Scaling Model Checking of Dataraces Using

Dynamic Information

Ohad Shacham

Tel Aviv University and IBM

Mooly Sagiv
Tel Aviv University

Assaf Schuster
Technion

Datarace

- Happens when two threads access a memory location concurrently
    - At least one access is a write
- Unpredictable results
- Can indicate bugs
- Hard to detect
- Hard to reproduce

Datarace example

```
TicketPurchase(NumOfTickets)
{
    if (NumOfTickets = FreeTickets)
         FreeTickets -= NumOfTickets
    else
         Print “Full”;
}
```

Datarace example

{FreeTickets = 4}

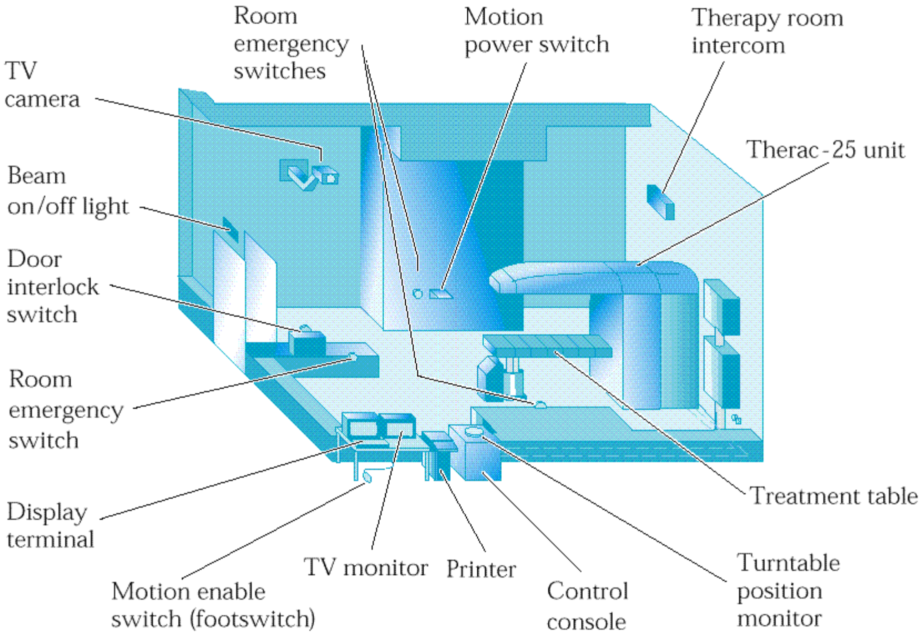
Thread I	**Thread II**
TicketPurchase(2)	
if (NumOfTickets = FreeTickets)	
	TicketPurchase(4)
	if (NumOfTickets = FreeTickets)
FreeTickets -= NumOfTickets	
	FreeTickets -= NumOfTickets
	{FreeTickets = -2}

Datarace example

```
TicketPurchase(NumOfTickets)
{
    Lock($lock^{FreeTickets}$)
    if (NumOfTickets = FreeTickets)
        FreeTickets -= NumOfTickets
    else
        Print “Full”;
    Unlock($lock^{FreeTickets}$)
}
```

Therac 25

- A medical radiation machine to treat cancer
- 6 patients got a radiation overdose
  - 4 died
  - 2 injured



Datarace detection

- Static datarace detection tools
  - Racex
  - rccjava
- Dynamic datarace detection tools:
  - Lamport’s *happens-before* partial order (Djit)
  - Lock based techniques (Lockset)

Difficulties in model checking dataraces

- Infinite state space
- Huge number of interleavings
- Huge transition systems
- Size problem

