Concurrent Program Verification With Invariant-guided Underapproximation

S. Prabhu P. Schrammel M. Srivas
M. Tautsching A. Yeolekar
presented by Jan Mrázek

Masaryk University
Brno, Czech Republic

12th March 2018

Bounded Model Checking

Basic idea of BMC for a system and a property ϕ:

- unroll the system up to k-step
  - number of context switches
  - number of loop iteration
  - ...
- transition function of the unrolled system as SAT or SMT
- if $\neg\phi \wedge P$ is:
  - unsat, property holds up to k steps
  - sat, property is violated

Bounded Model Checking

Basic idea of BMC for a system and a property ϕ:

- unroll the system up to k-step
  - number of context switches
  - number of loop iteration
  - ...
- transition function of the unrolled system as SAT or SMT
- if $\neg\phi \wedge P$ is:
  - unsat, property holds up to k steps
  - sat, property is violated

Observations:

- satisfiability check is the bottle neck
- solvers work the best when formula is under- or over-specified

Main Idea

Add more constraints to overspecify the formula.

Main Idea

Add more constraints to overspecify the formula.

- good candidates: program invariants
- possible candidates: likely invariants

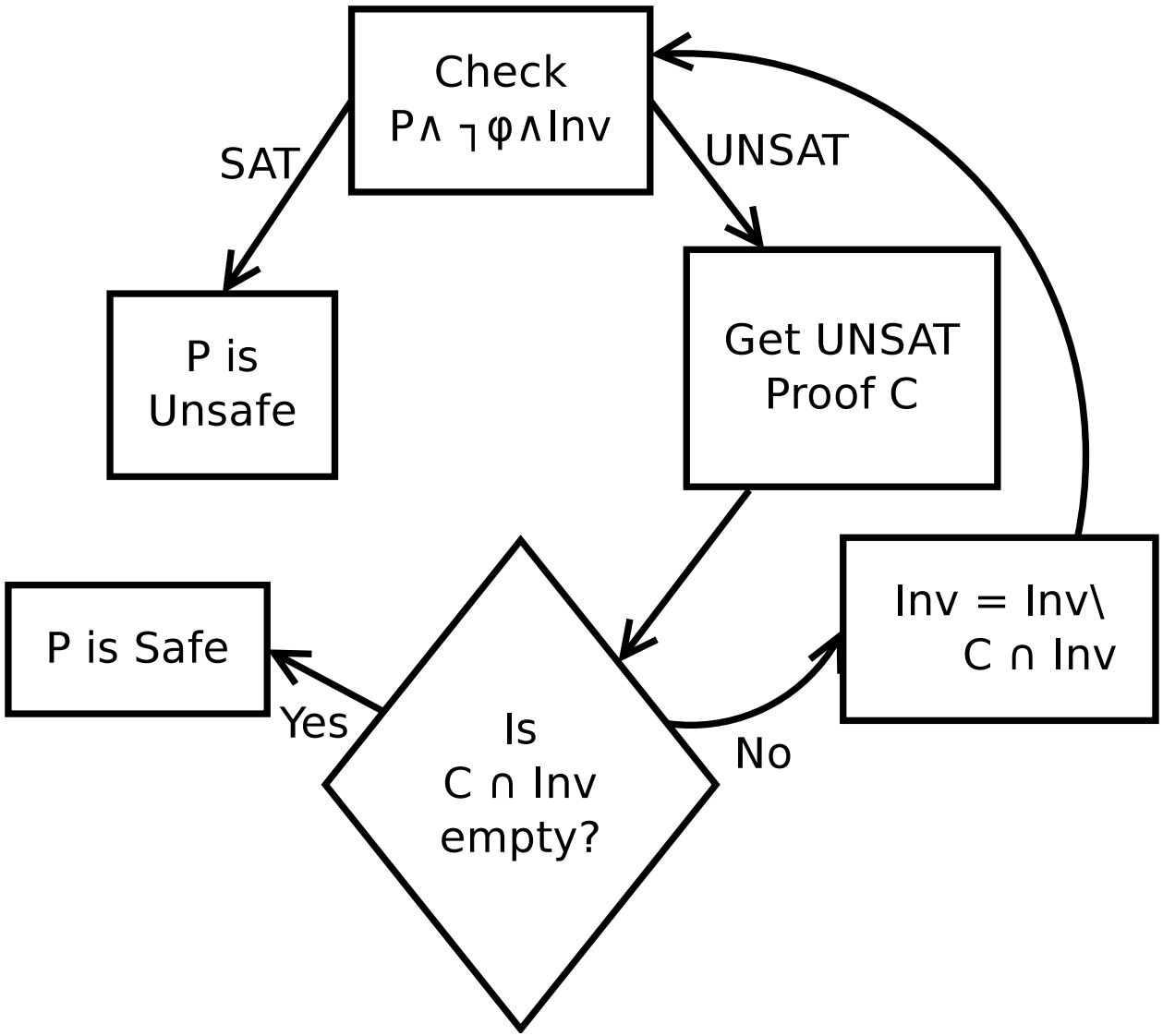
Main Idea

Add more constraints to overspecify the formula.

- good candidates: program invariants
- possible candidates: likely invariants

- simply add them to the conjunction
- the result might be underapproximation

BMC Underapproximation



Definition-Use Likely Invariants

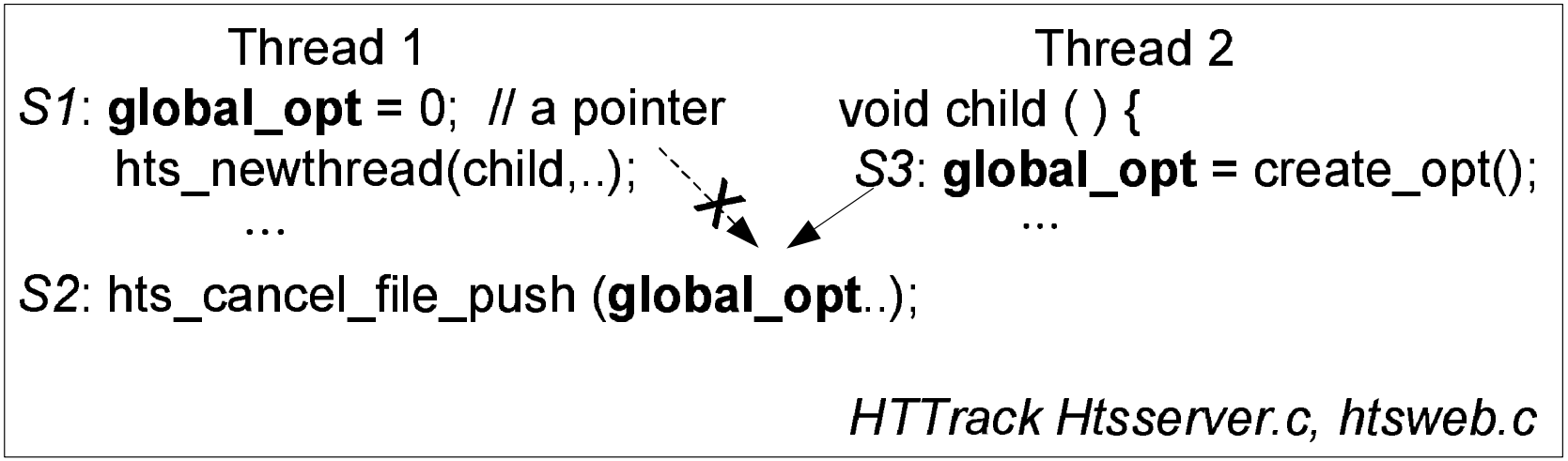
- paper: Do I Use the Wrong Definition? (Yao Shi et al.)
- tool DefUse

