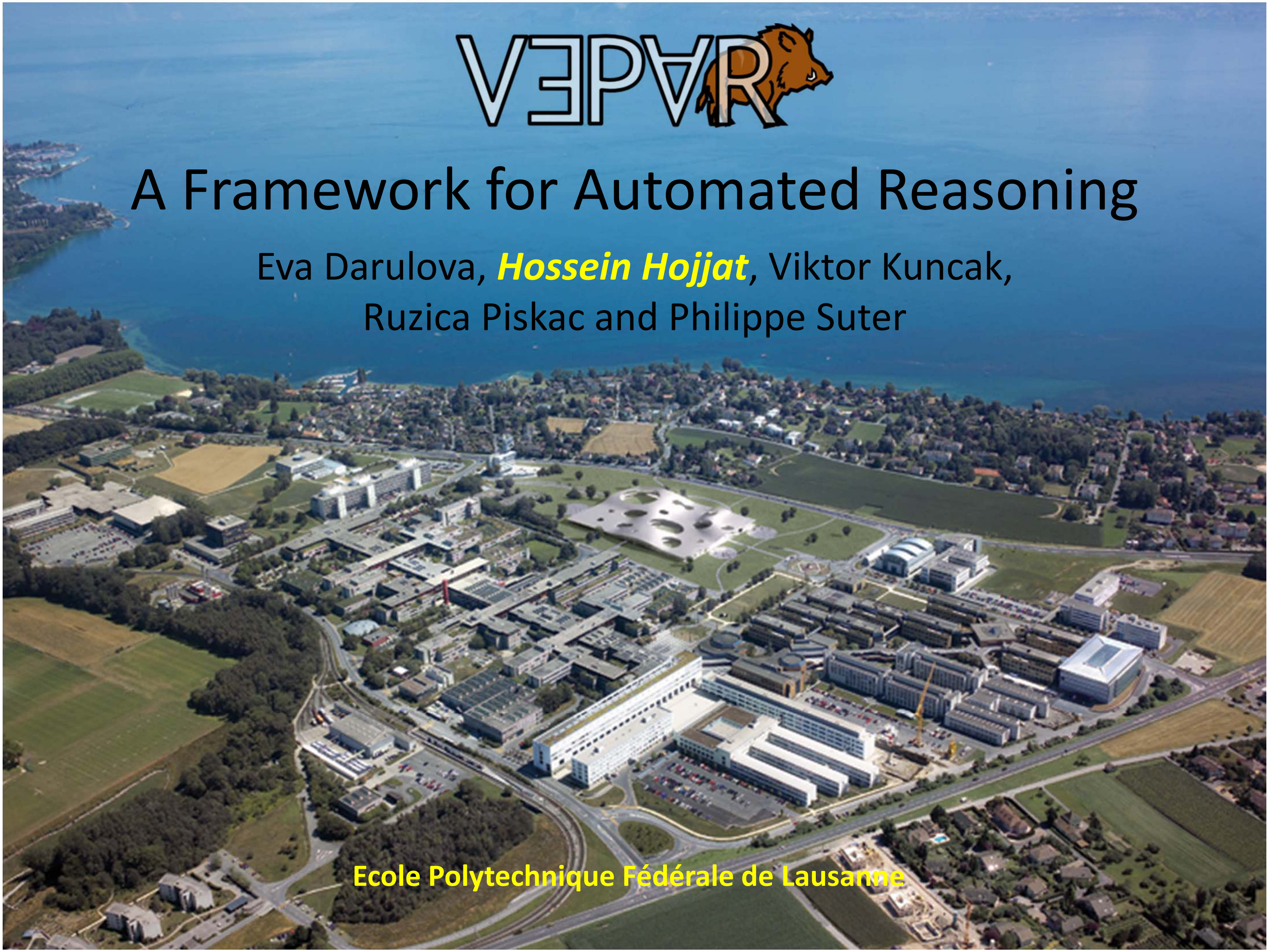
