Semi-Automated Reasoning About Non-Determinism in C Expressions

Léon Gondelman

joint work with Dan Frumin and Robbert Krebbers

Radboud University Nijmegen

23 February, 2019 @ EUTypes, Krakow, Poland
(in relation with Lambda Days 2019)

C quiz

```c
int main() {
  int x;
  int y = (x = 3) + (x = 4);
  printf("%d, %d\n", x, y);
}
```

What is the expected outcome of this program ?

C quiz

```c
int main() {
  int x;
  int y = (x = 3) + (x = 4);
  printf("%d, %d\n", x, y);
}
```

a small experiment with existing compilers gives

compiler	**outcome**	**warnings**
compcert	4, 7	no
clang	4, 7	yes
gcc-4.9	4, 8	no

C quiz

```
int main() {
  int x;
  int y = (x = 3) + (x = 4);
  printf("%d, %d\n", x, y);
}
```

according to C standard, the program is allowed do... anything,
it is even allowed to crash

this program violates the sequence point restriction:

- the order of evaluation in C expressions is unspecified
- concurrent memory access is allowed
- multiple unsequenced modifications result in undefined behavior

this talk

the problem: sequence point violations may cause a C program to crash or to have arbitrary results

the goal: guarantee the absence of undefined behavior in a given C program for any evaluation order

in this talk:

(1) use concurrent separation logic to reason about C
(previous work, Krebbers POPL'14)

(2) turn it into a semi-automated reasoning procedure
(our contributions)

(Krebbers POPL'14)

observation: view non-determinism through concurrency

idea: use the concurrent separation logic

$$\frac{\{P_1\}\,\mathtt{e}_1\,\{\Psi_1\} \quad \{P_2\}\,\mathtt{e}_2\,\{\Psi_2\} \quad \forall \mathtt{v}_1\,\mathtt{v}_2.\,\Psi_1\ \mathtt{v}_1 * \Psi_2\ \mathtt{v}_2 \vdash \Phi(\mathtt{w}_1\ [\![\circledcirc]\!]\ \mathtt{w}_2)}{\{P_1 * P_2\}\,\mathtt{e}_1 \circledcirc \mathtt{e}_2\,\{\Phi\}}$$

using the rules of this logic we can

- split the memory resources into two disjoint parts
- independently prove that each subexpression executes safely

(Krebbers POPL'14)

observation: view non-determinism through concurrency
idea: use the concurrent separation logic

$$\frac{\{P_1\}\,\mathtt{e}_1\,\{\Psi_1\} \quad \{P_2\}\,\mathtt{e}_2\,\{\Psi_2\} \quad \forall \mathtt{v}_1\,\mathtt{v}_2.\,\Psi_1\ \mathtt{v}_1 * \Psi_2\ \mathtt{v}_2 \vdash \Phi(\mathtt{w}_1\ [\![\circledcirc]\!]\ \mathtt{w}_2)}{\{P_1 * P_2\}\,\mathtt{e}_1 \circledcirc \mathtt{e}_2\,\{\Phi\}}$$

limitations:

- no support for automation
- difficult to conduct even a manual proof in Coq

weakest preconditions

instead of Hoare triples, we model our program logic
using **weakest precondition calculus**

$$\mathsf{wp}\ \mathsf{e}\ \{\Phi\}$$

- e is safe (has defined behavior),
- if e terminates with a value $\mathtt{v}$, then $\mathtt{v}$ satisfies the predicate Φ

the non-determinism is reflected in a similar but more concise way:

$$\frac{\mathsf{wp}\ \mathsf{e}_1\ \{\Psi_1\} \quad \mathsf{wp}\ \mathsf{e}_2\ \{\Psi_2\} \quad (\forall \mathtt{w}_1 \mathtt{w}_2.\ \Psi_1\ \mathtt{w}_1 * \Psi_2\ \mathtt{w}_2 \mathrel{-\!\!*} \Phi(\mathtt{w}_1\ \llbracket \circledcirc \rrbracket\ \mathtt{w}_2))}{\mathsf{wp}\ (\mathsf{e}_1 \circledcirc \mathsf{e}_2)\ \{\Phi\}}$$

dereferencing

a possible candidate for the load operation:

$$\frac{\mathsf{wp}\ \mathsf{e}\ \{\mathtt{l}.\ \exists \mathtt{w}.\ \mathtt{l} \mapsto \mathtt{w} * (\mathtt{l} \mapsto \mathtt{w} \mathrel{-\!\!*} \varPhi\ \mathtt{w})\}}{\mathsf{wp}\ (\mathsf{*e})\ \{\varPhi\}}$$