Definition-Use Likely Invariants

- paper: Do I Use the Wrong Definition? (Yao Shi et al.)
- tool DefUse

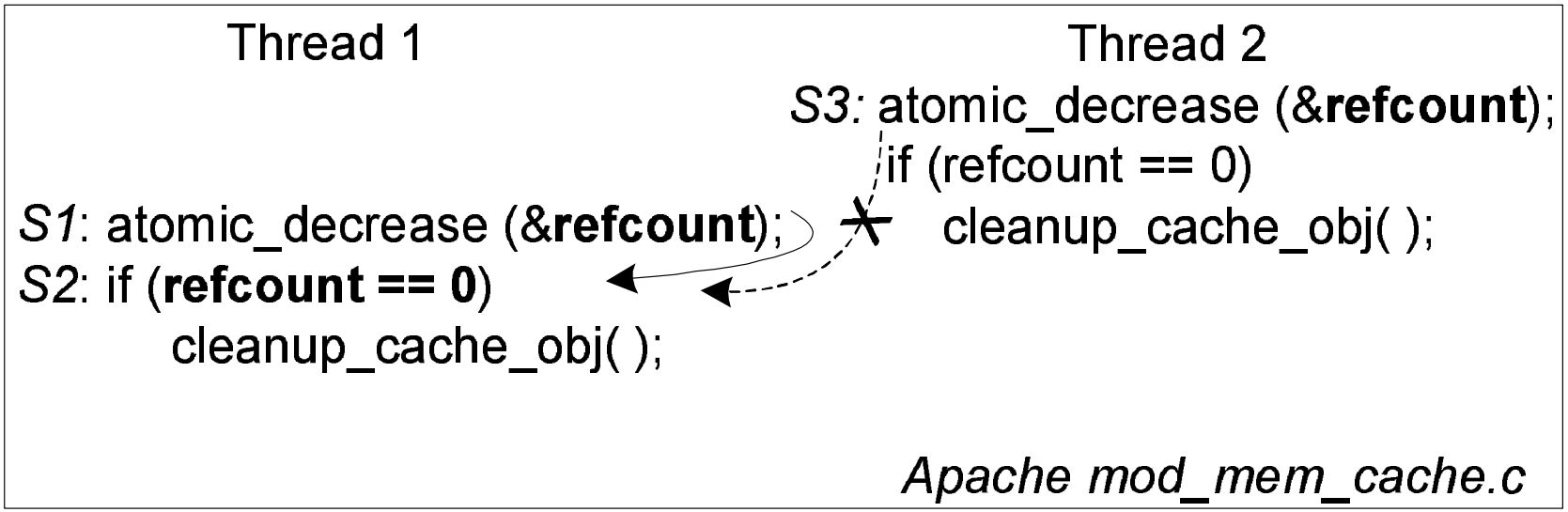
- invariants for reads and writes (load/store):
  - LR invariants
  - follower invariants
  - DSet invariants
- capturing likely invariants:
  - instrument a program
  - run it multiple times
  - collect data
  - statistically choose good candidates

