Formal Development of a Social Network Application Progress Report

Sorren Harvey, [2] Nestor Catano, [2] Camilo Rueda, [1] Timothy Wahls[3]

[1]Universidad Javeriana-Cali [2]Madeira University [3]Dickinson College

SecSy 2010

Motivation

$$
\begin{aligned}
&\forall pe.((pe \in dom(friendship)) \Rightarrow \\
&\quad \forall bf.((bf \in best\_friends[\{pe\}]) \Rightarrow \\
&\quad\quad \forall sf.((sf \in social\_friends[\{pe\}]) \Rightarrow \\
&\quad\quad\quad \forall rc.((rc \in rawcontent) \Rightarrow \\
&\quad\quad\quad\quad \forall ac.((ac \in OPS) \Rightarrow \\
&\quad\quad\quad\quad\quad ((pe = owner(rc) \wedge pe \mapsto rc \in content \\
&\quad\quad\quad\quad\quad\quad \wedge rc \mapsto ac \mapsto pe \in act \wedge bf \mapsto rc \in content \\
&\quad\quad\quad\quad\quad\quad \wedge sf \mapsto rc \in content \wedge rc \mapsto ac \mapsto sf \in act) \\
&\quad\quad\quad\quad\quad\quad \Rightarrow \\
&\quad\quad\quad\quad\quad\quad rc \mapsto ac \in act^{-1}[\{bf\}])))))) \\
&\wedge \\
&\forall pe.((pe \in dom(friendship)) \Rightarrow \\
&\quad (wallaccess[\{pe\}] \cap social\_friends[\{pe\}] \neq \varnothing \Rightarrow \\
&\quad\quad best\_friends[\{pe\}] \subseteq wallaccess[\{pe\}]))
\end{aligned}
$$

Motivation: SN proliferation

Social Networks

- Have become extremely popular
  E.g. Facebook, MySpace, LinkedIn, Hi5, Twitter, Sapo
- Each supporting millions of active users
- Used to
  - Publish media content
  - Share personal info
  - Make buisiness contacts

Motivation: SN privacy?

Social-networks and Media in general have replaced personal communication as communication force , but...

Information in social-networks is security and privacy-sensitive

Privacy: SN user behavior

- R. Gross and A. Acquisti (2006)
  - Analyzed the behaviour of 4,000 CMU students on a social-network catered to colleagues
  - Evaluated information students disclose and study how they use social-network site privacy settings

A minimal percentage of users change the highly permeable privacy preferences

SN Privacy

Existing Social Networks

- *Do not enforce privacy of media content*
- *They have conflicting goals*
  - E.g. Expanding the network vs. exposing users content