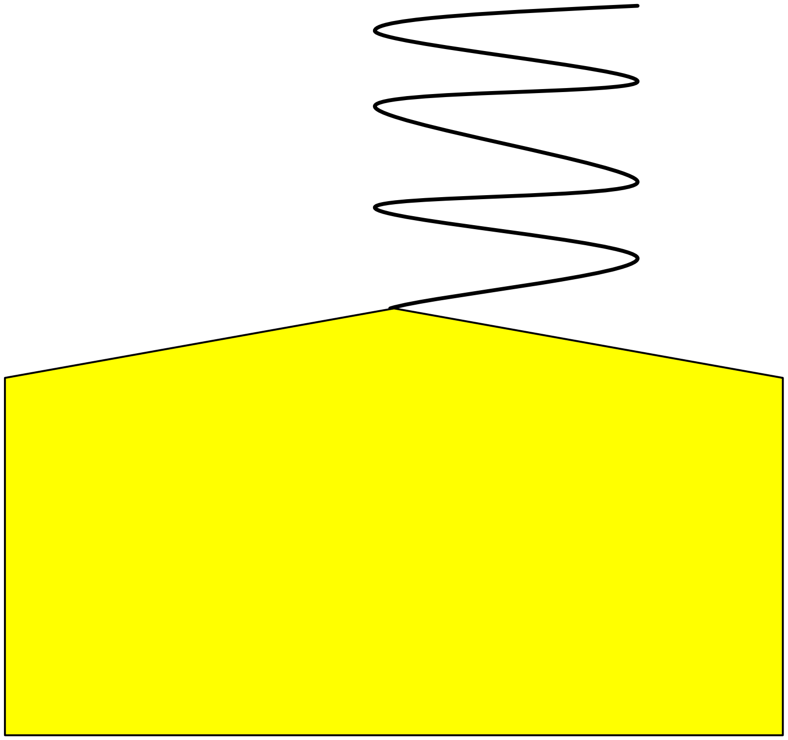
Observation

Dataraces free programs maintain a locking discipline

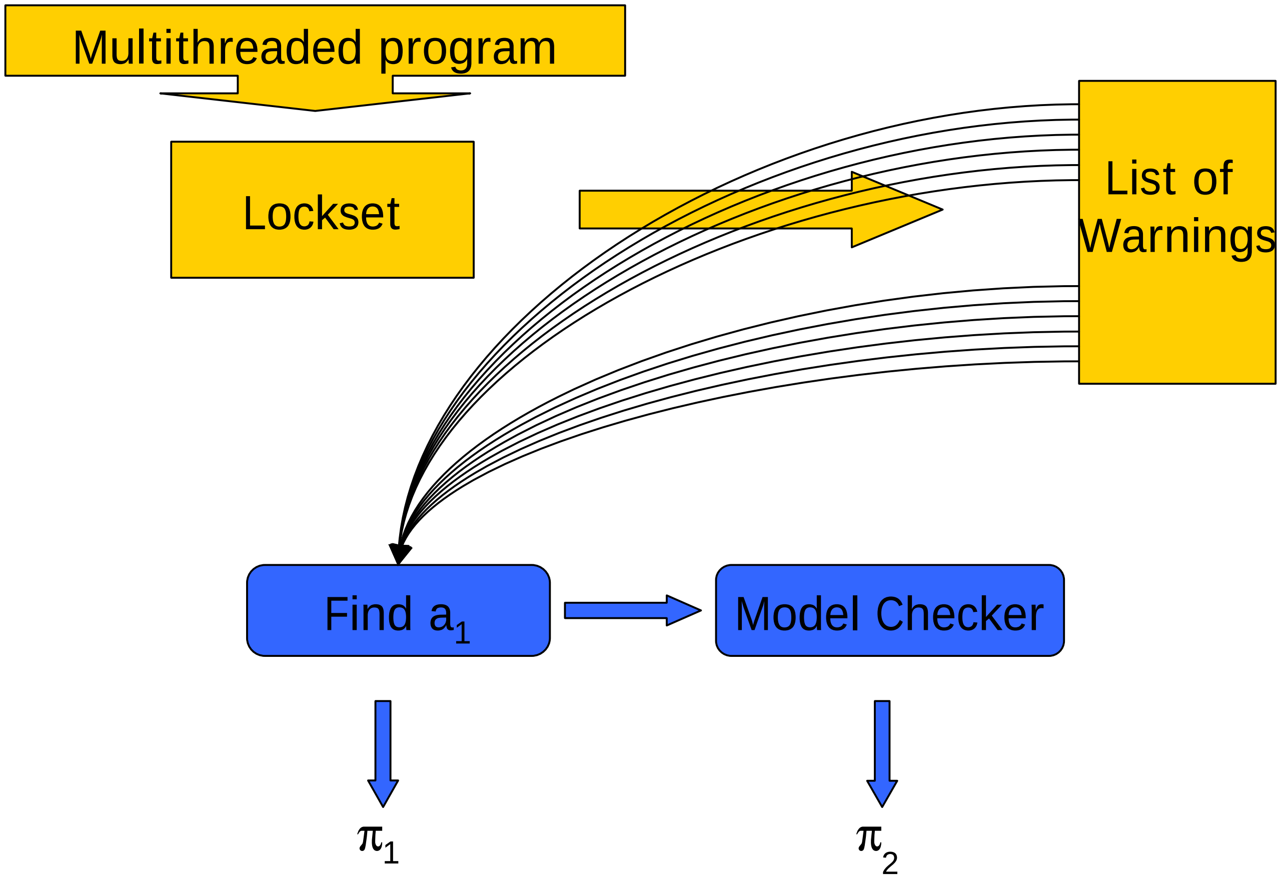
Hybrid solution

- Dynamically check the locking discipline
- Produce witnesses for dataraces using a model checker
    - Explore suffixes of the trace

Basic idea



Algorithm flow



Lockset invariant

Multiple accesses to a specific memory location are guarded by a unique lock

Lockset example

Thread I		**Thread II**		**C(X)**
Lock(lock^x)	f			{lock^x, lock^y}
X = 7	{lock^x}			{lock^x}
Unlock(lock^x)				
Lock(lock^y)	f			
Z = Y	{lock^y}			
Unlock(lock^y)				
		Lock(lock^y)	ϕ	
		Y = 2	{lock^y}	
		Unlock(lock^y)		
