Booleans

Julia Lawall (Inria/LIP6)

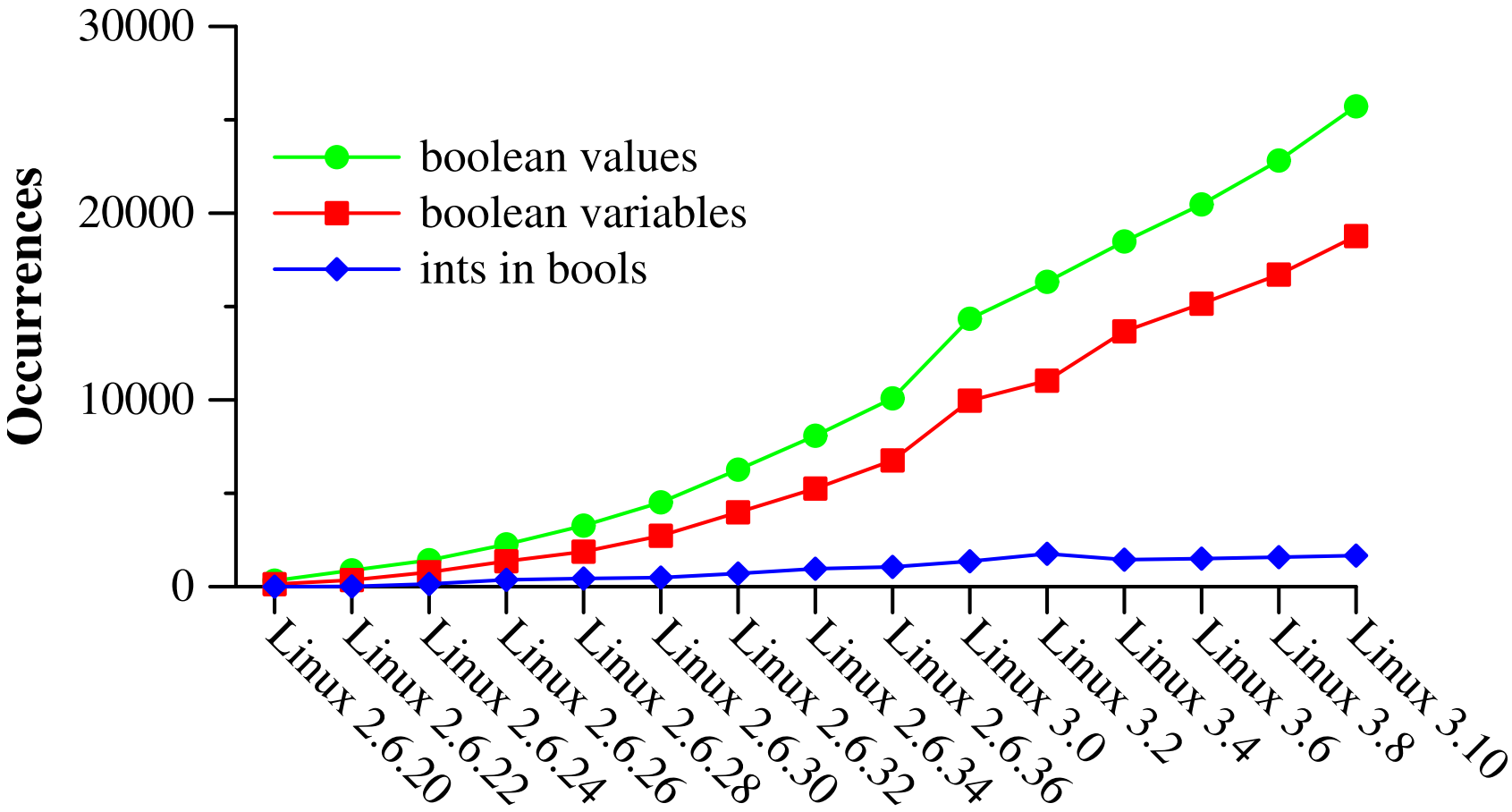
http://coccinelle.lip6.fr

October 31, 2013

The problem

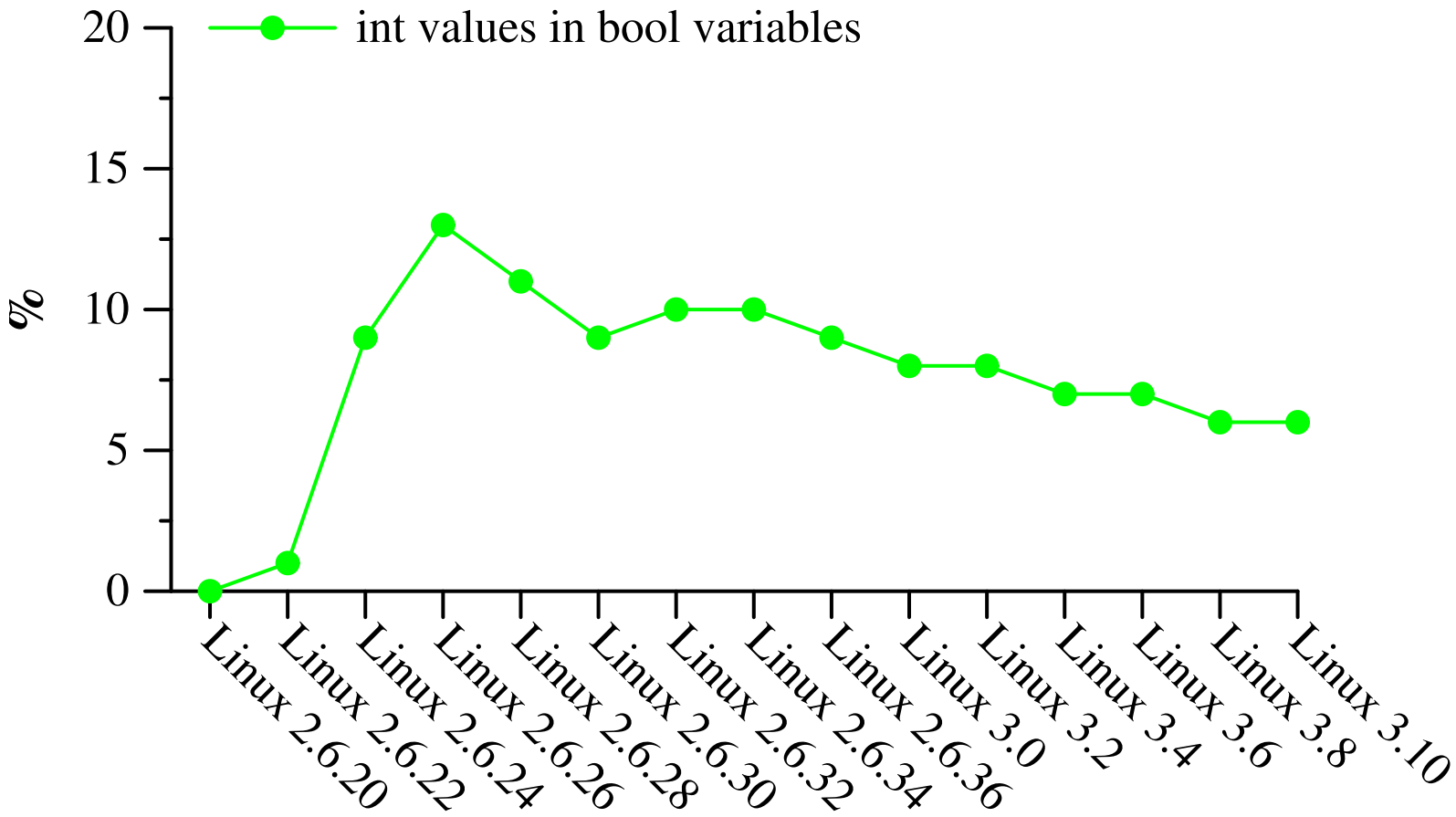
- Linux code commonly uses the type `bool` with values `true` and `false`.
- Some `bool` variables used with `0`, `1`, etc.
- Potentially confusing, because `0` often means success.
- Goal: limit boolean variables to `true` and `false`.