Goal

Developing a SN with parental control facilities, that is

Secure, extensible and *privacy-enforcing*

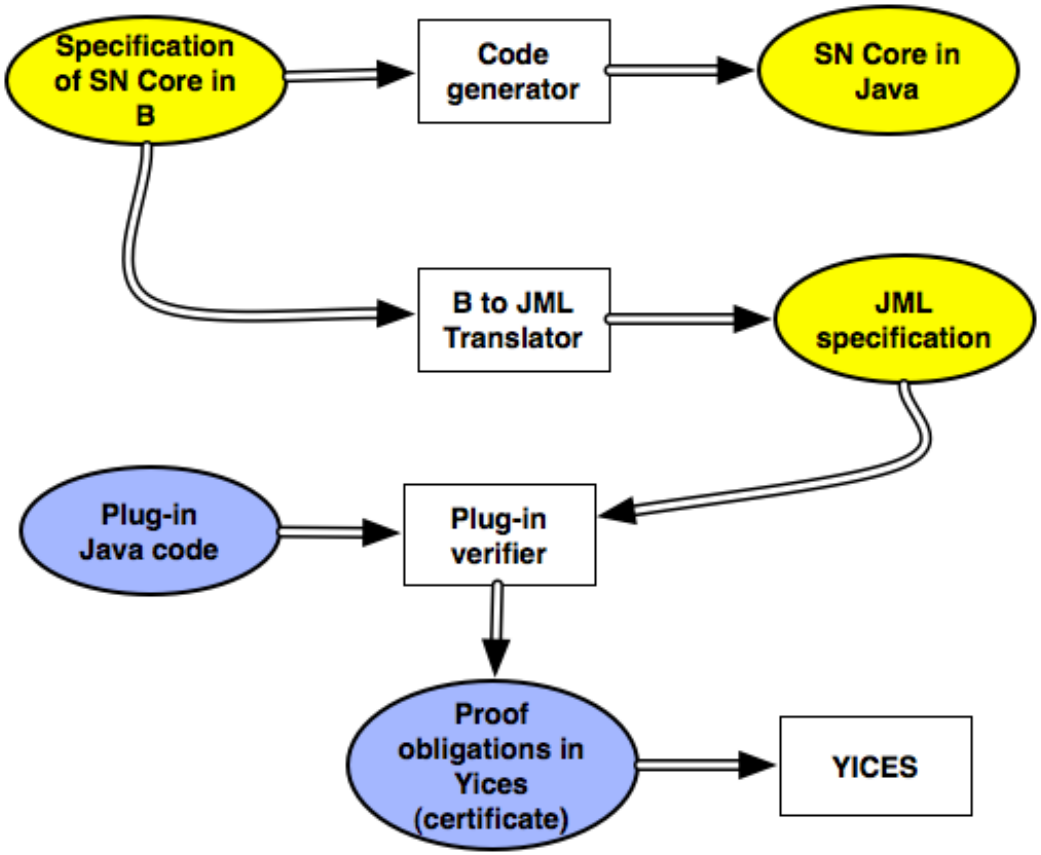
in such a way that those properties

can be formally proved

Outline

Outline

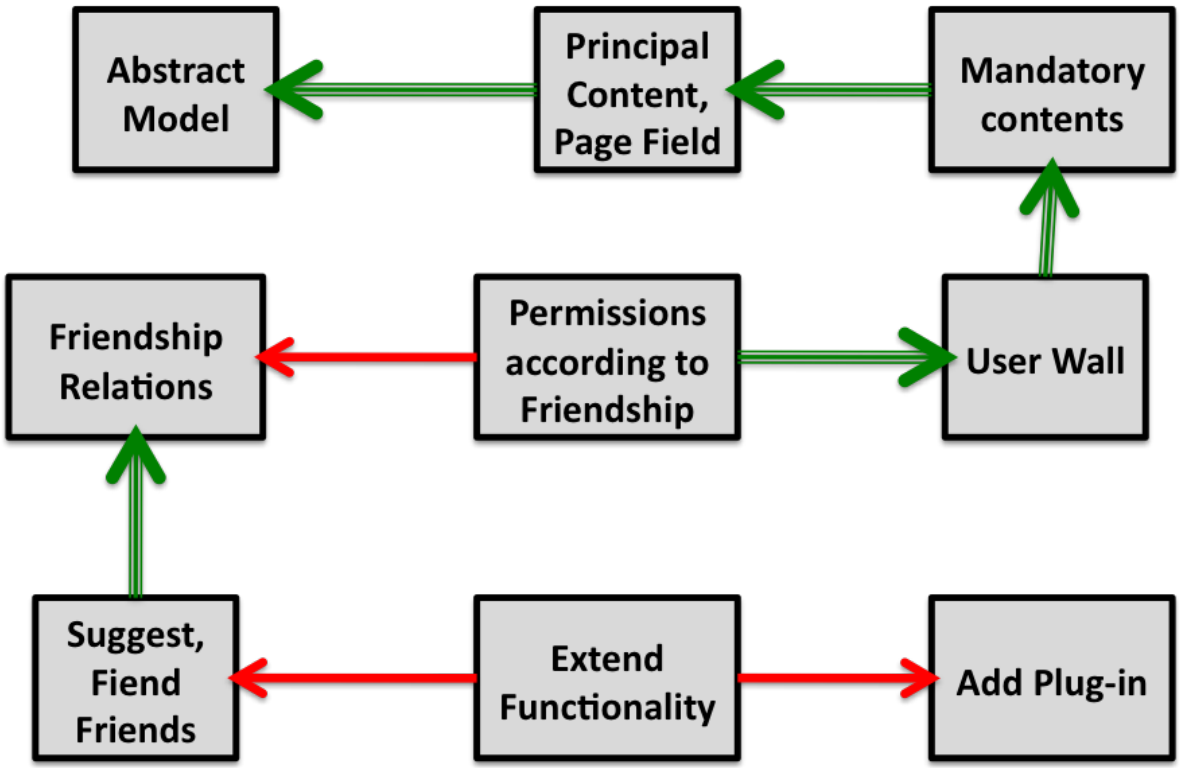
Architecture



Social network Core

- Write privacy and security social network policies as an initial predicate calculus-based abstract specification
- Refine the initial abstract specification and obtain a social network core application that adheres to stipulated policies
- Privacy is modeled as access permissions on content

The core in B


Abstract Model
Principal Content, Page Field
Mandatory contents
Friendship Relations
Permissions according to Friendship
User Wall
Suggest, Fiend Friends
Extend Functionality
Add Plug-in

Abstraction: state

Some defined types, i.e. sets: PERSON, RAWCONTENT, OPS
And some basic variables:

$$person \subseteq PERSON$$
$$rawcontent \subseteq RAWCONTENT$$
$$content \in person \leftrightarrow rawcontent$$
$$visible \subseteq content$$
$$act \in (rawcontent \times OPS) \leftrightarrow person$$

permissions

Abstraction: operations

- Create content, make it visible, edit, comment, hide,...
- Transmit some content rc from the owner ow to some person pe

$$
\begin{array}{l}
transmit\_rc(rc, ow, pe) = \\
\quad pre\ rc \in rawcontent \wedge pe \in person \wedge \\
\qquad ow = owner(rc) \wedge ow \neq pe \wedge \\
\qquad\quad (pe, , rc) \notin content \wedge pe \notin act[\{(rc, , view)\}] \\
\quad then \\
\quad\ content := content \cup \{(pe, , rc)\} \parallel \\
