Deductive Program Verification with Why3

Why3 team:

François Bobot, Martin Clochard Jean-Christophe Filliâtre,
Léon Gondelman, Claude Marché,
Guillaume Melquiond, Andrei Paskevich, Mário Pereira

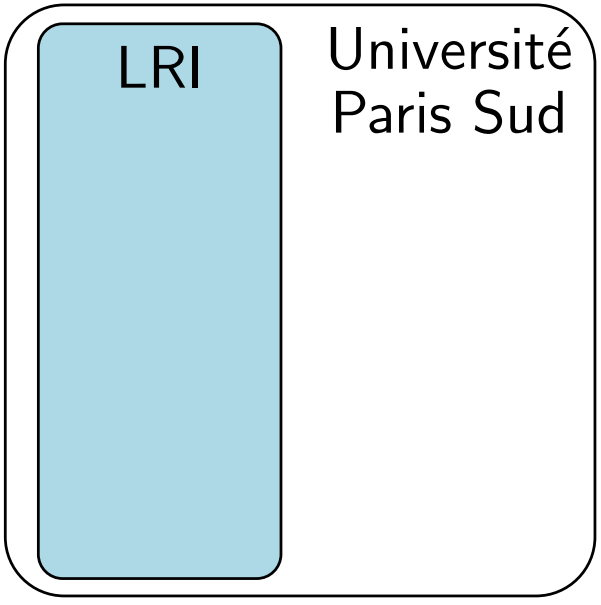
CNRS, Université Paris Sud, Inria

Inria Junior Seminar
April 19, 2016

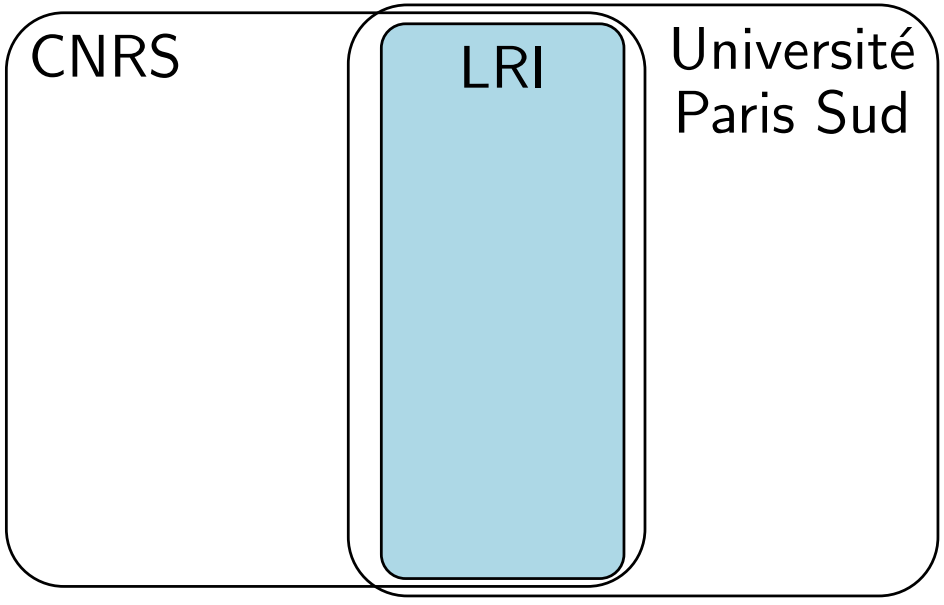
