Towards Combined Static and Runtime Verification of Distributed Software

Wolfgang Ahrendt

Chalmers University of Technology, Gothenburg, Sweden

Seminar on Distributed Runtime Verification
Bertinoro, 18 May 2016

My Profile

verification	static	combined static/runtime	runtime
sequential applications	deductive verif. of oo KeY	StaRVOOrS	runtime verif. of oo Larva
distributed applications	compositional ded. verif. of distributed oo	my next project (w. Schneider,Pace)	asked Borzoo ?to organise this seminar

Part I

verification	static	combined static/runtime	runtime
sequential applications	deductive verif. of oo KeY	StaRVOOrS	runtime verif. of oo Larva
distributed applications	compositional ded. verif. of distributed oo	my next project (w. Schneider,Pace)	

KeY

- Theorem prover for source code verification
- Functional (data-heavy) properties
- First-order dynamic logic as program logic
- Deductive verification, using a sequent calculus
- Verification = symbolic execution + induction/invariants
- Prover is interactive + highly automated
- most elaborate KeY instance KeY-Java
  - Java as target language
  - Java dynamic logic
  - Supports specification language JML

Supported Specification Language: JML

Java Modeling Language
Specification language for Java

International community effort lead by Gary T. Leavens, Iowa State

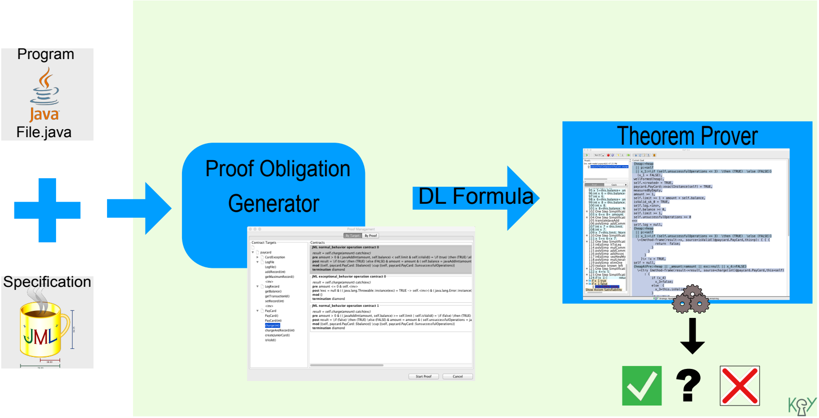
JML example

```
/*@ public normal_behavior
  @ requires a != null;
  @ ensures (\forall int j; j >= 0 && j < a.length;
  @                          \result >= a[j]);
  @ ensures a.length > 0 ==>
  @         (\exists int j; j >= 0 && j < a.length;
  @                          \result == a[j]);
  @*/
public static int max(int[] a) {
    int max = a[0], i = 1;
    while ( i < a.length ) {
      if ( a[i] > max ) max = a[i];
      ++i;
    }
    return max;
}
```

JML + Java translated to Dynamic Logic

```
  a != null
->
  <
    int max = 0;
    if ( a.length > 0 ) max = a[0];
    int i = 1;
    while ( i < a.length ) {
      if ( a[i] > max ) max = a[i];
      ++i;
    }
  >
    \forall int j; (j >= 0 & j < a.length -> max >= a[j])
    &
    (a.length > 0 ->
      \exists int j; (j >= 0 & j < a.length & max = a[j]))
```

The KeY Verification System



- Long-term collaboration: Karlsruhe–Darmstadt–Chalmers

References

- Ahrendt, Beckert, Bubel, Hähnle, Schmitt, Ulbrich
  The KeY Book
  Springer, LNCS, to appear

Major Case Study with KeY: Timsort

Timsort

Hybrid sorting algorithm (insertion sort + merge sort) optimised for partially sorted arrays (typical for real-world data).

Facts

- Designed by Tim Peters (for Python)
- Since Java 1.7 default algorithm for non-primitive arrays/collections
- `java.util.Collections.sort` and `java.util.Arrays.sort` implement Timsort

Major Case Study with KeY

Verification with KeY

- Attempt to verify OpenJDK implementation
- 460 lines of specification vs. 928 lines of code
- KeY verification revealed bug
- Fixed version formally verified
- whole project 2.5 person month

Aftermath

- Bug affected
  - Java (OpenJDK + Oracle)
  - Android
  - Python
  - Apache: Lucene, Hadoop, Spark++
  - Go, D, Haskell
- blog with >3 million page views
- top news on ycombinator, reddit, Hacker News etc.