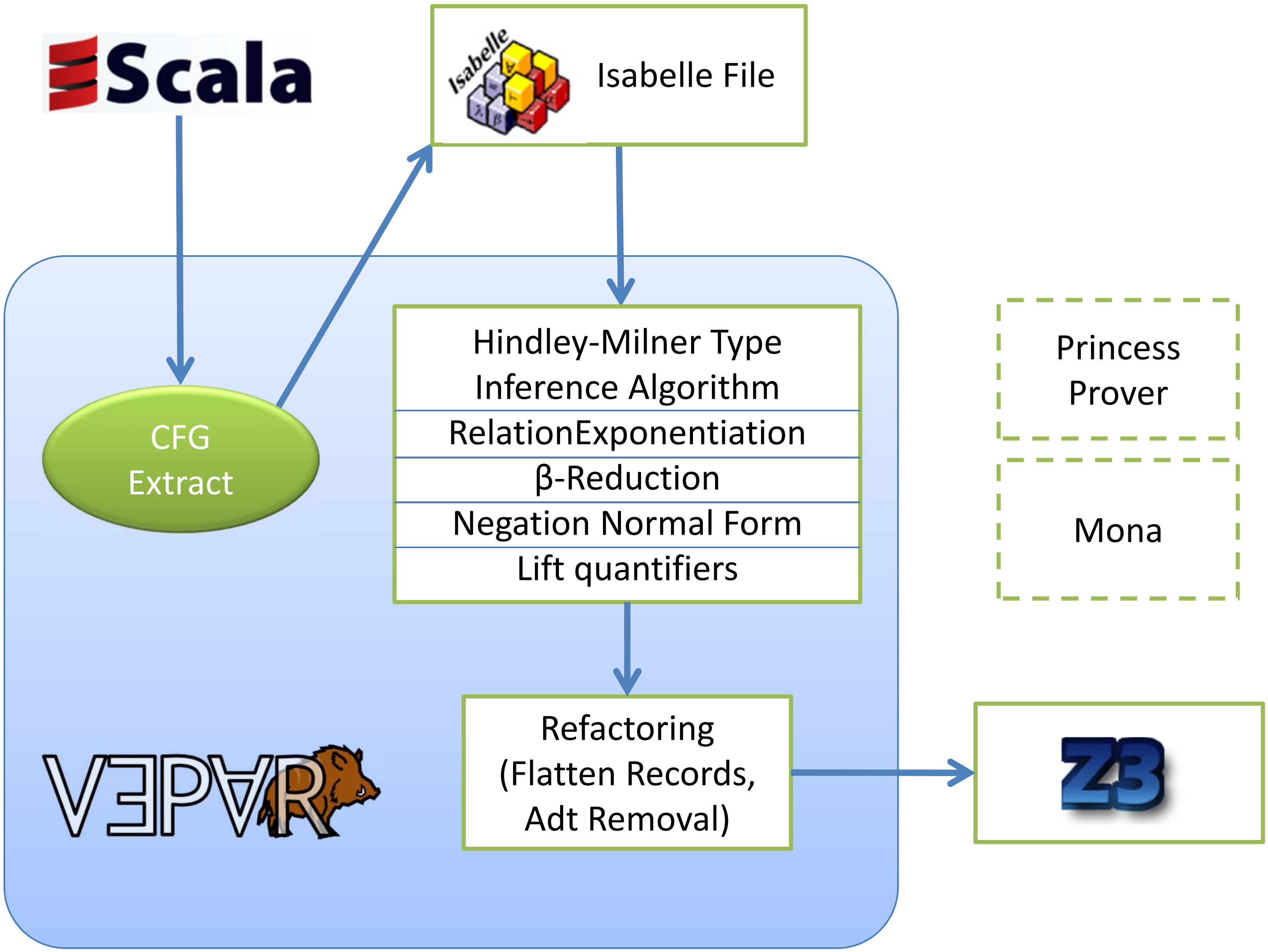