the VALS team — http://vals.lri.fr/

UNIVERSITÉ PARIS SUD
université PARIS-SACLAY

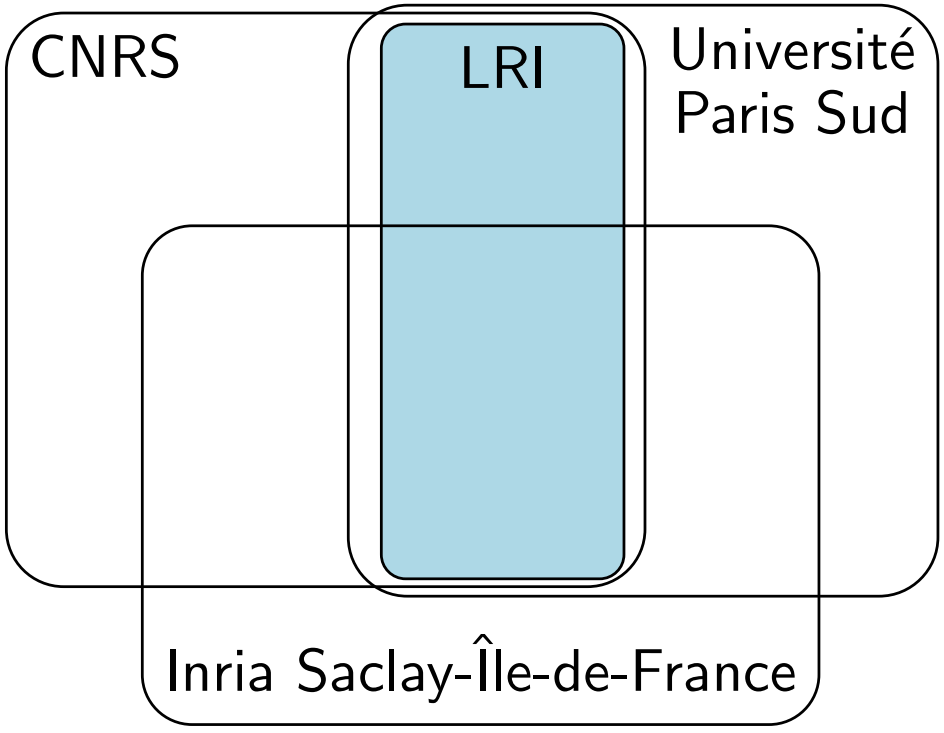
the VALS team — http://vals.lri.fr/



the VALS team — http://vals.lri.fr/



the VALS team — http://vals.lri.fr/



UNIVERSITÉ PARIS SUD
université PARIS-SACLAY

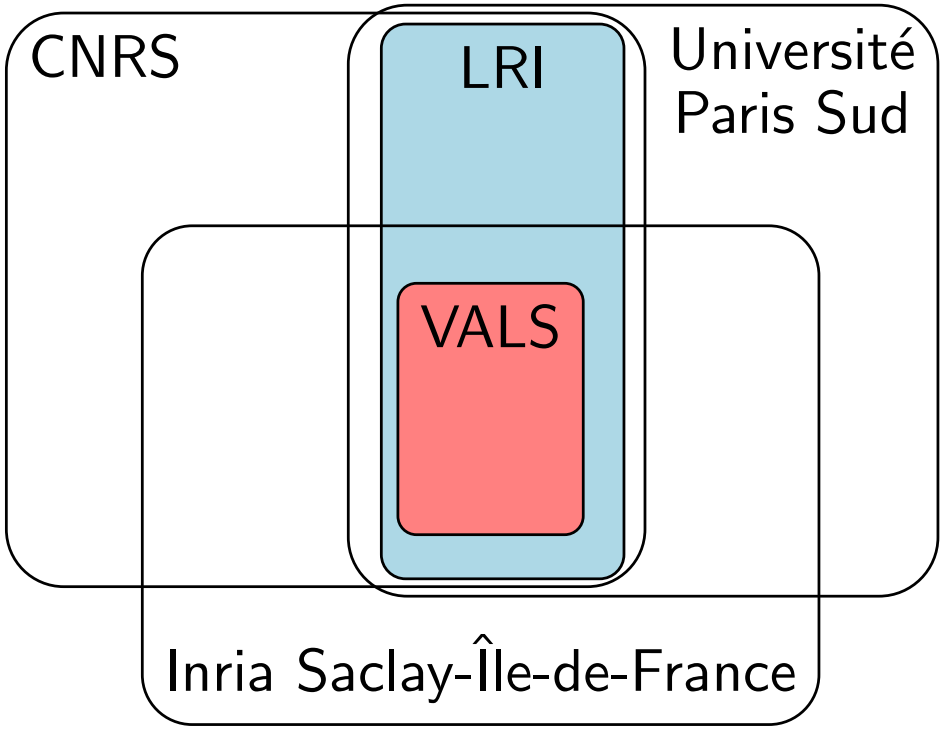
LABORATOIRE DE RECHERCHE EN INFORMATIQUE

