Trace Analysis using Model Checking

Hubert Garavel and Radu Mateescu

Inria and LIG / Convecs team

http://convecs.inria.fr

Outline

1. A few definitions
2. A bit of concurrency theory: Beware of traces!
3. The SEQ format for off-line traces
4. The SEQ.OPEN tool of CADP
5. Regular and non-regular properties
6. Concluding remarks

A FEW DEFINITIONS

Definitions

- Traces = chronological list of events
- Events are related to the system under study:
  - *black box*: inputs, outputs, exceptions raised
  - *white box*: internal states or actions
- Goal: Check the correctness of traces
- Nondeterminism = multiple runs of the system with the same inputs produce different traces

Off-line vs on-line traces

- Off-line trace = trace stored in a "log file"
  - verification by trace-based model checking
  - advantage: verifying several properties on one trace is easier (as multiple runs may produce different traces due to nondeterminism)

- On-line trace = trace generated on the fly as the system executes
  - verification by run-time monitoring
  - advantages: errors can be detected earlier, as soon as they occur

A BIT OF CONCURRENCY THEORY:

BEWARE OF TRACES!

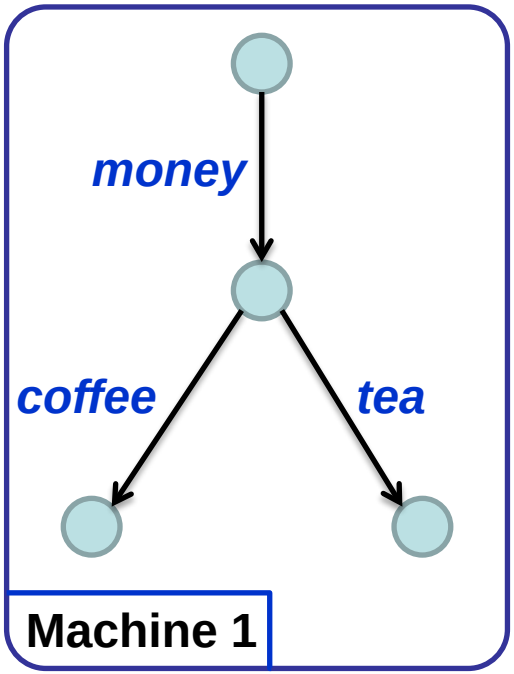
Traces are useful but insufficient

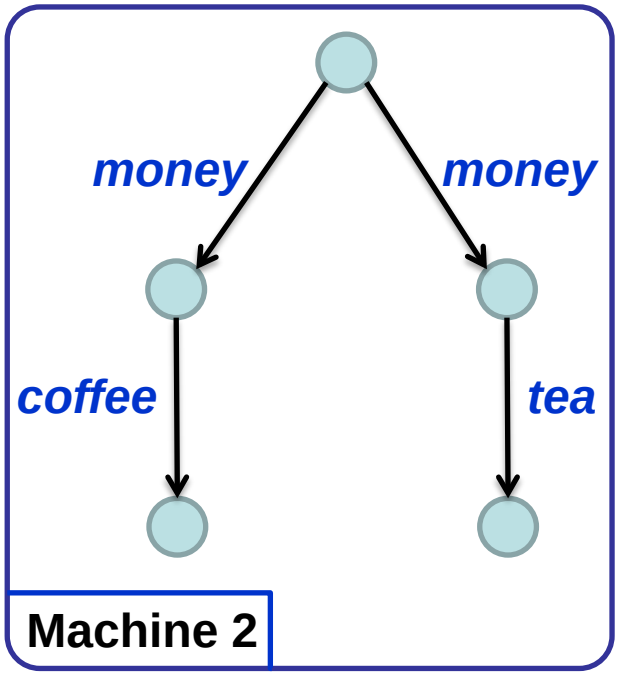
- Trace equivalence between components:
  $C_1 \approx C_2 \Leftrightarrow_{def} C_1$ and C_2 have the same set of traces
- A simple and convenient definition
- Unfortunately, in a distributed setting (i.e., with nondeterminism), trace equivalence does not guarantee substitutivity:

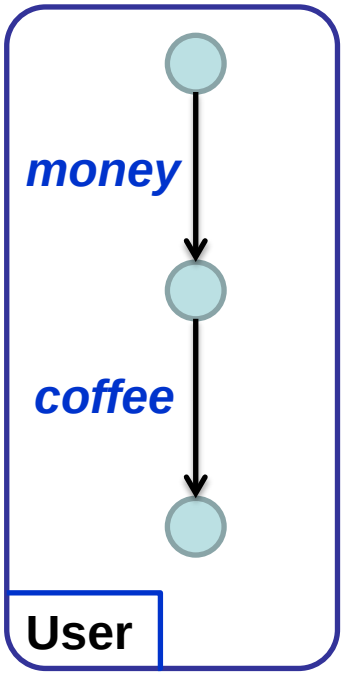
  Given some component X

  $C_1 \approx C_2$ does not imply $C_1 \,||\, X \approx C_2 \,||\, X$
- One needs stricter equivalences (bisimulations)