\quad\ act := act \cup \{((rc, , view), , pe)\} \\
end
\end{array}
$$

Abstraction: invariants

The owner of some data has all permissions for it:

$$\forall rc \in rawcontent \Rightarrow \\ \forall op \in OPS \Rightarrow \ ((rc, , op), , owner(rc)) \in act$$

If you can see some content, then you have "view" permission:

$$\forall (rc, pe).(rc \in rawcontent \wedge pe \in person) \\ \Rightarrow \\ ((pe, , rc) \in visible) \Rightarrow ((rc, , view, , pe) \in act))$$

Modeling friendship relations

Content is partitioned by three relations

$$friendship \in friend \leftrightarrow friend$$
$$best\_friends \subseteq friendship$$
$$social\_friends \subseteq friendship$$
$$acquaintances \subseteq friendship$$
$$best\_friends \cap social\_friends = \varnothing$$
$$etc.$$

Your best friends are not your social friends

Enforcing privacy

pe: the owner of some content rc,
bf: a best friend of pe,
sf: a social friend of pe,
ac: a permission sf has over rc

$$((rc, , ac), , sf) \in act \Rightarrow ((rc, , ac), , bf) \in act$$

Your best friends have always at least
the permissions you have given to your social friends

The whole B specification

- 6 machines
- About 450 Proof Obligations (all discharged)
- Some interactive proofs were rather ellaborate

Drawback

How to express the property:

if you have now permission op for data d then
(1) the owner gave d to you with the op permission previously
(2) the owner did not take the op permission away later

Solution (may be):
implement the core in the ProB model-checker
express the property in ProB's temporal logic

B to JML translator: Why?