A Framework for Automated Reasoning

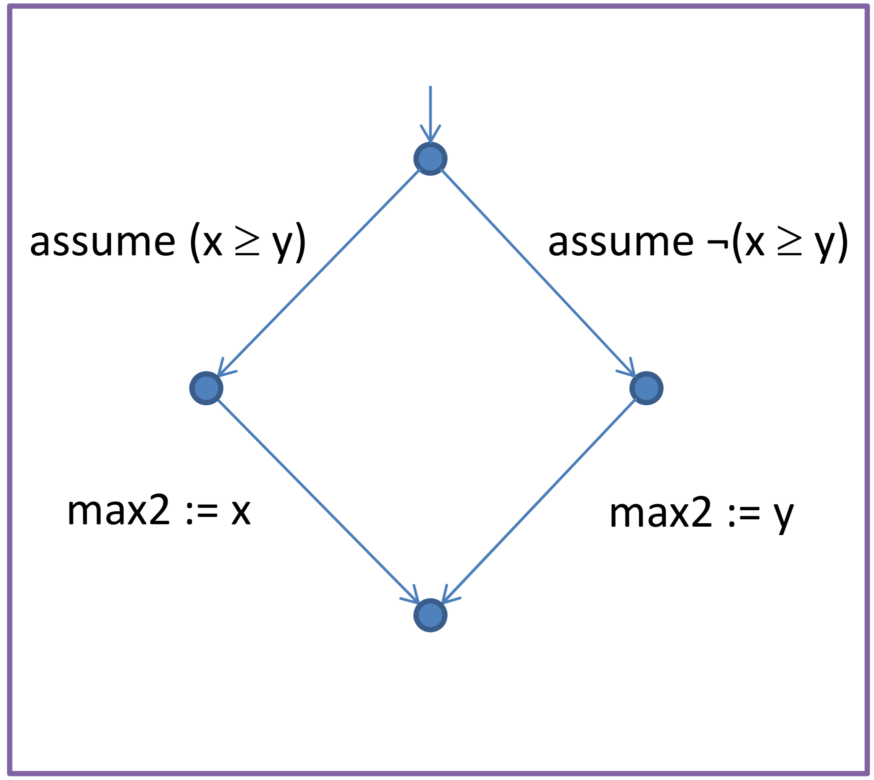
Eva Darulova, ***Hossein Hojjat***, Viktor Kuncak,
Ruzica Piskac and Philippe Suter

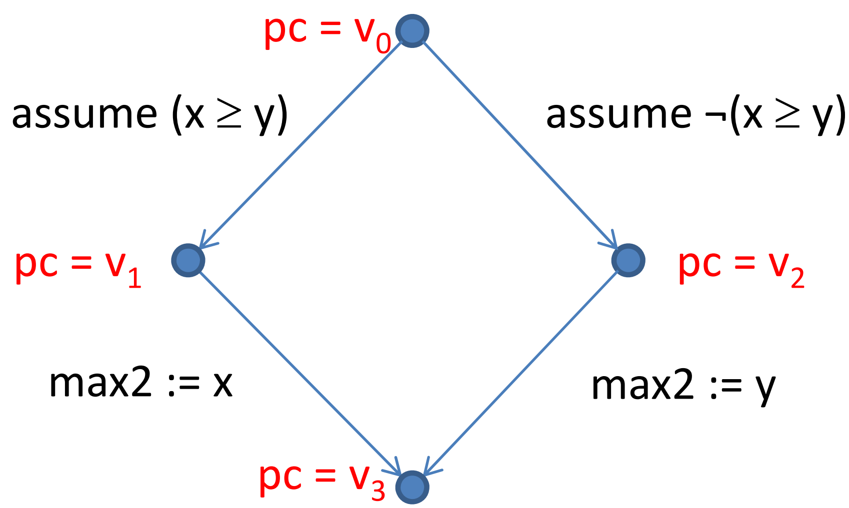
Ecole Polytechnique Fédérale de Lausanne


Scala
Isabelle File
CFG
Extract
Hindley-Milner Type
Inference Algorithm
RelationExponentiation
β-Reduction
Negation Normal Form
Lift quantifiers
Princess
Prover
Mona
Refactoring
(Flatten Records,
Adt Removal)
Z3

