Principper for Samtidighed og Styresystemer

Coccinelle

René Rydhof Hansen

April 2008

Sjov og ballade

- Ekstraforelæsning: tirsdag den 20. maj kl. 10–12 i auditorium 0.1.95 (formentlig).
- Spørgetime: tirsdag den 17. juni kl. 13–15 i auditorium 0.1.95.

Opgaver

- Opgave 1: Disk schedulering

Mål

- At kunne forklare hvad collateral evolution er
- At kunne forklare hvordan Coccinelle kan bruges i forbindelse med collateral evolution
- At få et indblik i aktuel forskning omkring styresystemer

Device drivers: a notorious weak point in OS code

Developed by device-experts, not core kernel experts.

- Protocols and coding conventions not always understood.
- Drivers can fall behind the rest of the kernel.
- Errors in driver code lead to system crash.

Previous (and ongoing) work:

- Verification and bug detection [Ball, Engler, Foster, etc.]
- Protocol detection [Engler, Zhou, etc.]
- Safe or domain-specific languages [SPIN, Cyclone, Devil, etc.]

Our focus: automating device driver collateral evolution.

Collateral evolution

Collateral evolution: an update required in a library client in response to an evolution affecting the interface of a library.

In the case of Linux:

- Drivers rely heavily on internal Linux libraries.
- These libraries and their usage protocols are evolving rapidly.
- Updating the drivers that rely on these libraries (collateral evolution) is time-consuming and error prone.
- Updating drivers outside the Linux source tree additionally requires a transfer of expertise.

Example

To access `Scsi_Host` information:

Old protocol: call get, check the result, process, call put.

- New protocol: receive the information as an argument.
- Used by 19 scsi drivers in the Linux source tree.

Goals

Document and automate collateral evolutions.

Need a notation that:

- Is easy to write, easy to read. WYSIWYG approach.
- Expresses code-level transformations.
- Provides confidence in the result.
- Is acceptable to driver maintainers.
- Focuses on device-driver relevant issues
  (eg, simple structure, wide use of copy-paste).

Example: The `Scsi_Host` protocol in more detail

`scsi_host_hn_get` and `scsi_host_put`:

- Access and release a `Scsi_Host` typed structure.
- Manage a reference count. Dangerous.

Linux 2.5.71:

- `scsi_host_hn_get` and `scsi_host_put` no longer exported.
- Functions using them get `Scsi_Host` value as an argument.
- In practice, only affects `proc_info` functions.

A scsi driver: scsiglue.c (simplified proc_info function)

```
static int usb_storage_proc_info (
         char *buffer, char **start, off_t offset,
         int length, int hostno, int inout)
{
  struct us_data *us;
  struct Scsi_Host *hostptr;

  hostptr = scsi_host_hn_get(hostno);
  if (!hostptr) { return -ESRCH; }

  us = (struct us_data*)hostptr->hostdata[0];
  if (!us) {
    scsi_host_put(hostptr);
    return -ESRCH;
  }

  SPRINTF("   Host scsi%d: usb-storage\n", hostno);
  scsi_host_put(hostptr);
  return length;
}
```

A scsi driver: scsiglue.c (simplified proc_info function)

```
static int usb_storage_proc_info (
         char *buffer, char **start, off_t offset,
         int length, int hostno, int inout)
{
  struct us_data *us;
  struct Scsi_Host *hostptr;

  hostptr = scsi_host_hn_get(hostno);
  if (!hostptr) { return -ESRCH; }

  us = (struct us_data*)hostptr->hostdata[0];
  if (!us) {
    scsi_host_put(hostptr);
    return -ESRCH;
  }

  SPRINTF("   Host scsi%d: usb-storage\n", hostno);
  scsi_host_put(hostptr);
  return length;
}
```

