Efficient temporal pointcuts through dynamic advice deployment

Eric Bodden[1] Volker Stolz[2]

[1]Sable Research Group
McGill University, Montréal, Canada

[2]MOVES: Software Modeling and Verification
RWTH Aachen University, Aachen, Germany

Bonn, March 20th, 2006

Outline

Trace matching

What is it?

Watch an application's execution trace and when some pattern occurs, do something.

Trace matching

Tools and formalisms

Tool	Formalism	Developed at ...
J-LO / Tracechecks	LTL	RWTH Aachen, McGill [Bod05]
Tracematches	Regular Expressions	Oxford, McGill [AAS+05]
Tracecuts	CFGs	University of Calgary [WV04]
PQL	CFGs	Stanford University [SGA04]
JAsCo	NFAs	Vrije Universiteit Brussel [SVJ03]

All use automata, all except PQL and Tracecuts use finite automata even.

Static approaches taken so far

What has been optimized *statically* so far?

- Minimization of the generated automata
  - Might mean to minimize NFA ! (PSPACE complete)
- Optimization of data structures [ATB+06]
- Specialization w.r.t. base program [SGA04]
- Sophisticated memory management [AAS+05, Bod05]

Brought trace matching from *infeasible* to *feasible*.

Our goal here: Close the efficiency gap between those and plain Java/AspectJ programs.

Trace matching

What does it look like?

A temporal specification:

Whenever a enumerator e is claimed for a collection c, do not modify c while e is in use.

Trace matching

What does it look like?

As a tracematch:

```
tracematch(Vector c, Enumeration e) {
    sym create after returning(e):
      call(Enumeration+.new(..)) && args(c);
    sym next before:
      call(Object Enumeration.nextElement()) && target(e);
    sym update after:
      vector_update() && target(c);

    create next∗ update+ next
    {
            throw new ConcurrentModificationException();
    }
}
```

Trace matching

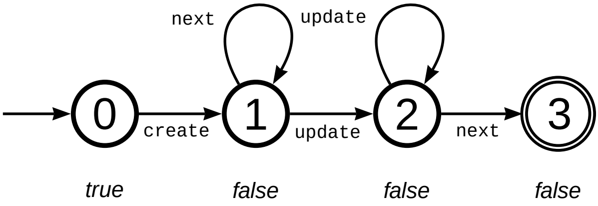
How does it work?

General evaluation strategy:

- Generate a (possibly finite) state machine according to the spec.
- Run that state machine along the actual execution trace.
- Events are triggered by advice.
- When we hit a final state, “blow the horn”.

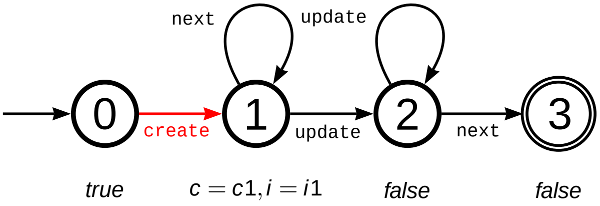
Trace matching

Evaluation by example



Trace matching

Evaluation by example

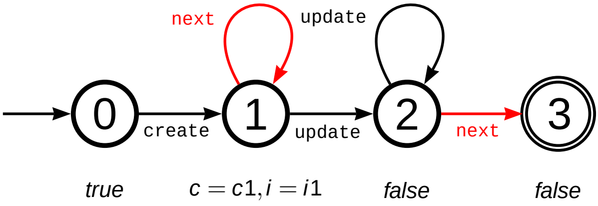


create(c=c1,i=i1)

Executed pieces of advice: 1

Trace matching

Evaluation by example

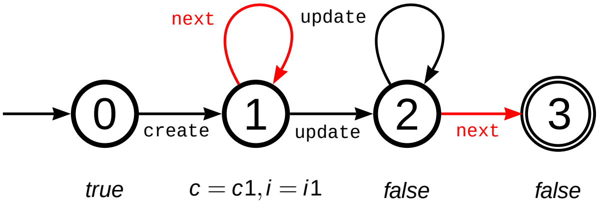


create(c=c1,i=i1) next(i=i1)

