Graphical Tracing of Supercompilation:

An Intermediate Report

Andrei P. Nemytykh
Program Systems Institute of Russian Academy of Sciences

VPT'13, 14 July 2013, Saint Petersburg, Russia

Outline

Introduction to Supercompilation

– *Super*vised *Compilation*.

– Semantic based program transformation technique (V.Turchin, 1960-70s).

– Can be used for optimization and specialization of (functional) programs.

– Much of the development has been done in the context of the Refal functional programming language.

– SCP4 is the most advanced implementation of supercompilation for Refal (A.Nemytykh, V.Turchin).

Supercompilation is a Method for Program Specialization

Given a program p and its functions (subprograms) f(x,y) и g(x).

- **f(x_0,y)**

- **f(g(x),y)**

Given a partially defined entry point Main, a supercompiler unfolds a potentially infinite tree T of all possible computations of p, starting at Main, optimizes T and folds it into a finite directed graph representing the result of supercompilation.

The problem is to generate an optimal residual program q (run-time optimization).

Supercompiler

– observes the behaviour of a functional program P running on partially defined input;

– unfolds a potentially infinite tree of all possible computations of P;

– reduces redundancy, e.g. by pruning *unreachable* fragments of code;

– folds the tree into a finite graph of parameterised configurations of P and transitions between them;

– basing on a graph of configurations constructs a new program, which is (almost) equivalent to the input program.

Resulting program defines a function which is an extention of the input function.

The initial specialization task Z, as well as intermediate tasks, can be automatically decomposed into a number of subtasks.

The supercompiler analyses a complicated structure – a forest of directed trees labeled with information on parameterized states at the trees' nodes and with conditions choosing the branches in a concrete computation of the program being supercompiled.

The supercompiler's task is to efficiently fold the forest into a finite directed graph.

The quality of the residual programs produced by supercompilation depends crucially on strategies chosen for the supercompilation process.

The most strong strategies may lead to:

- **a long supercompile-time for some source programs;**

- **or endless of the process.**

These data makes difficult understanding the linear tracing texts encoding the graphs and such an understanding frequently takes a long time and amounts to manual drawing the graphs.

We have developed and implemented a start version of a tool supporting graphical representation of SCP4's traces. The tool uses:

– free Graph Visualization Software (AT&T Research), which takes care of placement of output subgraphs in a way that is convenient for visual observation (a uniform subgraph density over the screen);

– a graph visualizer ZGRViewer (open software) allows users to navigate through such graphs and to scale them in various ways.

Implementation Principles of the Graphical Tracer

- **The graphical SCP4's tracer is based on a concept of a graphical step:**
  - **corresponds to a logical step changing the directed graph being analyzed and transformed;**
  - **such graphical steps are not uniform in the logical aspects, they correspond to the logically closed steps of different mechanisms of the supercompiler.**

The Graphical Step

– At the end of each such a step the tracer generates a graphical description of the SCP4 current state in a graph specification language Dot (AT&T). Dot is an abstract specification language not depending on any specific graphical format.

