Modular Verification

- Manually write precondition pr and postcondition po
- Prove automatically that for a "fucntion body"

$$pr \Longrightarrow WP(body, po)$$

- Or

$$\vdash WP(x : bool = pr; body, \neg x \vee po) \ where \ x \notin fv(body)$$

- Weakest precondition of function call
  - $C.pr$ and $C.po$ are the function pre and post conditions
  - x is the global variable (state) changed
  - y is the variable bound in Q representing the new value of x

$$WP(call\ C) = C.pr \vee \forall z.(C.po[z/y] \Rightarrow \phi[z/x])$$

Open Issues: Side effects

- there is not local variables
- all effects are global
- constraint the affected memory and registers

```
memOld = mem
pcOld = PC
spOld = SP
...
(PC=pcOld+4 & SP = spOld+4) &
(
  freevar != spOld =>
     memOld[freevar]=mem[freevar]
)
```

Open Issues: While conditions

- C condition is $sq <= x$
- Machine code condition is R_ZF or R_SF xor R_OF
- Enrich the loop invariant with

```
R_ZF or R_SF xor R_OF => (sq <= x)
&
~(R_ZF or R_SF xor R_OF) => (sq > x)
```

- Prove absence of overflow

Open Issues: While conditions

- C condition is $sq <= x$
- Machine code condition is R_ZF or R_SF xor R_OF
- Enrich the loop invariant with

```
R_ZF or R_SF xor R_OF => (sq <= x)
&
~(R_ZF or R_SF xor R_OF) => (sq > x)
```

- Prove absence of overflow

Open Issues: Unstructured control flow

- Bap manages simple non cyclic control flow (e.g. abs)
- Machine code does not have notion of functions
- Machine code does not have notion of while
- Formalize class of control flow that can be managed
- Formalize the algorithms that must be applied

Open Issues: Tools

- STP binary version infinitely loops (manually compiled version works)
- CVC3 never terminates
- BAP lifting of flag updates is non-deterministic (Debian 6 and Ubuntu 10.04 ok, Ubuntu 12.04 no). Partially solved with the option –always-vex
- BAP does not support ARM
  - arm_translate_ccall missing (3000 LOC)
  - arm_modify_flags missing (similar)
  - write a filter in ML (that is what is currently broken for x86)
- Formalize the algorithms that must be applied

Smaller Issues

- all operation exploits finite arithmetics (warning to theorem overflows)
- automatically apply the verification steps and transformations (Pre and Post condition Prover for Prosper $= p^4$ prototype)
- parse and transform the required structures
- provides a language tailored for our purposes