		Lock(lock^y)	ϕ	
		Y = X	{lock^y}	f

Lockset

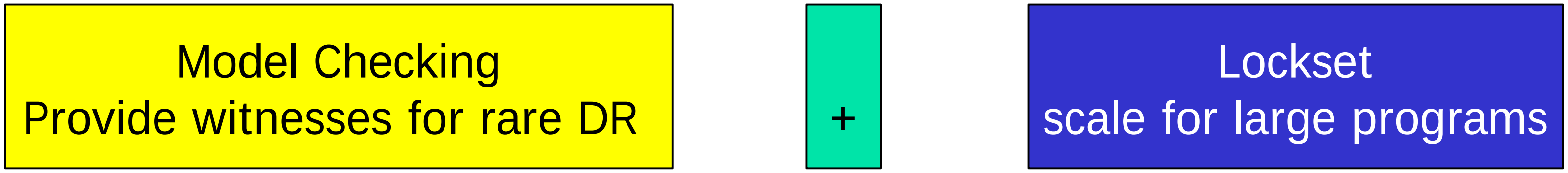
- Advantage
  - Predict dataraces which may occur in a different thread interleaving
- Disadvantages
  - Spurious dataraces
  - Hard to use
    - Lack of trace

Lockset strength

Thread I		**Thread II**		**C(X)**
Lock(lock^x);	f			{lock^x, lock^y}
X = 7;	{lock^x}			{lock^x}
Unlock(lock^x);				
Lock(lock^y);	f			
Z = Y;	{lock^y}			
Unlock(lock^y);				
		Lock(lock^y);	ϕ	
		Y = 2;	{lock^y}	
		Unlock(lock^y);		
		Lock(lock^y);	ϕ	
		Y = X;	{lock^y}	f

Our solution

- Combine Lockset & Model Checking
  - Provide witnesses for dataraces
    - Rare dataraces
    - Dataraces in large programs