From program to CFG

```
def  max2(x: Int, y: Int): Int = {
  if( x >= y) x else y
}
```





record state =
 pc :: label
 x :: int
 y :: int
 max2 :: int

definition program :: "(state × state) set" **where**

"program ≡ {(s,s'). $s' = (pc\ s = v_0) \wedge s⦇ pc := v_1 ⦈ \wedge (x\ s >= y\ s) \vee$
$(pc\ s = v_0) \wedge s' = s⦇ pc := v_2 ⦈ \wedge \neg(x\ s >= y\ s) \vee$
$(pc\ s = v_1) \wedge s' = s⦇ pc := v_3, max2 := x\ s ⦈ \vee$
$(pc\ s = v_2) \wedge s' = s⦇ pc := v_3, max2 := y\ s ⦈$}"

```
def  max2(x: Int, y: Int): Int = {
  if( x >= y) x else y
  assert ( max2 >= x && max2 >=y )
}
```

```
record state =
 pc      ::  label
 x       ::  int
 y       ::  int
 max2    ::  int
 error   ::  bool
definition program :: "(state × state) set" where
"program ≡  {(s,s'). s' = (pc s = v0) ∧ s⦇ pc := v1 ⦈ ∧ (x s >= y s) ∨
           (pc s = v0) ∧ s' = s⦇ pc := v2 ⦈ ∧ ¬(x s >= y s) ∨
           (pc s = v1) ∧ s' = s⦇ pc := v3, max2 := x s  ⦈ ∨
           (pc s = v2) ∧ s' = s⦇ pc := v3, max2 := y s  ⦈}"
           (pc s = v3) ∧ s' = s⦇ pc := ve, error := True⦈ ∧ (max2 s < x ∨ max2s <y)}"

lemma assert: "∀s. ∀s'. (pc s = v0) ∧ (error s = False) ∧ (program^*)(s,s') ⟶
      (error s' = False)"
```

Control Flow Graph in Isabelle

record state =

var_0 :: $type_0$

...

var_n :: $type_n$

error :: bool

program variables

definition program :: "(state × state) set" **where**
"program ≡ {(s,s').

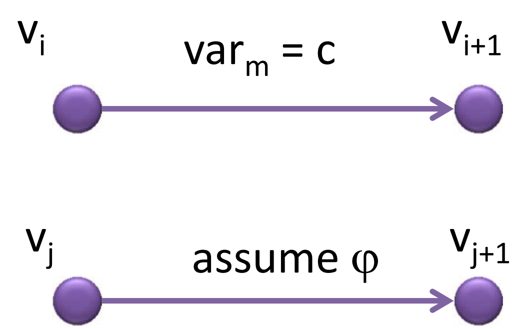
... ∨

(pc s = v_i) ∧ s' = s⦇ pc := v_{i+1} , var_m = c ⦈ ∨

... ∨

(pc s = v_j) ∧ φ ∧ s' = s⦇ pc := v_{j+1} ⦈ ∨

... } "



lemma assert: "∀s. ∀s'. (pc s = v0) ∧ (error s = False) ∧ (program^^n)(s,s') →
(error s' = False) "

Bounded model checking

```
record state =
    x      ::   int
    pc     ::   label
```

```
definition program :: "(state × state) set" where
"program ≡ {(s,s').
    ... ∨
        (pc s = v_i) ∧ s' = s⦇ pc := v_j , x = c ⦈
    ∨ ... } "
```

Record Flattening:
Replace records with their fields

```
definition program :: "(int × label × int × label) set" where
"program ≡ {(s_x, s_pc,s'_x,s'_pc).
    ... ∨
        (s_pc = v_i) ∧ (s'_pc = v_j ) ∧ (s'_x = c )
    ∨ ... } "
```

From Isabelle to SMT-LIB

record state =
 x :: int

definition program :: "(state × state) set" **where**

 "program ≡ {(s,s'). (s' = s⦇ x := 0 ⦈ }"

lemma dumb: " ∀s. ∀s'. state. (program(s,s')^^2) → (x s' = 0) "

