1918
TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Synthesis of On-line Planning Tester for Non-deterministic EFSM Models

Marko Kääramees
Jüri Vain
Kullo Raiend

Tallinn University of Technology
Eliko Competence Centre
Elvior

Overview

- Scope and main idea of the work
- Workflow of testing
- Off-line preparation algorithm and example
- On-line testing algorithm and example
- Implementation and complexity issues
- Conclusions

Scope of the work

- Black box model based testing
  - tests are generated from the model
- Model is non-deterministic
  - on-line testing needed
  - output observability assumed
- Several test goals are tackled at the same time
  - minimizing the amount and length of the tests

Testing non-deterministic models

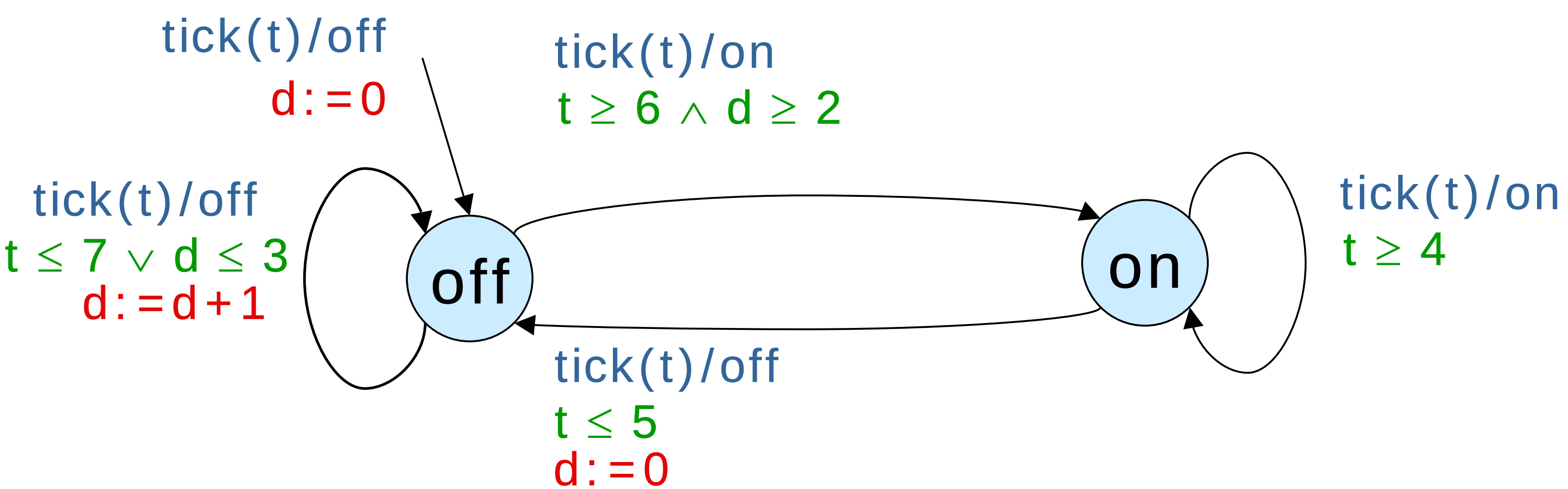
- Test cases cannot be prepared beforehand
- Tester must decide inputs during the test based on observed outputs and active goals
- Test planning is costly and not feasible on-line

Proposed solution

- Model is analysed off-line
- Result is expressed as a set of data constraints for each test goal
- Data instance generation is done on-line