Example: the coffee-vending machine







traces (Machine 1) = { *money*.*coffee* , *money*.*tea* }
traces (Machine 2) = { *money*.*coffee* , *money*.*tea* }
traces (Machine 1 || User) = { *money*.*coffee* }
traces (Machine 2 || User) = { *money*.*coffee* , *money* }

THE "SEQ" FORMAT FOR OFF-LINE TRACES

Motivation for the SEQ format

- Off-line traces ("log files") are easy to produce
- We want to handle large traces (~ 10^6-10^8 events)
- No *a priori* limitation on:
  - the length of traces (i.e., number of events)
  - the size of event strings (it is application-dependent)
- Traces should be encoded in simple text files
  - human-readable
  - XML too verbose, limited added value, CPU expensive
  - one event per line, enclosed between "…"
  - comments are allowed (any text not between "…")
  - multiple traces allowed, separated by []

Exemple of a SEQ file

1st trace begins here

```
"PRI_INIT !PRB0 !COH !PMWI_D !ADDR !WB !VECTOR(00001) !PRB0 !SRCTXNID !M0_SNC0 !0"
"ILU_RESP !PRB0 !COH !NOCMP !IRETRY !NORMAL !0 !VECTOR(1000000)"
"ILU_REQ !OP_NIL !SRCTXNID !M0_SNC0 !WB !PMWI_D !ADDR !COH !VECTOR(0000000) !PRB0 !0"
"PRR_UPD_CLCOL !PRB0 !COH !VECTOR(0000000) !NODATA !CMP !PCMP !NORMAL !0"
"ILU_RELEASE !PCMP !COH !WB !SRCTXNID !M0_SNC0 !CMP"
"PRR_UPD_CLCOL !PMWW !DIRUPDATE !DIR_E !VECTOR(1000000) !ADDR"
"PRBE_RELEASE !PRB0"
"PRR_REQ !OP_NIL !SRCTXNID !M0_SNC0 !UC !PRLC !COH !VECTOR(0000000) !VECT"
"ILU_RESP !PRB0 !COH !NOCMP !IRETRY !NORMAL !0 !VECTOR(0001000)"
...
```

2nd trace begins here

```
[]
"PRI_INIT !PRB0 !COH !PRLD !ADDR !WB !VECTOR(00001) !PRB0 !SRCTXNID !M0_SNC0 !0"
"PRR_UPD_CLCOL !PRB0 !COH !VECTOR(0000001) !DATA !NOCMP !PDATA !NORMAL !0"
"ILU_RESP !PRB0 !COH !NOCMP !NULL !NORMAL !0 !VECTOR(0001000)"
"PRR_UPD_CLCOL !NOT_PMWW !DIRNOUPDATE !DIR_NOE !VECTOR(1001000) !ADDR"
"PRI_INIT !PRB0 !COH !PMWE_D !ADDR !WM !VECTOR(00100) !PRB0 !SRCTXNID !M0_SNC0 !0"
"PRR_UPD_CLCOL !PRB0 !COH !VECTOR(0000000) !NODATA !CMP !PCMP !NORMAL !0"
"ILU_RELEASE !PCMP !COH !WM !SRCTXNID !M0_SNC0 !CMP"
"PRR_UPD_CLCOL !PMWW !DIRUPDATE !DIR_E !VECTOR(1000000) !ADDR"
...
```

THE "SEQ.OPEN" TOOL OF CADP

The CADP toolbox

http://cadp.inria.fr

- Main features:
  - formal specification (LNT, LOTOS, EXP, FSP, etc.)
  - equivalence checking (bisimulations)
  - model checking (MCL, XTL)

