NICTA

Predicate Abstraction for Reactive Synthesis

Adam Walker, Leonid Ryzhyk

Automatic Device Driver Synthesis

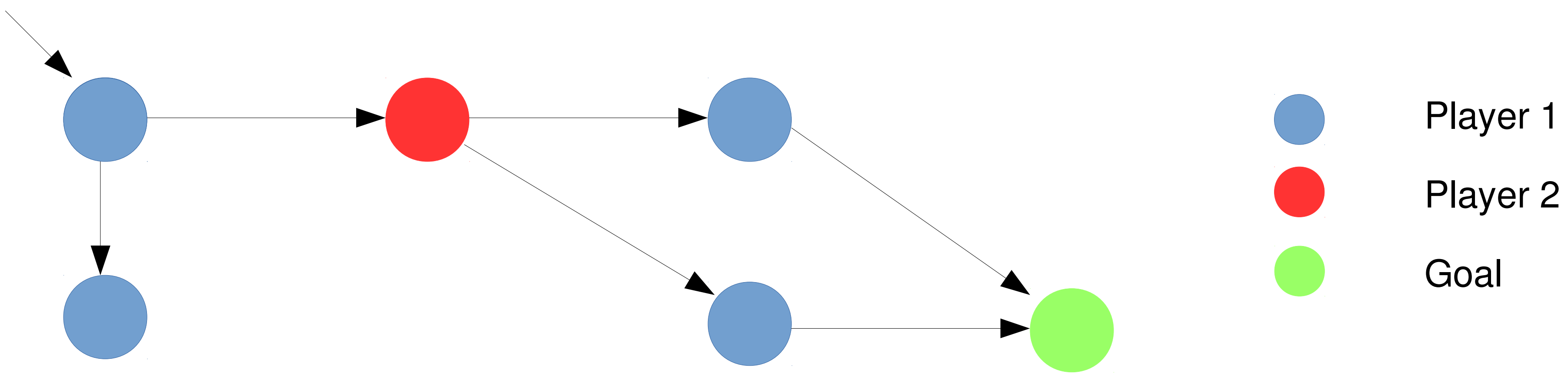
- A three-year project (2013-2016)
- Funded by a $1.2M grant from Intel

Motivation

- Termite device driver synthesis project (OSDI '14)
- Driver vs (OS || device)
- Upwards of $\mathbf{2}^{1000}$ states
- Predicate abstraction to capture relationships between variables

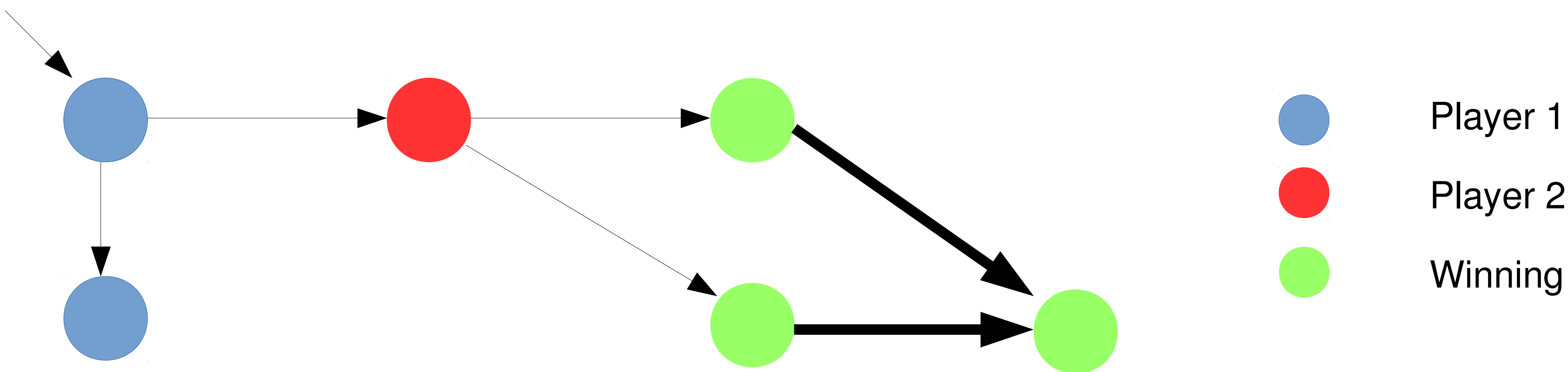
Background

- $G = \langle S, L, I, \tau_1, \tau_2, \delta \rangle$
- Reachability



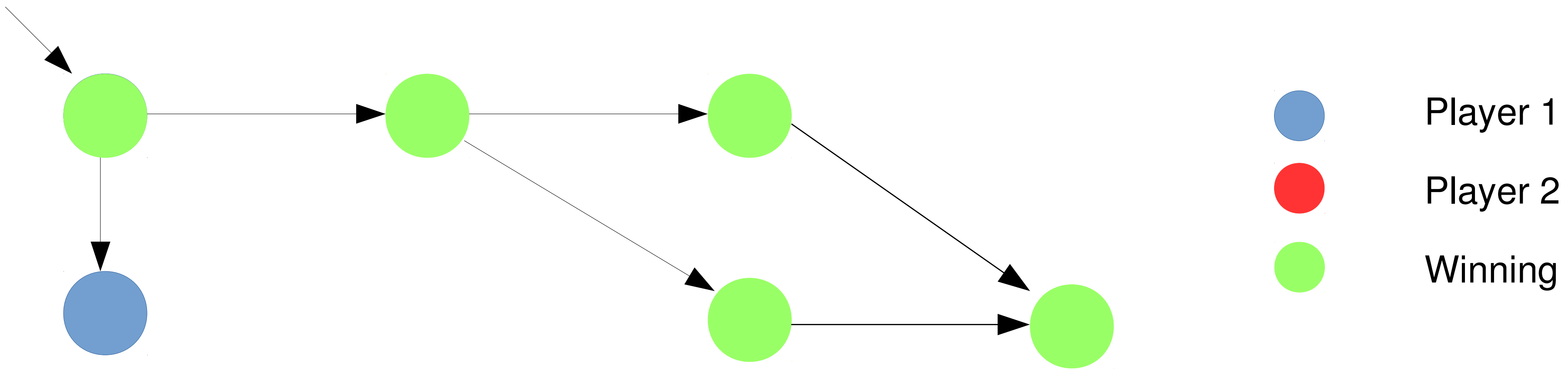
Controllable Predecessor

$$
\begin{aligned}
CPre(w) = \tau_1 \wedge \exists L.\forall N.\delta \rightarrow w' \\
\vee \tau_2 \wedge \forall L.\forall N.\delta \rightarrow w'
\end{aligned}
$$



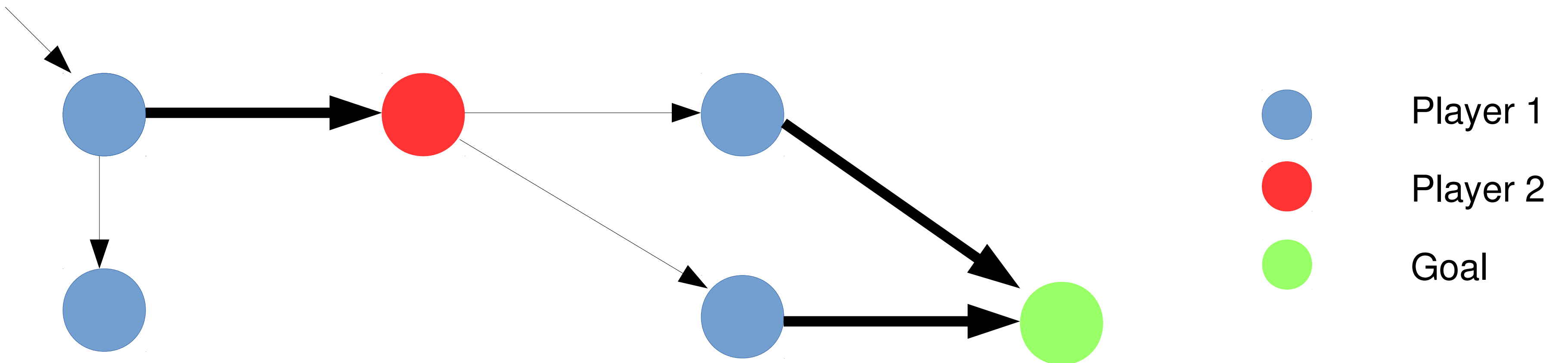
Controllable Predecessor

- Iterate to fixed point



Controllable Predecessor

- Extract strategy

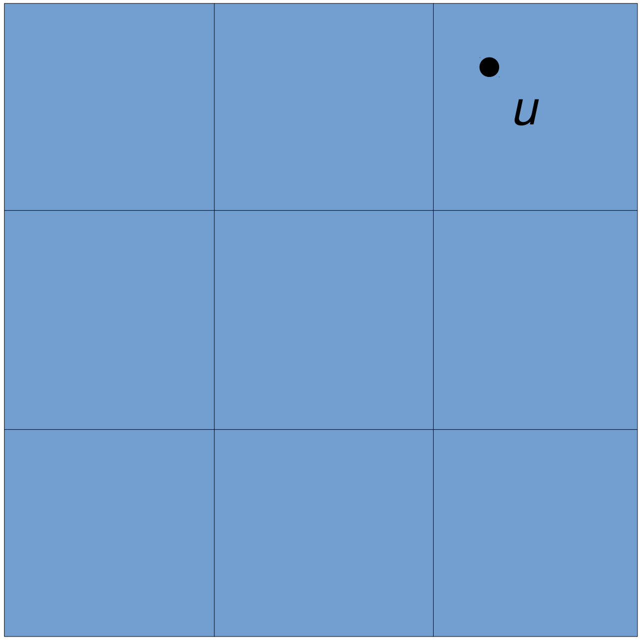


Symbolic Games

- Variables
  - State: X
  - Label: Y
- $\delta(X, Y, X')$
- $\text{Cpre}(\text{win}) = \tau_1 \wedge \exists l. \forall n. \delta \to win' \\ \vee \tau_2 \wedge \forall l. \forall n. \delta \to win'$

