Thread Modular Configurable Program Analysis

Pavel Andrianov, andrianov@ispras.ru

Motivation

Linux module drivers/net/irda/w83977af_ir.ko: 10 000 LOC

```
static void w83977af_change_speed(struct
w83977af_ir *self , __u32 speed ){
  ...
  self->io.speed = speed;
  ...
}
```

```
static void w83977af_hard_xmit(struct
sk_buff *skb , struct net_device *dev){
  ...
  speed = irda_get_next_speed(skb);
  tmp_speed = self->io.speed;
  assert(self->io.speed == tmp_speed);
  if ((speed != self->io.speed) && ...) {
  …
  }
}
```

SMACK: memory limit
CBMC: time limit
Yogar-CBMC: segmentation fault
Mu-Cseq: –, UNKNOWN
CPALockator: 15 sec

Existing approaches

Fast static analysis

RELAY, Locksmith,...

Unsound bug finding

Precise model checking

CBMC, SMACK,...

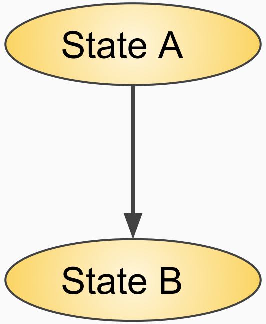
Theoretically sound approach

Adjustable combination?

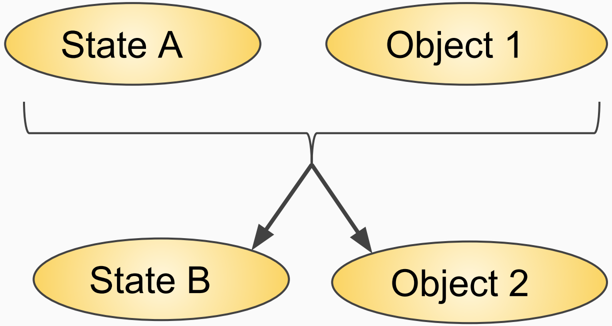
The goals of a new theory

An idea for theory extension

Introduce new objects: inference objects, which describe applied action.



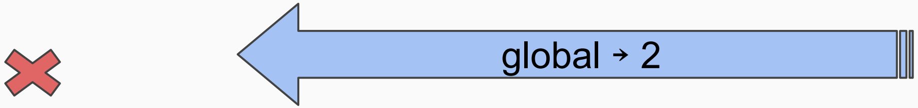
Classic transfer



Extended transfer

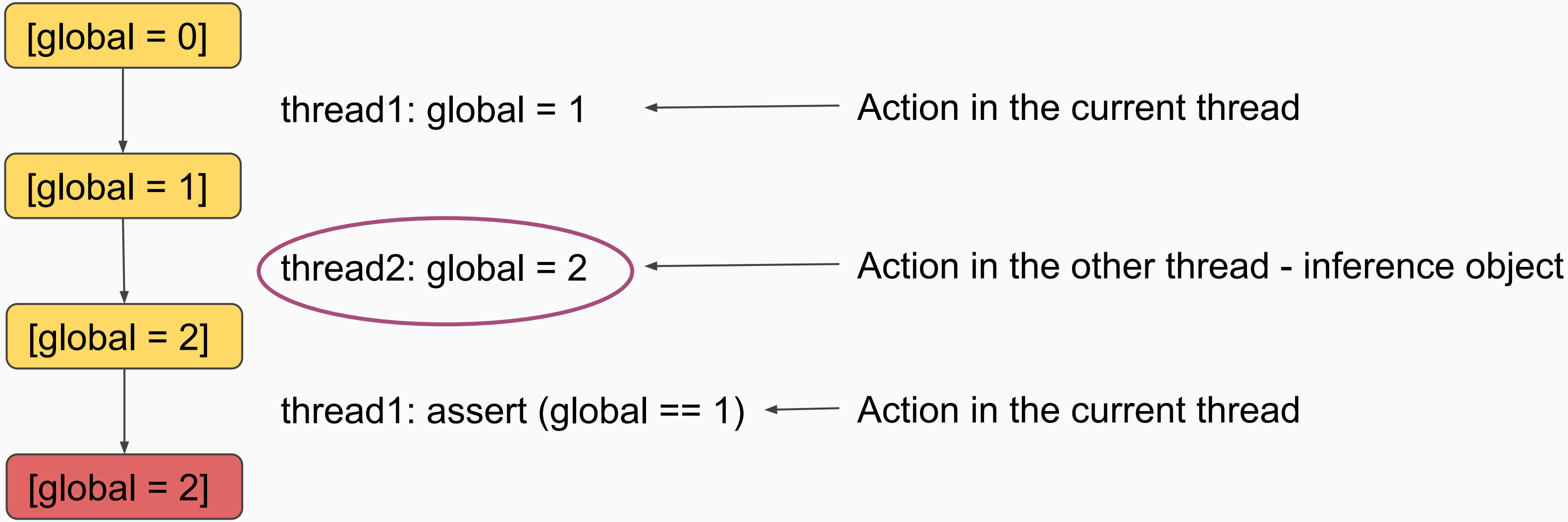
An example

```
thread1(..):

…

global = 1;

assert(global == 1);
```



