Synthesis of Test Purpose Directed Reactive Planning Tester for Nondeterministic Systems

Jüri Vain
Dept. of Computer Science
Tallinn University of Technology

Lecture plan

- Preliminaries
  - Model-Based Testing
  - Online testing
- Reactive Planning Tester (RPT)
- Constructing the RPT
- Performance of the approach
- Demo

Context: Model-Based Testing

- Given
  - a specification model and
  - an Implementation Under Test (IUT),
- Find
  - whether the IUT conforms to the specification.

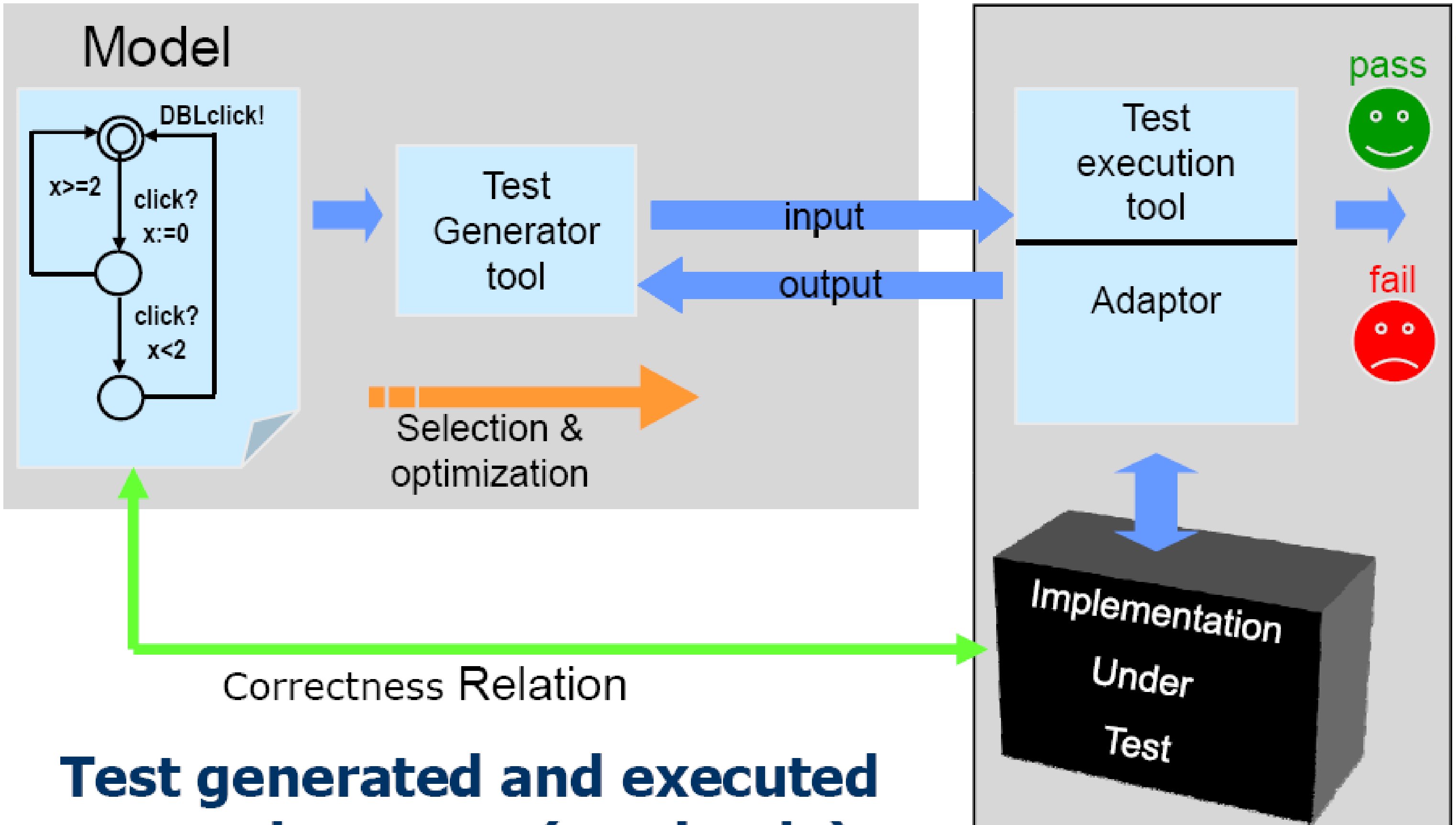
Model-Based Testing

- The specification needs to be formalised. We assume models are given as
  - Extended Finite State Machines
  - XTA
  - …

Online testing

- Denotes test generation and execution algorithms that
  - *compute successive stimuli at runtime* directed by
    - the test purpose and
    - the observed outputs of the IUT

Online Testing



Test generated and executed event-by-event (randomly)

A.K.A on-the-fly testing

see, e.g., Uppaal family tools for online testing

Online testing

- Advantages:
  - The *state-space explosion* problem *is reduced* because only a limited part of the state-space needs to be kept track of at any point in time.
- Drawbacks:
  - *Exhaustive planning is diffcult* due to the limitations of the available computational resources at the time of test execution.