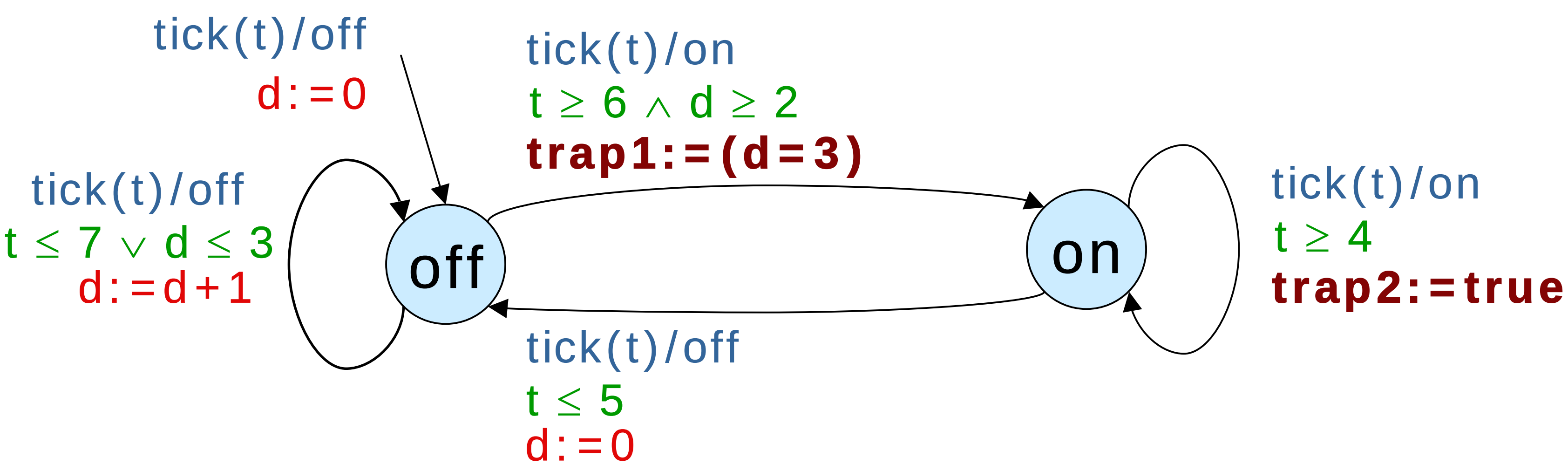
Model of SUT

- Model is given as EFSM
  - input/output, guard, update
  - input parameter t [temp] and variable d [delay]
- Requirements
  - fridge must switch off when t is 4..5
  - fridge must switch on when t is 6..7 and it has been off 20..39 seconds (tick every 10 seconds)