The use of booleans over time



(Roughly every 6 months, February 2007 - June 2013).

The use of booleans over time - rate of bad code



Example

```
static bool overlapping_resync_write(struct drbd_conf *mdev, ...) {
    struct drbd_peer_request *rs_req;
    bool rv = 0;
    spin_lock_irq(&mdev->tconn->req_lock);
    list_for_each_entry(rs_req, &mdev->sync_ee, w.list) {
        if (overlaps(peer_req->i.sector, peer_req->i.size,
                     rs_req->i.sector, rs_req->i.size)) {
            rv = 1;
            break;
        }
    }
    spin_unlock_irq(&mdev->tconn->req_lock);
    return rv;
}
```

Where can the problem occur?

- Assignments to boolean variables.
  - `bool b = 0;`
  - `b = 1;`
- Return values.
  - `bool f(...)  { ... return 0; }`
- Function arguments.
  - Function definition available.
  - Function prototype available.

Another example

```
static bool vgic_ioaddr_overlap(struct kvm *kvm)
{
  phys_addr_t dist = kvm->arch.vgic.vgic_dist_base;
  phys_addr_t cpu = kvm->arch.vgic.vgic_cpu_base;

  if (IS_VGIC_ADDR_UNDEF(dist) || IS_VGIC_ADDR_UNDEF(cpu))
    return 0;
  if ((dist <= cpu && dist + KVM_VGIC_V2_DIST_SIZE > cpu) ||
      (cpu <= dist && cpu + KVM_VGIC_V2_CPU_SIZE > dist))
    return -EBUSY;
  return 0;
}
```

linux-3.10/virt/kvm/arm/vgic.c

What problems can occur?

- The expected values: `true`, `false`
- Another variant: `TRUE`, `FALSE`
- Integers: `0`, `1`, `(bool) 0`, `(bool) 1`
- Integer variables: `ret`
- Other things: `-EBUSY`, `OFF`, `ON`, etc.

Semantic patch design strategy

- Four groups of rules, one for each boolean context:
  - Assignment to a boolean variable.
  - Return value.
  - Function argument, when a function definition is available.
  - Function argument, when a function prototype is available.
- For each group, transform the boolean-like values, e.g., 0 and 1.
- Generate an error message for the non-boolean-like values.
  - Issue: What about `#define` constants (rename 0/1)?

Assignment to a boolean variable

First attempt:

```
@@
bool b;
@@

b =
(
- 0
+ false
|
- 1
+ true
)
```

Issue: What about variable declarations?

Some results

```
diff -u -p a/arch/um/kernel/process.c b/arch/um/kernel/process.c
@@ -425,7 +425,7 @@ unsigned long arch_align_stack(unsigned
 unsigned long get_wchan(struct task_struct *p) {
        unsigned long stack_page, sp, ip;
-       bool seen_sched = 0;
+       bool seen_sched = false;

        if((p == NULL) || (p == current) || (p->state == TASK_RUNNING))
                return 0;
@@ -447,7 +447,7 @@ unsigned long get_wchan(struct task_stru
                ip = *((unsigned long *) sp);
                if (in_sched_functions(ip))
                        /* Ignore everything until ... */
-                       seen_sched = 1;
+                       seen_sched = true;
                else if (kernel_text_address(ip) && seen_sched)
                        return ip;
```

Variable declarations handled by an isomorphism.