A scsi driver: scsiglue.c (simplified proc_info function)

```
static int usb_storage_proc_info (struct Scsi_Host *hostptr,
         char *buffer, char **start, off_t offset,
         int length, int hostno, int inout)
{
  struct us_data *us;







  us = (struct us_data*)hostptr->hostdata[0];
  if (!us) {

    return -ESRCH;
  }

  SPRINTF("   Host scsi%d: usb-storage\n", hostno);

  return length;
}
```

scsiglue patch file

```
--- scsiglue_old.c      Tue Feb 13 13:31:56 2007
+++ scsiglue_new.c      Tue Feb 13 13:31:49 2007
@@ -1,20 +1,14 @@
-static int usb_storage_proc_info (
+static int usb_storage_proc_info (struct Scsi_Host *hostptr,
          char *buffer, char **start, off_t offset,
          int length, int hostno, int inout)
 {
   struct us_data *us;
-  struct Scsi_Host *hostptr;
-
-  hostptr = scsi_host_hn_get(hostno);
-  if (!hostptr) { return -ESRCH; }

   us = (struct us_data*)hostptr->hostdata[0];
   if (!us) {
-    scsi_host_put(hostptr);
     return -ESRCH;
   }

   SPRINTF("   Host scsi%d: usb-storage\n", hostno);
-  scsi_host_put(hostptr);
   return length;
```

Observations

Code must be added and removed.

The context of the API usage can be affected as well.

Affected code may be scattered.

Another scsi driver: sym53c8xx.c

```
static int sym53c8xx_proc_info(
                               char *buffer, char **start, off_t offset,
                               int length, int hostno, int func) {
     struct Scsi_Host *host;
     struct host_data *host_data;
     ncb_p ncb = 0;
     int retv;

     printk("sym53c8xx_proc_info: hostno=%d, func=%d\n", hostno, func);
     host = scsi_host_hn_get(hostno);
     if (!host) return -EINVAL;
     host_data = (struct host_data *) host->hostdata;
     ncb = host_data->ncb;
     retv = -EINVAL;
     if (!ncb) goto out;
     if (func) {
       retv = ncr_user_command(ncb, buffer, length);
     } else {
       if (start) *start = buffer;
       retv = ncr_host_info(ncb, buffer, offset, length);
     }
out: scsi_host_put(host);
     return retv;
```

Another scsi driver: sym53c8xx.c

```
static int sym53c8xx_proc_info(struct Scsi_Host *host,
                               char *buffer, char **start, off_t offset,
                               int length, int hostno, int func) {

     struct host_data *host_data;
     ncb_p ncb = 0;
     int retv;

     printk("sym53c8xx_proc_info: hostno=%d, func=%d\n", hostno, func);


     host_data = (struct host_data *) host->hostdata;
     ncb = host_data->ncb;
     retv = -EINVAL;
     if (!ncb) goto out;
     if (func) {
       retv = ncr_user_command(ncb, buffer, length);
     } else {
       if (start) *start = buffer;
       retv = ncr_host_info(ncb, buffer, offset, length);
     }
out:
     return retv;
```

sym53c8xx.c patch file

```
@@ -1,14 +1,11 @@
-static int sym53c8xx_proc_info(
+static int sym53c8xx_proc_info(struct Scsi_Host *host,
                                char *buffer, char **start, off_t offset,
                                int length, int hostno, int func) {
-  struct Scsi_Host *host;
   struct host_data *host_data;
   ncb_p ncb = 0;
   int retv;

   printk("sym53c8xx_proc_info: hostno=%d, func=%d\n", hostno, func);
-  host = scsi_host_hn_get(hostno);
-  if (!host) return -EINVAL;
   host_data = (struct host_data *) host->hostdata;
   ncb = host_data->ncb;
   retv = -EINVAL;
@@ -20,6 +17,5 @@
       *start = buffer;
     retv = ncr_host_info(ncb, buffer, offset, length);
   }
-out: scsi_host_put(host);
-  return retv;
+out: return retv;
```

Observations

Code must be added and removed.

The context of the API usage can be affected as well.

Affected code may be scattered.

Need to be able to abstract over local variable names, device-specific computations, etc.

- `hostptr` vs. `host`, etc.

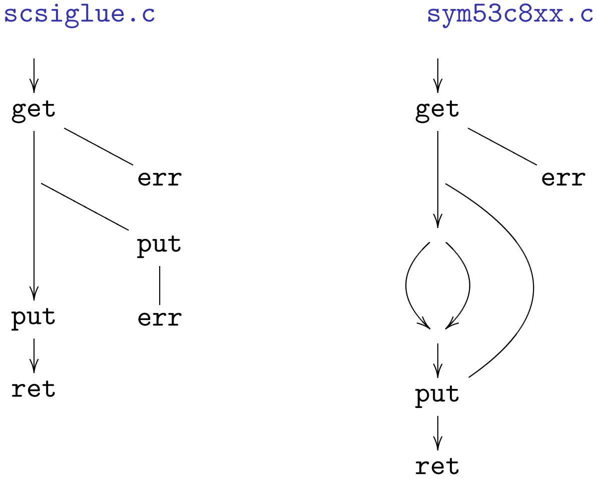
Need to be precise.