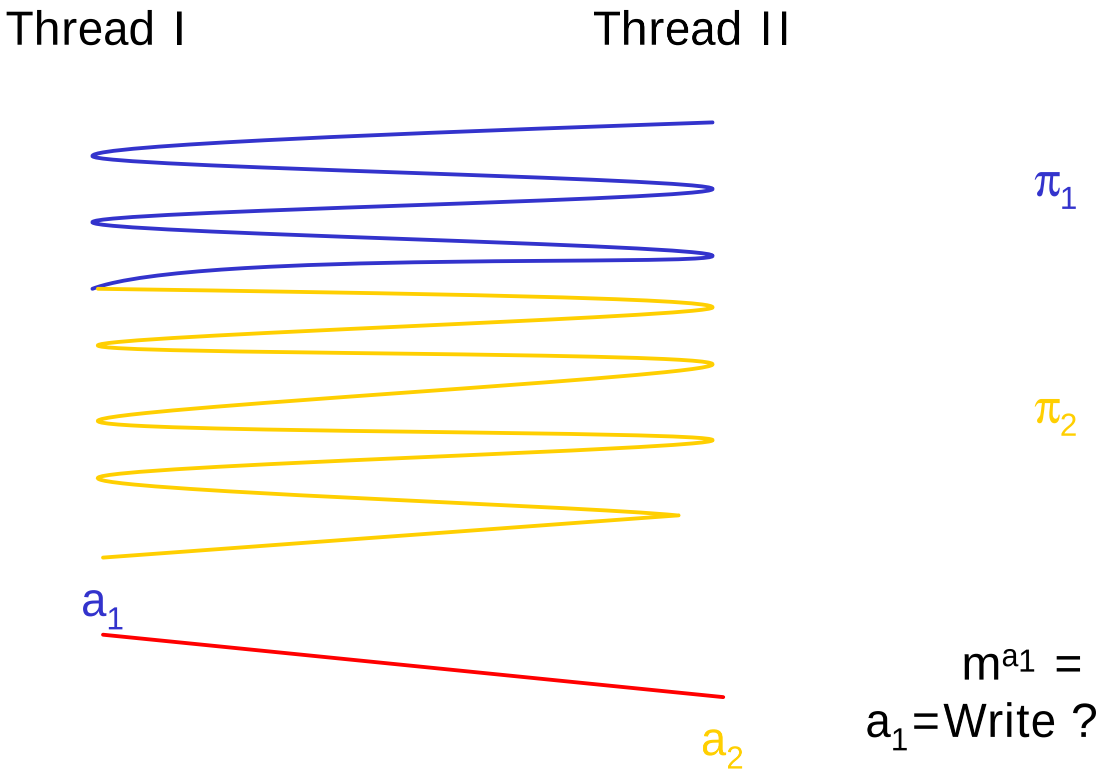


Model Checking
Provide witnesses for rare DR

+

Lockset
scale for large programs

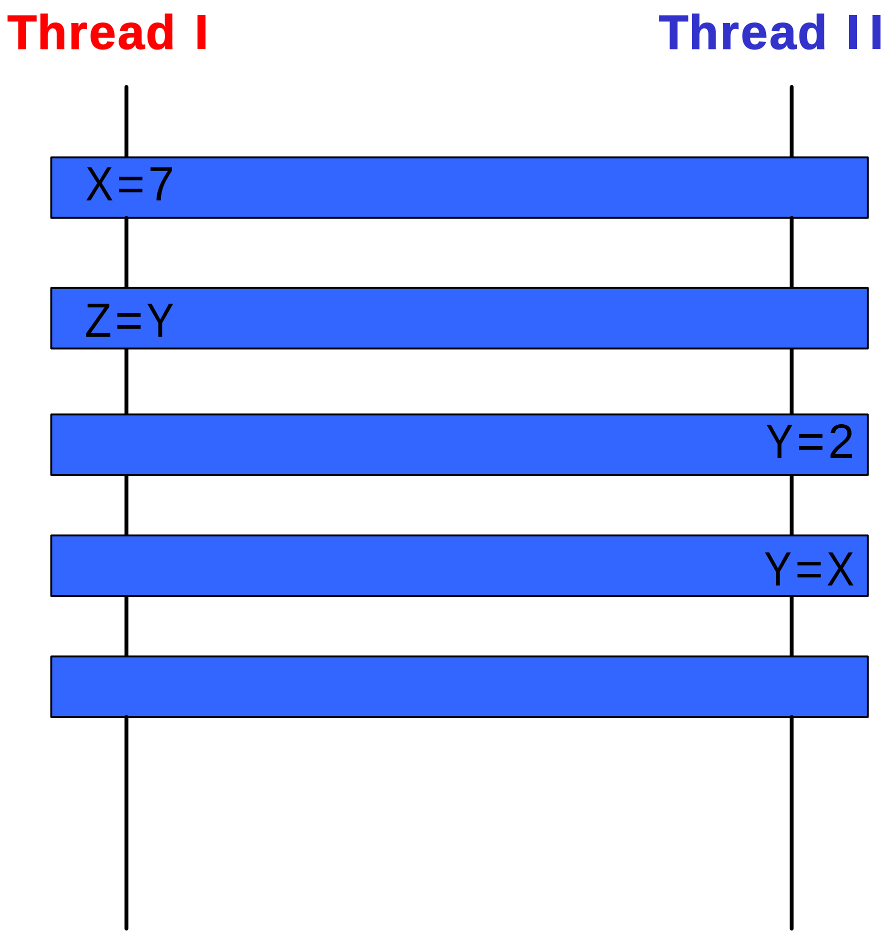
A witness for a datarace



$m^{a1} = m^{a2}$

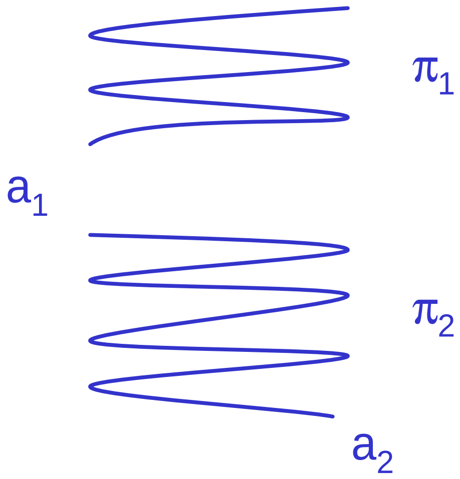
a_1=Write ? a_2=Write

Required data from Lockset



Using Lockset data

- Lockset provides for each warning only a single access event a_2



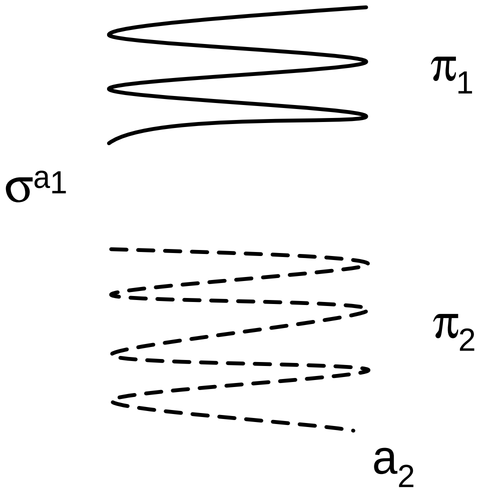
- Find a prior access event a_1 which can take part in a race with a_2

