

# Model Checking Race-Freeness

## MCC'08

---

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems  
Uppsala University

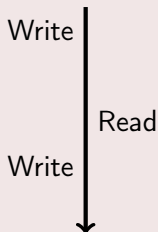
Friday, November 28<sup>th</sup>, 2008



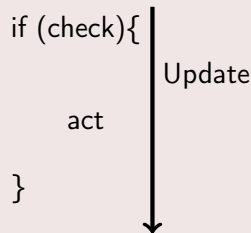
# Race Condition

Situation in which one process changes a variable which another process has previously read and the other process does not get notified of the change.

## Classic



## Check-then-act



i.e. getting the right answer relies on lucky timing.

# Related Work

- Feasible races  
vs. False positives
- Coverage
- Expressiveness
- Annotations

## On-the-fly

- dynamic tools

## Ahead-of-time

- static analysis
- compile-time heuristics

## Post-mortem

Combination of static and dynamic techniques

# Outline

- 1 Language
- 2 Model
- 3 Backward Reachability Analysis
- 4 Experiments and Future Work



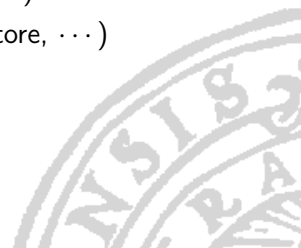
# Language

We analyze programs

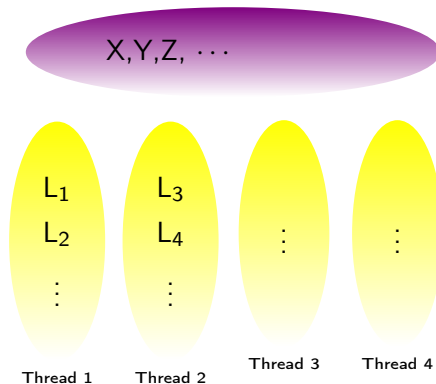
- written in C
- using POSIX threads.

The language we allow

- abstracts away CPU operations (add, sub, ...)
- narrows down to *movers* operations (load, store, ...)
- thread synchronization and bookkeeping



# Language



# Language

## Control statements

- if
- if-then-else
- while
- for-loop

↔ *branch and label*

## Movers

- Read X
- Write X

## Lock – mutex

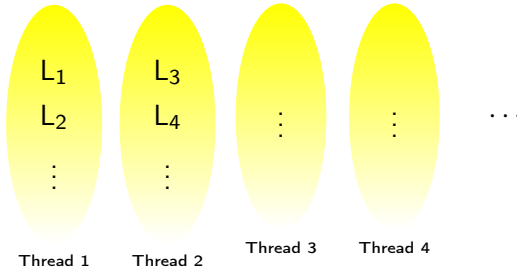
- acquire
- release

## Condition Variable

- wait
- signal

# Language

$X, Y, Z, \dots, M_1, M_2, \dots, CV_1, CV_2, \dots$



# Language – examples

```
int counter;  
pthread_mutex_t L;  
  
pthread_mutex_lock(L);  
counter++;  
pthread_mutex_unlock(L);
```

```
shared counter, L;  
  
acquire L;  
read counter;  
write counter;  
release L;
```



# Language – examples

```
int buffer; pthread_mutex_t L;
pthread_cond_t cvEmpty, cvFull;

//Many Producers
pthread_mutex_lock(L);
while(true){ /*branch*/
    pthread_cond_wait(cvEmpty, L);
    buffer = data;
    pthread_cond_signal(cvFull);
}
pthread_mutex_unlock(L);

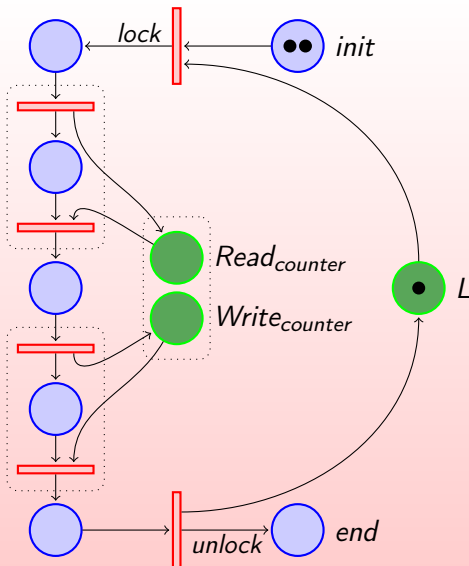
//Many Consumers
pthread_mutex_lock(L);
while(true){ /*branch*/
    pthread_cond_wait(cvFull, L);
    val = buffer;
    pthread_cond_signal(cvEmpty);
}
pthread_mutex_unlock(L);
```

```
shared buffer, L, cvEmpty, cvFull;

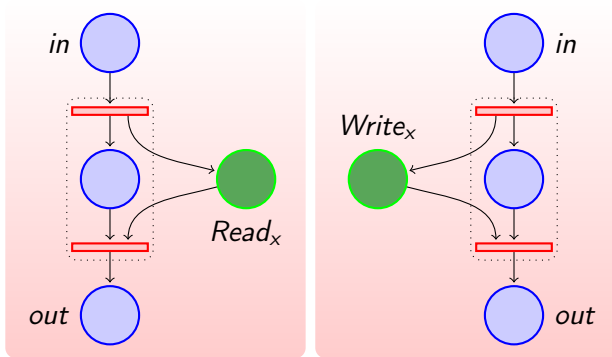
//Many Producers
acquire L;
while(true){ /*branch*/
    wait cvEmpty, L;
    write buffer;
    signal cvFull;
}
release L;

//Many Consumers
acquire L;
while(true){ /*branch*/
    wait cvFull, L;
    read buffer;
    signal cvEmpty;
}
release L;
```

