SmPL: SimPLe SamPLEs to Update Device Drivers
or
Patch Reloaded

Yoann Padoleau
Ecole des Mines de Nantes
Julia L. Lawall
DIKU, University of Copenhagen
Gilles Muller
Ecole des Mines de Nantes

THE PROBLEM

Evolution in API of a
generic library

Lots of Collateral Evolutions
in clients of this library 😞

Collateral evolutions are mostly
done manually, because hard to script.
The program transformations require working
on a high level representation of the program
(syntax and semantic, as in a compiler),
Not just sad,
→ time consuming
(may involve 100 files, 1000 code sites)
→ error prone

Then, the modifications are transmitted to
other Linux programmers via patch files.

Legend:
- scsi get/put function calls to delete
- dependent code to delete
- code to add

OUR SOLUTION: A declarative easy-to-use transformation language to specify collateral evolutions.

Linux programmers exchange, read, and manipulate program modifications in terms of patches.
→ Our language is based around the idea and syntax of a patch, extending patches to Semantic Patches.

A single small Semantic Patch can modify hundreds of files, at thousands of code sites. 😁

Semantic Patch Language (SmPL) by example

```c
struct SET sbt;
local function proc_info_func;
  sbt.proc_info = &proc_info_func;

identifier hostptr, hostto, buffer, length, inout;

proc_info_func (s)
  + struct ScsiHost *hostptr,
  char *buffer, int length,
  - int hostto,
  - int inout) {
    ...
    - struct ScsiHost *hostptr;
    ...
    hostptr = scsi.get(hostto);
    ...
    if(!hostptr) { ... }
    ...
    scsi.put(hostptr);
    ...
  }
```

→ Automation
→ Documentation

1. looks like real code, looks like a real patch

A developer can construct a "smpl patch" by top-pasting existing driver code and then modifying and generalizing it to generate the semantic patch.

2. abstracts away differences in spacing, indentation, comments

3. abstracts away specific names given to variables and expresses constraints between code sites by declaring and using metavariabes

4. declares arbitrary intervening code sequences, including straight-line code and arbitrary branching, with the "..." operator

5. abstracts away other variations using isomorphisms (e.g. if(!hostptr) ? if(hostptr==NULL) )

Features of SmPL that make semantic patches Generic
→ to accommodate the many variations in device driver coding style.