- the notation translated into may permit the use of tools that do not support the notation translated from.
- the notation translated from may not directly support implementation in the desired programming language.
- the user may simply be more proficient and comfortable with the notation translated into.
- One would like the same specification languages for Plug-in and Core

B to JML translator: JML

JML is a way to annotate java programs with specifications

- Specifications are of the form:
  precondition: requires P
  poscondition: ensures Q
- there are also class invariants (with so-called ghost variables)

JML tools verify an annotated java program

B to JML

B Model	JML Model
Machine	Class
Machine Refinement	Sub-class
PRE	requires
Operation Body	ensures + \old
INVARIANT	invariant
Operation	JML Method specification
ANY x	\forall x

B to JML rules

jml (PRE P THEN S END) =
  public normal_behavior; requires $jml(P)$:
    assignable $mod(S)$; ensures $jml(S)$;
    also public exceptional_behavior; requires !$jml(P)$;
      assignable \nothing;
      signals (java.lang.Exception e);

What S, above, could be:

$jml($ $x := E \parallel y := F$ $) =$
  x == \old(jml(E)) && y == \old(jml(F))

B to JML rules (2)

jml (ANY x WHERE P THEN S END) =
∃ *type_of*(x) x; $\old$ (*jml*(P))&& *jml*(S))

Example:

jml (ANY x WHERE $x \in PERSON \wedge x \notin person$
THEN $person := person \cup \{p\}$ END) =
∃ PERSON x; !$\old$ (person.has(x))
&& person == $\old$ (person.union(singleton(x)))

Specifications in JML: translating a B operation

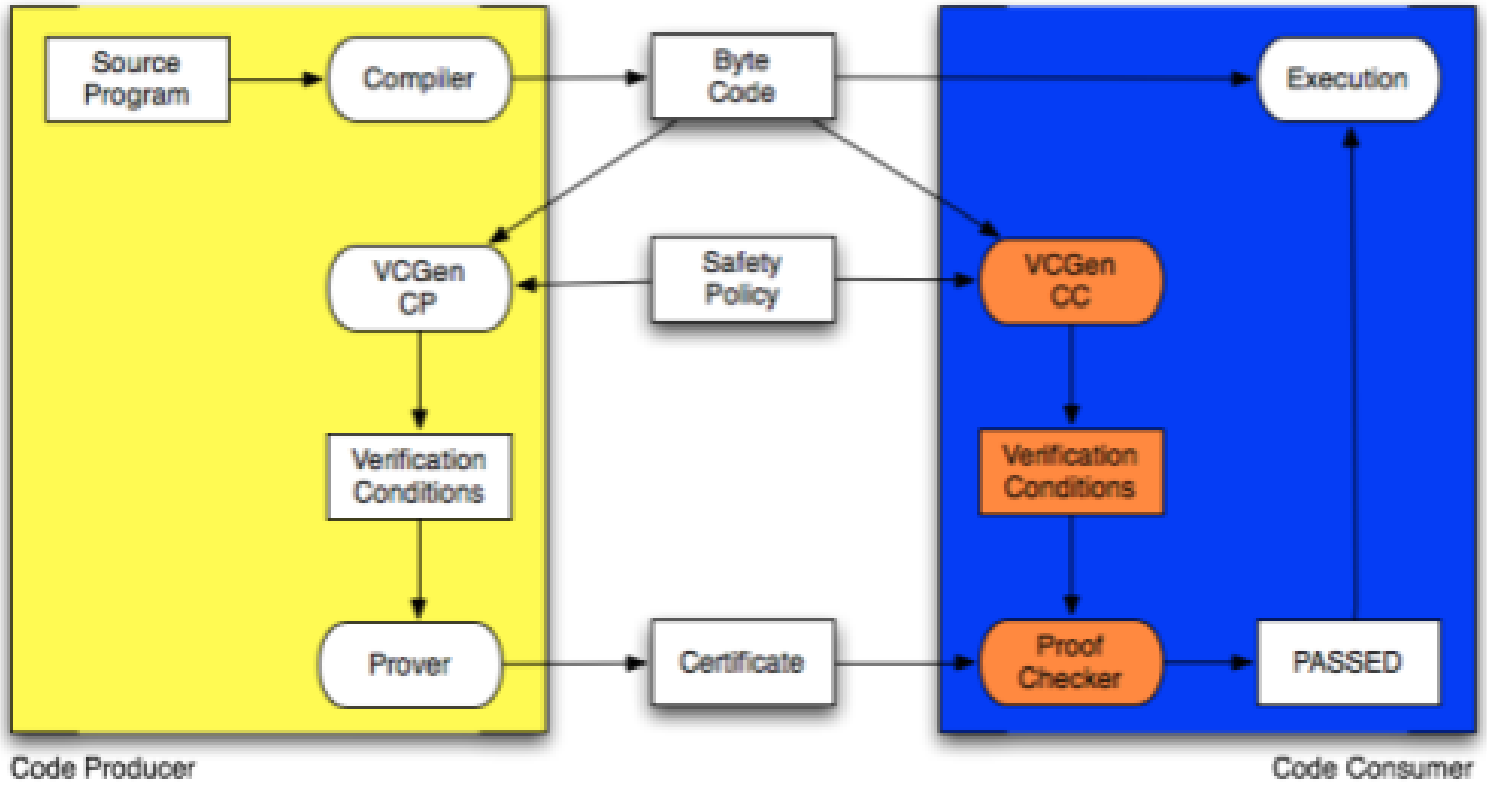
```
public abstract JMLEqualsSequence< Object >
        transmit_rc(Integer rc, Integer ow, Integer pe);
 /*@ requires RAWCONTENT.has(rc) && rawcontent.has(rc) &&
PERSON.has(pe) && person.has(pe)
 && content.has(pe, rc) && !visible.has(pe, rc)
 && act.has(new JMLEqualsEqualsPair< Integer, OPS >
                    (rc, OPS.view), pe);
 assignable visible;
 ensures visible.equals(
   \old(visible.union(JMLEqualsToEqualsRelation.singleton(pe, rc))))
 && \result.int_size() == 0;
```