Abstraction

- Equivalence relation over states that behave similarly



Abstraction Example

```
Bool X, Y, P, Q := False;

Action1 : X := True
Action2 : Y := P
Action3 : P := True
Action4 : Q := True

Goal := (X == True) && (Y == True)
```

Abstraction Example

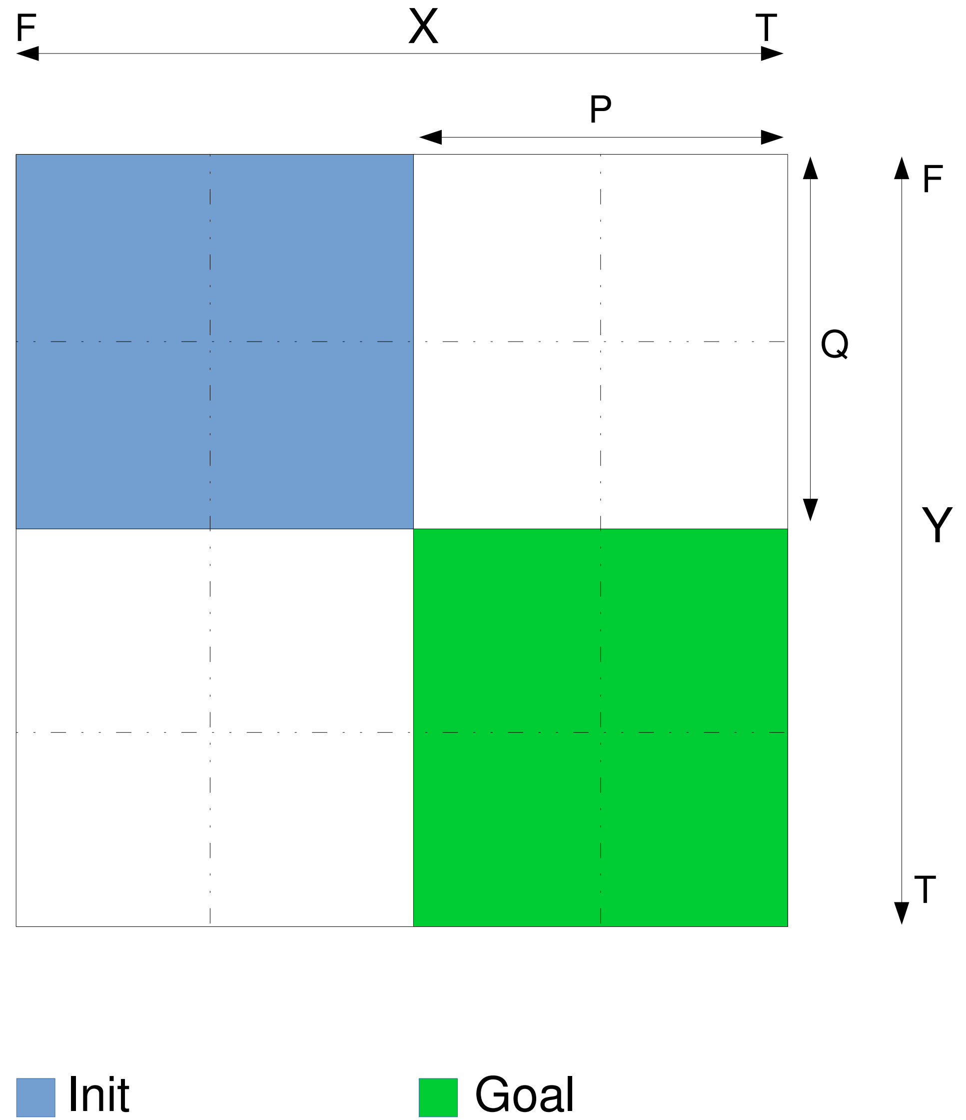
```
Goal := (X == True)
&& (Y == True)

Action1 : X := True

Action2 : Y := Q

Action3 : P := True

Action4 : Q := True
```

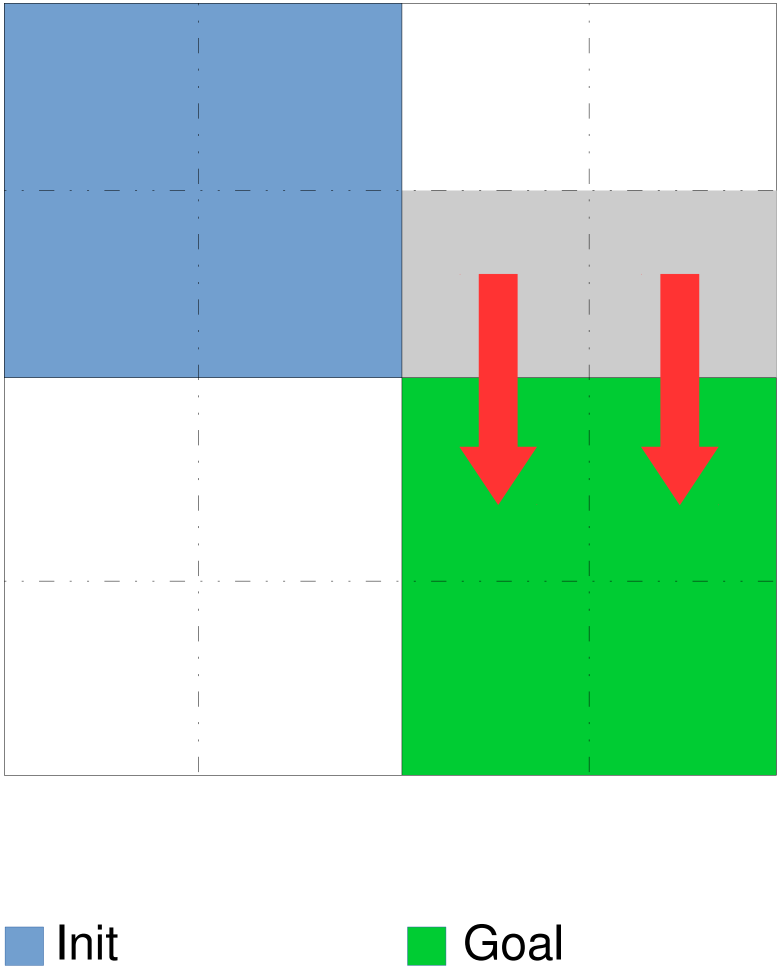


Three Valued Game Solving

- Abstraction introduces imprecision
- De Alfaro and Roy, 2007
- Classify states
  - Must win
  - May win
  - Must lose

Three Valued Game Solving

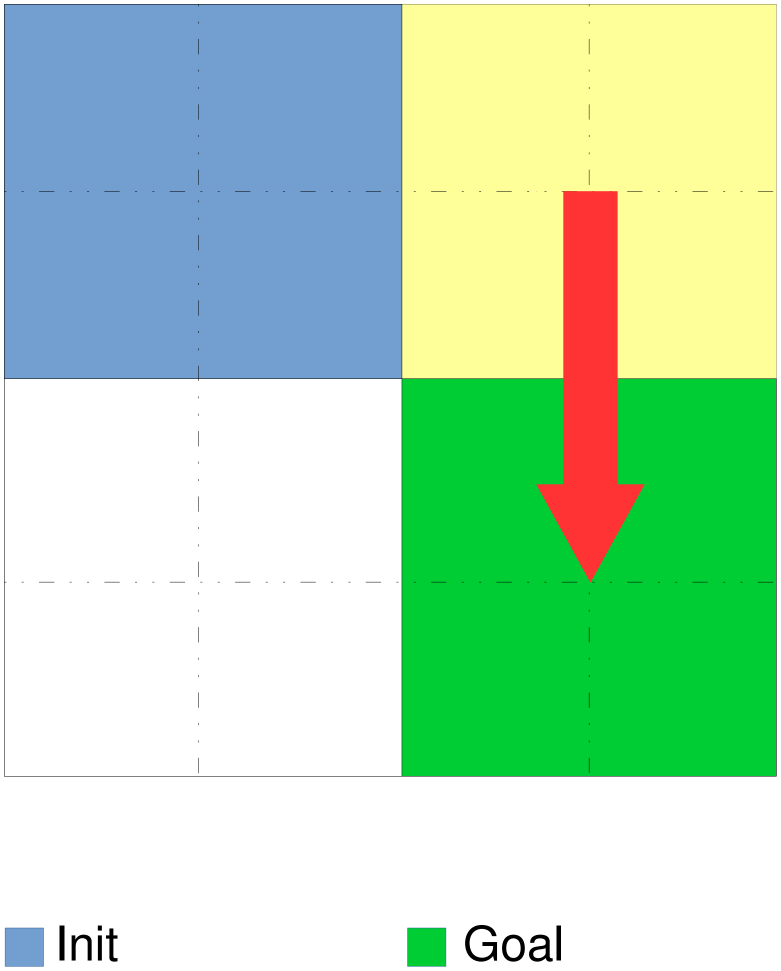
$$CPre(w) = \exists U. \exists L. \forall N. \delta \rightarrow w'$$



Winning?