Executed pieces of advice: 3

Trace matching

Evaluation by example

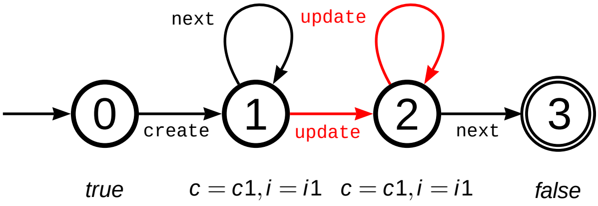


create(c=c1,i=i1) next(i=i1) next(i=i1)

Executed pieces of advice: 5

Trace matching

Evaluation by example

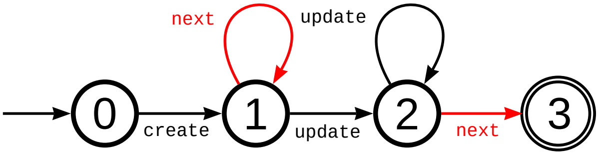


create(c=c1,i=i1) next(i=i1) next(i=i1) update(c=c1)

Executed pieces of advice: 7

Trace matching

Evaluation by example



true $c = c1, i = i1$ $c = c1, i = i1$ $c = c1, i = i1$

create(c=c1,i=i1) next(i=i1) next(i=i1) update(c=c1) next(i=i1)

Executed pieces of advice: 9

Matching today and in the future

Approach today:
Always match on everything. Works of course, but can be slow.

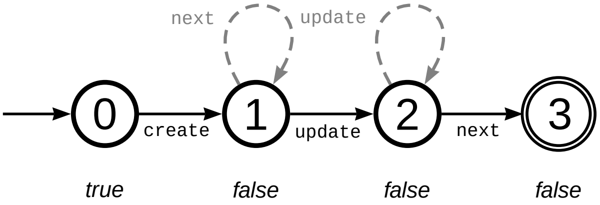
More clever approach:
At any time only match on what you are interested in!

How to tackle the problem

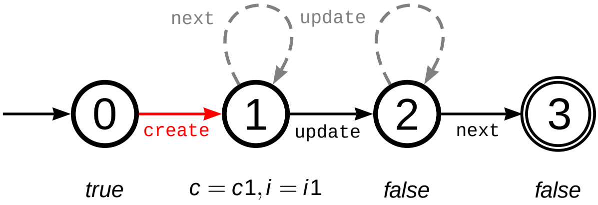
How can we do a better job?

1.) Statically remove loops. (sound for “non-overlapping” symbols)

Removing the loops



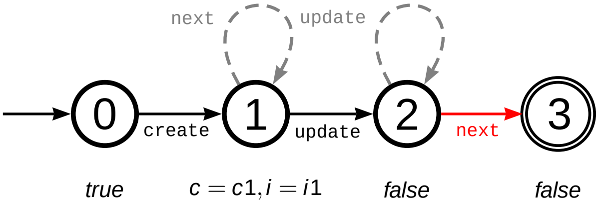
Removing the loops



create(c=c1,i=i1)

Executed pieces of advice: 1

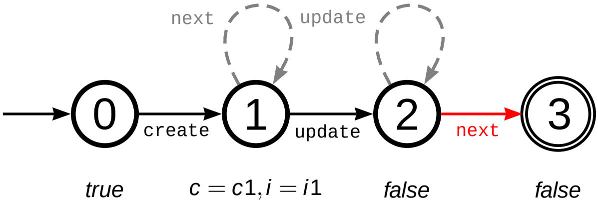
Removing the loops



create(c=c1,i=i1) next(i=i1)