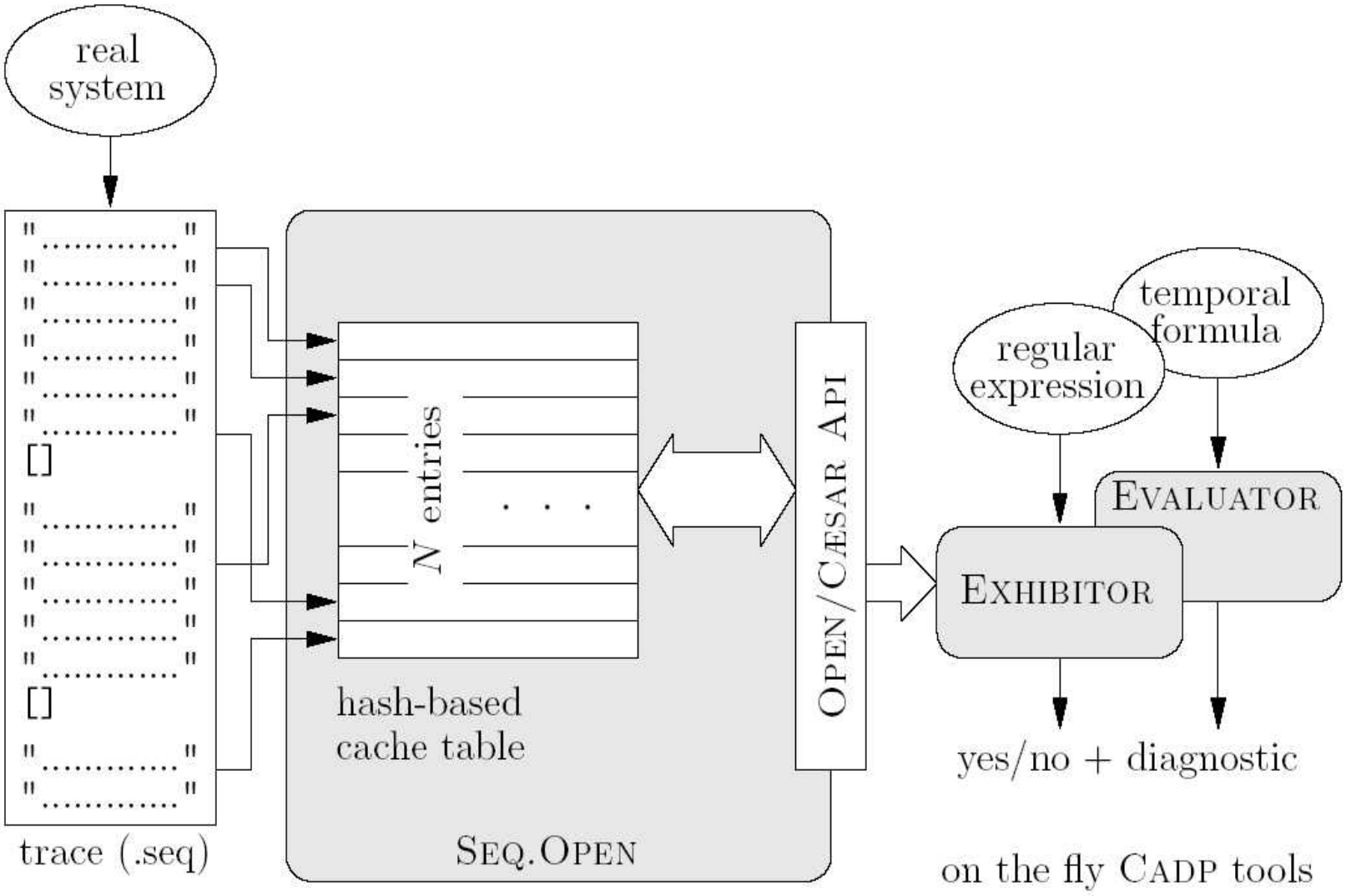
  } enumerative / on the fly / compositional / distributed

  - visual checking (graph drawing)
  - interactive, guided, and random simulation
  - (sequential and distributed) code generation
  - test case generation
  - performance evaluation
- Dissemination:
  - 10,000th license granted in 2012
  - more than 156 published case studies
  - more than 71 external research tools

The SEQ.OPEN tool

- SEQ.OPEN reads a SEQ file containing one or many traces
- These traces are partially loaded in main memory
  - loading in memory ensures faster access
  - but traces may be too large to be loaded entirely
  - hash-based encoding and cache tables are used
- These traces are checked using other CADP tools
- Examples of CADP tools relevant for traces:
  - BISIMULATOR: check for trace inclusion in an graph
  - EVALUATOR: model checking of temporal formulas
  - EXHIBITOR: search for regular expressions
  - etc.