Using Lockset data

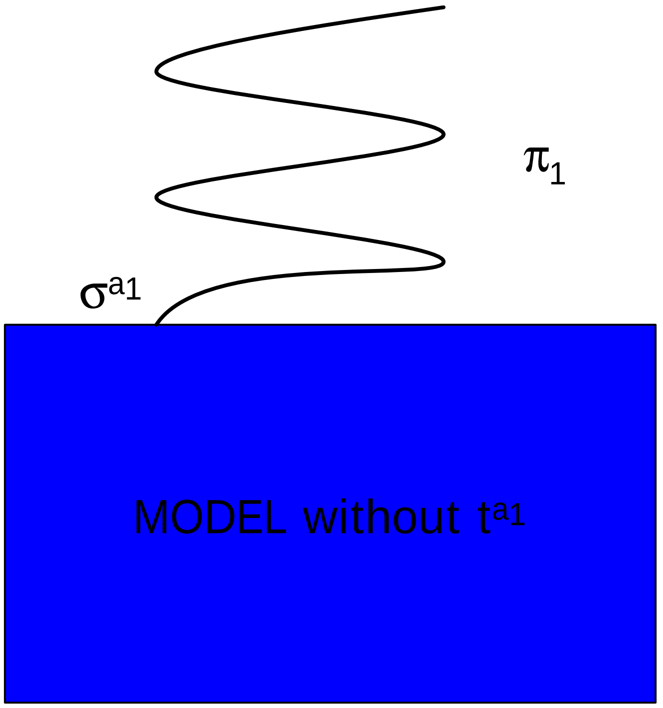
X = 7	{lock^x}	X=7
Z = Y	{lock^y}	Z=Y
Y = 2	{lock^y}	Y=2
Y = X	{lock^y}	Y=X