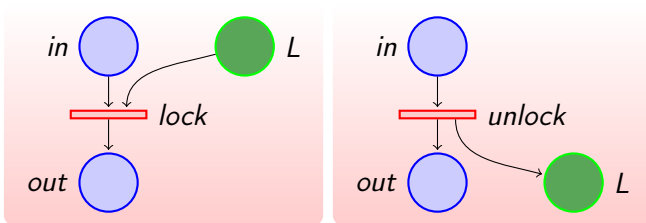
# Model – Petri Nets



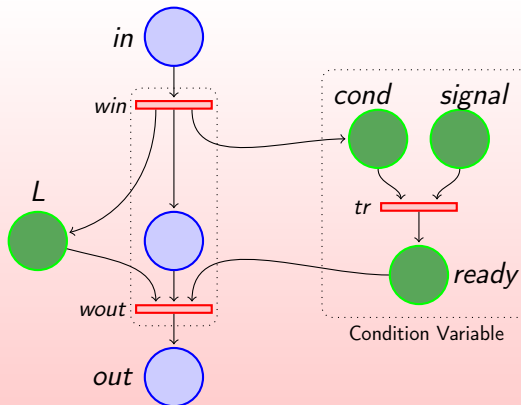
# Reading and Writing a shared variable



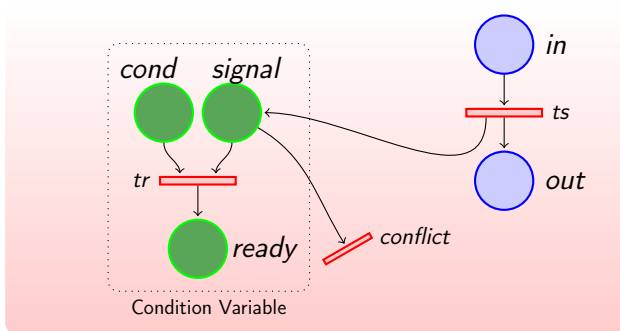
# Acquiring and releasing a lock



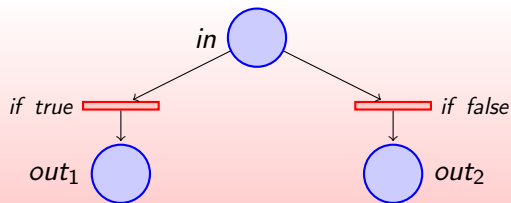
# Waiting on a condition variable



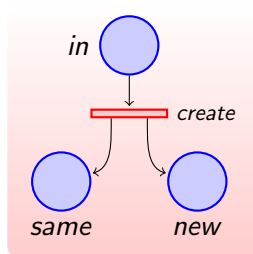
# Signaling a condition variable



# Branching and Jumps

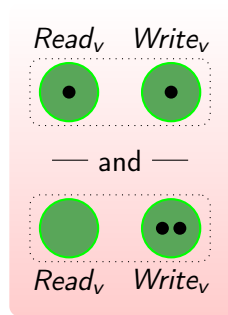


# Creating a new thread



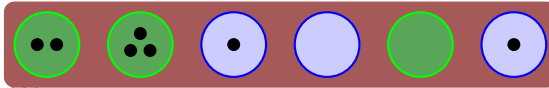
# Data race in our model

A data race occurs when multiple threads access a shared variable and at least one has the intention of changing it, without any synchronization constraints (i.e. event ordering).