Online testing: spectrum of methods

- Random walk (RW): select test stimuli in random
  - inefficient - based on random exploration of the state space
  - leads to test cases that are unreasonably long
  - may leave the test purpose unachieved
- RW with reinforcement learning (anti-ant)
  - the exploration is guided by some reward function
- .... ???
- Exploration with exhaustive planning
  - MC *provides possibly an optimal* witness trace
  - the *size of the model is critical* in explicit state MC
  - state explosion in "combination lock" or deep loop models

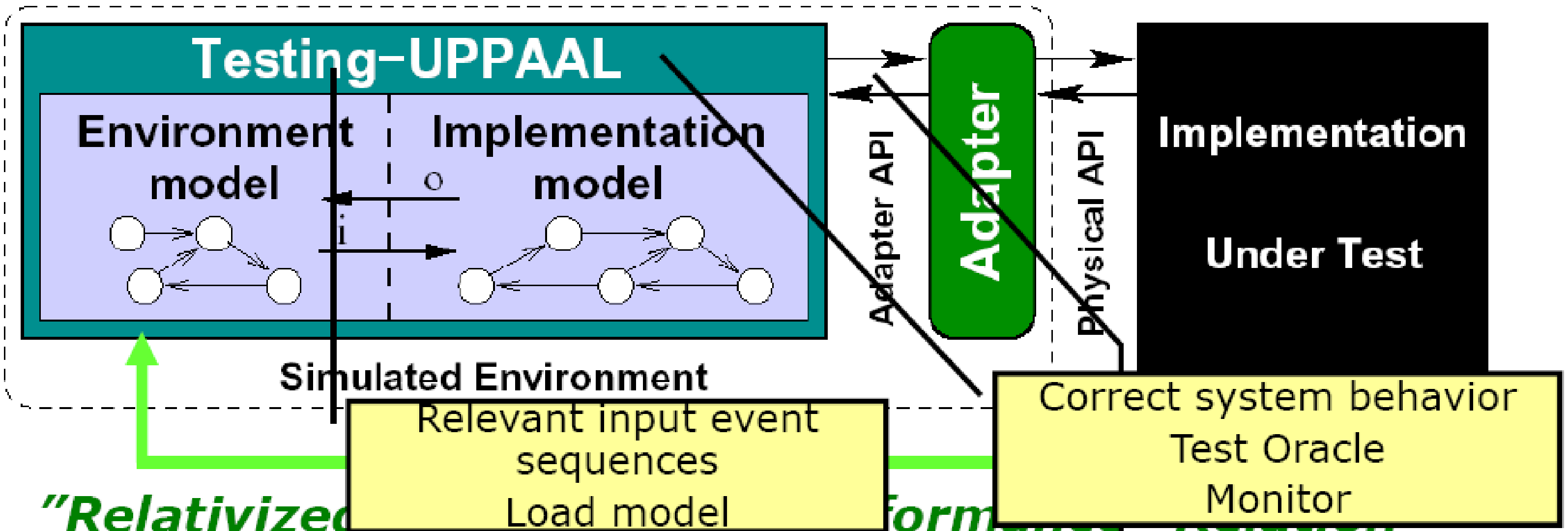
Online testing: spectrum of methods

- Random walk (RW): select test stimuli in random
  - inefficient - based on random exploration of the state space
  - leads to test cases that are unreasonably long
  - may leave the test purpose unachieved
- RW with reinforcement learning (anti-ant)
  - the exploration is guided by some reward function
- .... ⟵ Planning with limited horizon!
- Exploration with exhaustive planning
  - MC *provides* *possibly an optimal* witness trace
  - the *size of the model is critical* in explicit state MC
  - state explosion in "combination lock" or deep loop models

Tron Framework

UppAal-TRON: Testing **R**eal-Time Systems **On**line

Spec = UppAal Timed Automata *Network: Env || IUT*



Timed Trace: $i_1.2\frac{1}{2}.o_1.3.o_2.19.i_2.5.i_3$

Reactive Planning

- Instead of a complete plan with branches, a set of *decision rules* is derived
- The rules direct the system towards the planning goal.
- Just one subsequent input is computed at every step, based on the current context.
- Planning horizon can be adjusable

Reactive Planning
[Brian C. Williams and P. Pandurang Nayak, 96 and 97]

- A Reactive Planning works in 3 phases:
  - Mode identification (MI)
  - Mode reconfiguration (MR)
  - Model-based reactive planning (MRP)
- MI and MR set up the planning problem identifying initial and target states
- MRP generates a plan

Reactive Planning Tester

- MI – Where are we? Observe the output of the IUT to determine the current mode (state of the model)
- MR – Where do we want to go? Determined by still unsatisfied subgoals
- MRP – How do we get there? Gain guards choose the the next transition with the shortest path to the next subgoal