Extension to more non-boolean values

```
@@
bool b;
@@
b =
(
- 0
+ false
|
- 1
+ true
|
- (bool) 0
+ false
|
- (bool) 1
+ true
|
- FALSE
+ false
|
- TRUE
+ true
)
```

Other non-boolean values

Examples: `-EBUSY`, `OFF`, `ON`, etc.

- `OFF` and `ON` may be defined as `0` and `1`, respectively.
- `-EBUSY` is not likely to be a boolean.

Strategy:

- Match an assignment of a boolean variable to a constant
  - `true` and `false` are not considered constants.
- See if there is a `#define` of the constant to `0` or `1`.
- If not, print a warning message.

Collect constants stored in boolean variables

```
@@
bool b;
constant c;
@@
b = c
```

How will we want to use the collected information?

- Check for a `#define`.
  - Problem: A defined name should be an identifier, not a constant (expression).
- Print the position of the problem.
  - Problem: Need to detect the position in the match

Matching an identifier constant

```
@@
bool b;
constant c;
identifier i;
@@
b = \(c@i\|c\)
```

- Left disjunct checks for a match of both `c` and `i`.
  - Checks that the expression is both a constant and an identifier, e.g., `OFF`.
- Right disjunct checks for other constants.
  - E.g., `-1`.

Checking for #defines

```
@r1@
bool b;
constant c;
identifier i;
@@
b = \(c@i\|c\)

@r1def@
identifier r1.i;
@@
#define i \(0\|1\)
```

Rule name `r1` allows `i` to be used later.

Print an error message if needed

```
@@ bool b; @@ // simple version, for conciseness
b =
(
- 0
+ false
|
- 1
+ true
)

@r1@ bool b; constant c; identifier i; @@
b = \(c@i\|c\)

@r1def@ identifier r1.i; @@
#define i \(0\|1\)

@script:python depends on !r1def@ c << r1.c; @@
print "value %s on line %s of file %s" % c ???
```

Print an error message if needed

```
@@ bool b; @@ // simple version, for conciseness
b =
(
- 0
+ false
|
- 1
+ true
)

@r1@ bool b; constant c; identifier i; position p; @@
b = \(c@i@p\|c@p\)

@r1def@ identifier r1.i; @@
#define i \(0\|1\)

@script:python depends on !r1def@ c << r1.c; p << r1.p; @@
print "value %s on line %s of file %s" % c p[0].line p[0].file
```

Remaining cases

- Return values.
  - `bool f(...)  { ... return 0; }`
  - Similar to assignment.
- Function definition available.
  - Similar to prototype.
- Function prototype available.

Function prototype case

First attempt:

```
@prot@
identifier f,b;
type T; typedef bool;
@@
T f(...,bool b,...);

@@
identifier prot.f;
@@
f(...,
(
- 0
+ false
|
- 1
+ true
)
 ,...)
```

Issues:

- Return type needed on prototype, to distinguish from a call.
  - Can be omitted on a function definition.
- `typedef` needed for `bool`.
  - Not inferred in the parameter case.
- The transformation is unsafe.
  - 0/1 might not match with the bool parameter.

Connecting arguments and parameters

```
@prot@
identifier f,b;
type T; typedef bool;
parameter list[n] ps;
@@
T f(ps, bool b, ...);

@@
identifier prot.f;
expression list[prot.n] es;
@@
f(es,
(
- 0
+ false
|
- 1
+ true
)
 ,...)
```

Extend to search for bad constants and to print error messages.

Exercise 11

1. Extend the rules for the function prototype case to check for non 0/1 constants and print an error message.
2. Implement the rules for the function return value case.
3. Implement the rules for the function definition case.
4. Suppose a function is declared using both a prototype and its definition. Does it matter whether the rules for function prototypes come before or after the rules for function definitions?

Conclusion

Features

- Inherited variables.
- Position variables.
- Multiple metavariable matches (`c@i@p`).
- Python scripting.
- Parameter/expression list lengths.