Local/Remote Invariant Motivation



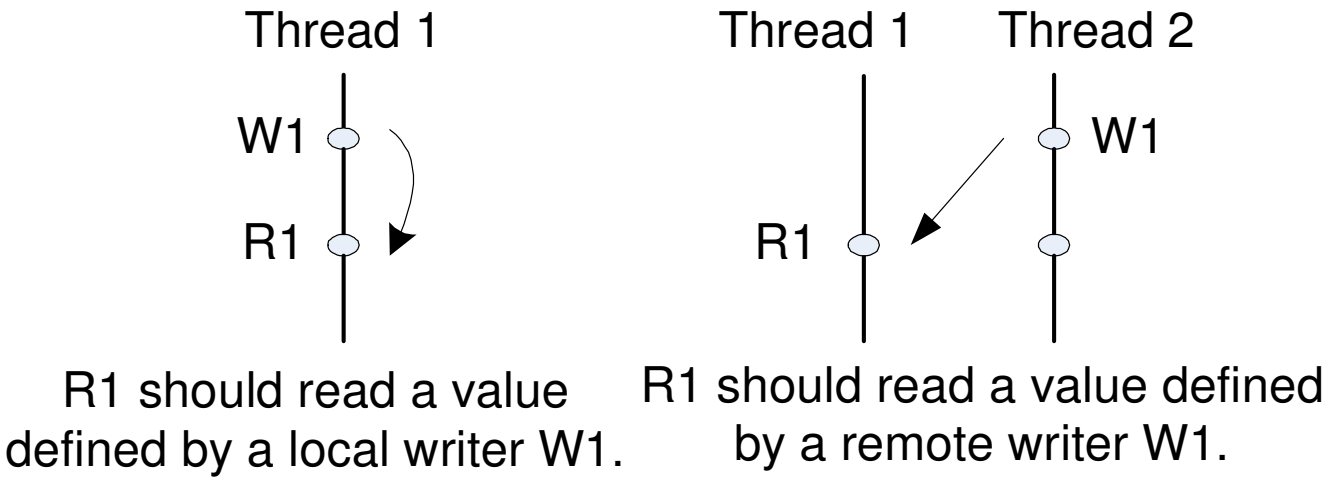
Root cause	Programmers' assumption that S3 always happens before S2 is not guaranteed in code.
Manifestation	S2 comes before S3 due to unexpectedly fast execution after thread creation.
Failure	Crash: null pointer dereference by *global_opt.*
Definition-use relation	S2 should use the remote definition of a *global_opt* by S3.

Local/Remote Invariant Motivation



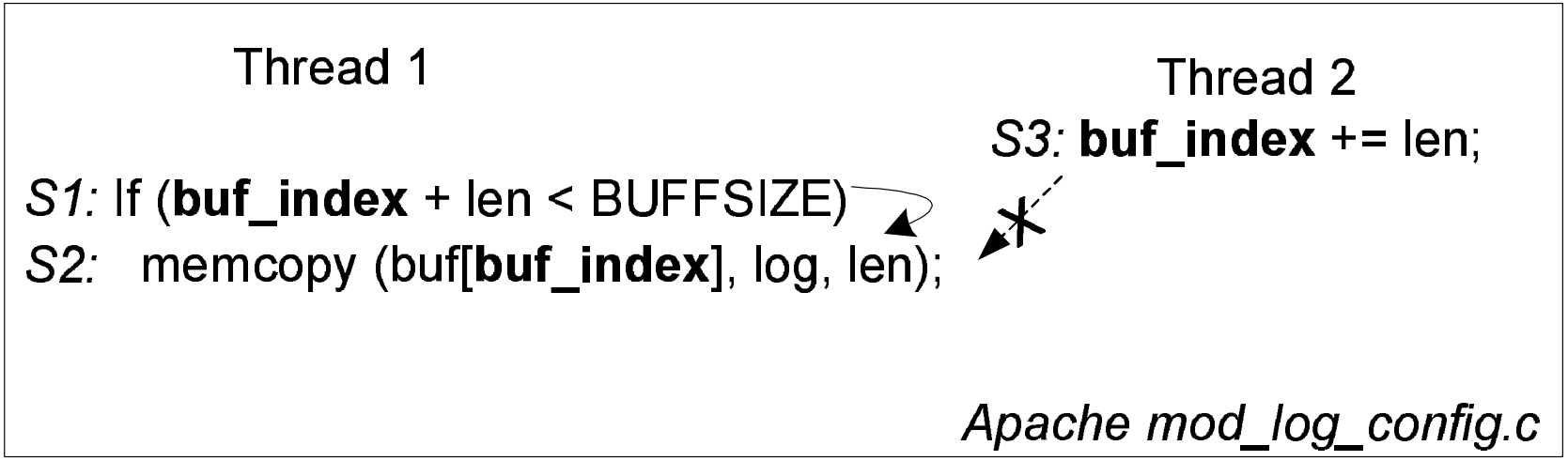
Root cause	S1 and S2 are not protected within an atomic section.
Manifestation	S2 uses the remote definition by S3 due to a wrong thread interleaving.
Failure	Crash: both threads try to free the same cache object (either dangling pointer or double free).
Definition-use relation	S2 should use the local definition of *refcount* By S1.