Reactive Planning Tester

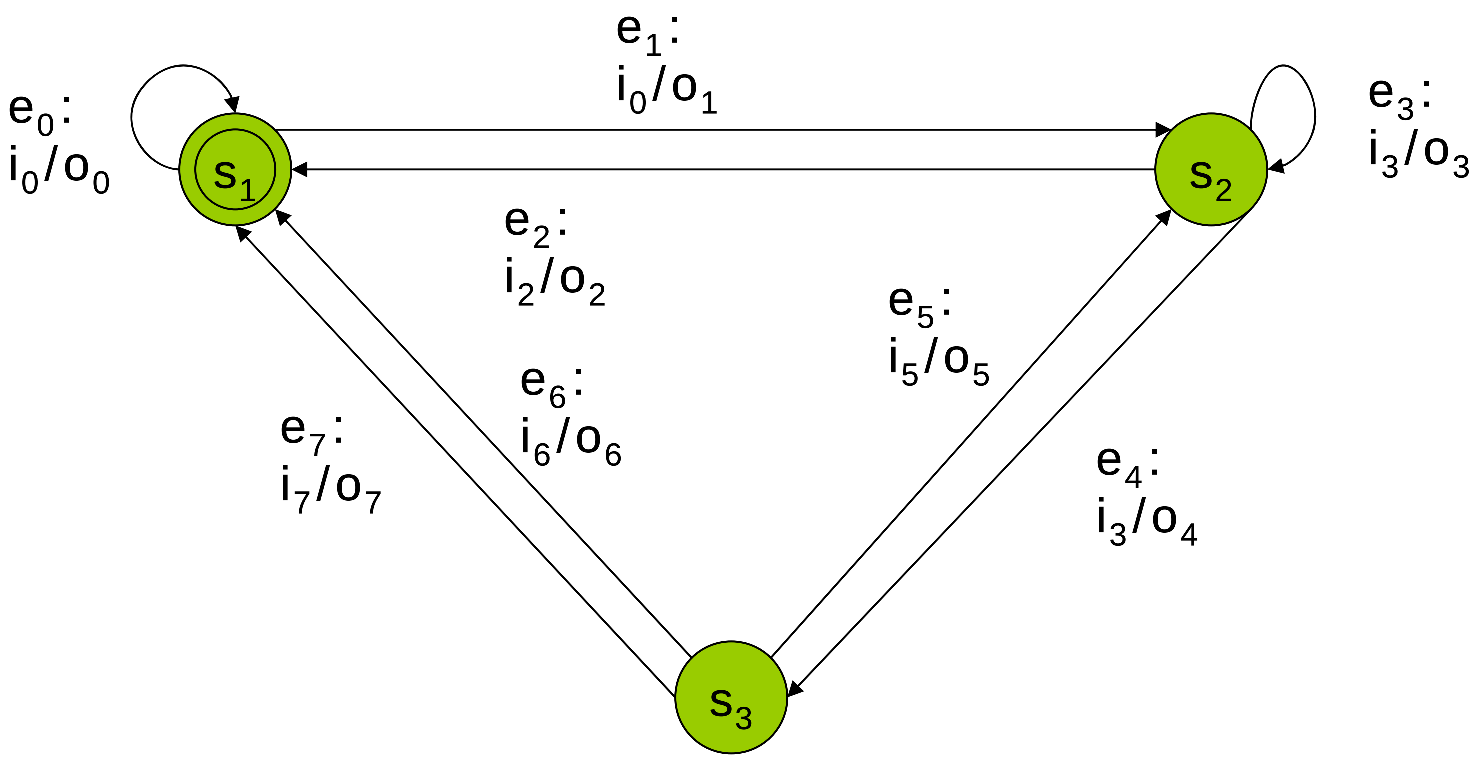
- Key assumptions:
  - Testing is guided by the (EFSM) model of the tester and the test purpose
  - Stimulae to the IUT *are tester outputs generated* by model execution
  - Responses from the IUT are *inputs* to the tester model
  - Decision rules of reactive planning are encoded in the *guards* of the transitions of the tester model
  - The rules are constructed by offline analysis based on the given IUT model and the test purpose.

The Model

- The IUT model is presented as an *output observable nondeterministic* EFSM in which all *paths are feasible*

- Algorithm of making EFSM feasible [Duale, 2004]