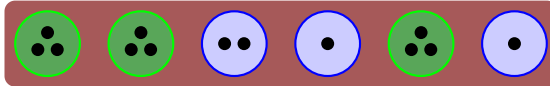
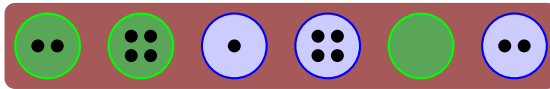


# Interesting properties

A bad configuration:



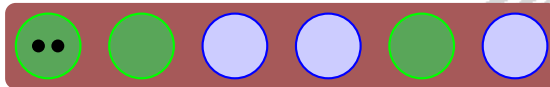
$Write_v$



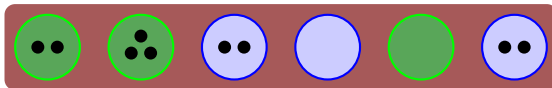
Bad state:



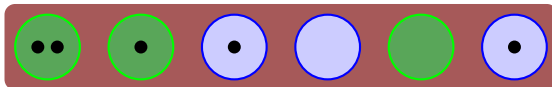
$Write_v$



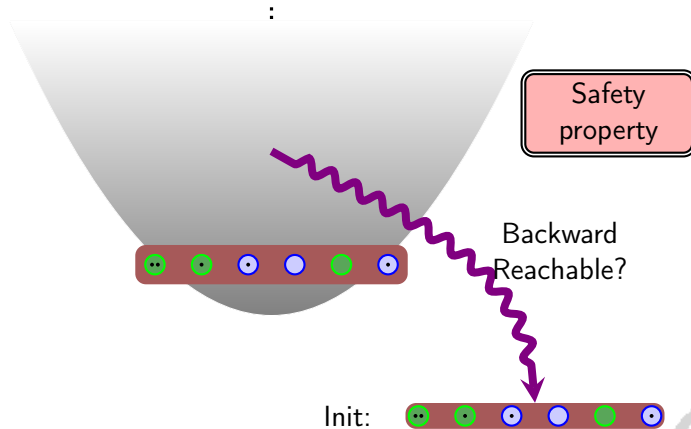
# Ordering



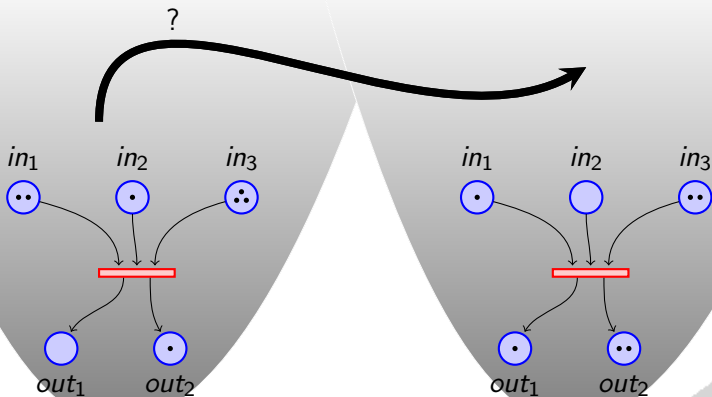
⊆



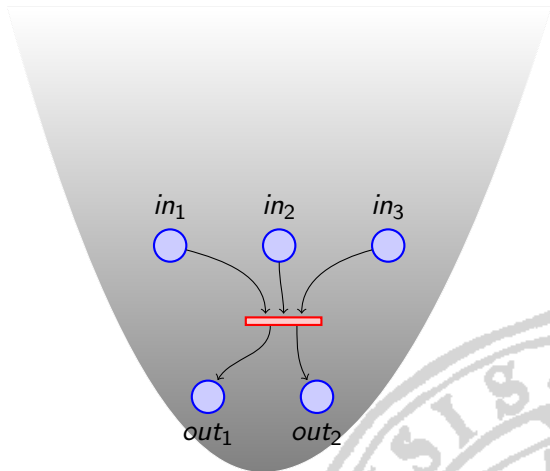
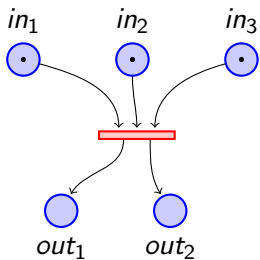
# Verification Method: Reachability