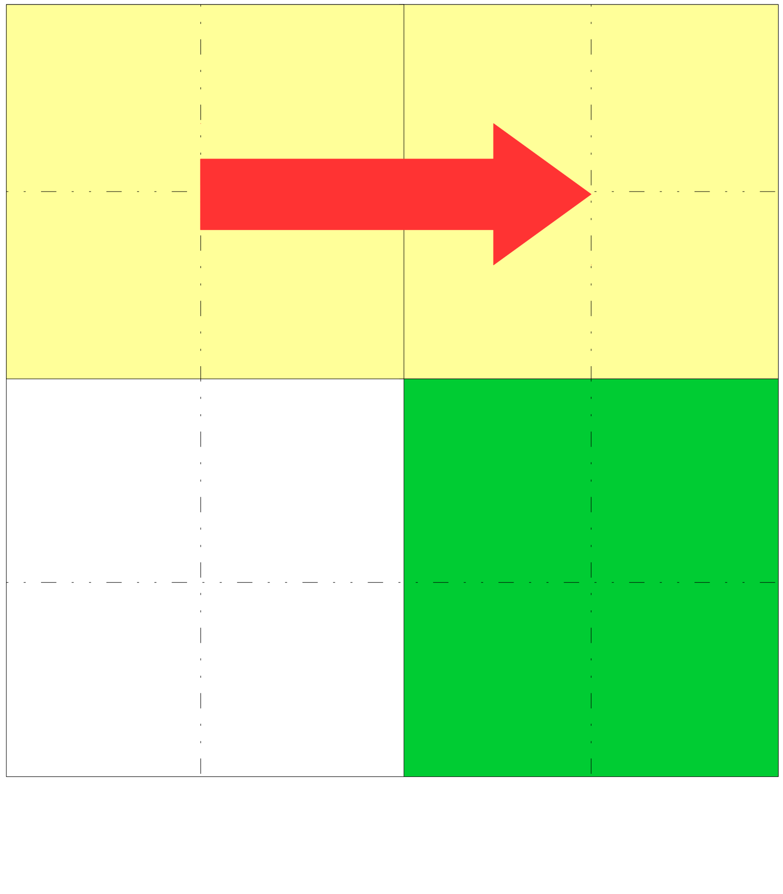
- Yes if Q==T
- No if Q==F

Three Valued Game Solving



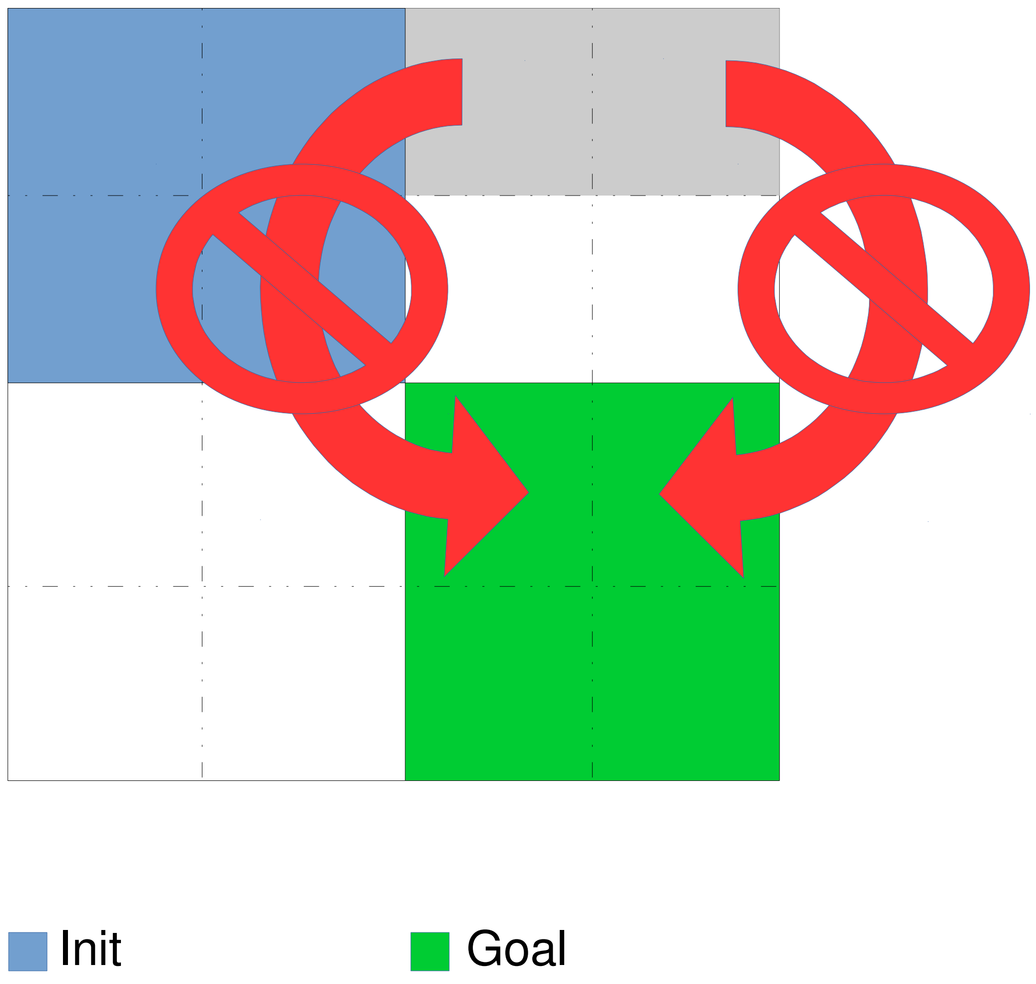
Winning?
- Yes if Q==T
- No if Q==F

Three Valued Game Solving

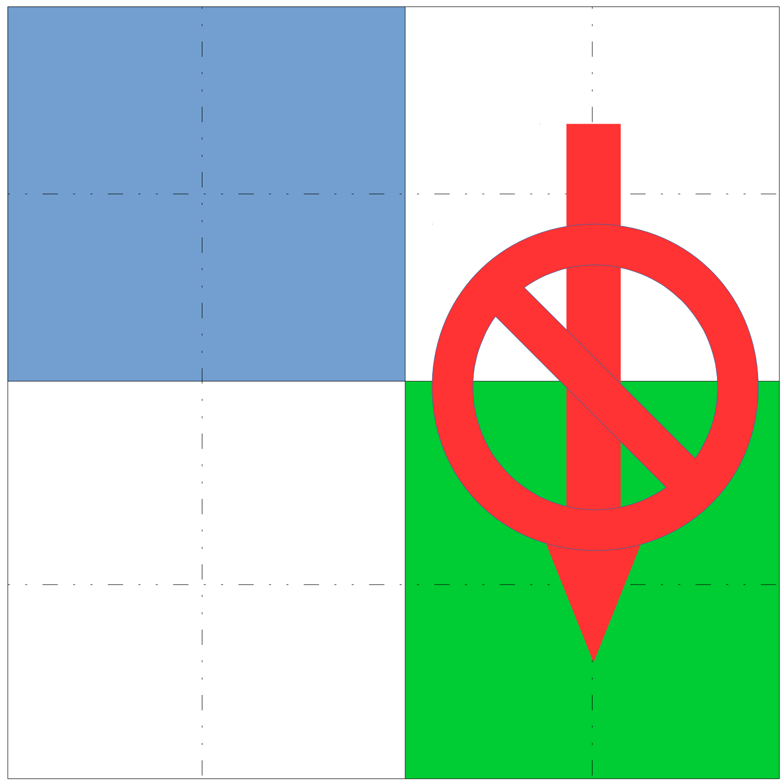


Three Valued Game Solving

$$CPre(w) = \forall U.\exists L.\forall N.\delta \rightarrow w'$$



Three Valued Game Solving