cnrs
dépasser les frontières

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA
centre de recherche SACLAY - ÎLE-DE-FRANCE

the VALS team — http://vals.lri.fr/



UNIVERSITÉ PARIS SUD
université PARIS-SACLAY

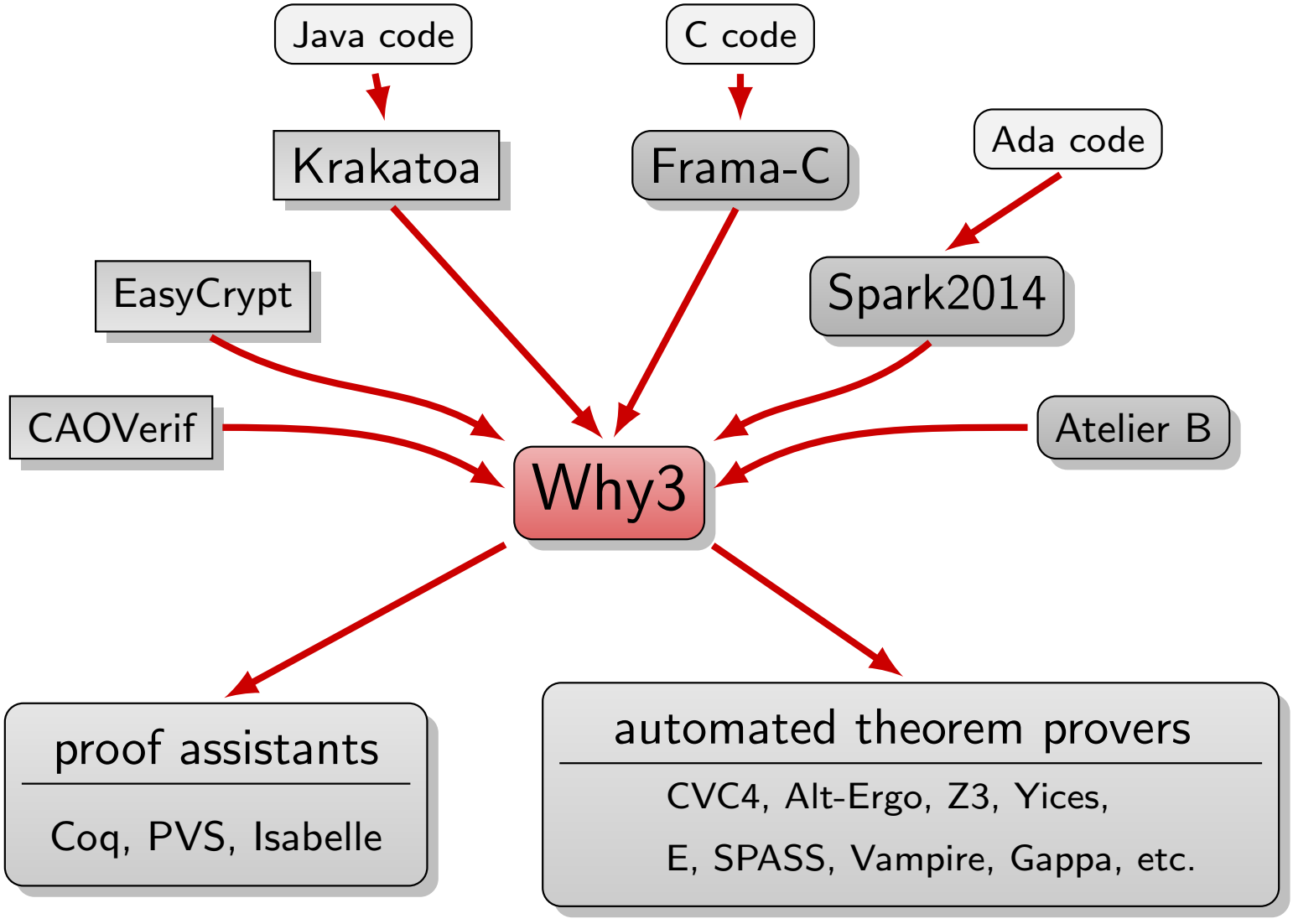
LABORATOIRE DE RECHERCHE EN INFORMATIQUE

cnrs
dépasser les frontières

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche SACLAY - ÎLE-DE-FRANCE

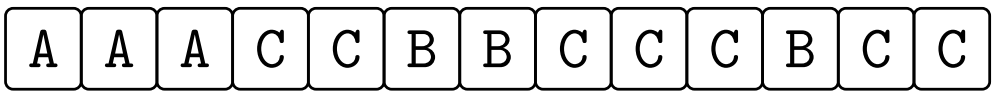
Deductive program verification is the art of turning the correctness of a program into a mathematical statement and then proving it.