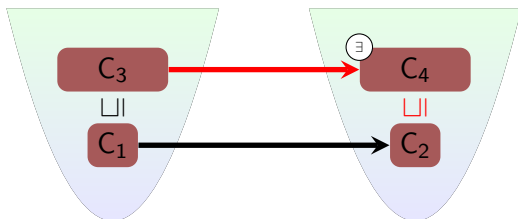
# Computing Pre



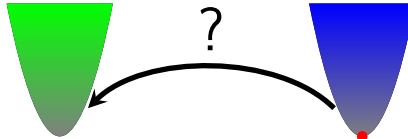
# Computing Pre



# Monotonicity



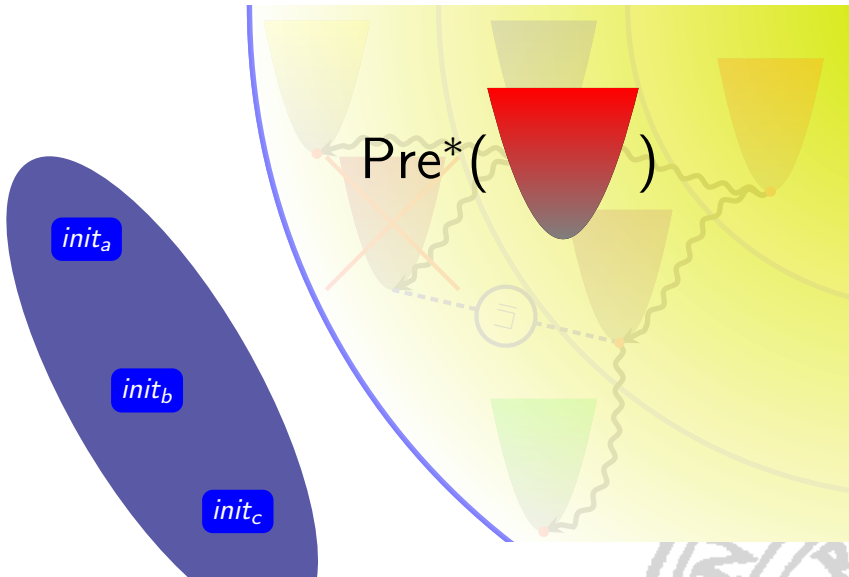
# Question!



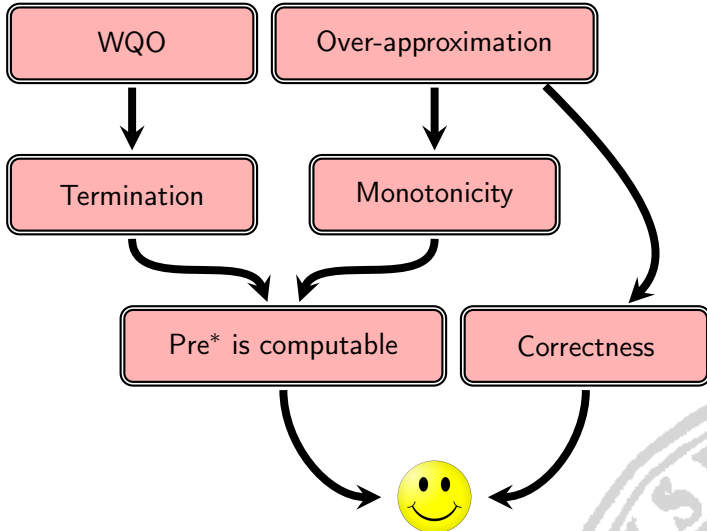
Is  $\text{Pre}(\text{set})$  an upward-closed set?

Is that enough to take  
to pre of the generator?

# Backward reachability analysis



# Termination and Correctness



# Experiments

Programs	#Conf.	#Subsum.	#Iter.	Safe?
Counter	10	3	4	-
CounterWithLock	14	7	6	✓
CheckThenAct	20	12	5	-
CheckThenAct-Lock	13	4	4	✓
Prods/Cons	310	645	19	✓
Prods/Cons 2	290	561	17	✓
Lock-ReadWriteOnly	25	13	8	✓

# Previous Work

- Mutual exclusion protocols, bus protocols, cache coherency protocols, telecommunication protocols
- Parameterized Tree Systems ← FORTE'08, Tokyo, June 2008
- Shape Analysis ← CAV'08, Princeton, July 2008



# Conclusions

- Race-Freeness: Safety Property
  - Backward Reachability analysis
  - Symbolic representation
- 

- Just the beginning...
- Analyze the race conditions themselves
- No annotations

## Future Work

- More pthreads constructs and maybe other libraries
- Usefulness of the method: false positives and coverage
- Compare with other tools
- Better Error reporting when a race is found