers/net/wireless`.
3. Are there any files where your semantic patch should not transform the code, but it does?

Exercise 9

One of the results for the `kmalloc` with no NULL test example is the following (linux-3.2/drivers/macintosh/via-pmu.c):

```
- pp = kmalloc(sizeof(struct pmu_private), GFP_KERNEL);
  if (pp == 0)
     return -ENOMEM;
- pp->rb_get = pp->rb_put = 0;
```

The code will not crash, but it is not as nice as it could be. Write a semantic patch to replace such bad uses of `0` by `NULL`.

Hints:

- A metavariable of type "`expression *`" matches any pointer typed expression.
- This exercise has nothing to do with dots.

Exercise 10

Clearing a region of memory using `memset` before the region of memory goes out of scope is useful to ensure that information cannot leak to the next user of that memory. Nevertheless, gcc may consider that the call to `memset` is useless and remove it. A new function `memset_explicit` has been introduced to get around this problem. The goal of this exercise is to introduce uses of this function.

1. Basically, we are concerned with a local variable declaring memory that is subsequently passed into the first argument of `memset`. Write one or more rules that express the possible cases. Does your rule match everything it should? Does it match anything it should not?

Exercise 10, contd.

An example of the desired transformation is as follows:

```
static int sha224_ssse3_final(struct shash_desc *desc, u8 *hash)
{
  u8 D[SHA256_DIGEST_SIZE];

  sha256_ssse3_final(desc, D);

  memcpy(hash, D, SHA224_DIGEST_SIZE);
- memset(D, 0, SHA256_DIGEST_SIZE);
+ memset_explicit(D, 0, SHA256_DIGEST_SIZE);

  return 0;
}
```

Exercise 10, contd.

2 Normally, `...` matches the shortest path from what comes before the `...` to what comes after. Furthermore, when transformation is performed, all paths must satisfy the pattern, but not those concerned with error-handling code (non strictness). Consider where the following can be placed in your semantic patch to improve the result.

- when any
- when exists
- when strict
- when != x

This exercise was inspired by the work of Daniel Borkmann

Positions and Python

```
@@ expression e,e1; identifier f; @@
  e = kmalloc(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
* e->f
)
```

Output reported as a diff:

- Useful in emacs (diff-mode).
- Perhaps less useful in other contexts.

Why is there no * on `kmalloc`?

Positions and Python

Goal:

- Collect positions of some matched elements.
- Print a helpful error message.

```
@r@
expression e,e1;
identifier f;
position p1, p2;
@@
  e = kmalloc@p1(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
  e@p2->f
)
```

```
@script:python@
p1 << r.p1;
p2 << r.p2;
@@
l1 = p1[0].line
l2 = p2[0].line
print "kmalloc on line %s not tested
       before reference on line %s" %
       (l1,l2)
```

A refinement

Exists:

- Require only a single matching execution path.
- Default for *.

```
@r exists@
expression e,e1;
identifier f;
position p1, p2;
@@
  e = kmalloc@p1(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
  e@p2->f
)
```

```
@script:python@
p1 << r.p1;
p2 << r.p2;
@@
l1 = p1[0].line
l2 = p2[0].line
print "kmalloc on line %s not tested
       before reference on line %s" %
       (l1,l2)
```

Exercise 11

Rewrite a semantic patch that you have implemented previously, so that it prints the line numbers on which a change is needed, rather than making the change.

Useful terms:

- `p[0].file` is the name of the file represented by `p`.
- `p[0].line` is the number, as a string, of the line represented by `p`.
- `p` is an array, because there can be many matches.

Exercise 12

1. The following semantic patch rule matches an initialization of a platform driver structure:

```
@platform@
identifier p, probefn, removefn;
@@
struct platform_driver p = {
  .probe = probefn,
  .remove = removefn,
};
```

   Extend this semantic patch to find the name of the first parameter of the probe function and of the remove function, and to print the function names and the corresponding parameter names using python.

2. Some platform driver probe functions use the function `kzalloc` to allocate memory. Adjust the previous semantic patch to find and print the positions of these calls.

Exercise 12, contd.

3. `Kzallocd` memory must be freed using `kfree`, but this is easy to forget. The function `devm_kzalloc` is like `kzalloc` but the driver library manages the freeing. Adjust the previous semantic patch to replace calls to `kzalloc` by calls to `devm_kzalloc`.

4. The code produced by the previous rule does not compile, because `devm_kzalloc` requires a device argument. This can be constructed from the first parameter of a platform driver probe function. If this parameter is named `x`, then the corresponding device value is `&x->dev`. Adjust the previous semantic patch to add this as the first argument of each generated call to `devm_kzalloc`.

Exercise 12, contd.

5. The code resulting from the previous semantic patch has double frees, because `devm_kzalloc` causes an implicit free, and the code still contains calls to `kfree`. A more complete, but not perfect, solution to this problem is found at `http://kernelnewbies.org/JuliaLawall_round8`. Study and explain the semantic patch code.

6. Test the `devm_kzalloc` semantic patch found at `http://kernelnewbies.org/JuliaLawall_round8`, e.g., on drivers/net/ethernet, and explain and try to resolve any deficiencies.

Summary

- Isomorphisms, for simplifying, eg NULL tests, parentheses, casts.
- Dots, for matching a sequence of statements, arguments, etc.
- When, for restricting the contents of sequences.
- Positions, for remembering the exact position of some code.
- Python, for printing error messages, managing hashtables, etc.