an intermediate language



a taste of program verification

given a multiset of N votes



determine the majority, if any

an elegant solution

due to Boyer & Moore (1980)
linear time
constant extra space

MJRTY—A Fast Majority Vote Algorithm[1]

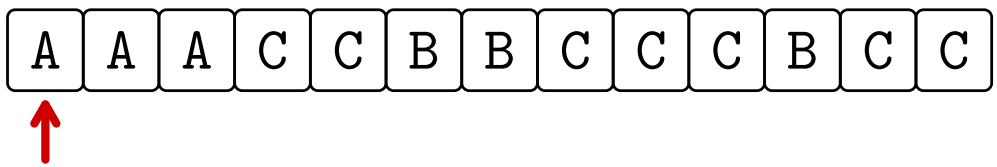
Robert S. Boyer and J Strother Moore

Computer Sciences Department
University of Texas at Austin
and
Computational Logic, Inc.
1717 West Sixth Street, Suite 290
Austin, Texas

Abstract

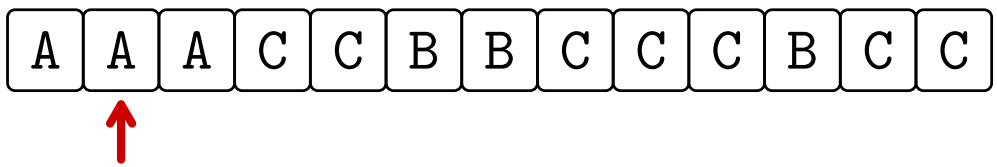
A new algorithm is presented for determining which, if any, of an arbitrary number of candidates has received a majority of the votes cast in an election.

Mjrty (Boyer & Moore)



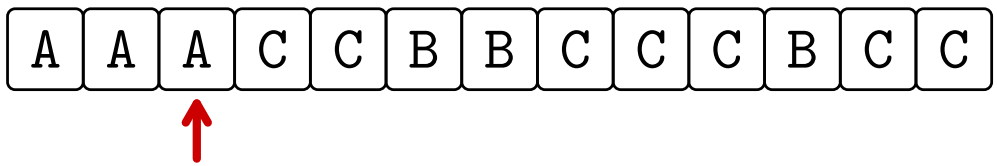

```
cand = A
k    = 1
```

Mjrty (Boyer & Moore)



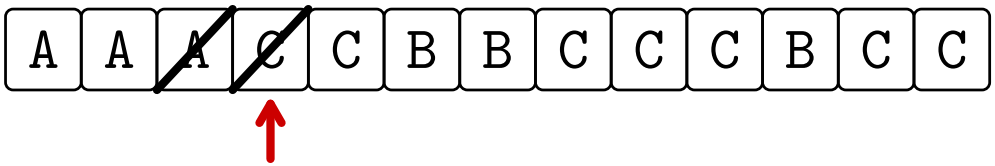

```
cand = A
k    = 2
```

Mjrty (Boyer & Moore)



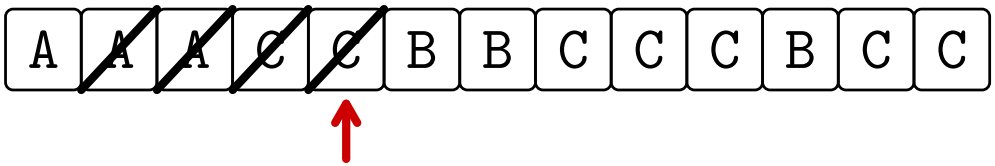

```
cand = A
k    = 3
```

Mjrty (Boyer & Moore)