dereferencing

a possible candidate for the load operation:

$$\frac{\mathsf{wp}\ \mathtt{e}\ \{\mathtt{l}.\ \exists\mathtt{w}.\ \mathtt{l} \mapsto \mathtt{w} * (\mathtt{l} \mapsto \mathtt{w} \mathrel{-\!\!*} \Phi\ \mathtt{w})\}}{\mathsf{wp}\ (\mathtt{*e})\ \{\Phi\}}$$

too weak: does not allow sharing *e.g.*, $\mathtt{*l} + \mathtt{*l}$

sharing resources

fractional permissions enable sharing of resources:

$$\mathtt{l} \xmapsto{q_1+q_2} \mathtt{v} \dashv\vdash \mathtt{l} \xmapsto{q_1} \mathtt{v} * \mathtt{l} \xmapsto{q_2} \mathtt{v}$$

so multiple subexpressions **can safely read** from the same location

the rule for load becomes

$$\frac{\mathsf{wp}\ \mathsf{e}\ \left\{\mathtt{l}.\ \exists \mathtt{w}\, q.\ \mathtt{l} \overset{q}{\mapsto} \mathtt{w} * (\mathtt{l} \overset{q}{\mapsto} \mathtt{w} \mathrel{-\!\!*} \Phi\ \mathtt{w})\right\}}{\mathsf{wp}\ (\mathtt{*}\mathsf{e})\ \{\Phi\}}$$

so we can now prove programs like $\mathtt{*l} + \mathtt{*l}$

assignment

a possible candidate for the assignment operation:

$$\frac{\mathsf{wp}\ \mathsf{e}_1\ \{\Psi_1\} \quad \mathsf{wp}\ \mathsf{e}_2\ \{\Psi_2\} \quad (\forall \mathtt{l}\ \mathtt{w}.\ \Psi_1\ \mathtt{l} * \Psi_2\ \mathtt{w} \mathrel{-\!\!*} \exists \mathtt{v}.\ \mathtt{l} \overset{1}{\mapsto} \mathtt{v} * (\mathtt{l} \overset{1}{\mapsto} \mathtt{w} \mathrel{-\!\!*} \mathit{\Phi}\ \mathtt{w}))}{\mathsf{wp}\ (\mathsf{e}_1 = \mathsf{e}_2)\ \{\mathit{\Phi}\}}$$

assignment

a possible candidate for the assignment operation:

$$\frac{\mathsf{wp}\ \mathsf{e}_1\ \{\Psi_1\} \quad \mathsf{wp}\ \mathsf{e}_2\ \{\Psi_2\} \quad (\forall \mathtt{l}\,\mathtt{w}.\ \Psi_1\ \mathtt{l} * \Psi_2\ \mathtt{w} \mathrel{-\!\!*} \exists \mathtt{v}.\ \mathtt{l} \overset{1}{\mapsto} \mathtt{v} * (\mathtt{l} \overset{1}{\mapsto} \mathtt{w} \mathrel{-\!\!*} \Phi\ \mathtt{w}))}{\mathsf{wp}\ (\mathsf{e}_1 = \mathsf{e}_2)\ \{\Phi\}}$$

unsound: does not account for sequence point violations

for example, we could verify programs like $\mathtt{l} = (\mathtt{l} = 3)$

assignment

to account for sequence point violations, we decorate fractional permissions with two access levels :

$$\mathtt{l} \overset{q}{\mapsto}_{\xi} \mathtt{v}, \quad \xi \in \{L, U\}$$

- permission $\mathtt{l} \overset{q}{\mapsto}_{U} \mathtt{v}$ states that the location is **unlocked**, so one can read from/write to the location $\mathtt{l}$
- permission $\mathtt{l} \overset{q}{\mapsto}_{L} \mathtt{v}$ states that the location has been **locked**, someone is already writing to it, so reads/writes are forbidden