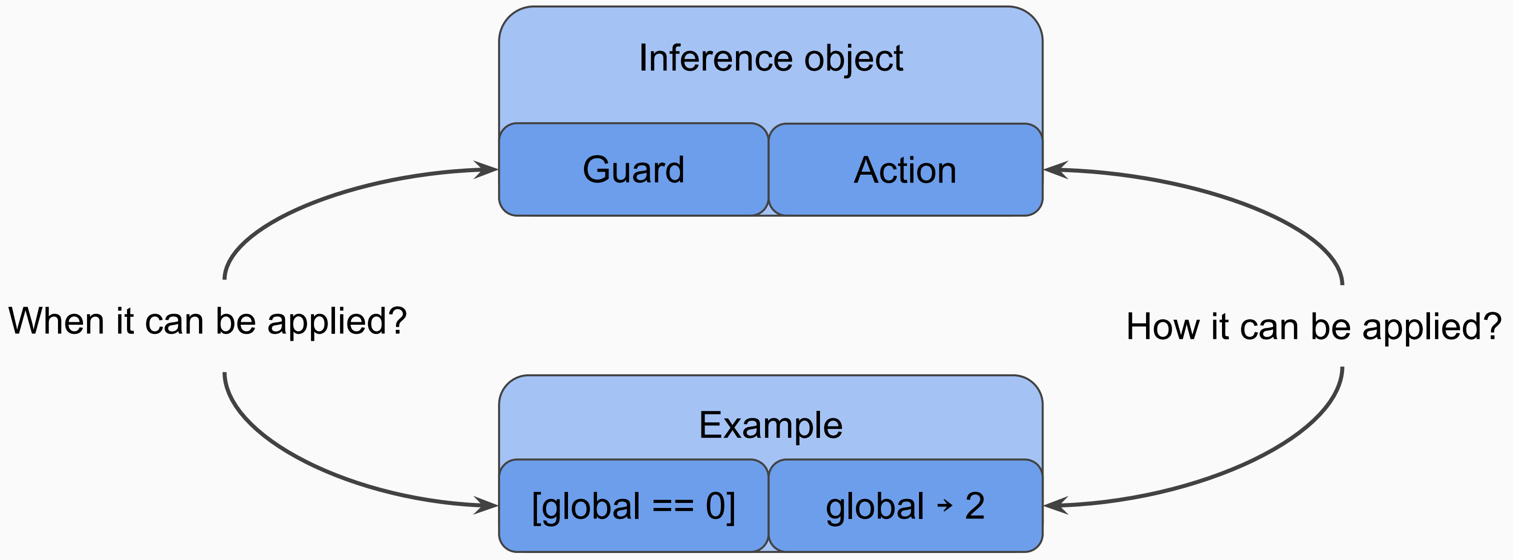

```
thread2(..):

…

global = 2;

…
```