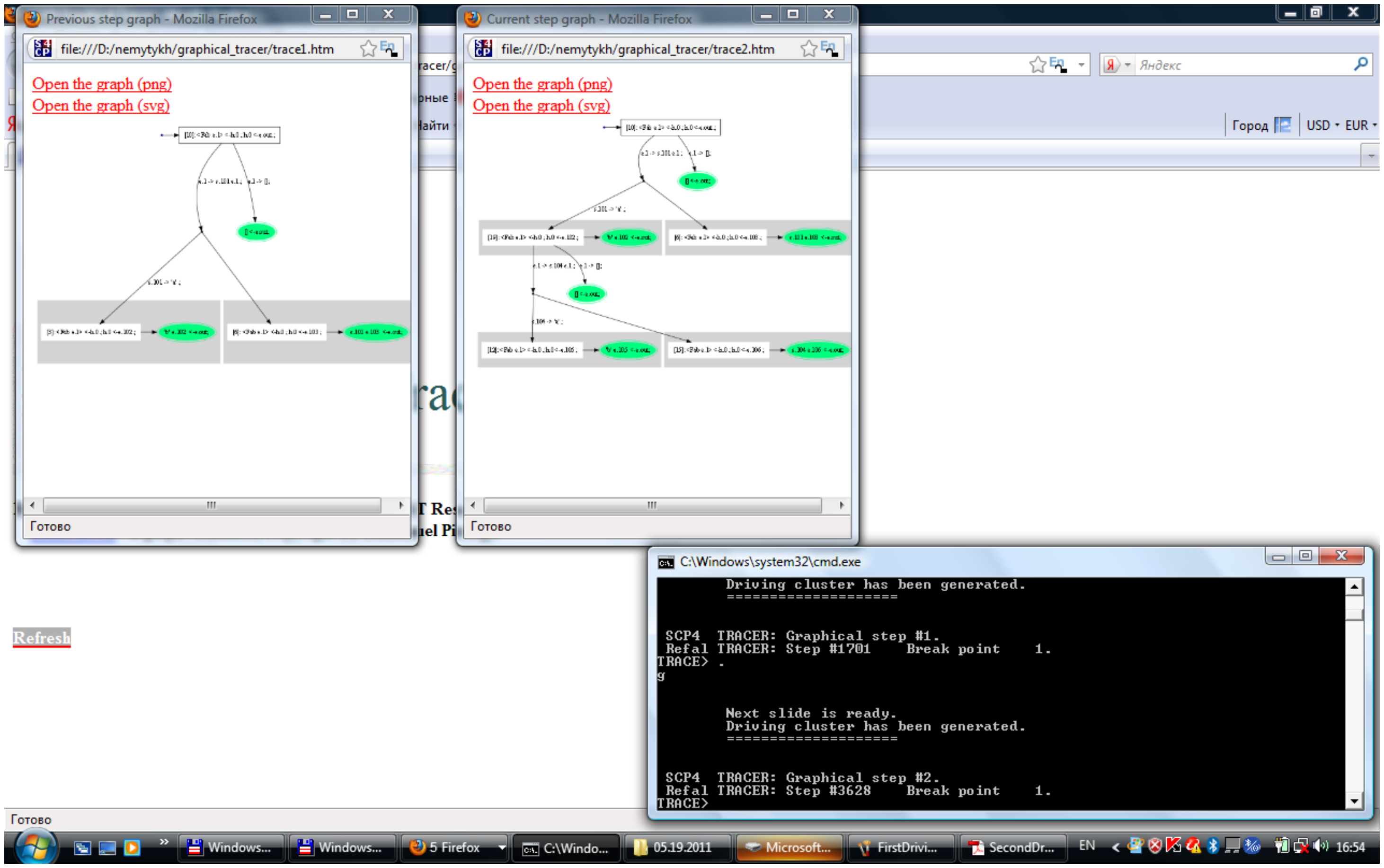
– A translator of the SCP4's states into Dot was implemented in REFAL-5 language. The same language is both the input and implementation language of SCP4.

Drawing and Navigation

– Given a Dot-description of a graph, the system Graphviz draws the graph on a computer screen. It supports all widely used graphical formats and is supported under many modern operating systems. The Graphviz interpreter is directly launched from the supercompiler.

– The system ZGRViewer allows users to navigate through such graphs and to scale them in various ways.

States of the current and previous tracer steps are displayed on the computer screen by means of a browser.


Previous step graph - Mozilla Firefox
file:///D:/nemytykh/graphical_tracer/trace1.htm
Open the graph (png)
Open the graph (svg)
Готово
Current step graph - Mozilla Firefox
file:///D:/nemytykh/graphical_tracer/trace2.htm
Open the graph (png)
Open the graph (svg)
Готово
C:\Windows\system32\cmd.exe
Refresh
Готово

```
         Driving cluster has been generated.
         ===================

 SCP4  TRACER: Graphical step #1.
 Refal TRACER: Step #1701     Break point    1.
TRACE> .
g

         Next slide is ready.
         Driving cluster has been generated.
         ===================

 SCP4  TRACER: Graphical step #2.
 Refal TRACER: Step #3628     Break point    1.
TRACE>
```

A console is used to control the tracer and allows to get additional text information not displayed on the graphical windows.

A Dot specification of the SCP4 state at the beginning of the current tracer step is generated on the base of such a Dot specification at the beginning of the previous step.

Regeneration of the whole Dot specification is not made.