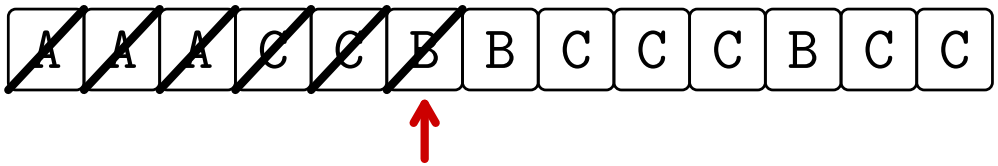

```
cand = A
k    = 2
```

Mjrty (Boyer & Moore)



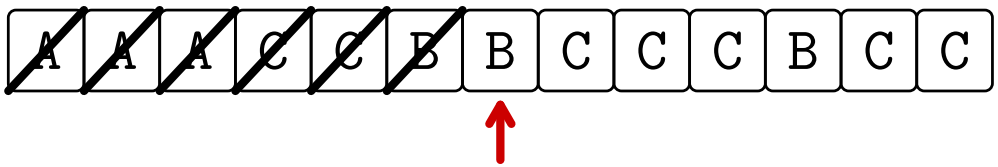

```
cand = A
k    = 1
```

Mjrty (Boyer & Moore)



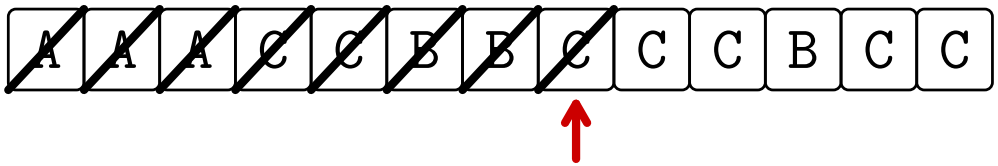

```
cand = A
k    = 0
```

Mjrty (Boyer & Moore)



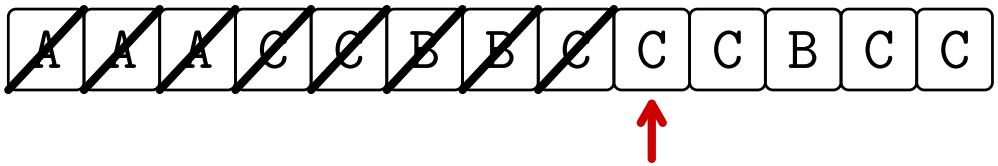

```
cand = B
k    = 1
```

Mjrty (Boyer & Moore)



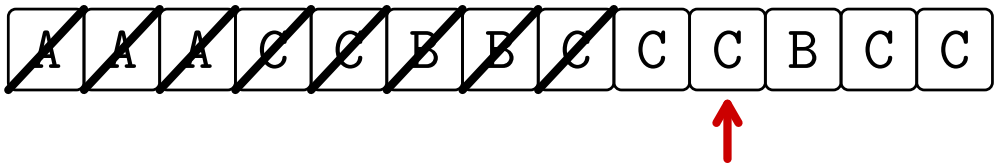

```
cand = B
k    = 0
```

Mjrty (Boyer & Moore)



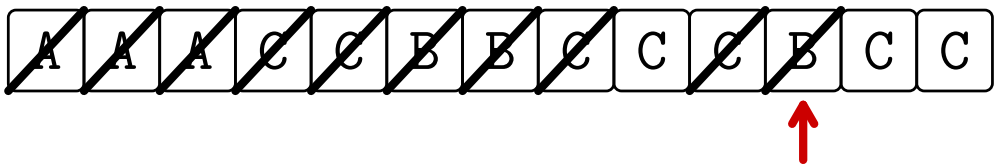

```
cand = C
k    = 1
```

Mjrty (Boyer & Moore)



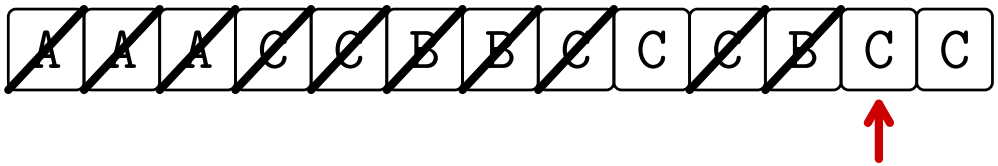

```
cand = C
k    = 2
```

Mjrty (Boyer & Moore)