References

- Stijn de Gouw, Jurriaan Rot, Frank S. de Boer, Richard Bubel, Reiner Hähnle, *OpenJDK's Java.utils.Collection.sort() Is Broken: The Good, the Bad and the Worst Case* CAV 2015

Part II

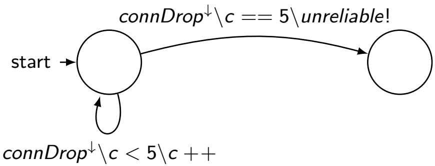
verification	static	combined static/runtime	runtime
sequential applications	deductive verif. of oo KeY	StaRVOOrS	runtime verif. of oo Larva
distributed applications	compositional ded. verif. of distributed oo	my next project (w. Schneider,Pace)	

Larva

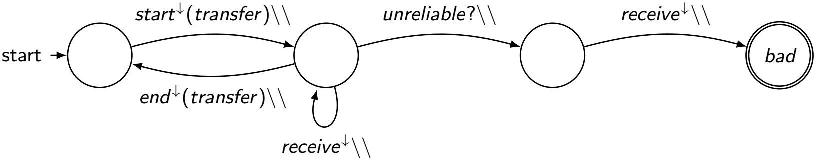
- **L**ogical **A**utomata for **R**untime **V**erification and **A**nalysis
- Property language:
  DATE (Dynamic Automata w. Timers and Events)
- Front ends: Duration Calculus, Lustre
- Monitor generation + Code instrumentation w. Aspects

DATE

- communicating automata, event-triggered transitions, timers
- events: method entry/exit, timer events, synchronising events



foreach transfer :



References

- C. Colombo, G.J. Pace, G. Schneider
  LARVA – A Tool for Runtime Monitoring of Java Programs
  SEFM'09, 2009. IEEE Computer Society

Part III

verification	static	combined static/runtime	runtime
sequential applications	deductive verif. of oo KeY	StaRVOOrS	runtime verif. of oo Larva
distributed applications	compositional ded. verif. of distributed oo	my next project (w. Schneider,Pace)	

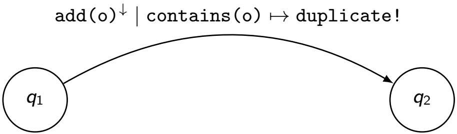
ppDATE :

- Specification Language for Data and Control Properties
- Extends DATE with
  pre/post-conditions, associated to the automata's states:

$$q_1 \xrightarrow{e|cond \mapsto act} q_2$$

$$\tau(q_1) = \{\, ..., \{pre\}\,\texttt{m}\,\{post\}, ...\,\}$$

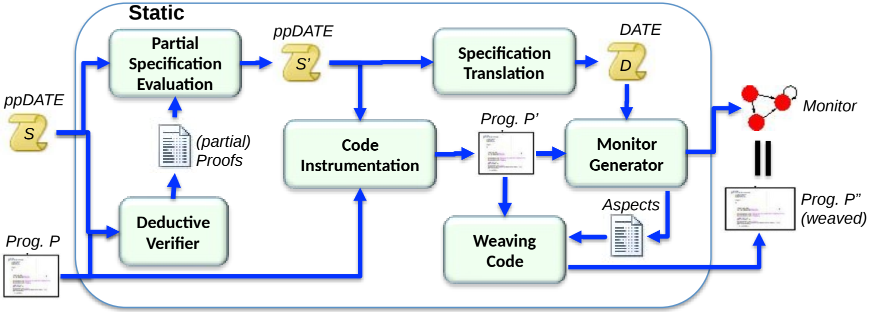
ppDATE Example: Scenario including a Hashtable



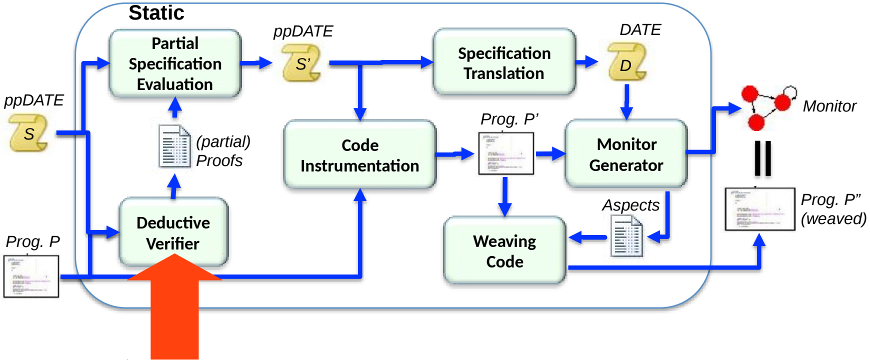
$$\tau(q_1) = \{ \quad \{\texttt{size} < \texttt{capacity}\}\,\texttt{add(o)}\,\{\exists\, i.\texttt{arr}[i] = \texttt{o}\} \quad \}$$