Example: Nondeterministic EFSM

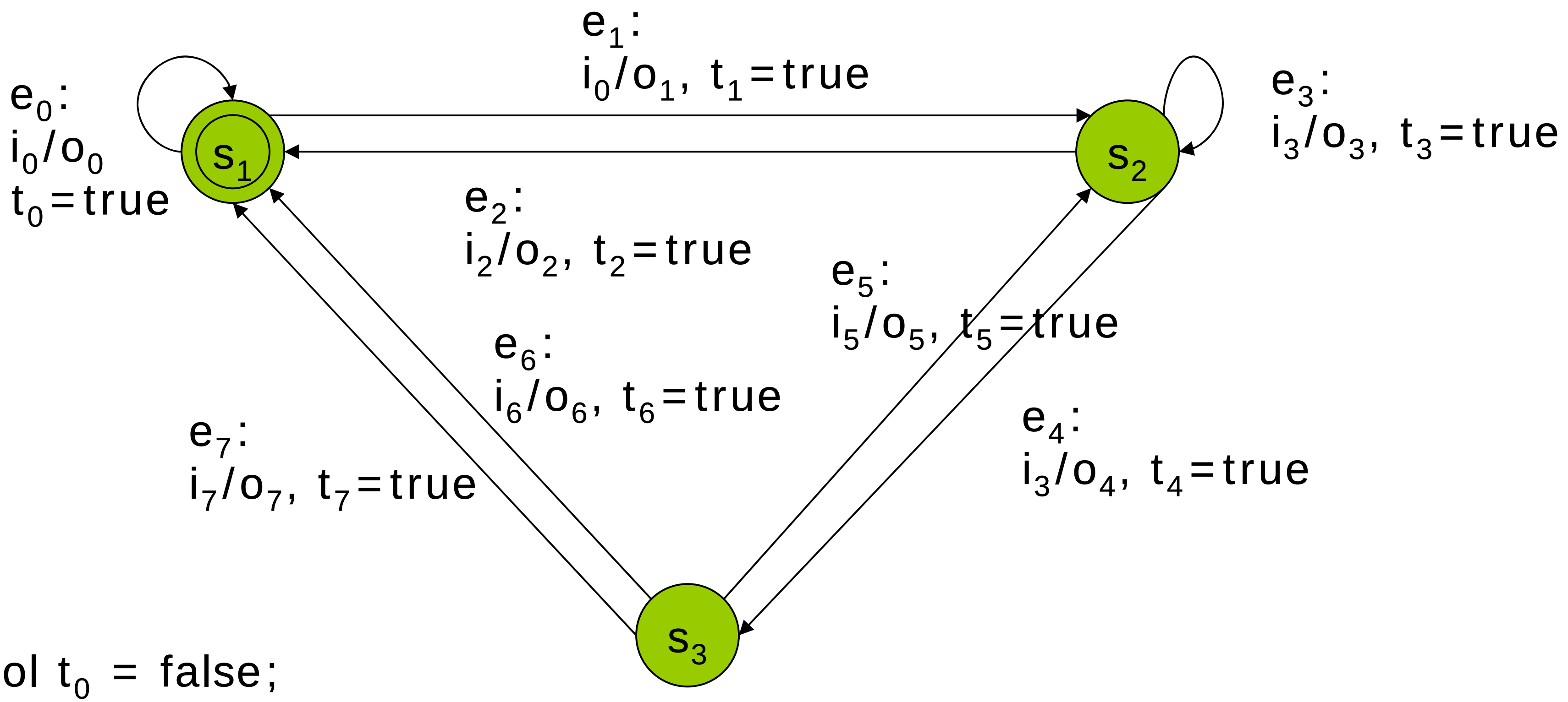


i_0 and i_3 are output observable nondeterministic inputs

Encoding the Test Purpose in IUT Model

- Trap - a boolean variable assignment attached to the transitions of the IUT model
- A trap variable is initially set to *false*.
- The trap update functions are executed (set to *true*) when the transition is visited.