A Warning on X

Prefix



Building a model

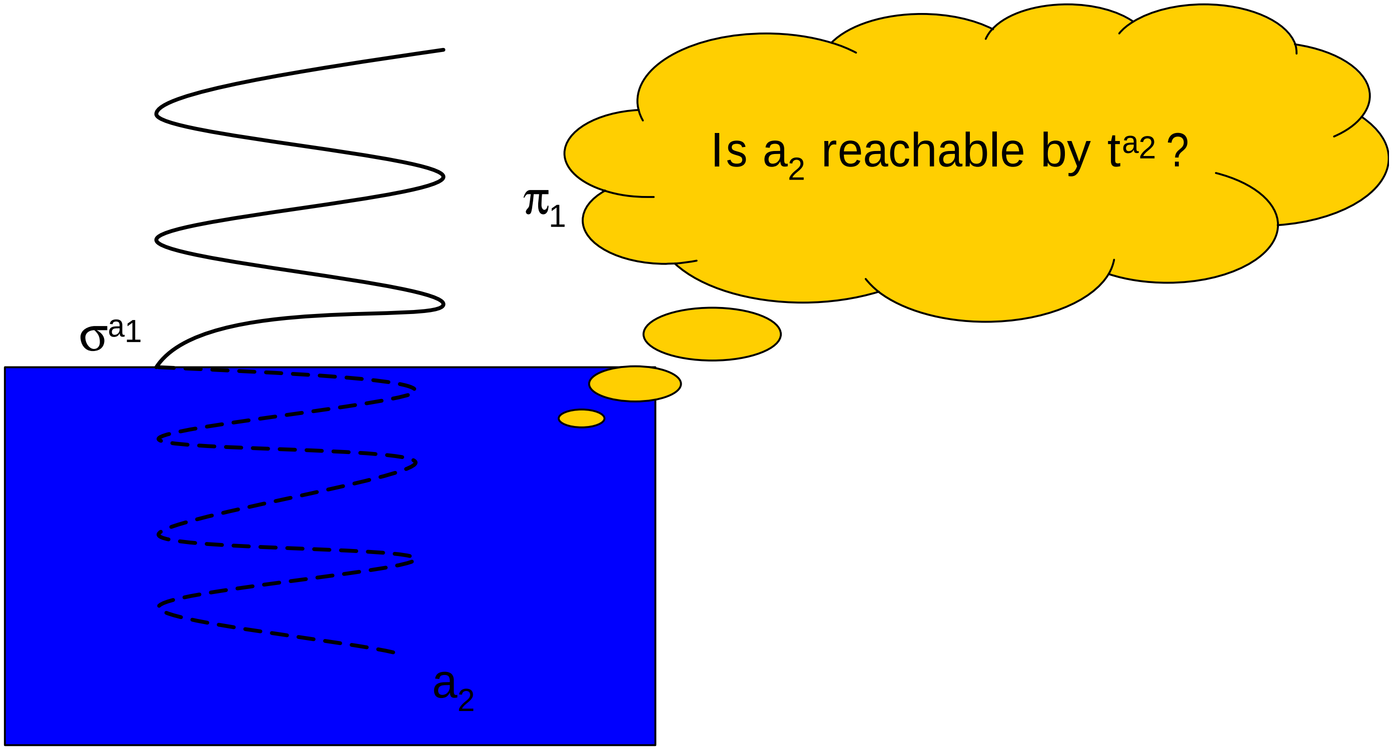


π_1

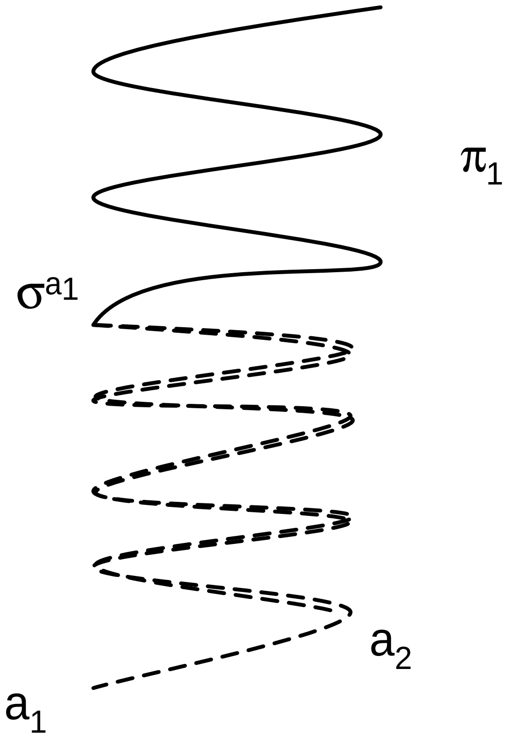
σ^{a1}

MODEL without t^{a1}

Using a model checker



Using a model checker



Reduce the model checker cost

- Reduction in the model size
- Elimination of thread t^{a_1}
- Providing a single new initial configuration
- Heuristically reducing the number of steps that the model checker should carry out