An example

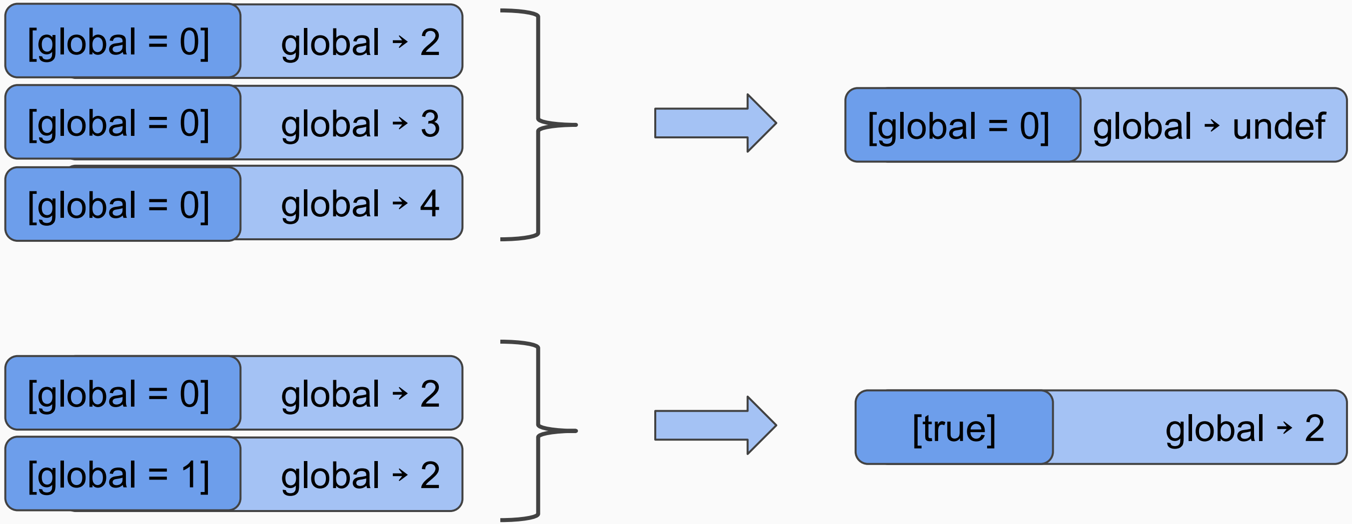


[global = 0]
thread1: global = 1
Action in the current thread
[global = 1]
thread2: global = 2
Action in the other thread - inference object
[global = 2]
thread1: assert (global == 1)
Action in the current thread
[global = 2]

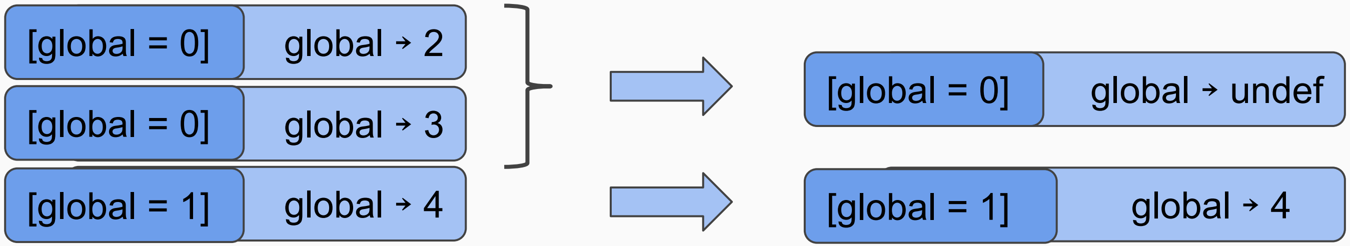
Structure of an inference object



Balancing between speed and precision



Balancing between speed and precision



Two options for extension of the theory

ThreadModular1

```
waitlist := frontier(∅, ∅, e_0, π_0);
reached := {e_0, π_0};
while waitlist ≠ ∅ do
    pop R̂ from waitlist;
    for each e' in R̂ ⇝ (e', π') do
        (ê, ê) = prec(e', π', reached);
        for each (e'', π'') ∈ reached do
            e_new = merge(ê, e'', π̂);
            if e_new ≠ e'' then
                waitlist := update(waitlist, reached, e'', π'', e_new, π̂);
                reached := reached \ {(e'', π'')} ∪ {(e_new, π̂)};
            end
        end
        if !stop(ê, reached, π̂) then
            waitlist := waitlist ∪ frontier(reached, ê, π̂);
            reached := reached ∪ {(ê, π̂)};
        end
    end
end
```

ThreadModular2

```
waitlist := {e_0} reached := {(e_0, π_0)};
while waitlist ≠ ∅ do
    pop e from waitlist;
    for each e' in (e, reached) ⇝ (e', π') do
        (ê, π̂) = prec(e', π', reached);
        for each (e'', π'') ∈ reached do
            e_new = merge(ê, e'', π̂);
            if e_new ≠ e'' then
                waitlist := waitlist \ {(e'', π'')} ∪ {(e_new, π̂)};
                reached := reached \ {(e'', π'')} ∪ {(e_new, π̂)};
            end
        end
        if !stop(ê, reached, π̂) then
            waitlist := waitlist ∪ {(ê, π̂)};
            reached := reached ∪ {(ê, π̂)};
        end
    end
end
```