Add Test Purpose



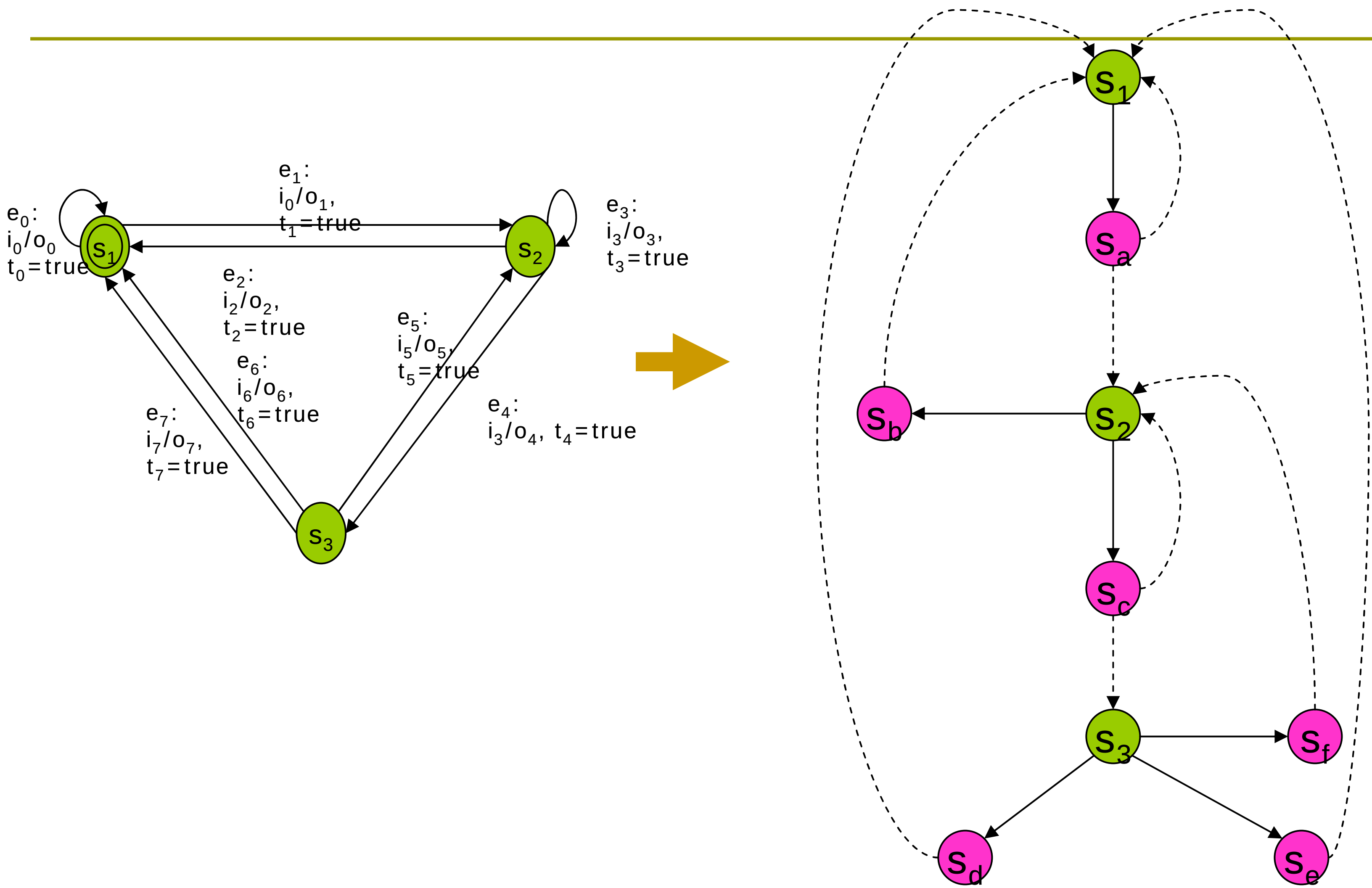
bool t_0 = false;
...
bool t_7 = false;

Model of the tester

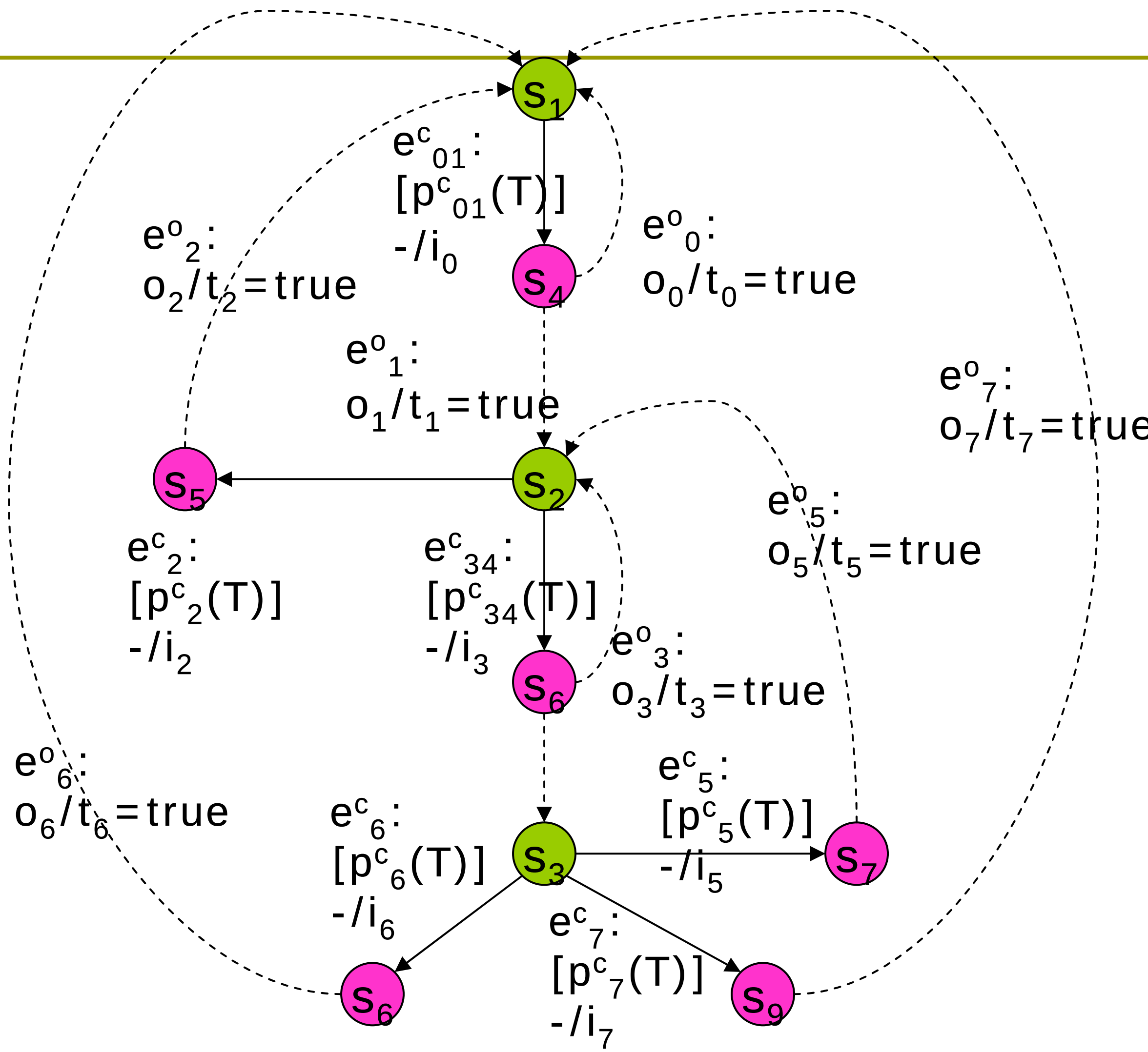
- Generated from the IUT model decorated with test purpose
- Transition guards encode the rules of online planning
- 2 types of tester states:
  - active – tester controls the next move
  - passive – IUT controls the next move
- 2 types of transitions:
  - Observable – source state is a passive state (guard ≡ *true),*
  - Controllable – source state is an active state (guard ≡ p_S /\ p_T where p_S – guard of the IUT transition; p_T – gain guard)