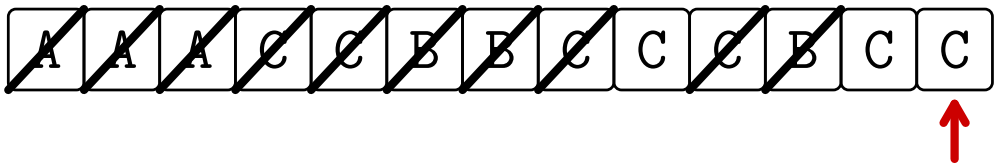

```
cand = C
k    = 1
```

Mjrty (Boyer & Moore)



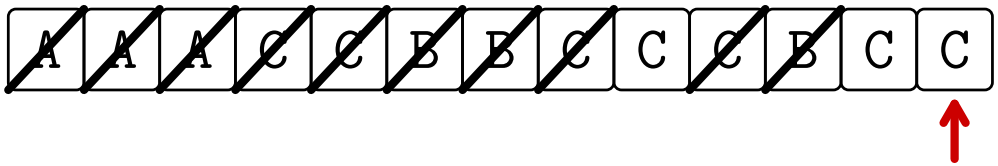

```
cand = C
k    = 2
```

Mjrty (Boyer & Moore)




```
cand = C
k    = 3
```

Mjrty (Boyer & Moore)



`cand = C`

`k = 3`

then we check if `C` indeed has majority, with a second pass
(in that case, it has: $7 > 13/2$)

Fortran code

```
      SUBROUTINE MJRTY(A, N, BOOLE, CAND)
      INTEGER N
      INTEGER A
      LOGICAL BOOLE
      INTEGER CAND
      INTEGER I
      INTEGER K
      DIMENSION A(N)
      K = 0
C     THE FOLLOWING DO IMPLEMENTS THE PAIRING PHASE. CAND IS
C     THE CURRENTLY LEADING CANDIDATE AND K IS THE NUMBER OF
C     UNPAIRED VOTES FOR CAND.
      DO 100 I = 1, N
      IF ((K .EQ. 0)) GOTO 50
      IF ((CAND .EQ. A(I))) GOTO 75
      K = (K - 1)
      GOTO 100
50    CAND = A(I)
      K = 1
      GOTO 100
75    K = (K + 1)
100   CONTINUE
      IF ((K .EQ. 0)) GOTO 300
      BOOLE = .TRUE.
      IF ((K .GT. (N / 2))) RETURN
C     WE NOW ENTER THE COUNTING PHASE. BOOLE IS SET TO TRUE
C     IN ANTICIPATION OF FINDING CAND IN THE MAJORITY. K IS
C     USED AS THE RUNNING TALLY FOR CAND. WE EXIT AS SOON
C     AS K EXCEEDS N/2.
      K = 0
      DO 200 I = 1, N
      IF ((CAND .NE. A(I))) GOTO 200
      K = (K + 1)
      IF ((K .GT. (N / 2))) RETURN
200   CONTINUE
300   BOOLE = .FALSE.
      RETURN
      END
```

Why3 code

```
exception Not_found
exception Found

let mjrty (a: array candidate) : candidate =
  requires { 1 ≤ length a }
  ensures  { 2 * numeq a result 0 (length a) > length a }
  raises   { Not_found → forall c: candidate.
               2 * numeq a c 0 (length a) ≤ length a }
  ...
  for i = 0 to n-1 do
    invariant { 0 ≤ !k ≤ numeq a !cand 0 i }
    invariant { 2 * (numeq a !cand 0 i - !k) ≤ i - !k }
    invariant { forall c: candidate. c ≠ !cand →
                  2 * numeq a c 0 i ≤ i - !k }
    ...
```

demo

what is going on in the Why3 world?

type system formalization (Léon Gondelman)

- ghost code
- regions
- refinement

proving transformations are sound (Martin Clochard)

bit vectors for all (Clément Fumex and Claude Marché)

effectful higher-order programming (it's me!)

effectful higher-order code in Why3

big dream:

- verify an OCaml library
- higher-order code with effects

first steps: higher-order iterators to first-order counterparts

A Modular Way to Reason About Iteration
J.-C. Filliâtre and M. Pereira, NFM'16
`https://hal.inria.fr/hal-01281759`

what next: CPS-programs

questions ?

---

http://why3.lri.fr

http://toccata.lri.fr/gallery/why3.en.html