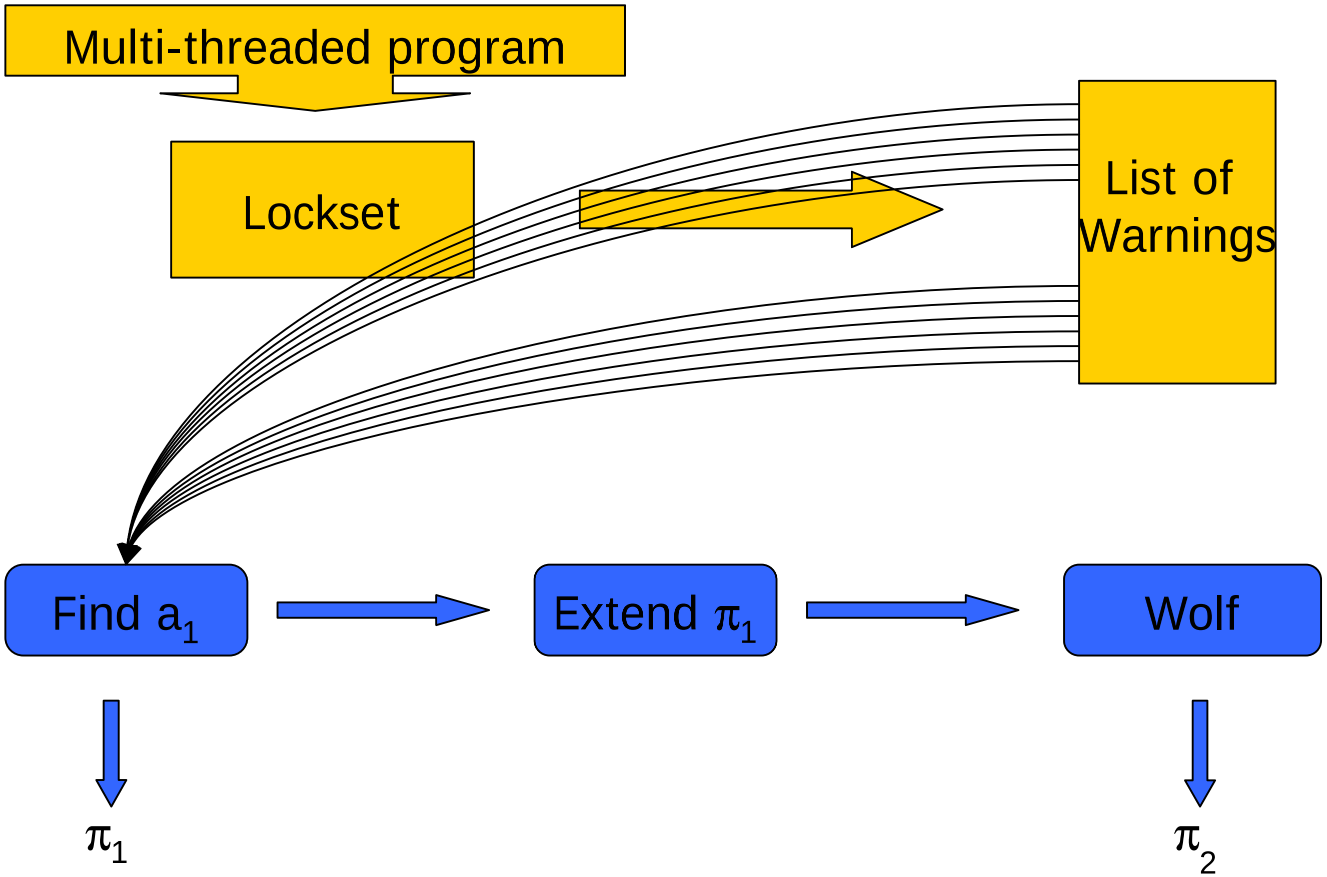
Example

	Thread I		**Thread II**		C(X)
π_1	{ Lock(lock^x);	f			{lock^x, lock^y}
	X = 7;	{lock^x}			{lock^x}
	Unlock(lock^x);				
			Lock(lock^y);		
	Lock(lock^y);	f			
	Z = Y;	{lock^y}	Y = 2;		
	Unlock(lock^y);		Unlock(lock^y);	} π_2	
			Lock(lock^y);		
			Y = X;		
	X = 7;		Lock(lock^y);		
			Y = 2;	{lock^y}	
			Unlock(lock^y);		
			Lock(lock^y);	ϕ	
			Y = X;	{lock^y}	f

Prototype implementation

- A prototype tool based on IBM tools
- Lockset – The IBM Watson tool
- Wolf – IBM Haifa’s software model checker

Prototype implementation



Benchmark programs

Program	Description	Lines
Tsp	traveling salesman from ETH	706
Our_tsp	Enhanced traveling salesman	708
mtrt	Multithreaded raytracer from specjvm98	3751
Hedc	Web Crawler Kernel from ETH	29948
SortArray	Parallel sort	362
PrimeFinder	Finds prime numbers in a given interval	129
Elevsim	Elevator simulator	150
DQueries	Shared DB simulator	166

Experimental results

Program	2 threads		3 threads		4 threads	
	Time (sec)	Memory (MB)	Time (sec)	Memory (MB)	Time (sec)	Memory (MB)
our_tsp	35069		Mem Out		Mem Out	
SortArray	569.3	123	1334.93	396	Mem Out	
PrimeFinder	888.7	116	2645.5	143	4547.1	168
ElevSim	33.02		67.92	33	147.9	48
DQueries	140.1	60	201.8	89	585.97	136
Hedc	2.66	11	7.33	12	9	17
tsp	35243	377	Mem Out		Mem Out	

Conclusion

- Hybrid technique which combines dynamic datarace detector and a model checker
- Provide witnesses for dataraces which occur only in rare interleavings
- Helps the user in analyzing the datarace
- No spurious dataraces