Architecture of SEQ.OPEN



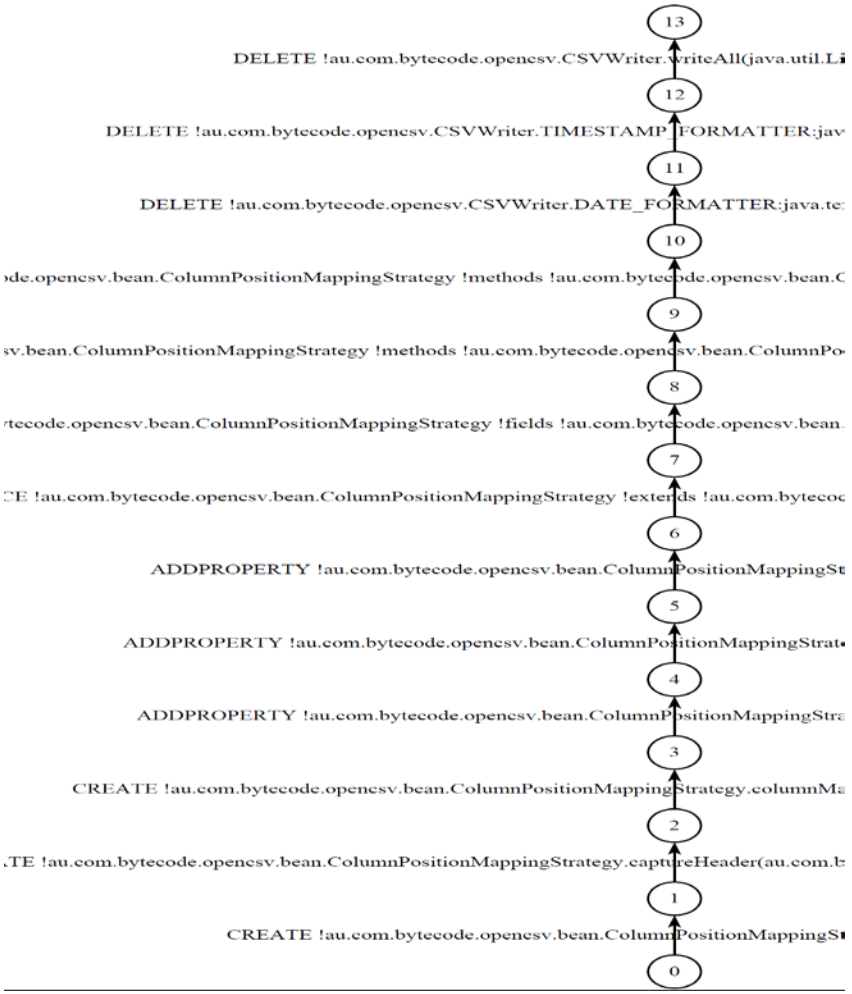
REGULAR AND NON-REGULAR PROPERTIES

Two applications

- Hardware design: Multiprocessor architectures
  - Bull "NovaScale" servers ("FormalFame" project)
  - random/guided simulation of Verilog designs
    $\Rightarrow$ large traces (>100,000 events)
  - correctness and coverage checking using SEQ.OPEN
- Software architectures
  - IST Project "Archware"
  - traces generated by execution of a multi-threaded VM
  - correctness checking using SEQ.OPEN

A recent example

- Harmony software mining project (Xavier Blanc, LaBRI)
- Goal: Analyze event histories in SVN repositories
- Queries expressed as temporal logic formulas in the MCL language of CADP



Example of a regular property

- Query: find all classes created between SVN revisions X and Y

- Corresponding MCL property:

```
[   true* .
    { CREATE  ?c:string  !"method"  ?V:int  ?any
                                    where (X < V) and (V < Y) }
]   print (c, "\n")
```

similar to a POSIX "grep", extended with data values

Example of a non-regular property

- Query: find all classes created between SVN revisions X and Y and later deleted

- Corresponding MCL property:

```
[  true* .
   { CREATE  ?C:string  ?M:string  ?V1:int  ?any
                                   where (X < V1) and (V1 < Y) } .
   true* .
   { DELETE !C !M ?V2:int ?any where V1<V2 }
]  print (C, "\n")
```

Other operations on traces

Besides checking properties, the CADP toolbox provides many other functionalities:

- generate traces from high-level languages
- hide events matching a given pattern
- minimize a trace to remove hidden events
- compare two traces (ignoring hidden events)
- merge a set of traces into a minimal tree

CONCLUDING REMARKS

Conclusion

- Traces: a pragmatic approach
  - trace checking is easily accepted in industry
  - beware of theoretical issues in a concurrent setting
- Software tools already exist for handling traces
  - do not redevelop a model checker for traces
  - reuse the existing CADP technology
- On-line traces are also supported
  - they can be encoded in the OPEN/CAESAR framework
  - the CADP on-the-fly tools also apply to on-line traces