- Hoare triples are described using JML-like notation

High-level description of the framework



High-level description of the framework



Deductive Verifier

- KeY tries to prove:

  $$\{\texttt{size} < \texttt{capacity}\}\,\texttt{add(o)}\,\{\exists\, i.\ \texttt{arr}[i] = \texttt{o}\}$$

- KeY cannot fully prove (automatically)
- proof branch

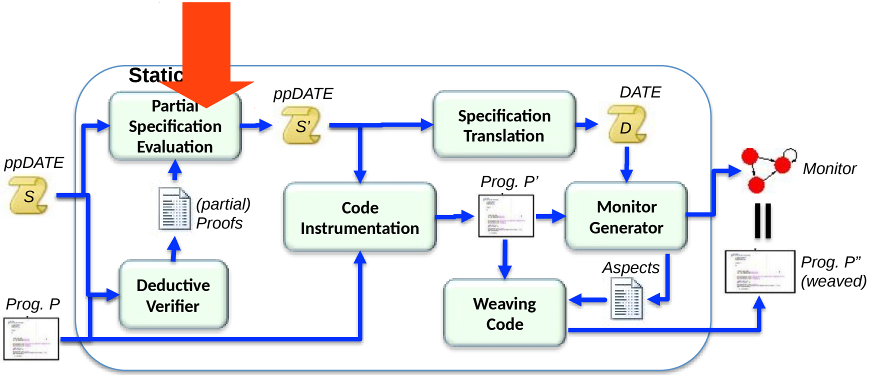
  $$\ldots, \texttt{arr[key\%capacity]} = \texttt{null} \ \vdash\ \ldots$$

  closed (automatically)
- proof branch

  $$\ldots, \neg\,\texttt{arr[key\%capacity]} = \texttt{null} \ \vdash\ \ldots$$

  not closed (automatically)

High-level description of the framework



Partial Specification Evaluation

- partial proof analysis synthesises additional pre-conditions, here

$$\neg\ \texttt{arr[key\%capacity]} = \texttt{null}$$

$\tau(q_1) =$
$\{ \quad \{\mathit{pre}\}\, \texttt{add(o)}\, \{\mathit{post}\} \quad \}$

Partial Specification Evaluation

- partial proof analysis synthesises additional pre-conditions, here

$$\neg\ \texttt{arr}[\texttt{key\%capacity}] = \texttt{null}$$

$$\tau(q_1) =$$
$$\{\quad \{\mathit{pre} \wedge \neg\ \texttt{arr}[\texttt{key\%capacity}] = \texttt{null}\}\, \texttt{add(o)}\, \{\mathit{post}\} \quad \}$$

StaRVOOrS: Mondex Case Study

Transactions	*no* monitoring	monitoring without static verif.	monitoring using static verif.
10	8 ms	120 ms	15 ms
100	50 ms	3,500 ms	90 ms
1000	250 ms	330,000 ms	375 ms

- Main reason for overhead: postcondition monitors
- KeY proves 2 Hoare triples fully
  $\Rightarrow$ never checked at runtime
- KeY proves 24 Hoare triples partially
  $\Rightarrow$ conditionally checked at runtime

References

- W. Ahrendt, G. Pace, G. Schneider
  A Unified Approach for Static and Runtime Verification:
  – Framework and Applications
  ISoLA 2012
- W. Ahrendt, M. Chimento, G. Pace, G. Schneider
  A Specification Language for Static and Runtime Verification of Data and Control Properties
  FM 2015
- W. Ahrendt, M. Chimento, G. Pace, G. Schneider
  StaRVOOrS*: A Tool for Combined Static and Runtime Verification of Java*
  RV 2015

Part IV

verification	static	combined static/runtime	runtime
sequential applications	deductive verif. of oo KeY	StaRVOOrS	runtime verif. of oo Larva
distributed applications	compositional ded. verif. of distributed oo	my next project (w. Schneider,Pace)	

(Static) Compositional Distributed Systems Verification

Here:

An object-oriented approach to distributed systems:

- asynchronous method calls
- incoming calls spawn object local threads
- shared memory is object local

target languages:

1. Creol [Univ. Oslo]
2. ABS [HATS project]

(Static) Compositional Distributed Systems Verification

- Properties of messages and their data
- Properties talk about communication history ($\mathcal{H}$)
- Outer specification of *interface* '`Bank`':

$$b.\mathrm{getBalance}(c) = \left(\sum_{b.\mathrm{deposit}(c,x) \in \mathcal{H}} (x) \right) - \left(\sum_{b.\mathrm{withdraw}(c,x) \in \mathcal{H}} (x) \right)$$

- Inner specification of *class* '`Swedbank implements Bank`':

$$\mathrm{getBalance}(c) = \mathrm{accounts.get}(c).\mathrm{balance}$$

(Deductive) Concurrent Verification

	shared memory	message passing
non-compositional	interference freedom tests [Owicky,Gries 75]	cooperation tests [Apt et.al. 80]
compositional	rely-guarantee [Jones 81]	assumption-commitment [Misra, Chandy 81]

"assume-guarantee"

- coined by Abadi&Lamport for union of compositional approaches
- also refers to compositional model checking techniques (Pnuelli, ..., Steffen, ..., Păsăreanu)

Assumption-Commitment

Component contract:

- *assumptions* about messages (+ data) *from* environment
- *commitments* about messages (+ data) *to* environment

Component internal invariant:

- refinement relation between external history and internal state

Component verification:

- Sequential verification,
  - using assumptions, showing commitments — of `this` component
  - showing assumptions, using commitments — of called components

Assumption-Commitment

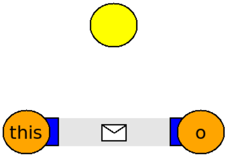
Variation of AC:

- Specs don't distinguish between assumption and commitment, but represent both in history invariant
- component has to guarantee that outgoing messages maintain invariant, given that incoming messages do

Calculus: Object Communication

$$\frac{\Gamma \vdash \{U_{\mathcal{H}}\} inv(\mathcal{H})^{\mathsf{this},o}_{caller,callee} \qquad\qquad \Gamma \vdash \{U_{\mathcal{H}}\}[rest]\phi}{\Gamma \vdash [\ l \ ! \ o.\mathrm{mtd}(\bar{x});\ rest]\phi}$$

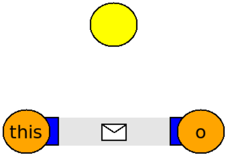
- *rest* : rest of the program
- $\mathcal{H}$: history
- *inv* : interface invariant
- $U_{\mathcal{H}}$: $\mathcal{H} := \mathrm{extend}(\mathcal{H}, Invoc(l, \bar{x}))$



Calculus: Object Communication

$$\frac{\Gamma \vdash \{U_{\mathcal{H},\bar{y}}\}[rest]\phi}{\Gamma \vdash [l?(\bar{y});\ rest]\phi}$$

- inv : interface invariant
- $\mathcal{H}$: history
- $U_{\mathcal{H},\bar{y}}$: append completion message
  $+$ assume invariant:



$$\mathcal{H}, \bar{y} := \text{some } H, \bar{v}.(\mathcal{H} \leq H \wedge Comp(H, l, \bar{v}) \wedge inv(H)^{\text{this},l.callee}_{caller,callee})$$

References

- W. Ahrendt, M. Dylla
  A System for Compositional Verification of Asynchronous Objects
  Science of Computer Programming
- C. Din, R. Bubel, R. Hähnle
  KeY-ABS: A deductive verification tool for the concurrent modelling language ABS
  CADE 2015

Part V

verification	static	combined static/runtime	runtime
sequential applications	deductive verif. of oo KeY	StaRVOOrS	runtime verif. of oo Larva
distributed applications	compositional ded. verif. of distributed oo	my next project (w. Schneider,Pace)	

Assume-Guarantee (AG) Runtime Verification?

AG mostly (or only?) used in *static* verification

Static AG has bottlenecks:

1. Verifying refinement of
   - communication contract

     to
   - inner state properties

   can require smart proof engineering.
2. It requires full access to implementation of all components.
   But in practice, some components are *closed*:
   proprietary components, legacy components, binaries.

Claim:
AG has great potential for combined static/runtime verification

Assume-Guarantee Static+Runtime Verification

Address bottlenecks of Assume-Guarantee:

For open components:

- *Attempt* static component verification,
  but refer ‘proof-holes’ to runtime verification

For closed components:

- formalise expected assumptions/guarantees ⇒ AG contract
- runtime verify compliance of *closed* components w. contract
- verify *open* components that use closed ones:
  using commitments / showing assumptions } of closed components they use
  (statically, or at runtime, or with combination)

References

- W. Ahrendt, G. Pace, G. Schneider
  StaRVOOrS – Episode II,
  Strengthen and Distribute the Force
  submitted