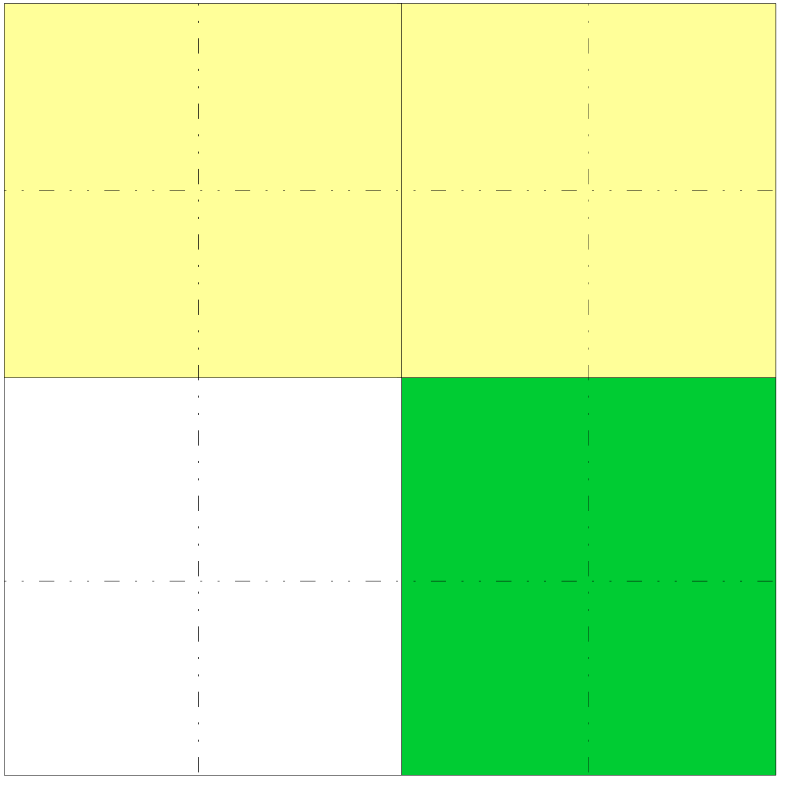


Winning?

- Yes if Q==T
- No if Q==F

Init　　Goal

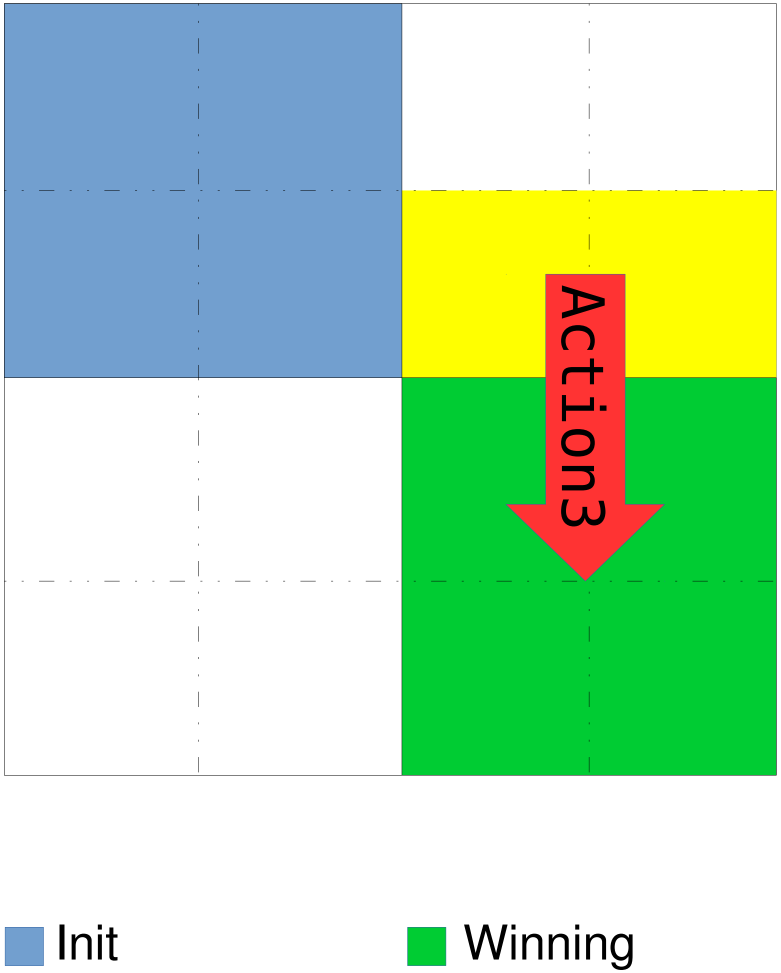
Three Valued Game Solving



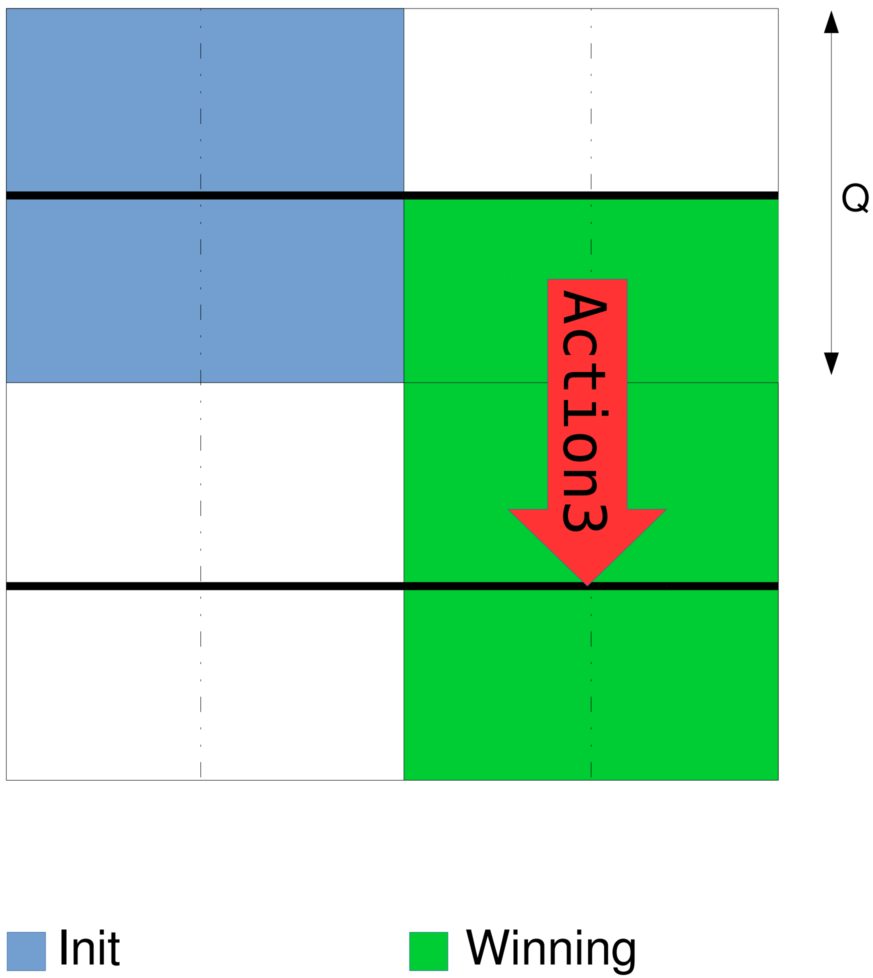
Maybe winning Winning

Abstraction Refinement

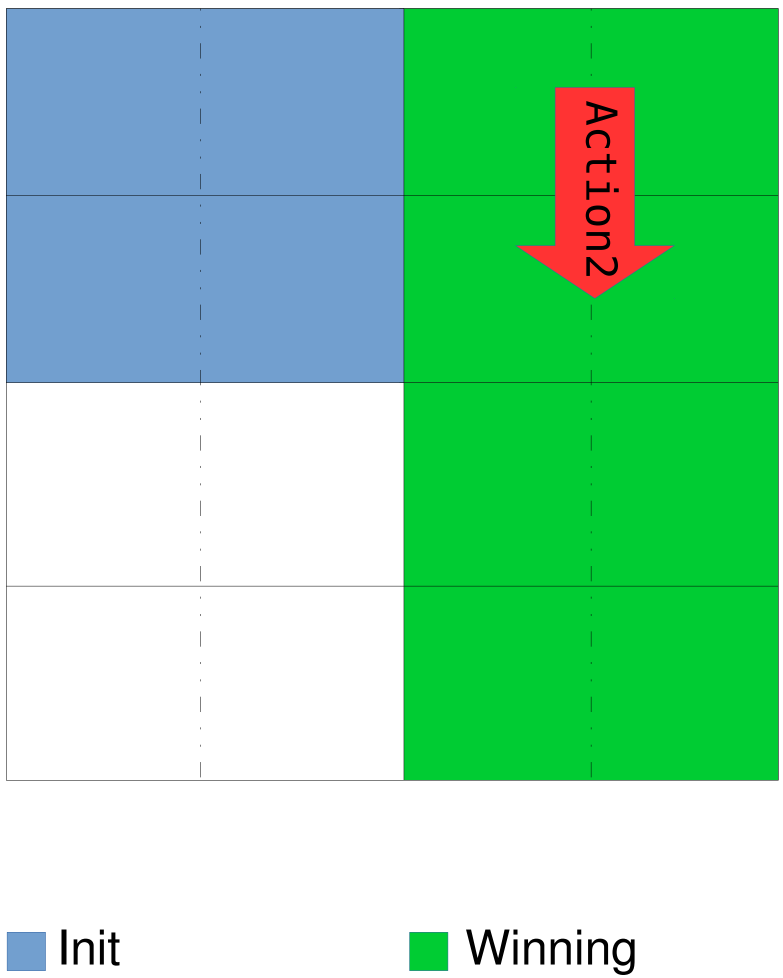
$$Cpre\_U(W) =$$
$$\exists L.\forall N.\delta \rightarrow W'$$



