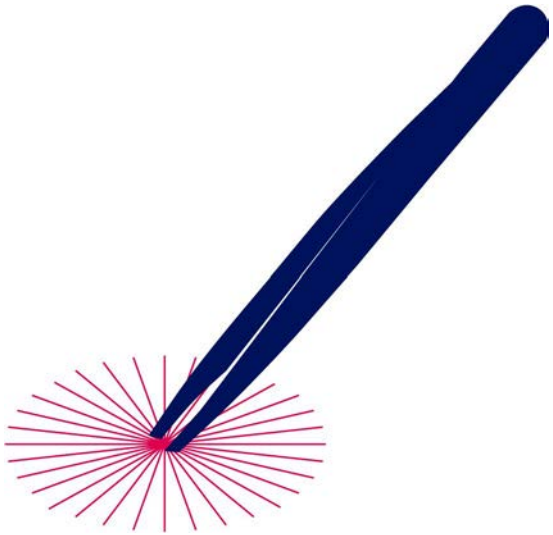
Verification of Software Upgrades



Natasha Sharygina

*FORMAL VERIFICATION LAB
University of Lugano

Motivation

- Software evolves
  - Small frequent upgrades
  - Complete re-verification impractical / infeasible
- Incremental verification
  - Store information from previous verification runs
  - Speed-up consecutive runs
- Local upgrade checks
  - Incremental Bounded model checking
  - Interpolation-based function summarization

