Kratos2

An SMT-Based Model Checker for Imperative Programs

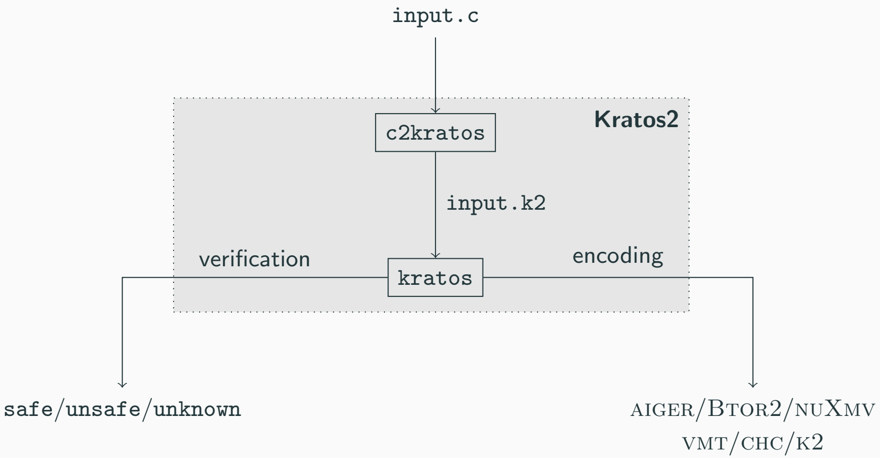
Alberto Griggio
Fondazione Bruno Kessler, Trento, Italy

Martin Jonáš
Masaryk University, Brno, Czechia

Slides at

https://is.muni.cz/go/cav23

What is Kratos2



K2 Intermediate Language

- easy to parse (s-expressions)
- value semantics
- value manipulations specified by underlying SMT theories
- support for arbitrary SMT theories (integers, reals, bit-vectors, floats, arrays, uninterpreted functions, . . .)
- limited side effects (assignment operator, havoc statement, function return values)
- no undefined/unspecified/implementation-defined behavior
- reachability and liveness specifications

K2 Intermediate Language: Example

<table>
<tr><th>C</th><th>K2</th></tr>
<tr><td><pre>int f(int x)
{
    x = x + 1;
    return x;
}

void main(void)
{
  int y;
  y = f(y);
  assert(y > 0);

}</pre></td><td><pre>(function f   ; function name
  ((var x (sbv 32)))   ; parameter list
  (return (var ret (sbv 32)))   ; return variables
  (locals)   ; local variables
  (assign ret (add x (const 1 (sbv 32)))))

(function main () (return) (locals (var y (sbv 32)))
  (seq
    (call f y y)
    (jump (label then) (label else))
    (label then)
    (assume (not (gt y (const 0 (sbv 32)))))
    (! (label err) :error assert-fail)
    (label else)))</pre></td></tr>
</table>

K2 Intermediate Language: Example

<table>
<tr><th>C</th><th>K2</th></tr>
<tr><td><pre>
int f(int x)
{
    x = x + 1;
    return x;
}

void main(void)
{
  int y;
  y = f(y);
  assert(y > 0);

}
</pre></td><td><pre>
(function f   ; function name
  ((var x (sbv 32)))   ; parameter list
  (return (var ret (sbv 32)))   ; return variables
  (locals)   ; local variables
  (assign ret (add x (const 1 (sbv 32)))))

(function main () (return) (locals (var y (sbv 32)))
  (seq
    (call f y y)
    (jump (label then) (label else))
    (label then)
    (assume (not (gt y (const 0 (sbv 32)))))
    (! (label err) :error assert-fail)
    (label else)))
</pre></td></tr>
</table>

K2 Intermediate Language: Example

C	K2

<pre>int f(int x)
{
    x = x + 1;
    return x;
}</pre>
	<pre>(function f ; function name
	  ((var x (sbv 32))) ; parameter list
	  (return (var ret (sbv 32))) ; return variables
	  (locals) ; local variables
	  (assign ret (add x (const 1 (sbv 32)))))</pre>

<pre>void main(void)
{
  int y;
  y = f(y);
  assert(y > 0);
 	
}</pre>
	<pre>(function main () (return) (locals (var y (sbv 32)))
	  (seq
	    (call f y y)
	    (jump (label then) (label else))
	    (label then)
	    (assume (not (gt y (const 0 (sbv 32)))))
	    (! (label err) :error assert-fail)
	    (label else)))</pre>