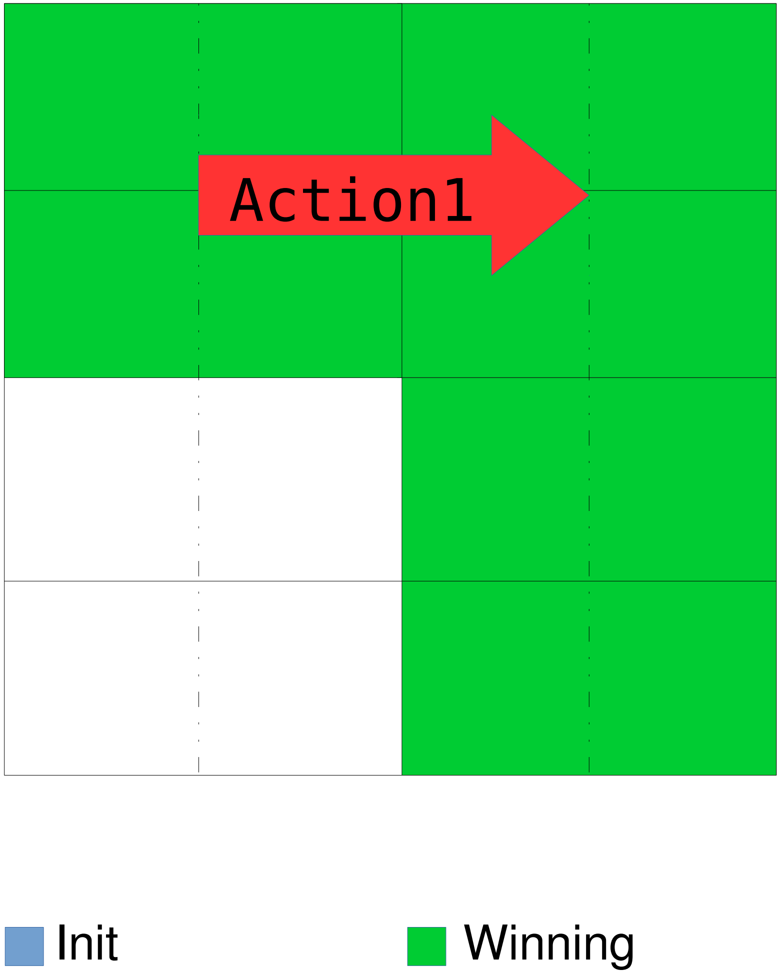
Abstraction Refinement



Abstraction Refinement



Abstraction Refinement



Abstraction Refinement

```
Solve(game, goal){
    abstraction ← create_initial_abstraction()









}
```

Abstraction Refinement

```
Solve(game, goal){
    abstraction ← create_initial_abstraction()
    win_must    ← False

    while(true) {
        win_must ← solve(win_must);









    }
}
```

Abstraction Refinement

```
Solve(game, goal){
    abstraction ← create_initial_abstraction()
    win_must    ← False

    while(true) {
        win_must ← solve(win_must);

        if(init → win_must){
            return True;
        } else {




        }
    }
}
```

Abstraction Refinement

```
Solve(game, goal){
    abstraction ← create_initial_abstraction()
    win_must    ← False

    while(true) {
        win_must ← solve(win_must);

        if(init → win_must){
            return True;
        } else {
            res ← refine_abstraction
            if(!res){
                return False;
            }
        }
    }
}
```

Abstraction Refinement

- We did not create an unneccessarily fine abstraction
- We performed refinement on demand
- We reused our previously computed winning set.
- Selection of variable to promote is cheap
- We do not need to recompute entire transition relation on each refinement

Abstraction Refinement

- Performs well if game outcome can be determined from a subset of state
  - Such as unsatisfiable subset of synthesis competition 2014 benchmarks