Comparison of the two approaches

ThreadModular1	ThreadModular2
An inference object is a special abstract state	A top-level abstract state is a pair of inner abstract state and an inference object
Waitlist is not a subset of a reached set	Waitlist is still a subset of a reached set
No problems with ARG implementation	Conflicting ARG and ThreadModular CPAs, which one should be top-level
Theoretical requirements are provided and a theorem about soundness was proven	

Linux drivers with known bugs

Approach	ThreadModular	ThreadModular2	Threading
False verdicts			
Correct	12	0	2
Incorrect	0	0	1
True verdicts	12	0	0
Unknowns	8	32	29
Time(s)	10 200	29 000	23 500

SV-COMP benchmarks

Approach	ThreadModular	ThreadModular2	Threading
False verdicts			
Correct	789	11	767
Incorrect	199	46	2
True verdicts	33	0	163
Unknowns	26	990	115
Time(s)	28 400	862 000	63 000

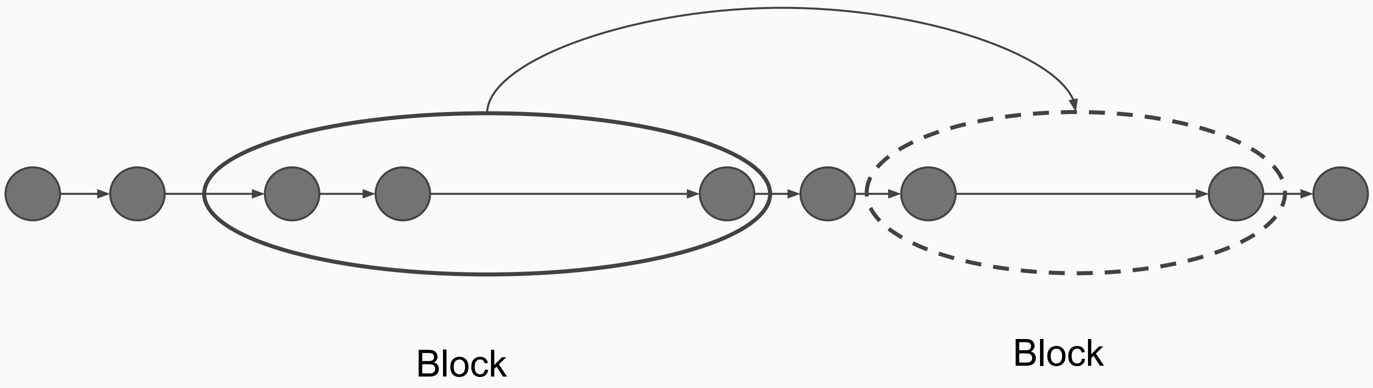
Pros and Contras

- The first way is fast, as it operates with states and inference objects distinctly
- The second requires less changes in basic algorithm
- Both of the options require a lot of changes in the CPAchecker core: reached set and waitlist, algorithms, CPA operators.