The *gain guard* (defined on trap variables) must ensure that only the outgoing edges with maximum gain are enabled in the given state.

Construction of the Tester



Add IO and Gain Guards



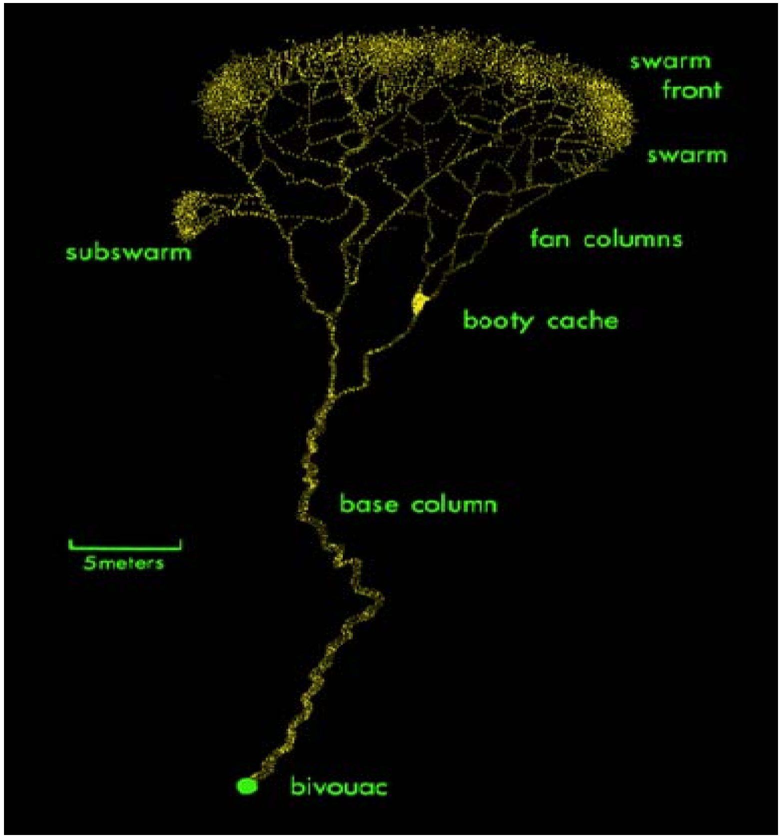
Constructing the gain guards (GG): intuition

- GG must guarantee that
  - each transition enabled by GG is a prefix of some locally optimal (w.r.t. test purpose) path;
  - tester should terminate after the test goal is reached or all unvisited traps are unreachable from the current state;
  - to have a quantitative measure of the gain of executing any transition *e* we define a gain function g_e that returns a distance weighted sum of unsatisfied traps that are reachable along *e.*