↓

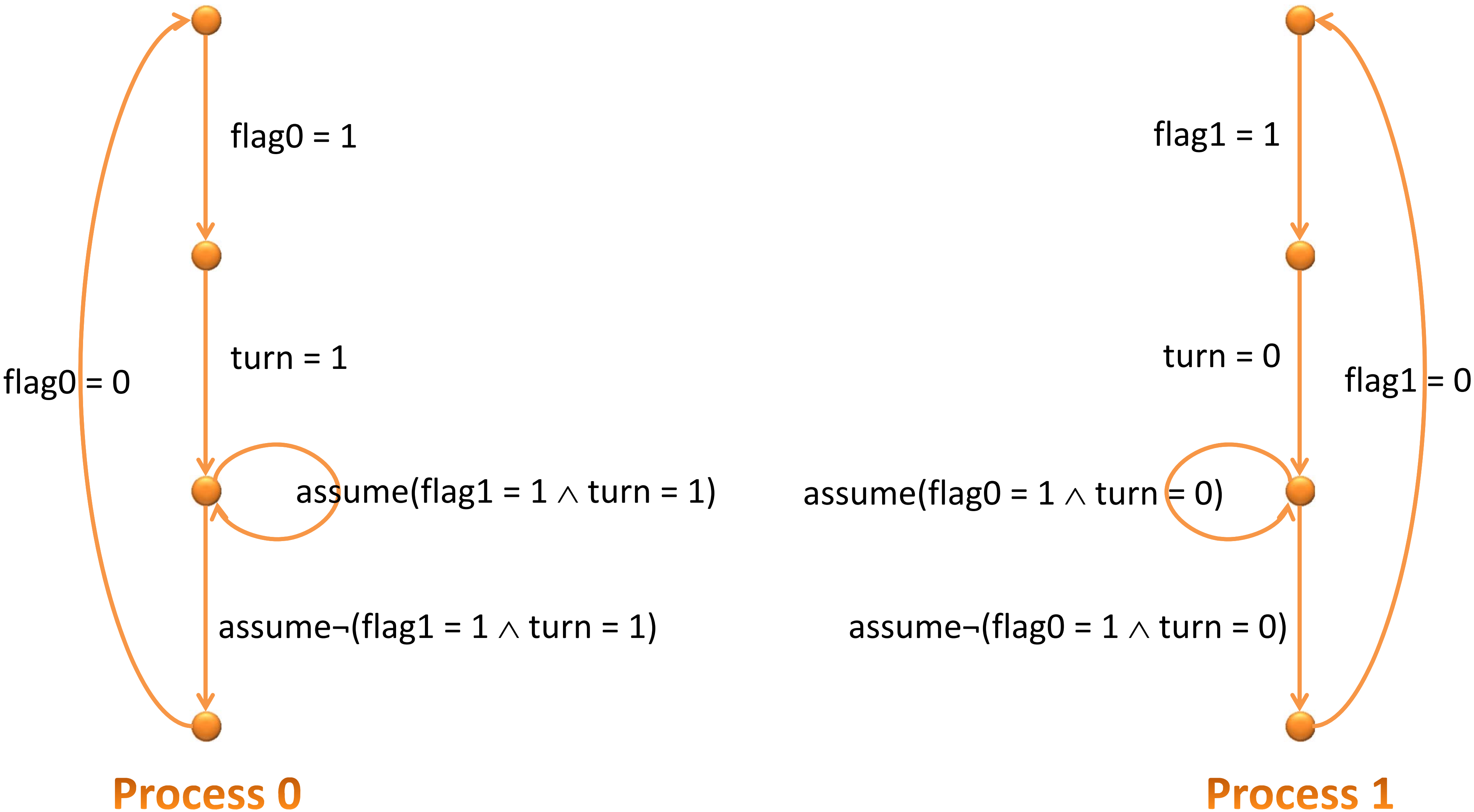
lemma dumb: "∃s_x. ∃s'_x. (s'_x = 0) ∧ (¬(s'_x = 0)) "