Local/Remote Invariant



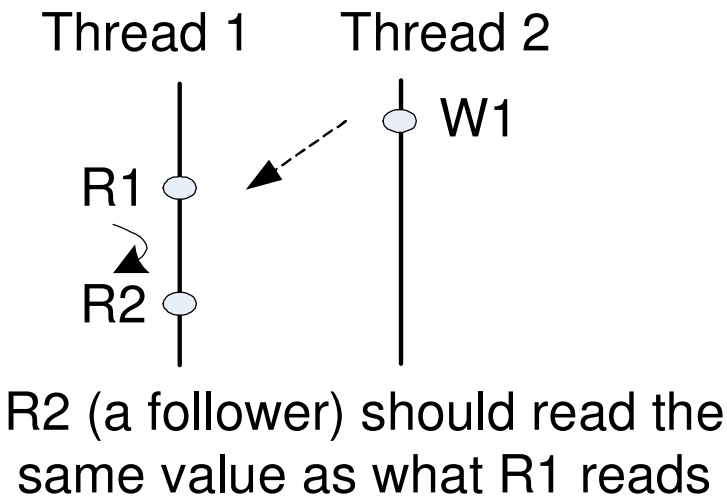
- read should get data **only** from writes:
  - in the same threads
  - in other threads

Follower Invariant Motivation



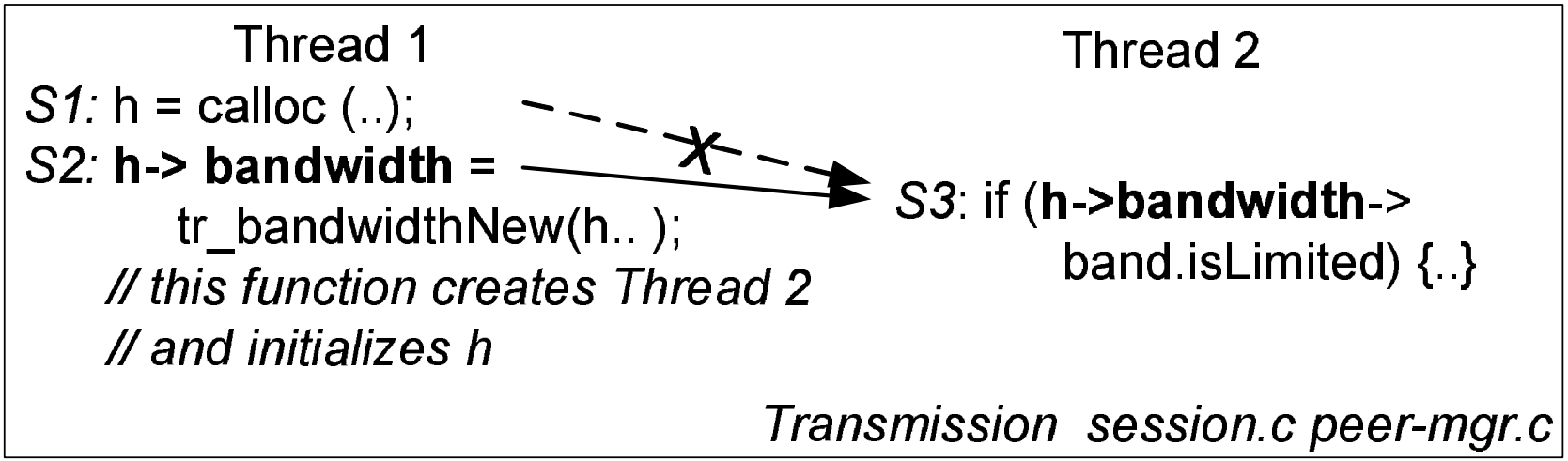
Root cause	S1 and S2 are not protected within an atomic section.
Manifestation	S2 uses updated *buf_index* by S3 after the array bound checking in S1.
Failure	Wrong results (corrupted log file) or crash.
Definition-use relation	S2 should consume the same definition of *buf_index* as S1 used.