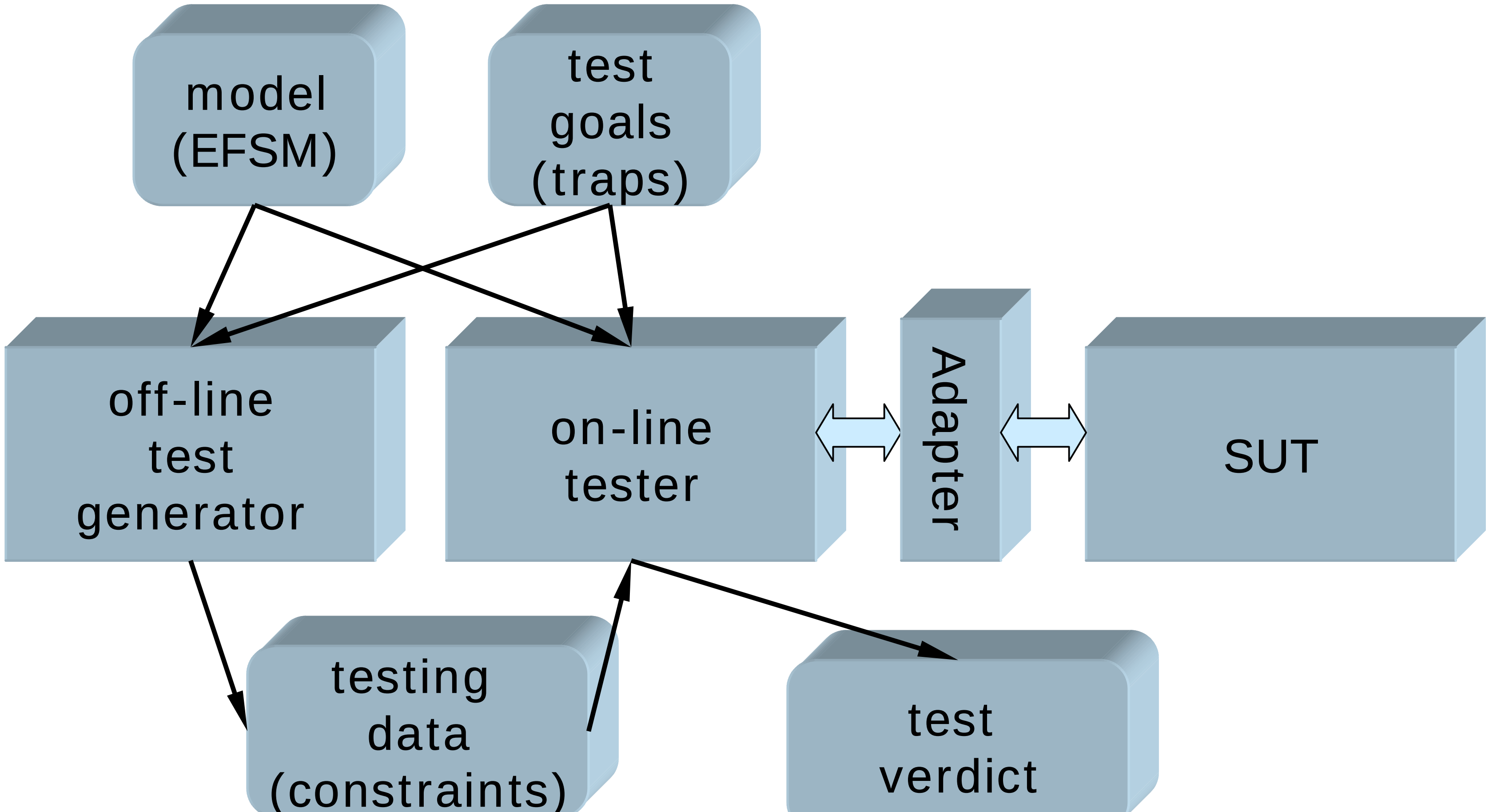


Modeling of test goals

- Test goals are expressed by traps
  - trap is a pair <transition,predicate>
  - expressed as update of trap variable in model
- Can express
  - transition coverage
  - transition sequence
  - repeated pass using auxiliary variable



Workflow


model
(EFSM)
test
goals
(traps)
off-line
test
generator
on-line
tester
Adapter
SUT
testing
data
(constraints)
test
verdict

Constraints

- A set of constraints is generated for every trap
  - help to guard the on-line tester towards the trap
- Constraints for states
  - Minimal path constraint C_s

    condition for the shortest paths to trap *tr* from state *s*
  - Maximal path constraint C^*_s

    condition for all paths to trap *tr* from state *s* that extend the constraint
- Constraints for transitions
  - Minimal C_t and maximal C^*_t as for states
  - Guarding constraint C^g_t

    if the shortest path to the trap starts with the transition
- Path lengths L_s , L^*_s , L_t and L^*_t are recorded also

Offline algorithm for trap *tr*

initialise C to *false*, L to 0

$C^*_t = guard_t \wedge condition_{tr}$

while fixpoint or search *depth* is reached

  for each state s on the depth level do

    $C^*_s = \text{simplify}(C^{*'}_s \vee \exists I{:}\vee C^*_{ti})$ // ti - t leaving from s; I - *input*