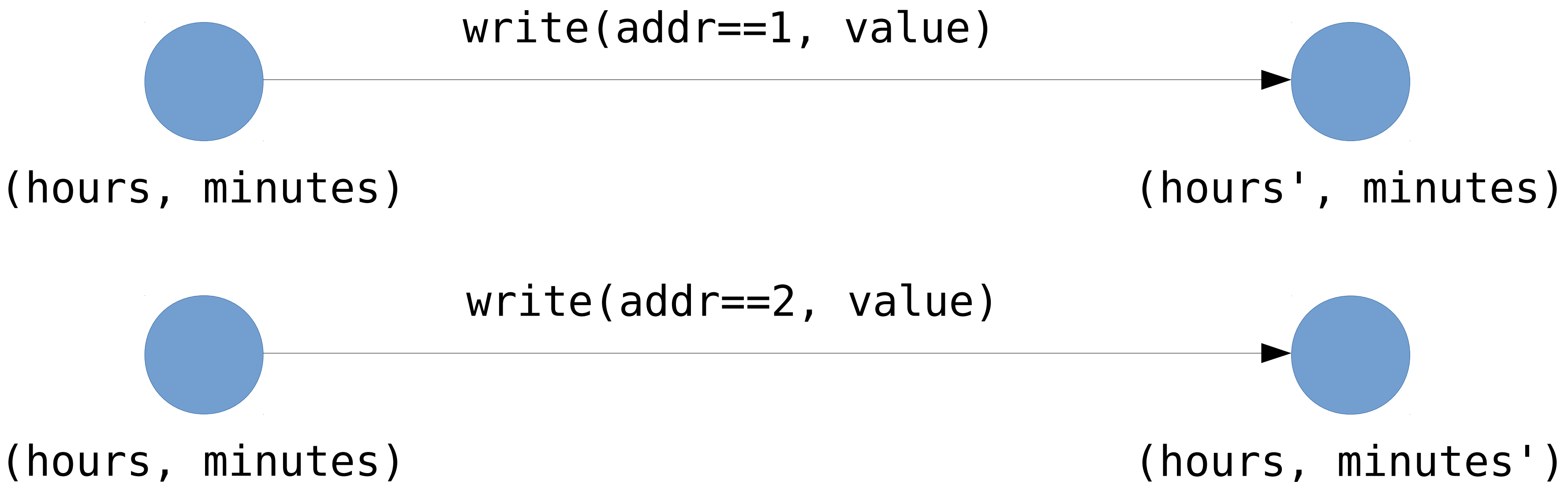
Predicate Abstraction

`write(1, 19)`

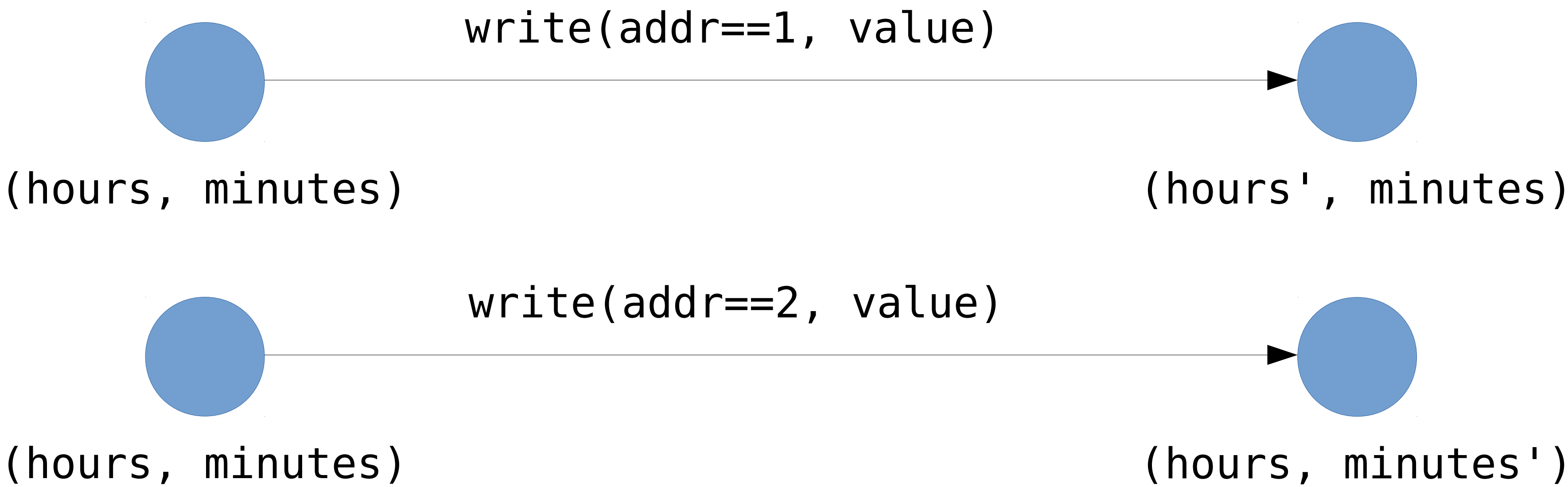
Predicate Abstraction

`write(2, 59)`

Predicate Abstraction

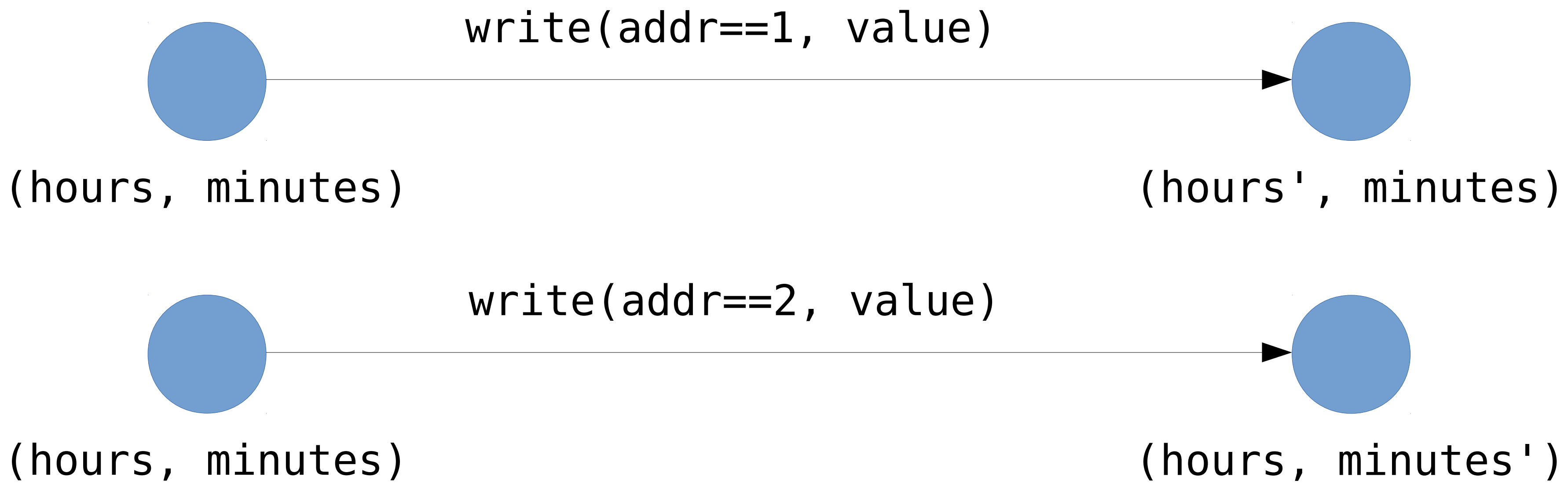


Predicate Abstraction




```
State predicates: (hours = req_hours)
                  (minutes = req_minutes)
```

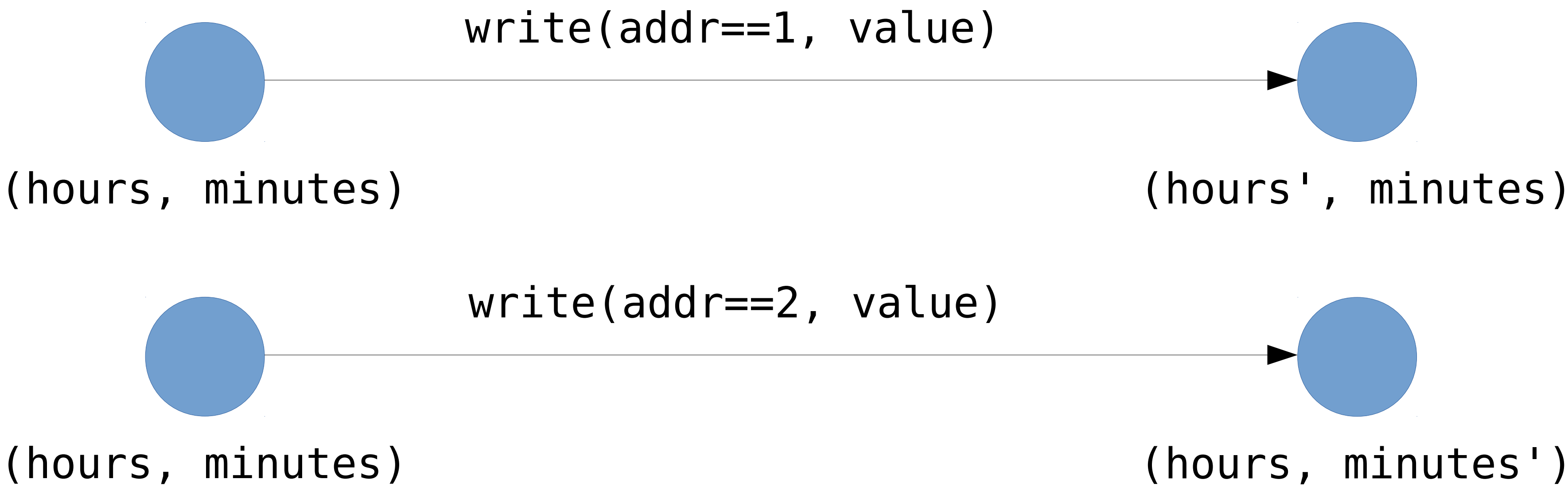
Predicate Abstraction




```
State predicates: (hours = req_hours)
                  (minutes = req_minutes)

Label predicates: (value = req_hours)
                  (value = req_minutes)
```

Predicate Abstraction



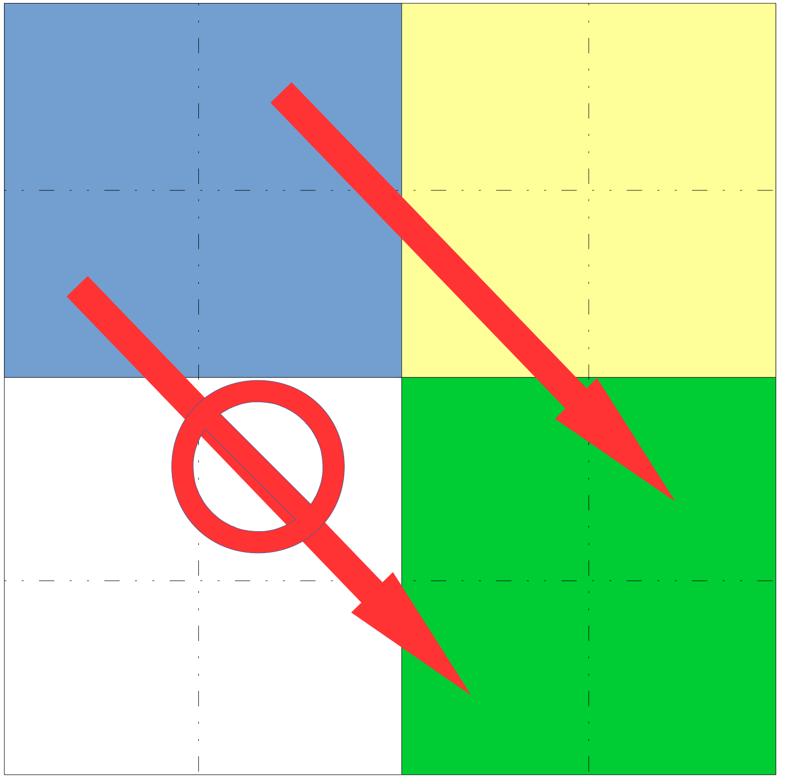

```
State predicates: (hours = req_hours)
                  (minutes = req_minutes)

Label predicates: (value = req_hours)
                  (value = req_minutes)
                  (addr = 1)
                  (addr = 2)
```