Outline

Extending the Core

- Plug-ins implementing functionalities
- Social Network Plug-in Validator
  - Proof Carrying Code (PCC), Necula, G.-C.
  - Plug-in consists of $java$ code implementing the functionality + proof of adherence to the B model of social-networks

PCC architecture



Extension adherence to policies

- Non-bypassable: the security functions cannot be circumvented
- Tamper-proof: subversive code cannot alter the function of the security functions by exhausting resources or overrunning buffers.

VC generator

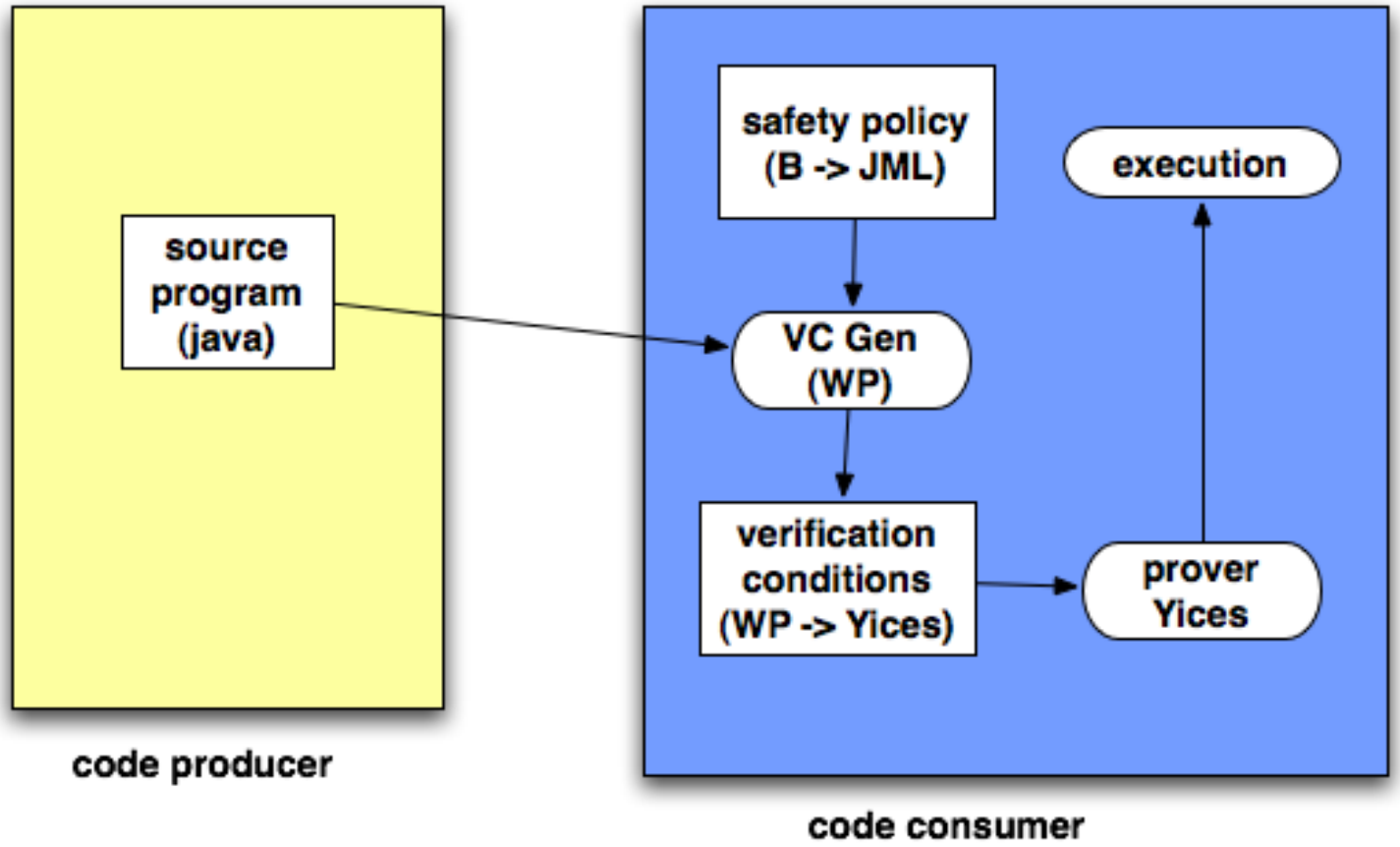
- As VC we use Dijkstra's weakest precondition calculus

> WP(S,Q): weakest condition that must hold before S
> to ensure that Q holds after executing S

- We devise rules to give WP semantics to a subset of $java$
- and translate WP rules to Yices formulae

Given a plug-in S : $\begin{cases} \textit{(1) compute } P = \textit{WP}(S, true) \\ \textit{(2) use Yices to verify } P \end{cases}$

Plugin verification: architecture



WP $java$ semantics

WP(S,$\langle Q, R, L \rangle$)

S	a $java$ statement
Q	a postcondition
R	exception-handling relation
L	a set of modified identifiers

- Relation R associates a WP with each catched $java$ exception
- L keeps track of assignable variables