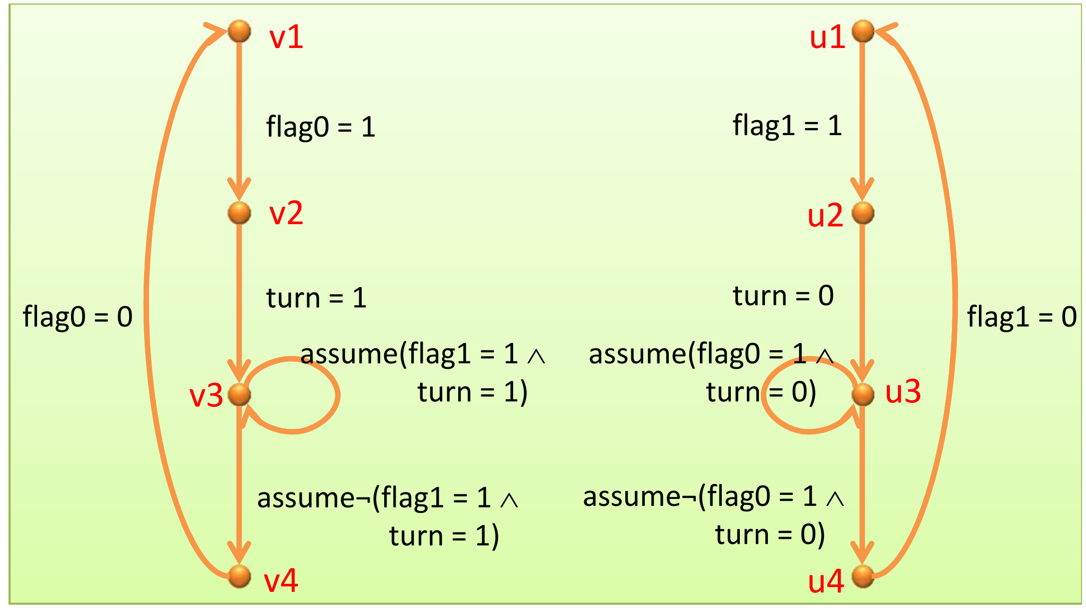
↓ Negation
Conversion to SMT-LIB

(and (= (s'_x) 0) (not (= (s'_x) 0))))

Demo

Peterson's algorithm






```
record state =
  pc0     ::    label
  pc1     ::    label
  flag0   ::    nat
  flag1   ::    nat
  turn    ::    nat
definition transition :: "(state × state) set" where
  "transition ==
      {(s,s').
      ((pc0 s) = v1) ∧ s' = s(|pc0 := v2,flag0 := 1|) |
      ((pc0 s) = v2) ∧ s' = s(|pc0 := v3,turn := 1|) |
      ((pc0 s) = v3) ∧ s' = s ∧ (flag1 s = 1) ∧ (turn s = 1) |
      ((pc0 s) = v3) ∧ s' = s(|pc0 := v4|) ∧ ¬ (flag1 s = 1 ∧ turn s = 1) |
      ((pc0 s) = v4) ∧ s' = s(|pc0 :=v1,flag0 := 0|) |
      ((pc1 s) = u1) ∧ s' = s(|pc1 := u2,flag1 := 1|) |
      ((pc1 s) = u2) ∧ s' = s(|pc1 := u3,turn := 0|) |
      ((pc1 s) = u3) ∧ s' = s ∧ (flag0 s = 1) ∧ (turn s = 0) |
      ((pc1 s) = u3) ∧ s' = s(|pc1 := u4|) ∧ ¬ (flag0 s = 1 ∧ turn s = 0) |
      ((pc1 s) = u4) ∧ s' = s(|pc1 :=u1,flag1 := 0|) |
      (s = s')
  }"
lemma critical: " ∀s. ∀s'.  (pc0 s = v1) ∧ (pc1 s = u1) ∧ (transition^^10)(s,s') → ¬ (pc0 s' = v4 ∧ pc1 s' = u4)"
```

Uncertainty on the unrolling number

Conclusions

- Framework for reasoning about programs
- Currently supports small subset of Scala programs
- Translation to Isabelle and from Isabelle to Z3

Future Work

- Incorporate more reasoning engines
- Support for Scala actors
- Connection to Isabelle/HOL