Comparing the two functions

```
static int usb_storage_proc_info (
 char *buffer,char **start,
 off_t offset,
 int length,int hostno,int inout) {
 struct us_data *us;
 struct Scsi_Host *hostptr;

 hostptr = scsi_host_hn_get(hostno);
 if (!hostptr) { return -ESRCH; }

 us=(struct us_data*)
       hostptr->hostdata[0];
 if (!us) {
   scsi_host_put(hostptr);
   return -ESRCH;
 }

 SPRINTF("...", hostno);
 scsi_host_put(hostptr);
 return length;
}
```

```
static int sym53c8xx_proc_info (
 char *buffer,char **start,
 off_t offset,
 int length,int hostno,int func) {
 struct Scsi_Host *host;
 struct host_data *host_data;
 ncb_p ncb = 0; int retv;

 printk("...", hostno, func);
 host = scsi_host_hn_get(hostno);
 if (!host) return -EINVAL;
 host_data =
  (struct host_data *) host->hostdata;
 ncb = host_data->ncb;
 retv = -EINVAL;
 if (!ncb) goto out;
 ...
out: scsi_host_put(host);
 return retv;
}
```

Comparing the control flow of the two functions



Each path does a get, then a put.

Observations

Code must be added and removed.

The context of the API usage can be affected as well.

Affected code may be scattered.

Need to be able to abstract over local variable names, device-specific computations, etc.

- `hostptr` vs. `host`, etc.

Need to be precise.

Need to describe control-flow paths, not abstract syntax trees.

Ideas

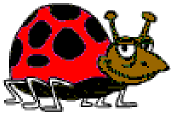
Follow the patch syntax!

- – for removed code.
- + for added code.
- context code.

Metavariables and "..." for irrelevant code fragments.

At the statement level, "..." represents a control-flow path, not an abstract syntax tree.

SmPL: a language for writing semantic patches



Creating a proc_info semantic patch

```
static int sym53c8xx_proc_info(

                               char *buffer, char **start, off_t offset,
                               int length, int hostno, int func) {
     struct Scsi_Host *host;
     struct host_data *host_data;
     ncb_p ncb = 0; int retv;

     printk("sym53c8xx_proc_info: hostno=%d, func=%d\n", hostno, func);
     host = scsi_host_hn_get(hostno);
     if (!host) return -EINVAL;
     host_data = (struct host_data *) host->hostdata;
     ncb = host_data->ncb;
     retv = -EINVAL;
     if (!ncb) goto out;
     if (func) { retv = ncr_user_command(ncb, buffer, length); }
     else { if (start) *start = buffer;
                retv = ncr_host_info(ncb, buffer, offset, length); }
out: scsi_host_put(host);
     return retv;
}
```

Creating a proc_info semantic patch

Drop the uninteresting parts

```
static int sym53c8xx_proc_info(

                               char *buffer, char **start, off_t offset,
                               int length, int hostno, int func) {
     struct Scsi_Host *host;
     ...



     host = scsi_host_hn_get(hostno);
     if (!host) return -EINVAL;
     ...








    scsi_host_put(host);
    ...
}
```

Creating a proc_info semantic patch

Indicate code to add and remove

```
static int sym53c8xx_proc_info(
+                              struct Scsi_Host *host,
                               char *buffer, char **start, off_t offset,
                               int length, int hostno, int func) {
-    struct Scsi_Host *host;
     ...




-    host = scsi_host_hn_get(hostno);
-    if (!host) return -EINVAL;
     ...








-    scsi_host_put(host);
     ...
}
```