Executed pieces of advice: 2

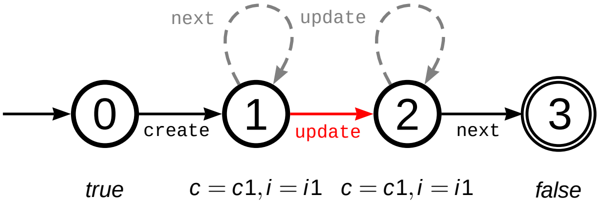
Removing the loops



create(c=c1,i=i1) next(i=i1) next(i=i1)

Executed pieces of advice: 3

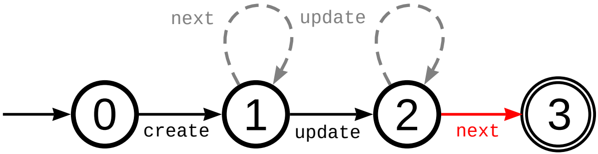
Removing the loops



create(c=c1,i=i1) next(i=i1) next(i=i1) update(c=c1)

Executed pieces of advice: 4

Removing the loops



true $\quad c = c1, i = i1 \quad c = c1, i = i1 \quad c = c1, i = i1$

create(c=c1,i=i1) next(i=i1) next(i=i1) update(c=c1) next(i=i1)

Executed pieces of advice: 5

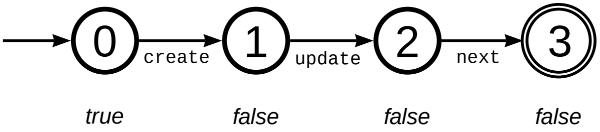
Let’s get dynamic

How can we still do better?

1.) Statically remove loops.

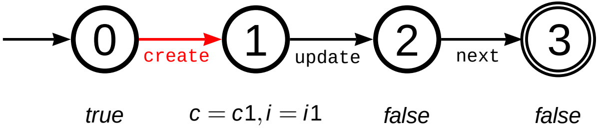
2.) Use dynamic advice deployment: At any time, only use those pieces of advice that are necessary.

Dynamic deployment



Deployed pieces of advice: { create }

Dynamic deployment

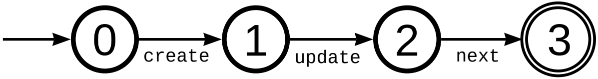


create(c=c1,i=i1)

Deployed pieces of advice: { create, update(c=c1) }

Executed pieces of advice: 1

Dynamic deployment



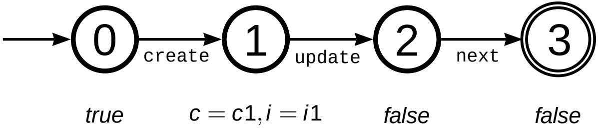
true $c = c1, i = i1$ *false* *false*

create(c=c1,i=i1) next(i=i1)

Deployed pieces of advice: $\{$ create, update(c=c1) $\}$

Executed pieces of advice: 1

Dynamic deployment

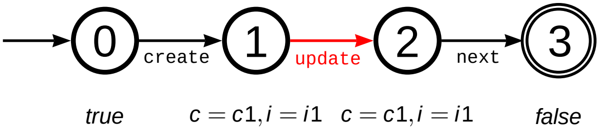


create(c=c1,i=i1) next(i=i1) next(i=i1)

Deployed pieces of advice: { create, update(c=c1) }

Executed pieces of advice: 1

Dynamic deployment

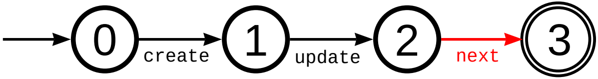


create(c=c1,i=i1) next(i=i1) next(i=i1) update(c=c1)

Deployed pieces of advice: { create, next(i=i1) }

Executed pieces of advice: 2

Dynamic deployment



true $c = c1, i = i1$ $c = c1, i = i1$ $c = c1, i = i1$

create(c=c1,i=i1) next(i=i1) next(i=i1) update(c=c1) next(i=i1)

Deployed pieces of advice: { create }

Executed pieces of advice: 3

Results

Went down from 9 updates in the unoptimized version over ...