    if SAT($\neg(C^*_s \Rightarrow C^{*'}_s)$) // C^*_s changed

      $L^*_s = depth$

      if not C_s // minimal constraint

       $C_s = C^*_s$; $L_s = L^*_s$

      for each transition t coming to s

        $C^*_t = \text{simplify}(C^{*'}_t \vee guard_t \wedge wp(update_t, C^*_s))$

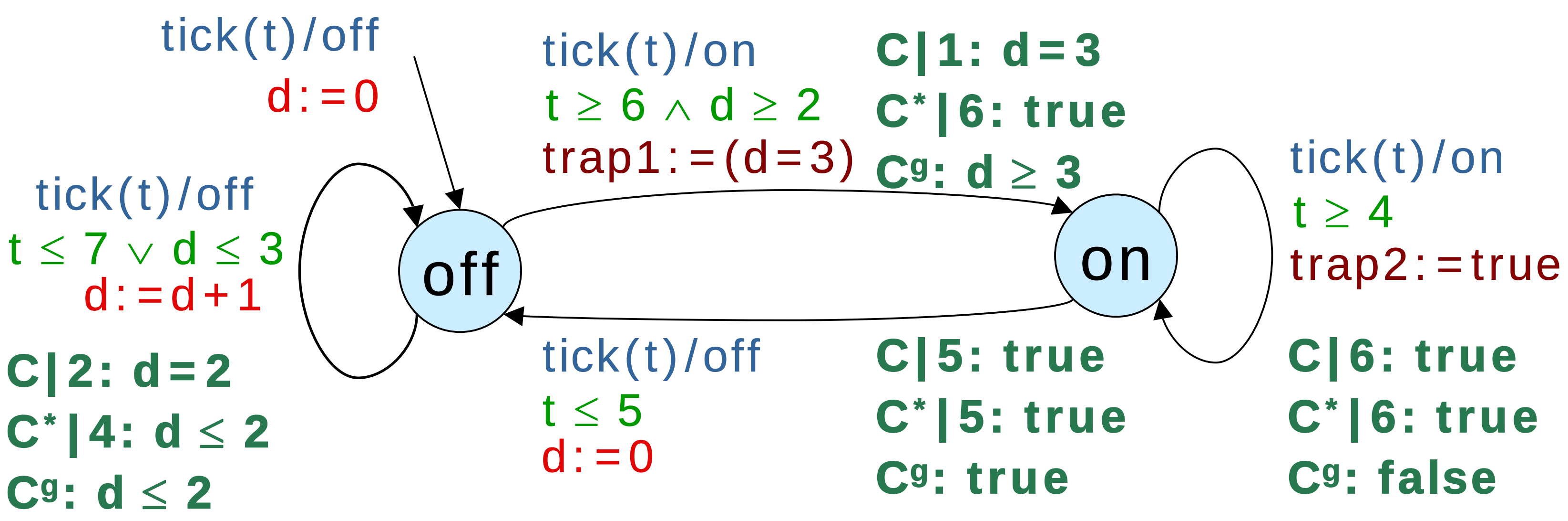
        record L^*_s, C_t, L_s if needed

        $C^g_t = \text{simplify}(C^{g'}_t \vee (\exists I{:}C^*_t \wedge \neg C^*_{source(t)}))$

Off-line constraint generation

Constraints for trap1:

- Constraints C|L give the condition and length for the shortest path
- Constraints $C^*|L^*$ give the condition and length for all paths up to fixpoint (or search depth)
- Constraints C^g give the condition for choosing the next transition depending on the values of variables



On-line algorithm (greedy)

while exist uncovered traps //at state *s*

select nearest reachable trap *tr* // using SAT()

select transition with C^g_t satisfiable // using SAT()

select input parameters valuation by

solving C_t or C^*_t // constraint solving

communicate the inputs to SUT

if the output does not conform to the model // using SAT()

stop(test_failed)