WP *java* semantics: the annotation of a method

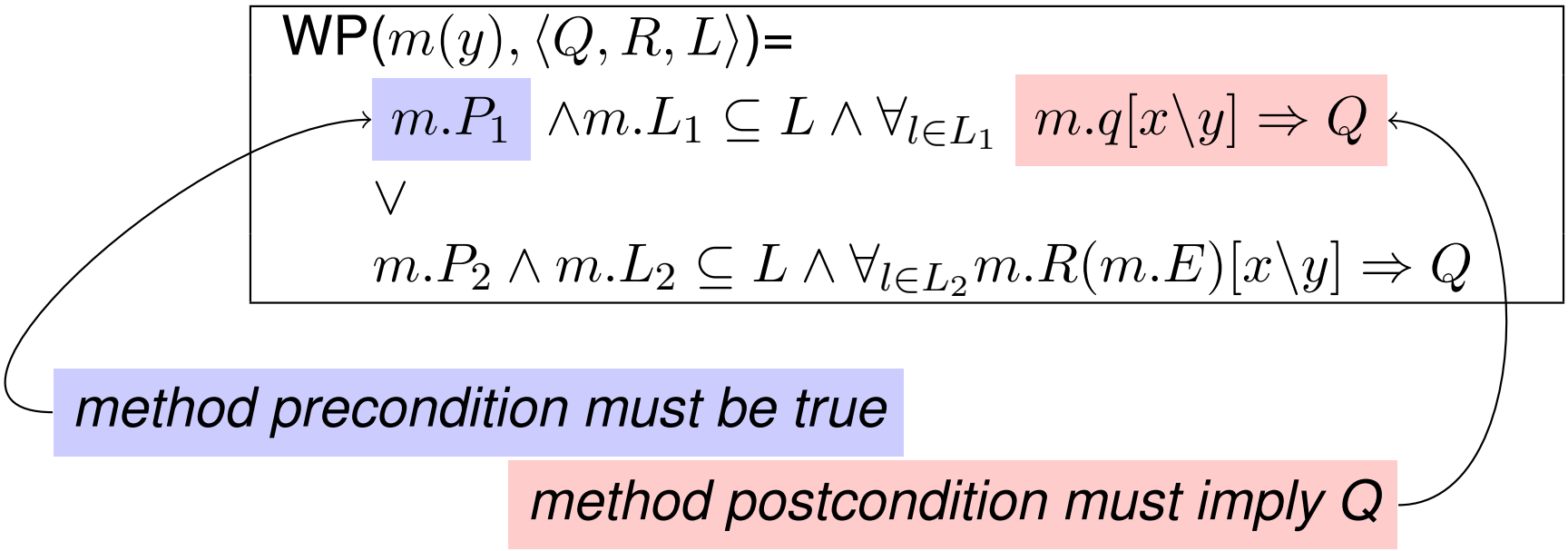
```
/@ public normal_behavior;
 @requires P1;                    precondition
 @assignable L1;                  modifed variables
 @ensures Q;                      postcondition
 @also
 @public exceptional_behavior;
 @requiresP2;                     exception condition
 @assignable L2;                  modified when exception
 @signals R;                      the exception
@/
public T m(x) throws E { block }
```

WP $java$ semantics: some rules

WP$(t, \langle Q, R, L\rangle)$=$Q[result\backslash t]$	terms
WP$(x = e, \langle Q, R, L\rangle)$=WP$(e, \langle Q[x\backslash result], R, L\rangle)$	assignment
WP$(S_1; S_2, \langle Q, R, L\rangle)$=WP$(S_1, \langle$ WP$(S_2, \langle Q, R, L\rangle), R, L\rangle)$	sequence
WP$(\mathit{throw}\ e, \langle Q, R, L\rangle)$=WP$(e, \langle R(e.type), R, L\rangle)$	raise exc
WP$(\mathit{try}\ \{S_1\}\ \mathit{catch}\ e\ \{S_2\}, \langle Q, R, L\rangle)$=	
WP$(S_1, \langle Q,$ *R.add(e,WP*$(S_2, \langle Q, R, L\rangle)$),L$\rangle)$	catch exc