... 5 updates after removal of loops ...

... to 3 updates updates through dynamic deployment.

So: Use of dynamic deployment yields potential for 40% speedup here!
And that only for one pair of participating objects!

Algorithmic approach

Algorithms involved are simple and can be calculated statically:

1. For each state, calculate relevant symbols, i.e. symbols which exists on outgoing non-loop edges.
2. In the matching advice, insert deploy/undeploy commands based on that information.
3. Generally, instance-based deployment should yield way better results.

Optimality

Can we still do any better?

Optimality

Can we still do any better?

... probably not.

Reason: At any point in time, we know that all which is done needs to be done.

But what about...?

Right, so what about Stack machines or Petri nets?

(may be used for context-free or counting expressions)

For both, reachability is decidable.

- Pushdown systems: use “p-automata”. Low polynomial overhead. [EHRS00]
- Petri nets: use “coverability graph”. Exponential overhead. [May81, EN94]

Why using trace matching at all?

“I don’t like those strange formalisms. Can we not do the same with plain AspectJ?”

Why using trace matching at all?

“I don’t like those strange formalisms. Can we not do the same with plain AspectJ?”

No! We cannot...

Pure AspectJ

```
aspect SafeEnum {

  after(Vector c) returning(Enumeration e):
  call(Enumeration+.new(..)) && args(c) {
    if(in state 0) {
      //take transition to state 1,
      //storing c and e
    }
  }

  before(Vector c):
  vector_update() && target(c) {
    if(in state 2 for object c) {
      //take transition to state 3 for c
    }
  }

  //advice for 3rd symbol "next" here
}
```

Problems / Future work

- How expensive is dynamic deployment?
- Undeploy vs. disable advice
- Does VM support really buy us anything?

Thank you for your attention.

Bibliography I

C. Allan, P. Avgustinov, A.S. Simon, L. Hendren, S. Kuzins, O. Lhoták, O. de Moor, D. Sereni, G. Sittamplan, and J. Tibble.
Adding Trace Matching with Free Variables to AspectJ.
In *OOPSLA '05, San Diego, California, USA*, October 2005.

Pavel Avgustinov, Julian Tibble, Eric Bodden, Ondřej Lhoták, Laurie Hendren, Oege de Moor, Neil Ongkingco, and Ganesh Sittampalam.
Efficient trace monitoring.
Technical report, AspectBench development group, `http://www.aspectbench.org`, 03 2006.

Eric Bodden.
J-LO - A tool for runtime-checking temporal assertions.
Master's thesis, RWTH Aachen University, Germany, Nov 2005.

J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon.
Efficient algorithms for model checking pushdown systems.
In *Proceedings of CAV'2000*, number 1855 in Lecture Notes in Computer Science, pages 232–247. Springer-Verlag, 2000.

J. Esparza and M. Nielsen.
Decidability issues for petri nets - a survey.
Bulletin of the European Association for Theoretical Computer Science, 52:245–262, 1994.

Bibliography II

Ernst W. Mayr.
An algorithm for the general petri net reachability problem.
In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 238–246, New York, NY, USA, 1981. ACM Press.

Robert O'Callahan Simon Goldsmith and Alex Aiken.
Light-Weight Instrumentation From Relational Queries Over Program Traces.
Technical Report UCB/CSD-04-1315, EECS Department, University of California, Berkeley, 2004.

Davy Suvée, Wim Vanderperren, and Viviane Jonckers.
JAsCo: an aspect-oriented approach tailored for component based software development.
In *AOSD*, pages 21–29, 2003.

Robert J. Walker and Kevin Viggers.
Implementing protocols via Declarative Event Patterns.
In *SIGSOFT FSE*, pages 159–169, 2004.