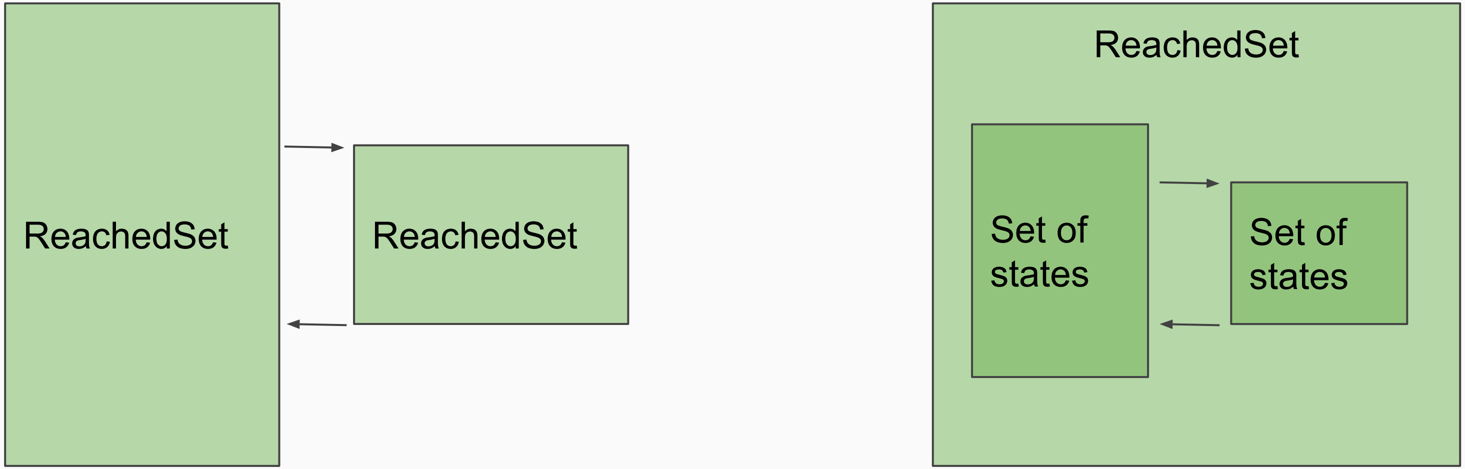
Other problems with theory

- ARG does not satisfy the theory (side-effects of operators)
- BAM does not operate with global reached set
- Global refinement procedure

BAM logic



BAM logic



Old idea New idea

Cons of the new idea

- Reducing and expansion of all internally states?
- Partial cache hits - is it possible?
- Parallel BAM - is it still real?

Conclusion

- The first way of the theory shows better results
- Discussion about the problem is open and welcome

Copy on Write Refinement (BAM-COW)

-noout -setprop statistics.memory=true -heap 13000M -ldv-bam -setprop cpa.predicate.useMemoryRegions=true -setprop cpa.predicate.bam.usePrecisionReduction=false -setprop cpa.predicate.bam.useAbstractionReduction=false -setprop cpa.predicate.refinement.predicateBasisStrategy=all					-noout -setprop statistics.memory=true -heap 13000M -ldv-bam -setprop cpa.predicate.useMemoryRegions=true -setprop cpa.predicate.bam.usePrecisionReduction=false -setprop cpa.predicate.bam.useAbstractionReduction=false -setprop cpa.predicate.refinement.predicateBasisStrategy=all -setprop cpa.bam.useCopyOnWriteRefinement=true				
cputime (s)	host	memUsage	status	walltime (s)	cputime (s)	host	memUsage	status	walltime (s)
901	felchbach	8060493824	timeout	794	132	apollon002	4389015552	false(unreach-call)	78.2
834	apollon021	7145263104	false(unreach-call)	691	901	nau	7901581312	timeout	763
901	apollon158	6196330496	timeout	796	901	apollon145	5992124416	timeout (assertion)	805
900	apollon016	7149158400	timeout	778	164	nassach	5315604480	true	104
93.4	apollon057	4380151808	false(unreach-call)	57.8	901	apollon093	5752844288	timeout	831
668	kirnach	11520987136	true	511	905	geltnach	13944238080	timeout	548
180	apollon159	5493026816	true	115	901	nau	12460052480	timeout	698
384	osterbach	7269740544	true	300	901	apollon082	7399866368	timeout	780
189	apollon075	5913460736	true	118	901	naab	10569953280	timeout	725
40.3	apollon148	2387091456	false(unreach-call)	21.8	948	krassach	14115983360	timeout	696
744	ranna	7710564352	true	650	901	frommbach	8057307136	timeout	795
624	sandrach	6283059200	true	548	903	apollon110	6813827072	timeout	790
901	apollon121	10880585728	timeout	687	689	apollon111	10321993728	false(unreach-call)	500
103	apollon143	2891935744	true	57.8	901	apollon048	11784228864	timeout	672
855	haselgraben	7469555712	true	759	902	nassach	7552999424	timeout	796
cputime (s)	host	memUsage	status	walltime (s)	cputime (s)	host	memUsage	status	walltime (s)
8320	–	100751405056	15	6880	11900	–	132371619840	15	9580
2350	–	41722630144	6	1790	821	–	14711009280	2	578
1420	–	30197215232	4	1040	0	–	–	–	–
927	–	11525414912	2	749	821	–	14711009280	2	578
0	–	–	–	–	164	–	5315604480	1	104
0	–	–	–	–	164	–	5315604480	1	104
0	–	–	–	–	0	–	–	–	–
–	–	–	10	–	–	–	–	-30	–