Recall lessons from nature: Collective Hunting Strategies

Benefits of Collective Hunting

- Maximizing prey localization
- Minimizing prey catching effort



Constructing the gain guards: the gain function

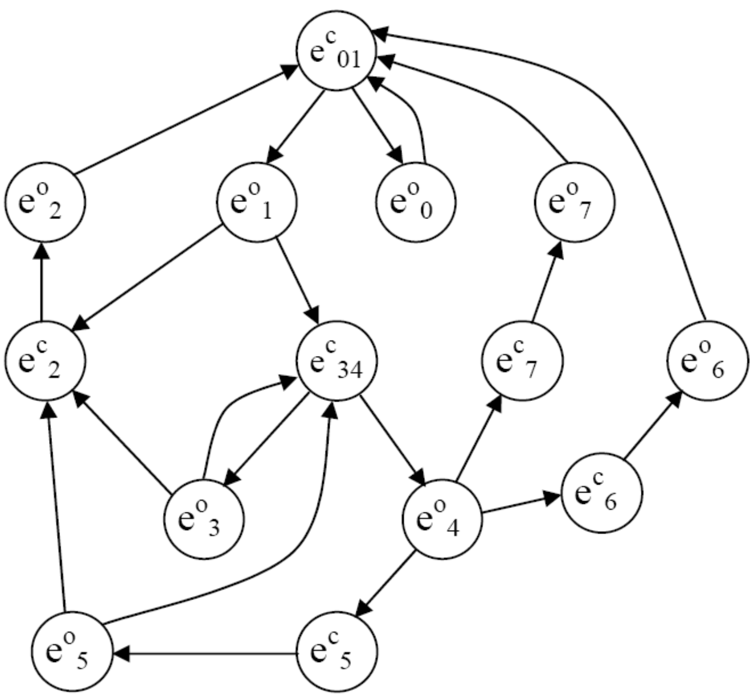
- $g_e = 0$, if it is useless to fire the transition e from the current state with the current variable bindings;
- $g_e > 0$, if fireing the transition e from the current state with the current variable bindings visits or leads closer to at least one unvisited trap;
- $g_{ei} > g_{ej}$ for transitions e_i and e_j with the same source state, if taking the transition e_i leads to unvisited traps with smaller distance than taking the transition e_j;
- Having gain function g_e with given properties define GG:

$$p_T \equiv (g_e = \max_k \; g_{ek}) \text{ and } g_e > 0$$

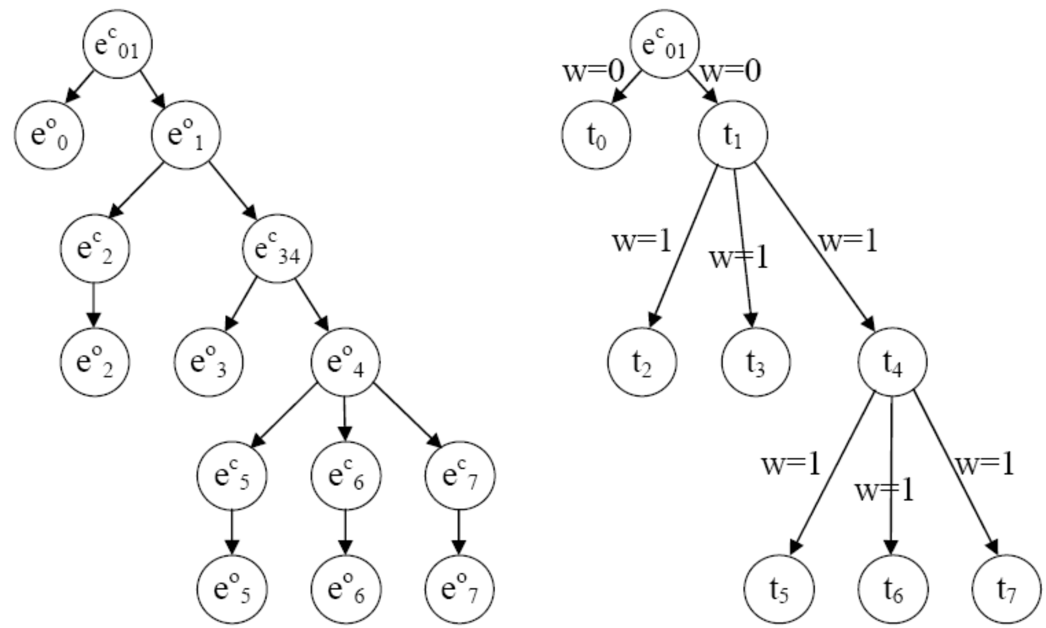
Constructing the Gain Functions: *shortest path trees*

- Reachability problem of trap labelled transitions can be reduced to *single-source shortest path problem*.
- Arguments of the gain function g_e are
  - Shortest path tree TR_e with root node e
  - V_T – vector of trap variables
- To construct TR_e we create a dual graph $G = (V_D, E_D)$ of the tester where
  - the vertices V_D of G correspond to the transitions of the M_T,
  - the edges E_D of G represent the pairs of subsequent transitions sharing a state in M_T (2-switches)

Constructing the Gain Guards: *shortest path tree (example)*



The dual graph of the tester model



The shortest-paths tree (left) and the reduced shortest-paths tree (right) from the transition e^c_{01}