Predicate Abstraction

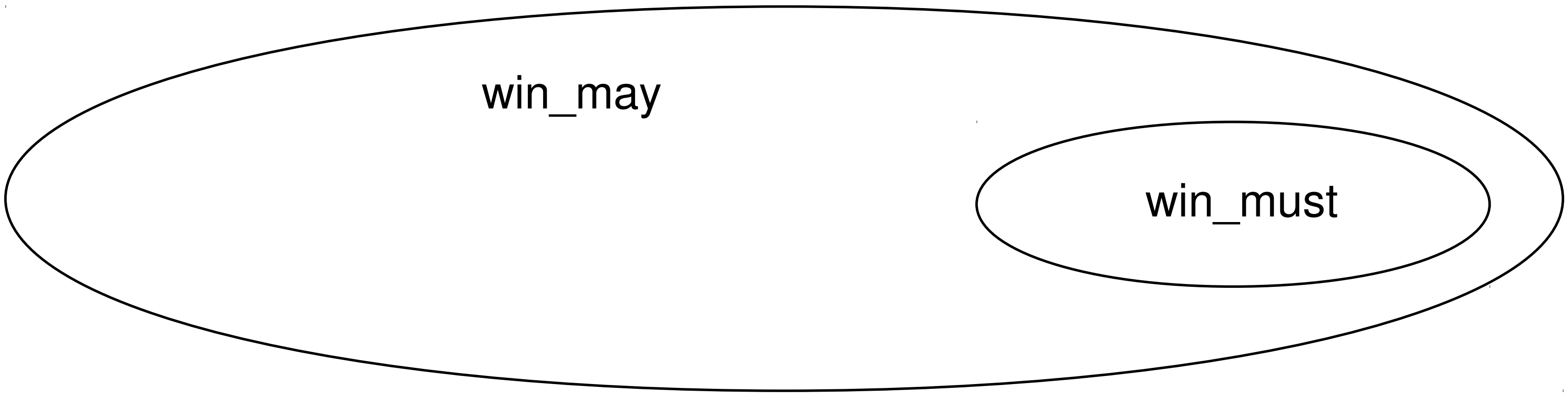
Predicate Abstraction

Predicate Abstraction

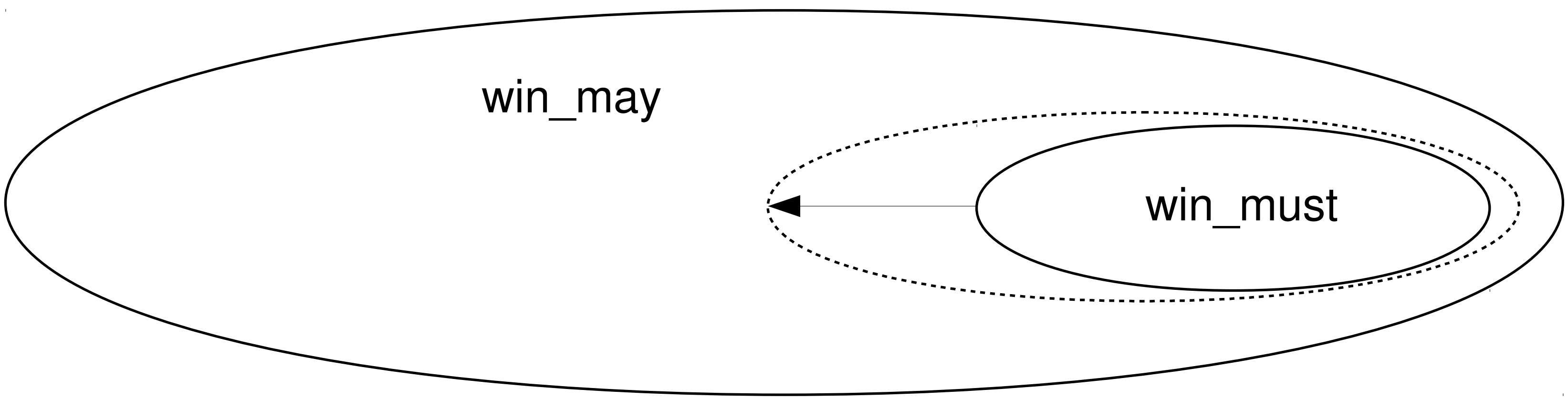


$$CPre(w) = \exists L.\forall N.((\delta \to w') \wedge consistent)$$

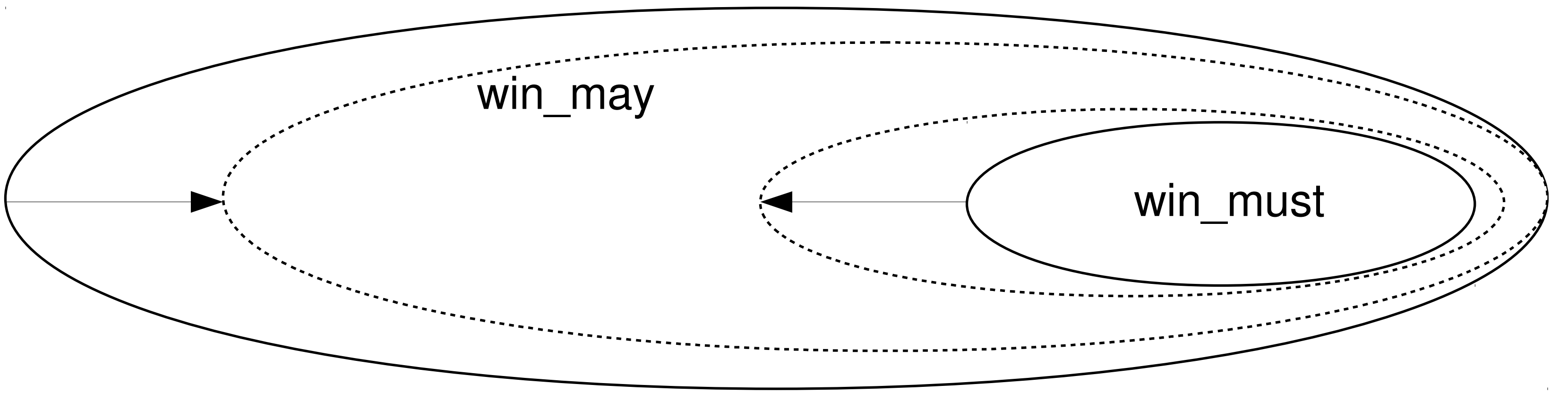
Predicate Abstraction



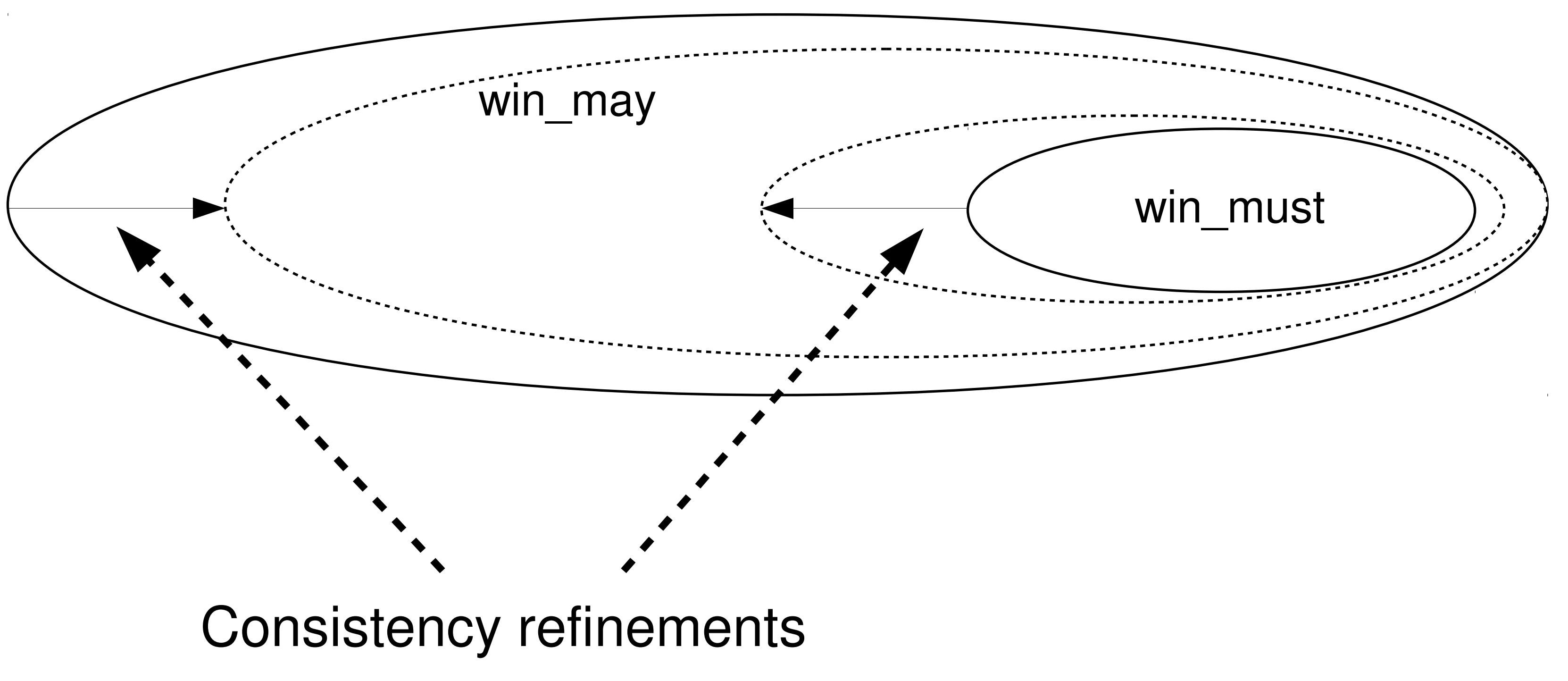
Predicate Abstraction



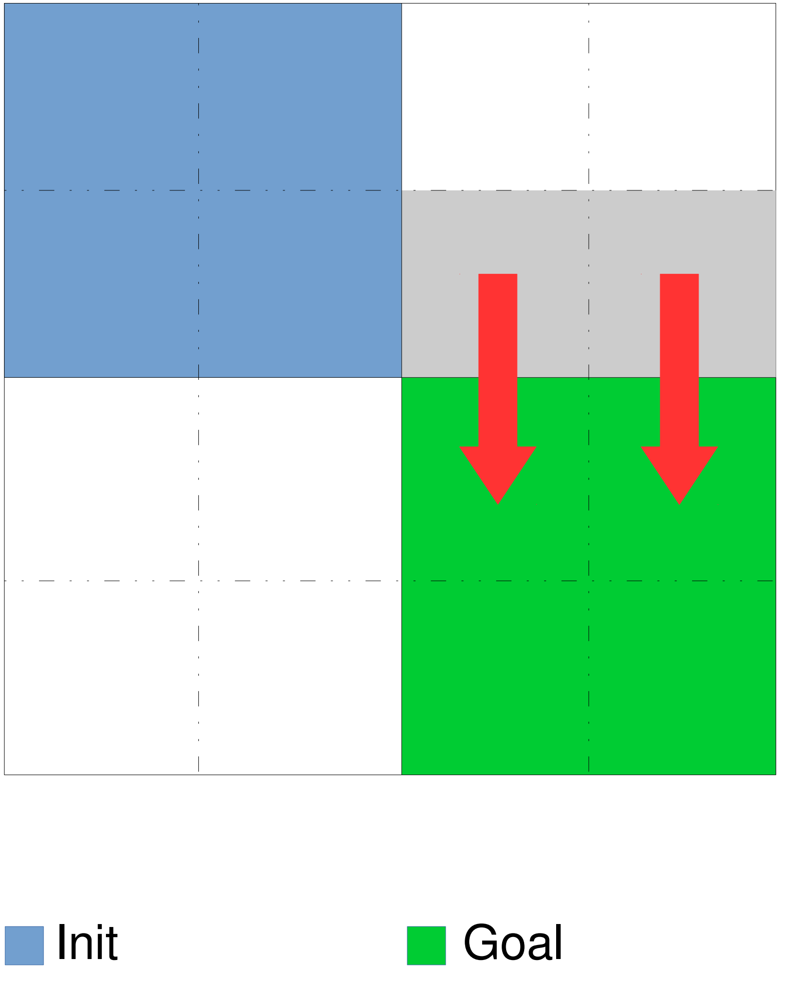
Predicate Abstraction



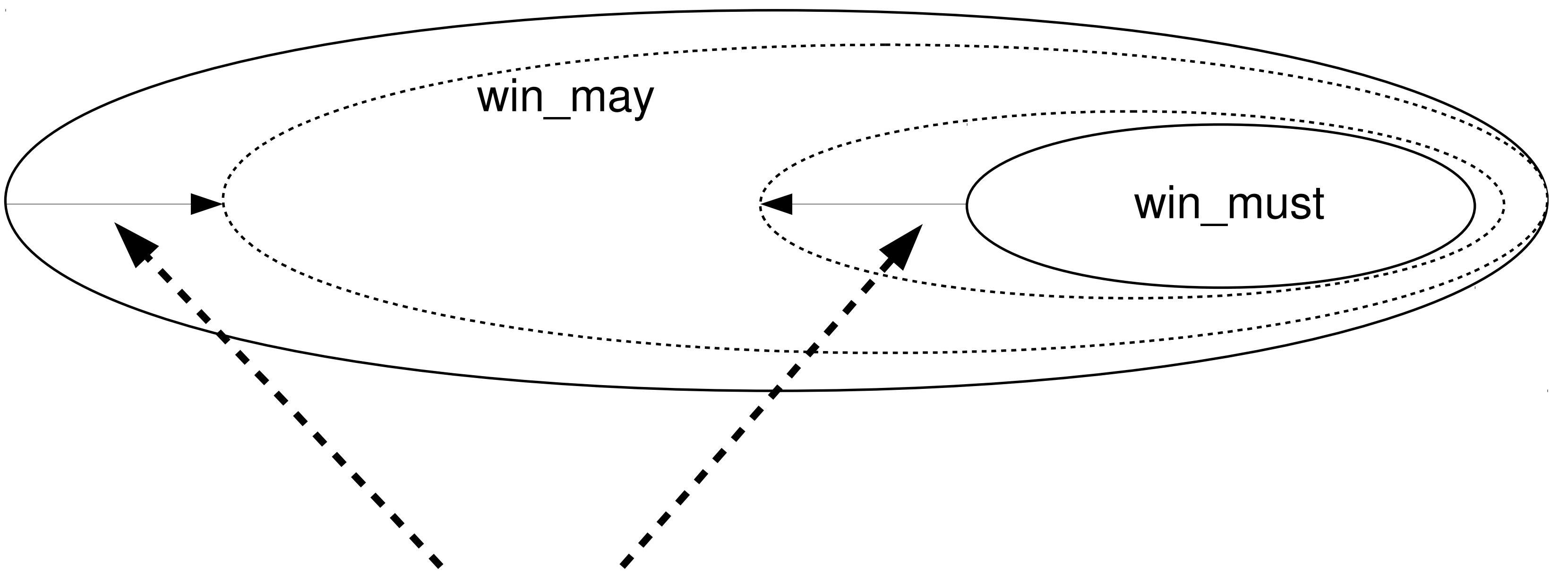
Predicate Abstraction



Three Valued Game Solving

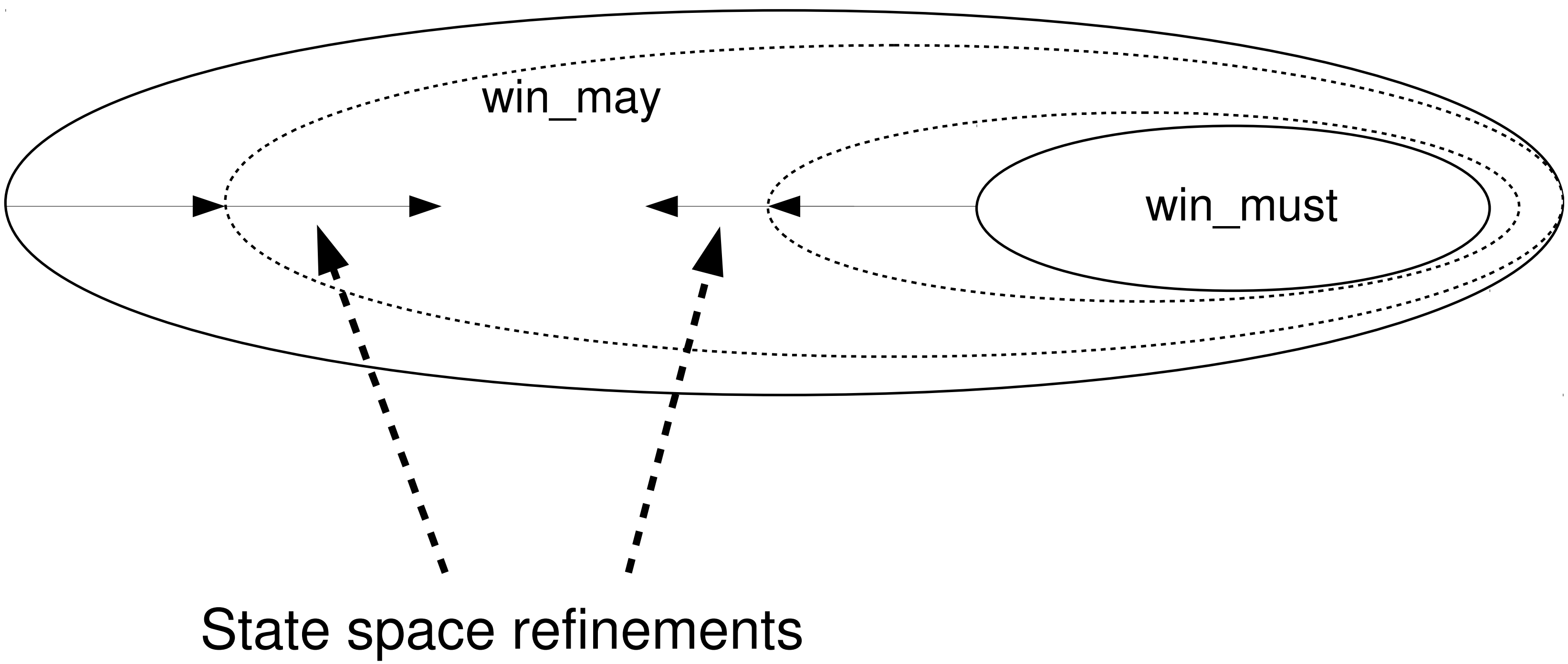


Predicate Abstraction



Consistency refinements

Predicate Abstraction



Implementation

- Part of the Termite driver synthesis toolkit
- GR(1) games
- ~2000 lines of code
- Binary decision diagram based

`termite2.org`

`www.github.com/termite2`

Results

Driver	State bits	Abstract state bits	Num refinements	Synthesis time (s)	Lines of code
Webcam	75908	30	47	462	113
16450 UART	335	33	60	50	51
Exynos UART	896	58	54	645	37
STM SPI	644	31	32	67	24
Exynos I2C	222	24	21	45	79
Real time clock	624	25	25	56	84
IDE disk	952	31	29	285	94

termite2.org