add (exception,predicate) pair

WP $java$ semantics: method call rule



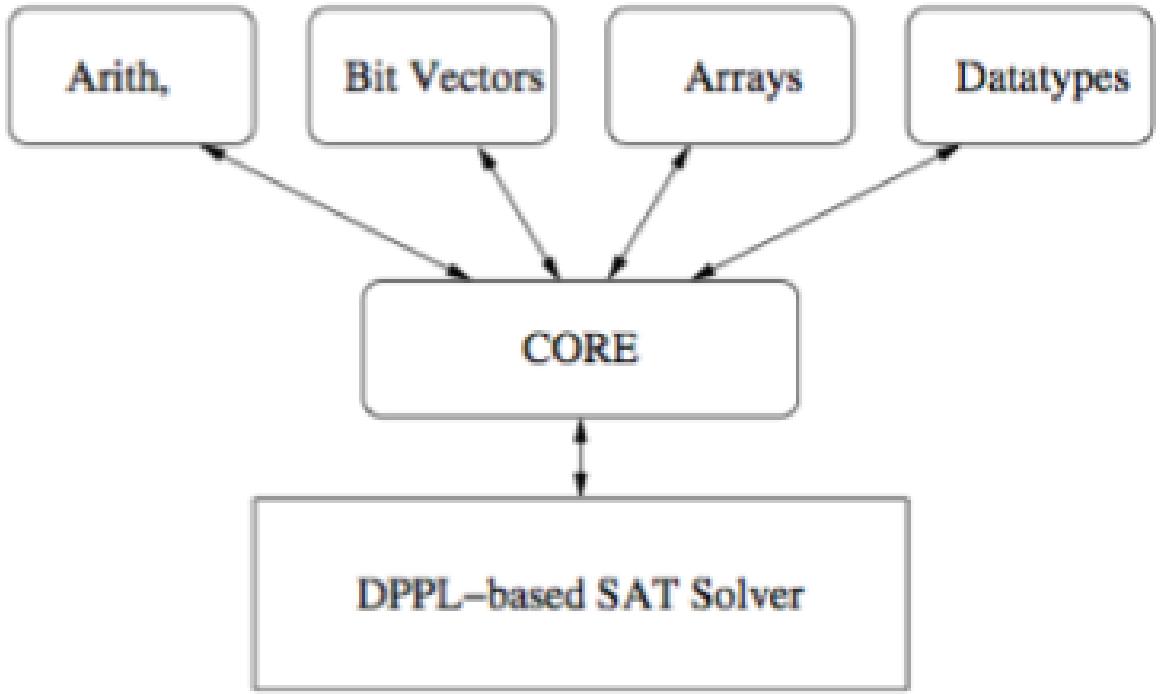
VC generator: plug-in example

```
/*@ PRECONDITION true; @ POSTCONDITION Q; @*/
public void install_plugin(SocialNetwork sn) {
    integer ow1 = sn.create_content(); // I1
    integer rc1 = sn.create_rc(); // I2
    sn.upload_rc(ow1,rc1) // I3
    integer pe1 = sn.create_content(); // I4
    sn.transmit_rc(rc1,ow1,pe1);  } // I5
```

$$VC = (true \Rightarrow WP(I1; I2; I3; I4, Q))$$
$$\equiv$$
$$WP(I1, WP(I2, WP(I3, WP(I4, WP(I5, Q)))))$$

Outline

Yices SAT solver



VC generator in Yices: representation

- Sets (and relations) are represented as bit-vectors
- B operations on sets are translated to bit-vector ops
- challenge: do the above efficiently (i.e. avoid recursion!)

$r1 \subseteq r2$
(*define* jmlrel-is-subset::(→ jmlrel jmlrel bool) (*lambda* (r1::jmlrel r2::jmlrel) (= r1 (*bv-and* r1 r2))))

How to implement $S \lhd r$ *??*

VC generator in Yices: example

WP(I1, WP(I2, WP(I3, WP(I4, WP(I5,Q))))) =

```
(define Q5
  (let ( (r5::bool (translation-pre (transmit_rc sn rc1 ow1 pe1)))
         (t5::bool (translation-post (transmit_rc sn rc1 ow1 pe1))) )
    (and r5 (⇒ t5 Q) ) ))
```

WP(I1, WP(I2, WP(I3, WP(I4, Q5))))

VC proof in Yices

```
(assert
 (and
    (= person4 person-prestate5)
    (= rawcontent4 rawcontent-prestate5)
    (= content4 content-prestate5)
    ...
    Q5 Q4 Q3 Q2 Q1
    ))
 (check )
```

Outline

Conclusions

Help! we need somebody,

not just anybody

We need students!!!