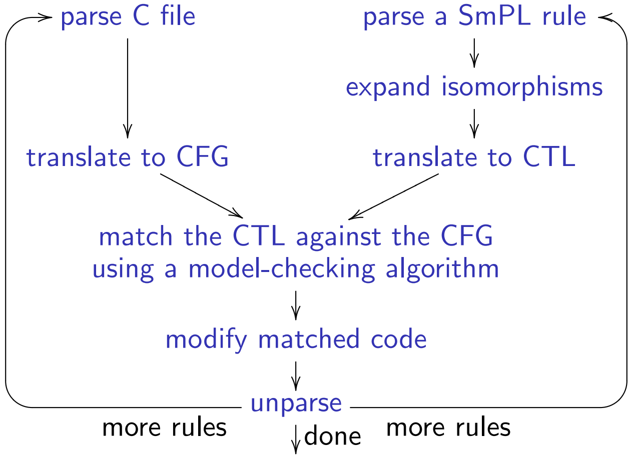
Creating a proc_info semantic patch

Abstract over variable names, arbitrary expressions, etc.

```
@@
identifier proc_info_fn;
identifier host, buffer, start, offset, length, hostno, func;
@@

static int proc_info_fn(
+                        struct Scsi_Host *host,
                         char *buffer, char **start, off_t offset,
                         int length, int hostno, int func) {
- struct Scsi_Host *host;
  ...
- host = scsi_host_hn_get(hostno);
- if (!host) return ...;
  ...
- scsi_host_put(host);
  ...
}
```

Function prototypes updated automatically.

Overview of the Coccinelle implementation



Implementation issues

Parse C file

- Maintain spacing, comments, preprocesser code.

Parse a SmPL rule

- Arbitrary interleaving of − and + code.
- Expansion according to isomorphisms:

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

Match and modify:

- A rule is matched against each function once (termination guaranteed).
- Transformation is only performed for a complete match.

The matching process

Ideas:

- Properties of paths are naturally expressed using temporal logic (CTL) [Lacey and de Moor].
- CTL has an efficient, easy-to-implement decision procedure: model checking.

```
@@ @@
  f();
  ...
- g();
+ h();
```

```
int main () {
  f();
  x();
  if (a < b) { g(); p(a); }
  else { p(b); g(); }
}
```

$$\mathtt{f()} \wedge \mathsf{AX}(\mathsf{A}\,[\neg(\mathtt{f()} \vee \mathtt{g()})\,\mathsf{U}\;\mathtt{g()}])$$

Adding metavariables

Introduce predicates, with finite domain

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

  f(E1);
  ...
- g(E1,E2);
+ h(E1,E2);
```

$$\begin{tikzcd}[column sep=small] & \text{1.int main()} \arrow[d] & \\ & \text{2.f(3);} \arrow[d] & \\ & \text{3.x();} \arrow[d] & \\ & \text{4.if (a < b)} \arrow[dl] \arrow[dr] & \\ \text{5.g(3,4);} \arrow[dr] & & \text{6.g(3,4);} \arrow[dl] \\ & \text{return;} & \end{tikzcd}$$

$$\texttt{f}(E1) \wedge \mathrm{AX}(\mathrm{A}[\neg(\texttt{f}(E1) \vee \texttt{g}(E1,E2))\,\mathrm{U}\,\texttt{g}(E1,E2)])$$

- $\texttt{f}(E1)$ matches at $(2, [E1 \mapsto 3])$
- $\texttt{g}(E1,E2)$ matches at $(5, [E1 \mapsto 3, E2 \mapsto 4])$, $(6, [E1 \mapsto 3, E2 \mapsto 4])$

These environments are compatible.

Negated bindings also allowed: constructive negation.

Finer-grained matching of metavariables

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

  f(E1);
  ...
- g(E1,E2);
+ h(E1,E2);
```

$$\begin{tikzcd}
 & \text{1.int main()} \arrow[d] & \\
 & \text{2.f(3);} \arrow[d] & \\
 & \text{3.x();} \arrow[d] & \\
 & \text{4.if (a < b)} \arrow[dl] \arrow[dr] & \\
\text{5.g(3,a);} \arrow[dr] & & \text{6.g(3,b);} \arrow[dl] \\
 & \text{return;} &
\end{tikzcd}$$

$$\texttt{f}(E1) \wedge AX(A[\neg(\texttt{f}(E1) \vee \texttt{g}(E1,E2))\,U\,\texttt{g}(E1,E2)])$$

- $\texttt{f}(E1)$ matches at $(2, [E1 \mapsto 3])$
- $\texttt{g}(E1,E2)$ matches at $(5, [E1 \mapsto 3, E2 \mapsto \texttt{a}])$, $(6, [E1 \mapsto 3, E2 \mapsto \texttt{b}])$

These environments are not compatible.

Solution: Add quantifiers:

$$\exists E1.\texttt{f}(E1) \wedge AX(A[\neg(\texttt{f}(E1) \vee \exists E2.\texttt{g}(E1,E2))\,U\,\exists E2.\texttt{g}(E1,E2)])$$

Collecting the match information

Traditionally, model checking says yes or no

- CTL model checking internally collects all states where a formula holds.