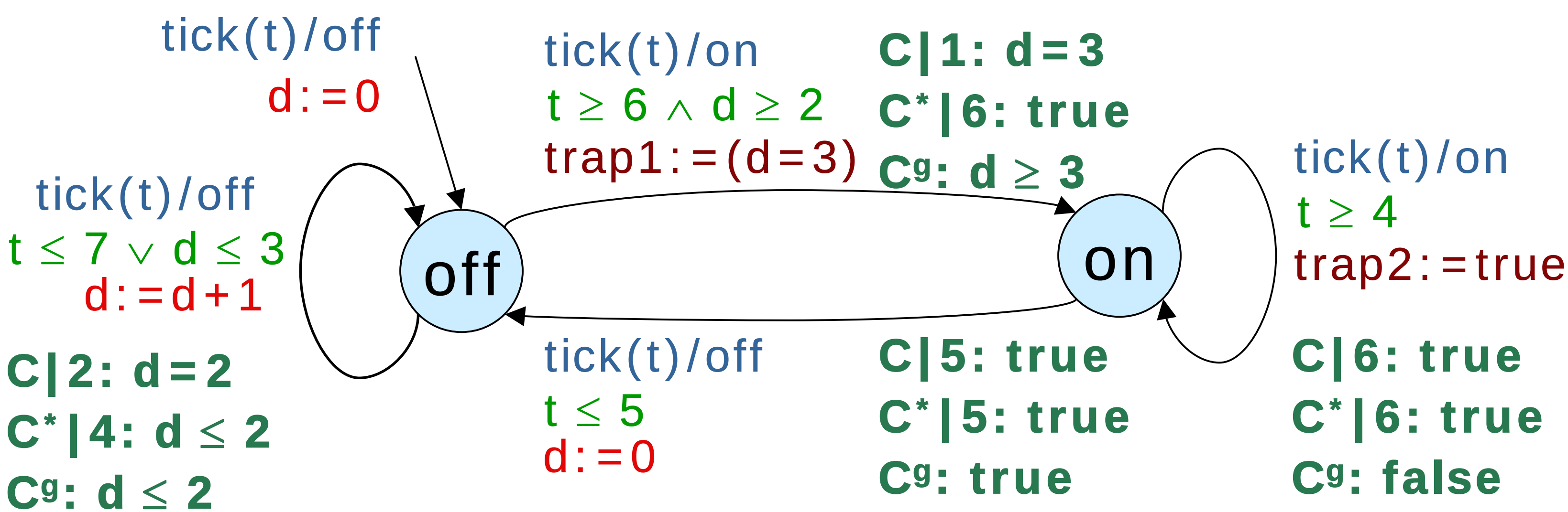
move to the next state

end while

stop(test_passed)

Example (on-line)

1.	tick(true):	off, d=0	
2.	tick(true):	off, d=1	
3.	tick(true):	off, d=2	
4.	tick($t < 6$):	off, d=3	
5.	tick($t \geq 6$):	on, d=3 trap1☺	off, d=4
6.	tick($t > 7$):		on, d=4
7.	tick($t > 5$):		on, d=4 trap2☺
8.	tick($t < 4$):		off, d=0