K2 Intermediate Language: Example

<table>
<tr><th>C</th><th>K2</th></tr>
<tr><td>

```
int f(int x)
{
    x = x + 1;
    return x;
}

void main(void)
{
  int y;
  y = f(y);
  assert(y > 0);


}
```

</td><td>

```
(function f                         ; function name
  ((var x (sbv 32)))                ; parameter list
  (return (var ret (sbv 32)))       ; return variables
  (locals)                          ; local variables
  (assign ret (add x (const 1 (sbv 32)))))

(function main () (return) (locals (var y (sbv 32)))
  (seq
    (call f y y)
    (jump (label then) (label else))
    (label then)
    (assume (not (gt y (const 0 (sbv 32)))))
    (! (label err) :error assert-fail)
    (label else)))
```

</td></tr>
</table>

K2 Intermediate Language: Example

<table>
<tr><th>C</th><th>K2</th></tr>
<tr><td><pre>
int f(int x)
{
    x = x + 1;
    return x;
}

void main(void)
{
  int y;
  y = f(y);
  assert(y > 0);

}
</pre></td><td><pre>
(function f   ; function name
  ((var x (sbv 32)))   ; parameter list
  (return (var ret (sbv 32)))   ; return variables
  (locals)   ; local variables
  (assign ret (add x (const 1 (sbv 32)))))

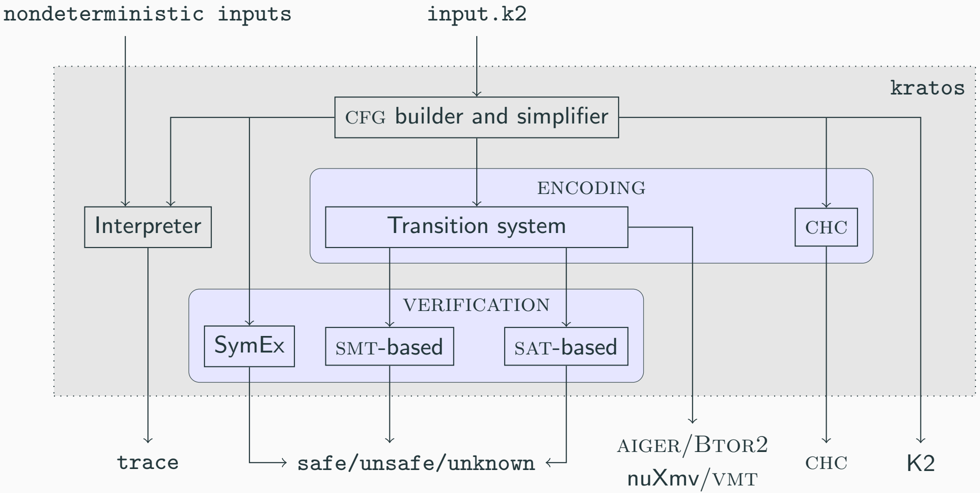
(function main () (return) (locals (var y (sbv 32)))
  (seq
    (call f y y)
    (jump (label then) (label else))
    (label then)
    (assume (not (gt y (const 0 (sbv 32)))))
    (! (label err) :error assert-fail)
    (label else)))
</pre></td></tr>
</table>

c2kratos

Customizable translation of C99 programs to K2

- Python API
- custom optimization passes
- custom implementation of C builtins (`assume`, `malloc`, `memset`, ...)
- choice of theories to encode C data types (e.g., integers or bit-vectors)

kratos: Architecture



kratos: Verification Engines

Symbolic Execution

- standard symbolic execution with iterative deepening DFS
- all theories implemented in MathSAT5 (UF, NIA, NRA, FP, BV, A)

SMT-Based Engines

- engines from nuXmv: BMC, k-induction, and IC3 via implicit predicate abstraction
- all theories implemented in MathSAT5

SAT-Based Engines

- engines from nuXmv: BMC, k-induction, and propositional IC3
- custom engine for arrays, based on abstraction and prophecy variables
- only for finite-state systems

Experimental Evaluation

RQ1: Is K2 expressive enough to capture realistic C programs?

RQ2: Are the verification engines efficient?