assignment

the rule for assignment becomes

$$\frac{\mathsf{wp}\ \mathsf{e}_1\ \{\Psi_1\} \quad \mathsf{wp}\ \mathsf{e}_2\ \{\Psi_2\} \quad (\forall \mathtt{l}\,\mathtt{w}.\ \Psi_1\ \mathtt{l} * \Psi_2\ \mathtt{w} \mathrel{-\!\!*} \exists \mathtt{v}.\ \mathtt{l} \overset{1}{\mapsto}_U \mathtt{v} * (\mathtt{l} \overset{1}{\mapsto}_L \mathtt{w} \mathrel{-\!\!*} \Phi\ \mathtt{w}))}{\mathsf{wp}\ (\mathsf{e}_1 = \mathsf{e}_2)\ \{\Phi\}}$$

programs like $\mathtt{l} = (\mathtt{l} = 3)$ cannot be verified any more

unlocking modality

remark: we want to access locked pointers later again

$$\mathtt{l} = \mathtt{4}\,;\mathtt{*l}$$

we use the unlocking modality $\mathbb{U}$ that unlocks
all locked locations at the sequence point ;

$$\frac{\mathsf{wp}\ \mathsf{e}_1\ \{\_.\ \mathbb{U}(\mathsf{wp}\ \mathsf{e}_2\ \{\Phi\})\}}{\mathsf{wp}\ (\mathsf{e}_1\,;\mathsf{e}_2)\ \{\Phi\}} \qquad \frac{\mathtt{l} \overset{q}{\mapsto}_L \mathtt{v}}{\mathbb{U}(\mathtt{l} \overset{q}{\mapsto}_U \mathtt{v})} \qquad \frac{P \twoheadrightarrow\!\!\ast\ Q}{\mathbb{U}P \twoheadrightarrow\!\!\ast\ \mathbb{U}Q}$$

reasoning about programs

usually we prove programs assuming some logical context:

$$P \vdash \mathsf{wp}\ \mathsf{e}\ \{\Phi\}$$

we intertwine the application of wp rules with other logical steps (*splitting resources, discharging side-conditions, ...*)

manual proof quickly becomes tedious even for small programs
e.g., to reason about binary operators we have to

- infer manually the intermediate postconditions
- subdivide resources all the time

key idea

turn program logic into an algorithm procedure

using a novel symbolic execution algorithm:

input		**output**
precondition		postcondition
program	--→	value
		frame = resources not used

example

$\mathtt{l} \mapsto \mathtt{v1} * \mathtt{k} \mapsto \mathtt{v2} * \mathtt{r} \mapsto \mathtt{v3}$

- -

$\mathtt{l} = \mathtt{*k} + 10$

postcondition: $\top$

frame: $\top$

example

$\mathtt{l} \mapsto \mathtt{v1} * \cancel{\mathtt{k} \mapsto \mathtt{v2}} * \mathtt{r} \mapsto \mathtt{v3}$

- -

$\mathtt{l} = \mathtt{v2} + \mathtt{10}$

postcondition: $\mathtt{k} \xmapsto{0.5} \mathtt{v2}$

frame: $\mathtt{k} \xmapsto{0.5} \mathtt{v2}$

example

$\cancel{\mathtt{l} \mapsto \mathtt{v1}} * \cancel{\mathtt{k} \mapsto \mathtt{v2}} * \mathtt{r} \mapsto \mathtt{v3}$

- -

$\mathtt{v2} + 10$

postcondition: $\mathtt{k} \overset{0.5}{\longmapsto} \mathtt{v2} * \mathtt{l} \mapsto_{L} (\mathtt{v2} + 10)$

frame: $\mathtt{k} \overset{0.5}{\longmapsto} \mathtt{v2}$

example

$\cancel{\texttt{l} \mapsto \texttt{v1}} * \cancel{\texttt{k} \mapsto \texttt{v2}} * \cancel{\texttt{r} \mapsto \texttt{v3}}$

- -

$\texttt{v2} + 10$

postcondition: $\texttt{k} \xrightarrow{0.5} \texttt{v2} * \texttt{l} \mapsto_L (\texttt{v2} + 10)$

frame: $\texttt{k} \xrightarrow{0.5} \texttt{v2} * \texttt{r} \mapsto \texttt{v3}$

example (continued)

$\mathtt{l} \mapsto \mathtt{v1} * \mathtt{k} \mapsto \mathtt{v2} * \mathtt{r} \mapsto \mathtt{v3}$

- -

$(\mathtt{l} = \mathtt{*k} + \mathtt{10}) + (\mathtt{r} = \mathtt{*k} + \mathtt{10})$

postcondition: $\top$

frame: $\top$

after executing the LHS

$\cancel{\texttt{l} \mapsto \texttt{v1}} * \cancel{\texttt{k} \mapsto \texttt{v2}} * \cancel{\texttt{r} \mapsto \texttt{v3}}$