Constructing the gain guards: *the gain function*

- Represent the reduced tree $TR(e_i, G)$ as a set of elementary sub-trees each specified by the production $\nu_i \leftarrow |_{j \in \{1,..n\}} \nu_j$
- Rewrite the right-hand sides of the productions as arithmetic terms:

$$\nu_i \rightarrow (\neg t_i)^{\uparrow} \cdot \frac{c}{d(\nu_0, \nu_i) + 1} + \max_{j=1,k}(\nu_j), \qquad (3)$$

  - $t\uparrow_i$ - trap variable t_i lifted to type $\mathbb{N}$,
  - c - constant for the scaling of the numerical value of the gain function,
  - $d(\nu_0, \nu_i)$ the distance between vertices ν_0 and ν_i, where

$$d(\nu_0, \nu_i) = l + \sum_{j=1}^{l} w_j$$

l - the number of hyper-edges on the path between ν_0 and ν_i

w_j – weight of j-th hyperedge

Constructing the gain guards: *the gain function* (continuation)

- For each symbol ν_i denoting a leaf vertex in *TR*(*e*,*G*) define a production rule

$$\nu_i \rightarrow (\neg t_i)^{\uparrow} \cdot \frac{c}{d(\nu_0, \nu_i) + 1} \qquad (4)$$

- Apply the production rules (3) and (4) starting from the root symbol ν_0 of *TR*(*e*,*G*) until all non-terminal symbols ν_i are substituted with the terms that include only terminal symbols $t\uparrow_i$ and $d(\nu_0, \nu_i)$

Example: Gain Functions

Transition	Gain function for the transition
e^c_{01}	$g_{e^c_{01}}(T) \equiv c \cdot max($ $\neg t_0/2,$ $\neg t_1/2 + max(\neg t_2/4, \neg t_3/4, \neg t_4/4+$ $max(\neg t_5/6, \neg t_6/6, \neg t_7/6)))$
e^c_2	$g_{e^c_2}(T) \equiv c \cdot (\neg t_2/2 + max($ $\neg t_0/4,$ $\neg t_1/4 + max(\neg t_3/6, \neg t_4/6+$ $max(\neg t_5/8, \neg t_6/8, \neg t_7/8))))$
e^c_{34}	$g_{e^c_{34}}(T) \equiv c \cdot max($ $\neg t_3/2 + \neg t_2/4 + max(\neg t_0/6, \neg t_1/6),$ $\neg t_4/2 + max(\neg t_5/4, \neg t_6/4, \neg t_7/4))$

Example: Gain Guards

Transition	Gain guard formula for the transition
e^c_{01}	$p^c_{01}(T) \equiv g_{e^c_{01}}(T) = max(g_{e^c_{01}}(T)) \wedge g_{e^c_{01}}(T) > 0$
e^c_2	$p^c_2(T) \equiv g_{e^c_2}(T) = max(g_{e^c_2}(T), g_{e^c_{34}}(T)) \wedge g_{e^c_2}(T) > 0$
e^c_{34}	$p^c_{34}(T) \equiv g_{e^c_{34}}(T) = max(g_{e^c_2}(T), g_{e^c_{34}}(T)) \wedge g_{e^c_{34}}(T) > 0$

Complexity of constructing and running the tester

- The complexity of the synthesis of the reactive planning tester is determined by the complexity of constructing the gain functions.
- For each gain function the cost of finding the TR_E by breadth-first-search is $O(|V_D| + |E_D|)$ [Cormen], where
  - $|V_D| = |E_T|$ - number of transitions of M_T
  - $|E_D|$ - number of transition pairs of M_T (is bounded by $|E_S|^2$)
- For all controllable transitions of the M_T the upper bound of the complexity of the computations of the gain functions is $O(|E_S|^3)$.
- At runtime each choice by the tester takes $O(|E_S|^2)$ arithmetic operations to evaluate the gain functions

Experimental results: All Transitions Test Purpose

Algorithm of the tester	Model 1 (8 trans.)	Model 2 (16 trans.)	Model 3 (32 trans.)
Random choice	56 ± 36	295 ± 130	1597 ± 1000
Anti-ant	21 ± 4	53 ± 13	218 ± 81
Reactive planner	17 ± 3	37 ± 6	80 ± 10

Experimental Results: One Transition Test Purpose

Algorithm of the tester	Model 1 (8 trans.)	Model 2 (16 trans.)	Model 3 (32 trans.)
Random choice	34 ± 35	120 ± 114	699 ± 719
Anti-ant	14 ± 7	36 ± 19	140 ± 70
Reactive planner	5 ± 2	8 ± 3	11 ± 3

Demo: "combination lock"

- Comparison of methods
  - Random search
  - Anti-ant
  - Reactive planning tester

Summary

- RP always drives the execution towards still unsatisfied subgoals.
- Efficiency of planning:
  - Number of rules that need to be evaluated at each step is relatively small (i.e., = the number of outgoing transitions of current state)
  - The execution of decision rules is significantly faster than looking through all potential alternatives at runtime.
  - Leads to the test sequence that is lengthwise close to optimal.

Questions?