Evaluation

- all 5400 C programs from the ReachSafety category of SV-COMP 2022
- comparison with VeriAbs 1.4.2 and CPAchecker 2.2
- 15 minutes CPU time, 8 GiB RAM

Experimental Evaluation: K2 expressivity

From all 5400 C programs, **5344** successfully converted to K2 by `c2Kratos`

Remaining 56 programs

- unsupported hexadecimal floating-point literals (`0x1.AE9p3`)
- unimplemented floating-point builtins (`rint()`, `fmod()`, `copysign()`, `...`)
- unsupported variable-length arrays (VLA)

Experimental Evaluation: Kratos2 Performance

Tool	Unsafe	Safe	Wrong	Unique
CPAchecker	1606	1925	4	48
Kratos2	1606	1710	0	23
VeriAbs	**1928**	**2494**	0	**1039**

Experimental Evaluation: Kratos2 Performance

Tool	Unsafe	Safe	Wrong	Unique
CPAchecker	1606	1925	4	48
Kratos2	1606	1710	0	23
VeriAbs	**1928**	**2494**	0	**1039**

Kratos2 solves the largest number of benchmarks in

- safe `bitvector`
- unsafe `controlflow`
- unsafe `floats`
- unsafe `sequentialized`

Kratos2

- efficient and extensible model checker of imperative programs
- reusable intermediate verification language K2 based on SMT semantics
- translation from C/K2 to multiple other formalisms
- free for academic and non-commercial use

Kratos2

Kratos2

- efficient and extensible model checker of imperative programs
- reusable intermediate verification language K2 based on SMT semantics
- translation from C/K2 to multiple other formalisms
- free for academic and non-commercial use

Try Kratos2

`https://kratos.fbk.eu/`

Bonus slides

Experimental Evaluation: Detailed Results

Family	CPAchecker			Kratos2			VeriAbs		
	U	S	W	U	S	W	U	S	W
arrays	70	5	0	75	7	0	**106**	**261**	0
bitvectors	13	31	0	13	**33**	0	**14**	31	0
combinations	**295**	36	0	282	47	0	277	**77**	0
controlflow	39	36	0	**40**	37	0	**40**	**47**	0
eca	223	481	0	210	365	0	**467**	**600**	0
floats	41	356	0	**43**	350	0	**43**	**393**	0
heap	**71**	118	1	67	102	0	70	**120**	0
loops	152	334	2	159	307	0	**192**	**427**	0
productlines	**265**	**332**	0	262	315	0	260	322	0
recursive	40	36	1	43	28	0	**46**	**41**	0
sequentialized	347	108	0	**361**	68	0	**361**	**123**	0
xcsp	50	**52**	0	51	51	0	**52**	**52**	0
Total	1606	1925	4	1606	1710	0	**1928**	**2494**	0

Experimental Evaluation: Configuration of `kratos`

Sequential portfolio

- 30 s – symbolic execution
- 200 s – SMT-based IC3 with implicit predicate abstraction
- 400 s – SAT-based IC3 with array abstraction via prophecy variables
- rest – SMT-based BMC

Translation of C99 programs to K2

- complex control flow $\rightarrow$ nondeterministic jumps and assumes
- compound instructions $\rightarrow$ sequence of assignments and auxiliary variables
- pointers and dynamic memory $\rightarrow$ theory of arrays
- structures $\rightarrow$ multiple variables
- unions $\rightarrow$ multiple variables $+$ assumptions on equality of all members

Applications

Autosar platform

- verification of automotive software

Taste development environment

- verification of C code automatically generated from AADL specifications

Railway interlocking systems

- verification of C code for railway interlocking systems automatically generated from the specifications in a controlled natural language

Benchmark generator

- producing symbolic transition systems from C programs to benchmark counter-example guided prophecy for array abstractions

Notable References i

[1] Dirk Beyer et al. **"Software model checking via large-block encoding".** In: *FMCAD*. IEEE, 2009, pp. 25–32.

[2] Per Bjesse. **"Word-Level Sequential Memory Abstraction for Model Checking".** In: *FMCAD*. Ed. by Alessandro Cimatti and Robert B. Jones. IEEE, 2008, pp. 1–9.

[3] Alessandro Cimatti et al. **"Infinite-state invariant checking with IC3 and predicate abstraction".** In: *Formal Methods Syst. Des.* 49.3 (2016), pp. 190–218.

[4] Alessandro Cimatti et al. **"Kratos – A Software Model Checker for SystemC".** In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 310–316.

Notable References ii

[5] Jakub Daniel et al. **“Infinite-State Liveness-to-Safety via Implicit Abstraction and Well-Founded Relations”.** In: *CAV (1)*. Vol. 9779. Lecture Notes in Computer Science. Springer, 2016, pp. 271–291.

[6] Sergey Grebenshchikov et al. **“Synthesizing software verifiers from proof rules”.** In: *PLDI*. ACM, 2012, pp. 405–416.

[7] Makai Mann et al. **“Counterexample-Guided Prophecy for Model Checking Modulo the Theory of Arrays”.** In: *Log. Methods Comput. Sci.* 18.3 (2022).