We need to know:

- Where to transform?
- With respect to what environment?

Solution:

- Collect witnesses for quantified metavariables.
- Introduce quantified metavariables for information we want to collect.

Matching with witnesses

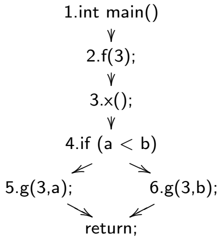
$$\exists E1.\mathtt{f}(E1) \wedge \mathsf{AX}(\mathsf{A}[\neg(\mathtt{f}(E1) \vee \exists E2.\mathtt{g}(E1,E2))\,\mathsf{U}\,\exists E2.\mathtt{g}(E1,E2)])$$

$$\begin{tikzcd}
& \text{1.int main()} \arrow[d] & \\
& \text{2.f(3);} \arrow[d] & \\
& \text{3.x();} \arrow[d] & \\
& \text{4.if } (a < b) \arrow[dl] \arrow[dr] & \\
\text{5.g(3,a);} \arrow[dr] & & \text{6.g(3,b);} \arrow[dl] \\
& \text{return;} &
\end{tikzcd}$$

- $\mathtt{f}(E1)$ matches at $(2, [E1 \mapsto 3], \emptyset)$
- $\mathtt{g}(E1,E2)$ matches at $(5, [E1 \mapsto 3, E2 \mapsto \mathtt{a}], \emptyset)$, $(6, [E1 \mapsto 3, E2 \mapsto \mathtt{b}], \emptyset)$
- $\exists E2.\mathtt{g}(E1,E2)$ matches at $(5, [E1 \mapsto 3], \{(5, [E2 \mapsto \mathtt{a}], \emptyset)\})$, $(6, [E1 \mapsto 3], \{(6, [E2 \mapsto \mathtt{b}], \emptyset)$

Collecting extra information

$\exists E1.\mathtt{f}(E1) \wedge AX(A[\neg(\mathtt{f}(E1) \vee \exists E2.\mathtt{g}(E1,E2))\, U\, \exists E2.\exists v.\mathtt{g}(E1,E2)^{v}])$



- $\mathtt{f}(E1)$ matches at $(2, [E1 \mapsto 3], \emptyset)$
- $\exists E2.\exists v.\mathtt{g}(E1,E2)^{v}$ matches at
  $(5,[E1 \mapsto 3],$
  $\{(5,[E2 \mapsto \mathtt{a}],$
  $\{(5, [v \mapsto \mathtt{g}(E1,E2)], \emptyset)\})),$
  $(6, \ldots, \ldots)$

Final answer:

$$(2, [\,], \{(2, [E1 \mapsto 3], \{(5, [E2 \mapsto \mathtt{a}], \{(5, [v \mapsto \mathtt{g}(E1,E2)], \emptyset)\}), (6, ...)\})\})$$

Interpretation:

- Transform at node 5 according to the transformation associated with $\mathtt{g}(E1,E2)$, where $E1$ is 3 and $E2$ is a.
- Similarly for node 6.

Benefits of our approach

CTL algorithm is easy to implement.

- Efficient enough for individual driver functions.

Flexible logic encoding:

- (Mostly) universal path quantification for transformation.
- Existential path quantification for searching.
- Either encoding can be extended to collect partial matches.

Current status

- SmPL compiler and CTL algorithm implemented.
- Semantic patches written for over 70 Linux collateral evolutions.
- Semantic patches applied to over 6000 relevant driver files, from recent Linux versions.
  - Often less than one second per relevant file.
- Some patches contributed to Linux.
- Soundness and completeness proofs well underway.

Conclusion

- Transformation of multiple program points related by control-flow relationships is interesting in practice.
- CTL with some extensions is a convenient target language for expressing such transformations.
- CTL model checking is efficient enough for pefrorming the matching required by such transformatiosn in practice.
- Perhaps our approach is not restricted to device drivers or to collateral evolutions, but we make no promises...

zzz

Opsummering og næste gang

- Collateral evolutions
- Coccinelle
- Næste gang: opsummering, mini-projekter (8., 13. og 15. maj)
- Ekstraforelæsning: 20. maj kl. 10–12
- Spørgetime: 17. juni kl. 13–15