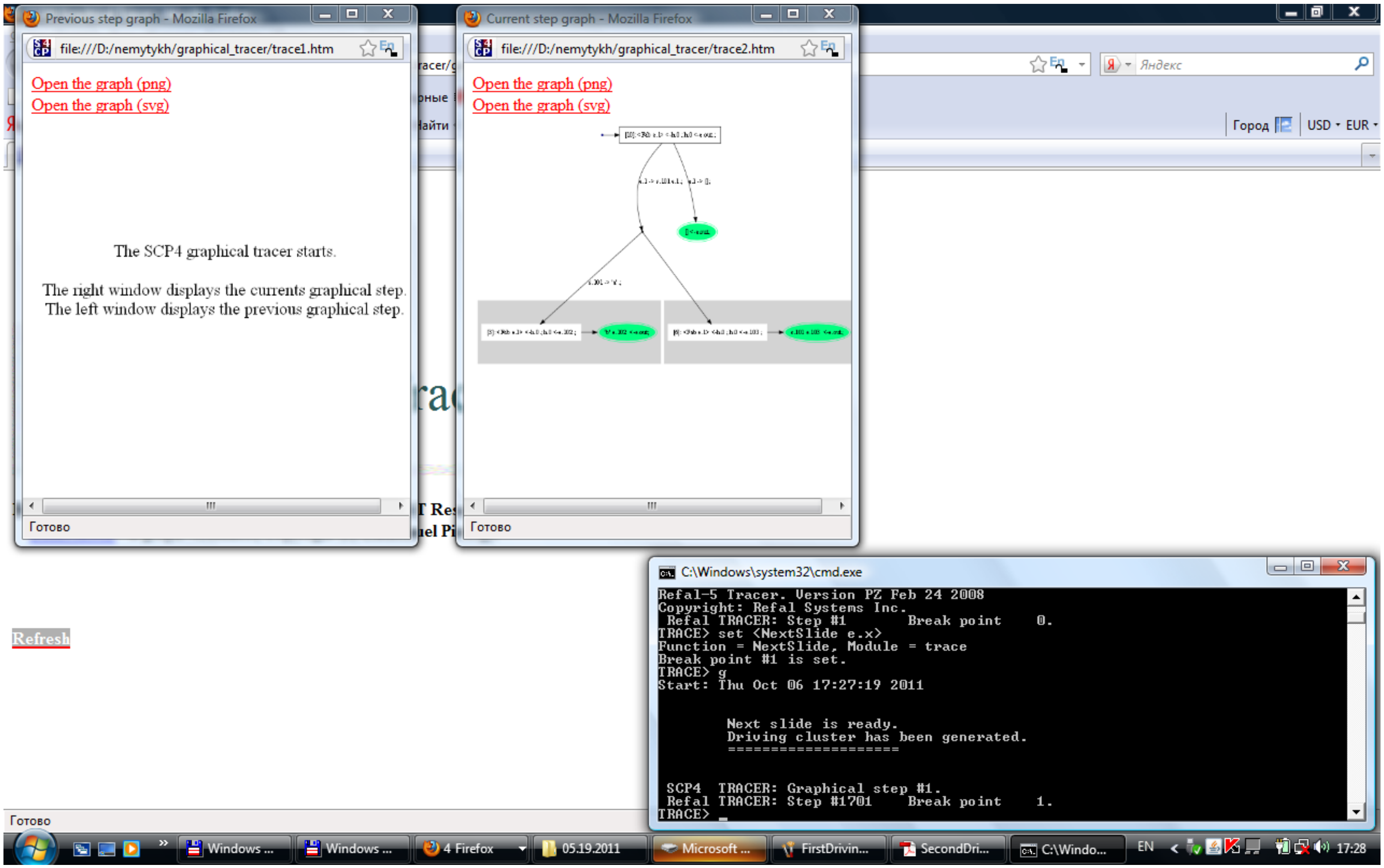
Example: a Result of the Driving Algorithm

As a rule, any interpreter Int of a given language L has the following structure:

```
 Int(p,d) {
  initialization;
  while (computations do not finished)
              { STEP; }
  return (resulting state);
}
```

The driving is a meta-extension of the L-machine step. Given a parameterized function stack, the driving produces the tree of all possible computation of this input stack.

A screenshot of a first call result of the driving algorithm.