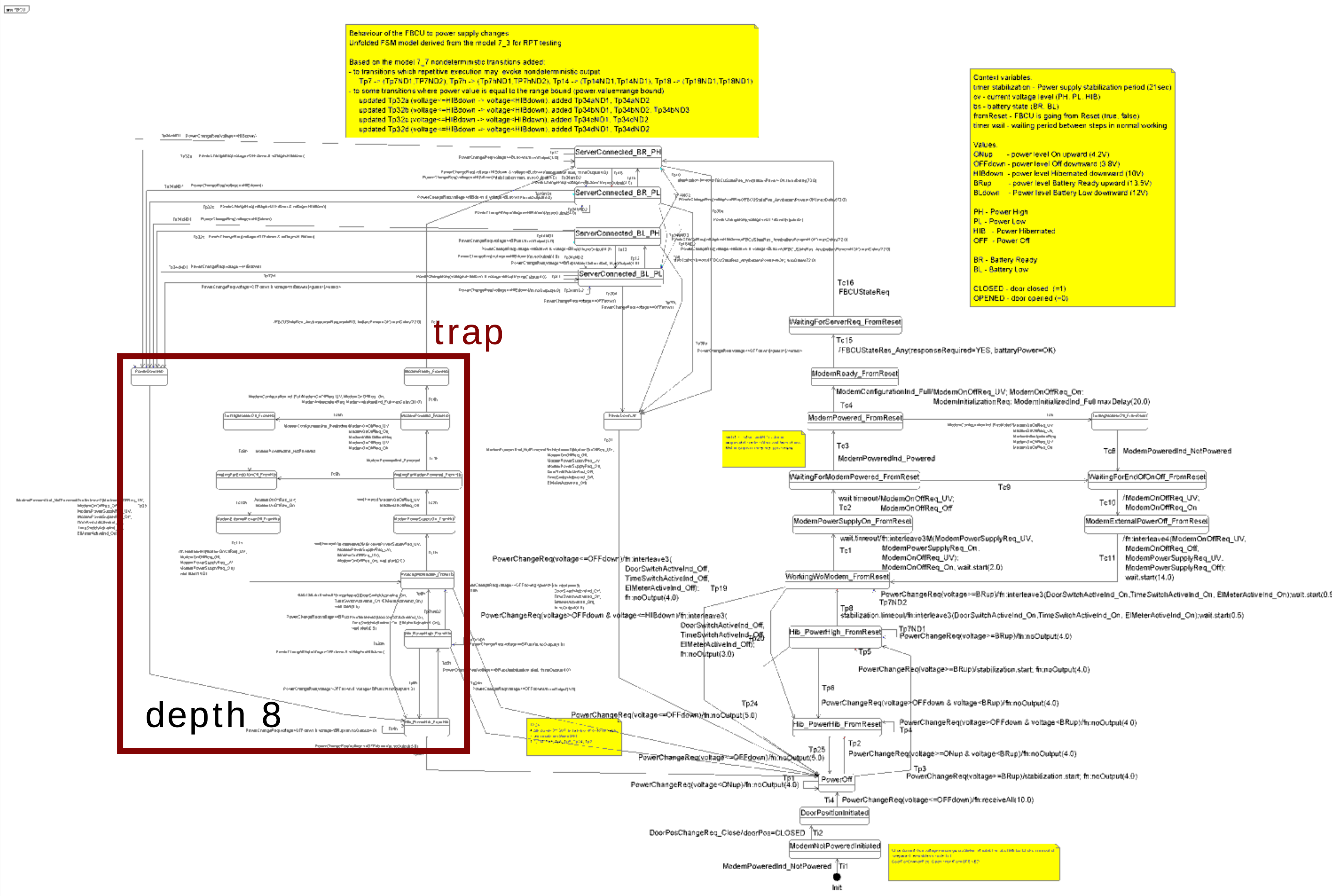
Implementation issues

- UPPAAL used for modelling (Uppsala & Aalborg U)
- Z3 SMT solver suite (Microsoft Research)
  - simplification of constraints
  - quantifier elimination
  - SAT solver
  - constraint solving (model generation)
- Python scripts for parsing and constraining generation algorithm implementation
- TestCast - TTCN3 toolset (Elvior)
  - running generated TTCN3 scripts

Complexity issues

- Constraints limited to decidable theories
  - linear arithmetic (+ others supported by solver)
- Theoretical limits
  - SAT problem is NP-complete
  - decision procedures and simplification of Presburger arithmetic is double-exponential
- Practical aspects
  - number of constraints is in O(traps*transitions)
  - Z3 does a good job in SAT and simplification
- Search depth
  - complexity of the constraints depends on the structure of the model and search depth
  - search depth can be constrained off-line when the time for the SAT check needed on-line exceeds the predefined limit

Constrained search



Main results

- Tester for non-deterministic EFSM
- Efficient on-line test planning
  - supported by off-line preparation
- Off-line computation is usable also for off-line test cases generation for deterministic models
- On-line planning drives the test towards uncovered test goals resulting a test with sub-optimal length