Follower Invariant



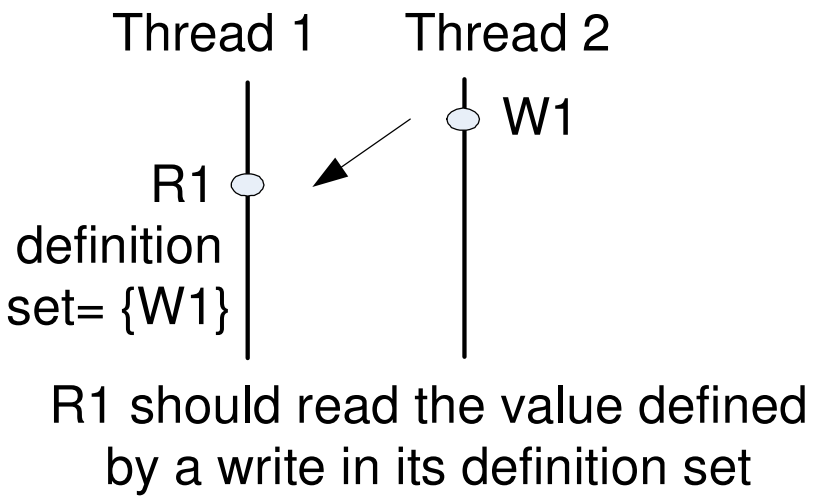
- atomicity assumption
- two following reads should get data from the same write

Definition Set Invariant Motivation



Root cause	Programmers' assumption that S2 always happens before S3 is not guaranteed in code.
Manifestation	S3 comes before S2 sets *h->bandwidth* properly and uses the definition from S1.
Failure	Crash: null-pointer dereference by *h->bandwidth*.
Definition-use relation	S3 should use the definition only from S2, not from S1.

Definition Set Invariant



- restricts the possible values a read can get
- define a set of writes

Implementation

- instrument reads and writes in the verified program
  - PIR instrumentation framework
- run it with random inputs
  - possibly with guided thread scheduling: CHESS, CTrigger
- construct the likely invariants based on ranks

Implementation Challenges

- monitored memory locations
  - granularity (DefUse: a byte)
  - memory location (DefUse: heap only)
- external definitions
  - external functions might loose invariants (`memset`, `memcpy`)
  - add annotation
- virtual address recycle
  - due to deallocations
  - intercept deallocation
- context sensitivity
  - small uninlined functions used by different threads
- training noise
  - incorrect oraculum

Results

- 50 random inputs and random interleaving for invariant mining
- LI = tool from the paper
- NoR = encoding without unnecessary writes

File	Type	U	CBMC	LI	Refinement	Writes Saved	NoR
1.c	Unsafe	10	14.06s	13.541s	87 to 87 in 1	1235/2390	*21.719s*
2.c	Unsafe	10	2.835s	2.034s	28 to 28 in 1	450/912	1.734s
3.c	Unsafe	20	21.127s	10.359s	58 to 58 in 1	1900/3727	*16.87s*
4.c	Safe	16	39.633s	23.987s	107 to 88 in 5	1266/4489	6.818s
5.c	Unsafe	16	28.273s	34.923s	93 to 93 in 1	2415/3710	5.813s
6.c	Unsafe	21	15.984s	11.416s	42 to 42 in 1	1720/3144	*12.832s*
7.c	Safe	6	48.716s	44.519s	22 to 0 in 4	0/599	0.598s
8.c	Unsafe	11	4.567s	5.909s	32 to 32 in 1	685/1194	5.41s
9.c	Unsafe	10	31.835s	17.196s	76 to 76 in 1	2115/3060	8.553s
10.c	Unsafe	10	101.484s	29.699s	76 to 76 in 1	1935/3060	*TO*
11.c	Unsafe	9	38.624s	20.81s	83 to 83 in 1	1744/2868	16.439s
12.c	Unsafe	9	62.895s	155.681s	68 to 68 in 1	1935/3060	3.03s
13.c	Safe	10	7.392s	10.993s	22 to 8 in 3	144/736	8.4s