- -

$(\texttt{v2} + \texttt{10}) + (\texttt{r} = \texttt{*k} + \texttt{10})$

postcondition: $\texttt{k} \xrightarrow{0.5} \texttt{v2} * \texttt{l} \mapsto_L (\texttt{v2} + \texttt{10})$

frame: $\texttt{k} \xrightarrow{0.5} \texttt{v2} * \texttt{r} \mapsto \texttt{v3}$

before executing the RHS

$\cancel{\mathtt{l} \mapsto \mathtt{v1}} * \mathtt{k} \xrightarrow{0.5} \mathtt{v2} * \mathtt{r} \mapsto \mathtt{v3}$

- -

$(\mathtt{v2} + 10) + (\mathtt{r} = \mathtt{*k} + 10)$

postcondition: $\mathtt{k} \xrightarrow{0.5} \mathtt{v2} * \mathtt{l} \mapsto_L (\mathtt{v2} + 10)$

frame: $\cancel{\mathtt{k} \xrightarrow{0.5} \mathtt{v2}} * \cancel{\mathtt{r} \mapsto \mathtt{v3}}$

executing the RHS

$$\cancel{\mathtt{l} \mapsto \mathtt{v1}} * \cancel{\mathtt{k} \xmapsto{0.5} \mathtt{v2}} * \mathtt{r} \mapsto \mathtt{v3}$$

- -

$$(\mathtt{v2} + \mathtt{10}) + (\mathtt{r} = \mathtt{*k} + \mathtt{10})$$

postcondition: $\mathtt{k} \xmapsto{3/4} \mathtt{v2} * \mathtt{l} \mapsto_L (\mathtt{v2} + \mathtt{10})$

frame: $\mathtt{k} \xmapsto{1/4} \mathtt{v2} * \cancel{\mathtt{r} \mapsto \mathtt{v3}}$

final result

$\cancel{\texttt{l} \mapsto \texttt{v1}} * \cancel{\texttt{k} \xmapsto{0.5} \texttt{v2}} * \cancel{\texttt{r} \mapsto \texttt{v3}}$

- -

$(\texttt{v2} + 10) + (\texttt{v2} + 10)$

postcondition: $\texttt{k} \xmapsto{3/4} \texttt{v2} * \texttt{l} \mapsto_L (\texttt{v2} + 10) * \texttt{r} \mapsto_L (\texttt{v2} + 10)$

frame: $\texttt{k} \xmapsto{1/4} \texttt{v2} * \cancel{\texttt{r} \mapsto (\texttt{v2} + 10)}$

algorithm

our symbolic execution algorithm is a partial function restricted to *symbolic heaps* ($m \in \mathsf{sheap}$):

$$\mathsf{forward} : (\mathsf{sheap} \times \mathsf{expr}) \to (\mathsf{val} \times \mathsf{sheap} \times \mathsf{sheap})$$

satisfying the following specification:

$$\frac{\mathsf{forward}(m, \mathtt{e}) = (\mathtt{w}, m_1^o, m_1)}{[\![m]\!] \vdash \mathsf{wp}\ \mathtt{e}\ \{\mathtt{v}.\ \mathtt{v} = \mathtt{w} * [\![m_1^o]\!]\} * [\![m_1]\!]}$$

limitations

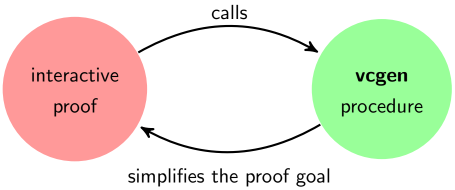
symbolic execution helps to make the wp rules algorithmic
but the algorithm itself may fail for several reasons:

- the program is not of the right shape
- the precondition is not a symbolic heap
- needed resource is missing in the precondition

to turn the program logic into an automated procedure
we integrate the symbolic executor algorithm into a
verification condition generator (vcgen)

key idea

design an interactive verification condition generator



vcgen automates the proof as long as forward does not fail,

and when forward fails,

- vcgen returns to the user a partially solved goal
- from which it can be called back after the user helped out

conclusions

main message:

symbolic execution with frames is a key to enable
a semi-automated about non-determinism in C
in an interactive theorem prover

other contributions:

- a definitional semantics to a fragment of C in Coq
- soundness proof for symbolic executor and vcgen
- development built on top of the Iris framework

thank you !