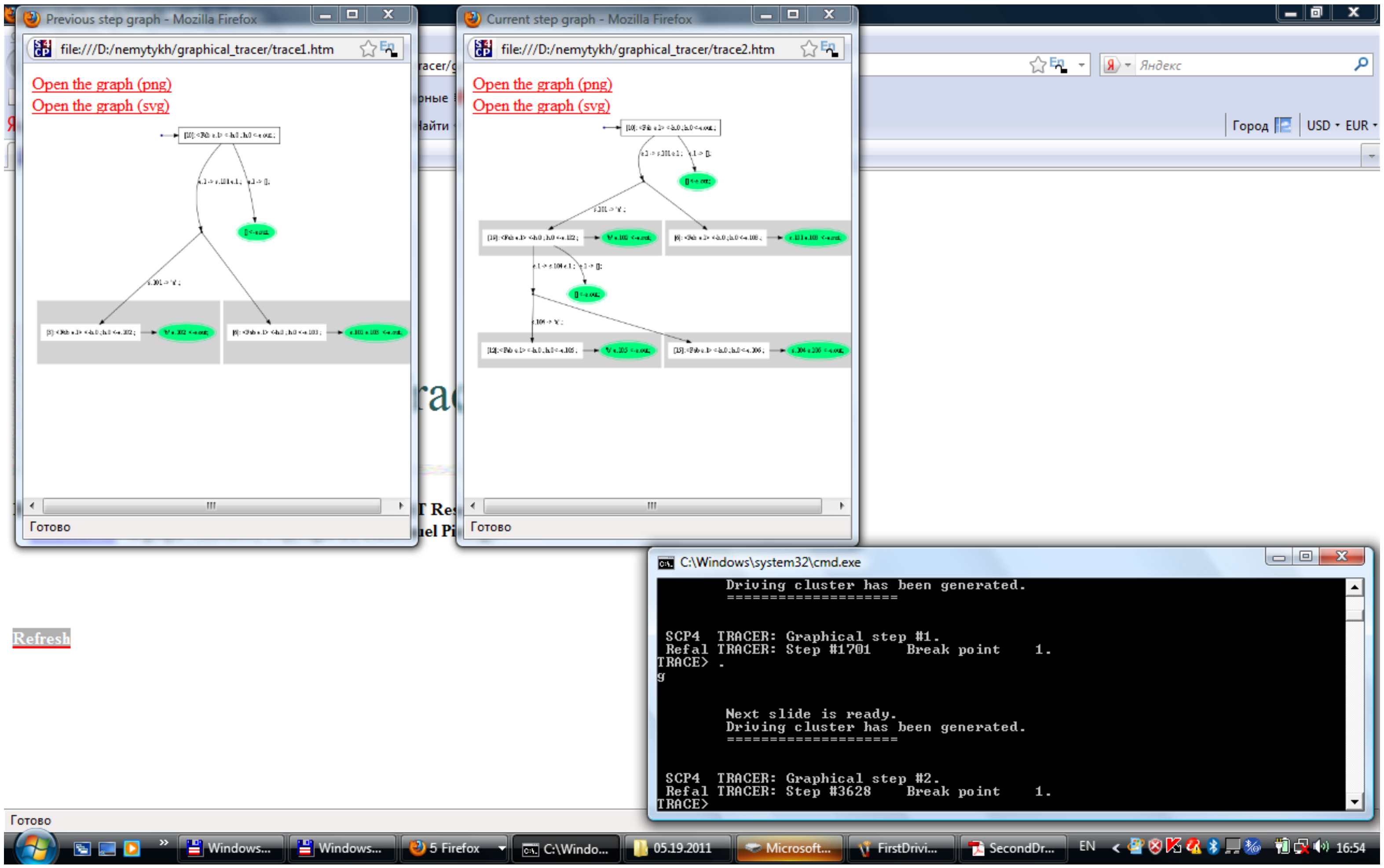


The SCP4 graphical tracer starts.

The right window displays the currents graphical step.
The left window displays the previous graphical step.

```
Refal-5 Tracer. Version PZ Feb 24 2008
Copyright: Refal Systems Inc.
 Refal TRACER: Step #1        Break point    0.
TRACE> set <NextSlide e.x>
Function = NextSlide, Module = trace
Break point #1 is set.
TRACE> g
Start: Thu Oct 06 17:27:19 2011


        Next slide is ready.
        Driving cluster has been generated.
        ====================


 SCP4  TRACER: Graphical step #1.
 Refal TRACER: Step #1701     Break point    1.
TRACE> _
```

The results of the first and second steps of the tracer.


Previous step graph - Mozilla Firefox
file:///D:/nemytykh/graphical_tracer/trace1.htm
Open the graph (png)
Open the graph (svg)

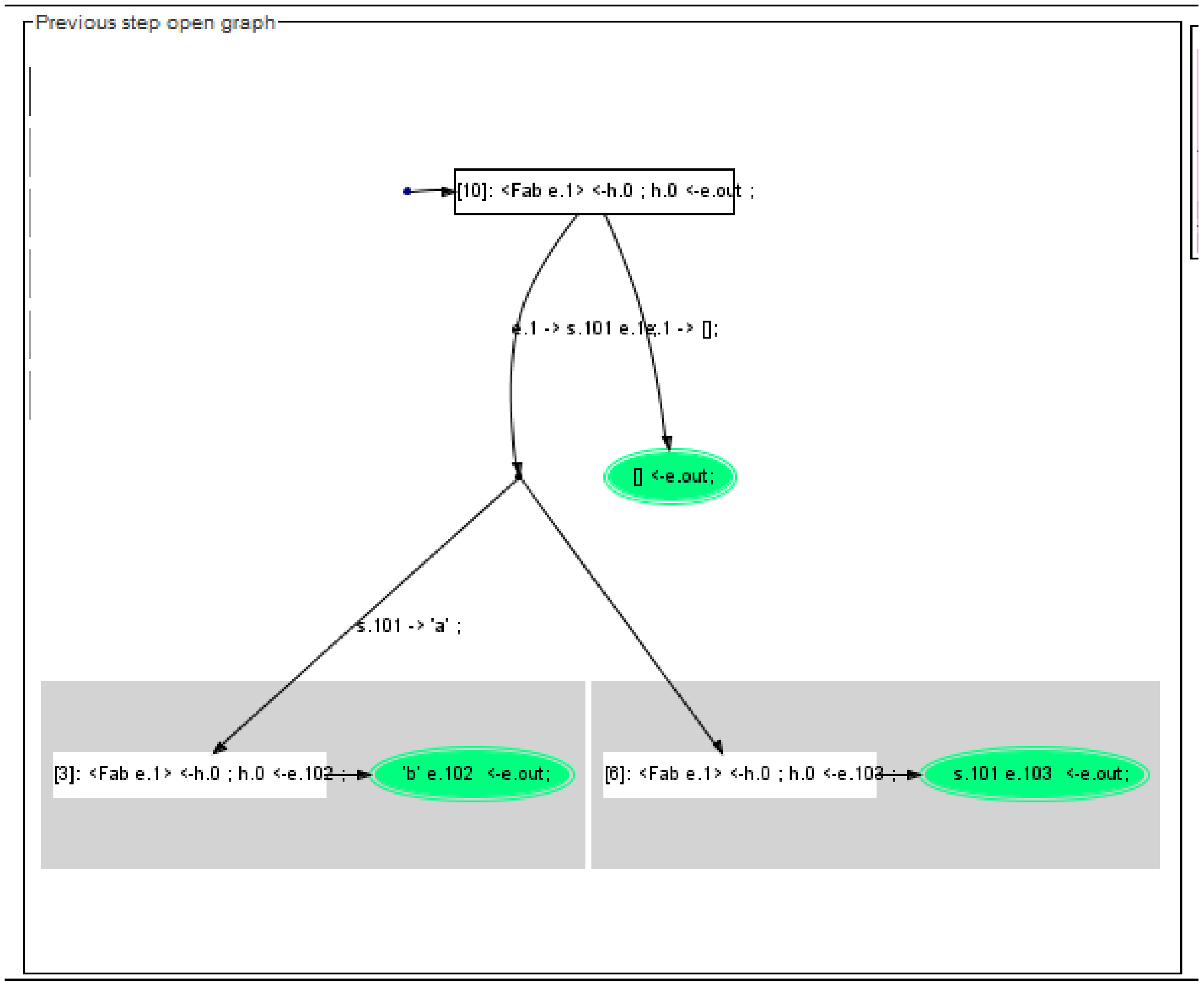
Current step graph - Mozilla Firefox
file:///D:/nemytykh/graphical_tracer/trace2.htm
Open the graph (png)
Open the graph (svg)

C:\Windows\system32\cmd.exe

```
        Driving cluster has been generated.
        ===================

 SCP4  TRACER: Graphical step #1.
 Refal TRACER: Step #1701     Break point    1.
TRACE> .
g


        Next slide is ready.
        Driving cluster has been generated.
        ===================

 SCP4  TRACER: Graphical step #2.
 Refal TRACER: Step #3628     Break point    1.
TRACE>
```

The red references given in the left upper angles of the windows provide possibility to enlarge the represented graph.

The result of the first driving call given in a large-scale representation.



The specialization task

```
<Fab #e.x>
```

The program

```
 Fab {
  'a' e.S = 'b' <Fab e.S>;
  s.1 e.S = s.1 <Fab e.S>;
          = ;
}
```

tracing

Verification of the MESI cache coherence protocol

<Mesi (#e.actions) (Invalid I #e.i) (Modified) (Shared) (Exclusive)>

The program

The MESI cache coherence protocol program model.

tracing

⇑

Thank you!