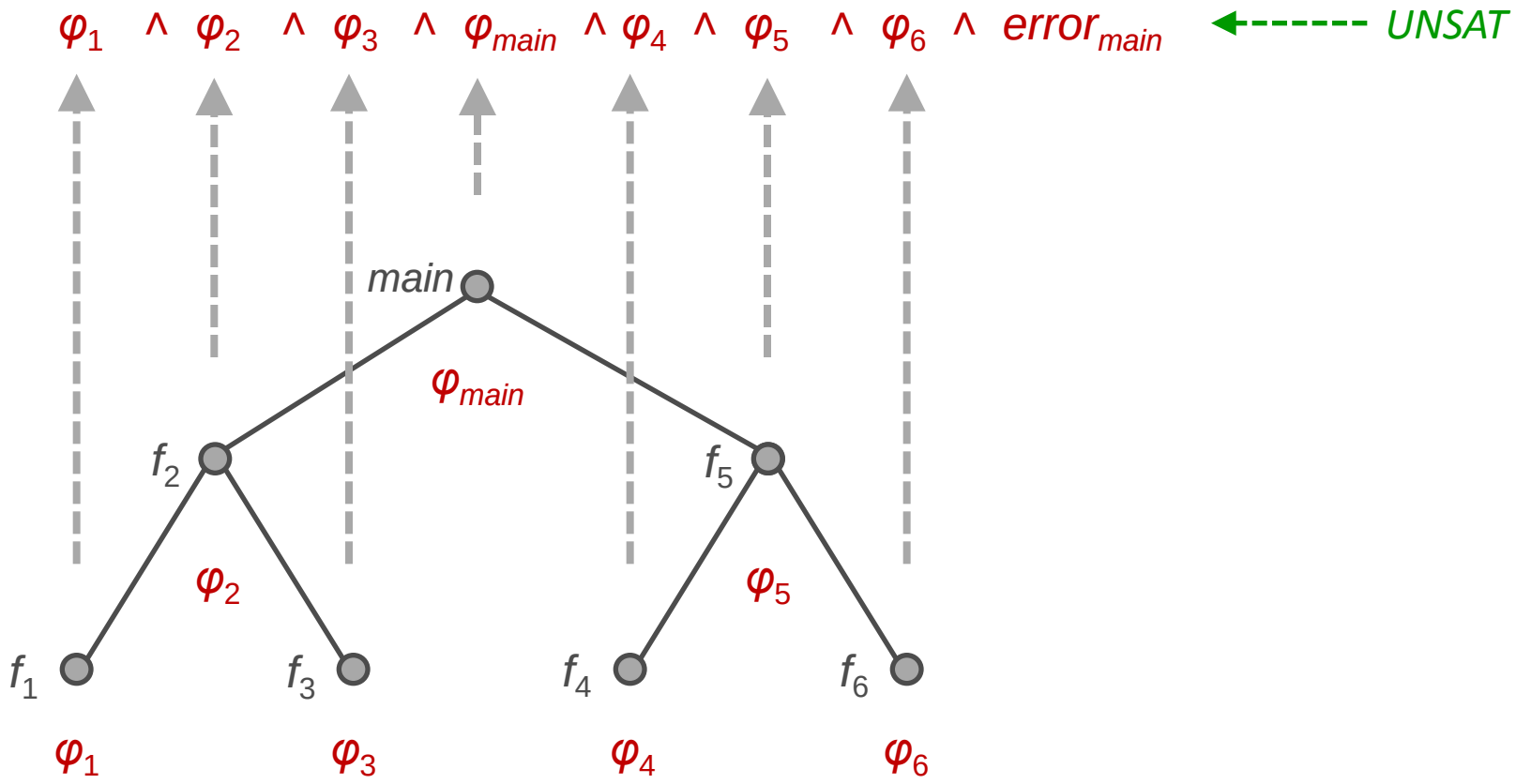
Context

bounded model checking

- Loops/recursion unwound
  - Up to a given bound
- Encoding into a BMC formula
- Satisfiability check by a solver
  - UNSAT → System is safe
  - SAT → Error found
    - Satisfying assignment identifies an error trace

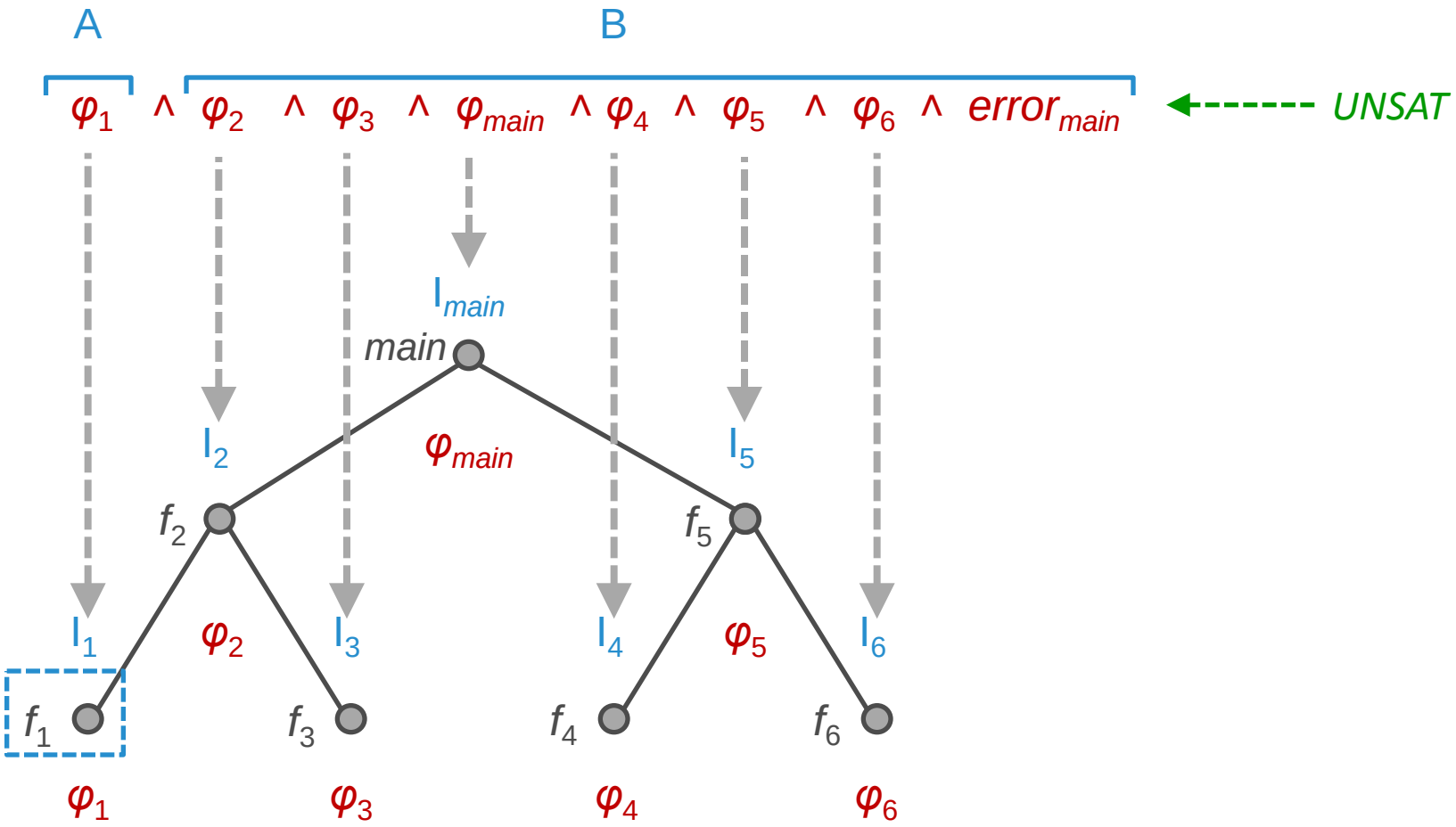
Partitioning BMC

formula construction



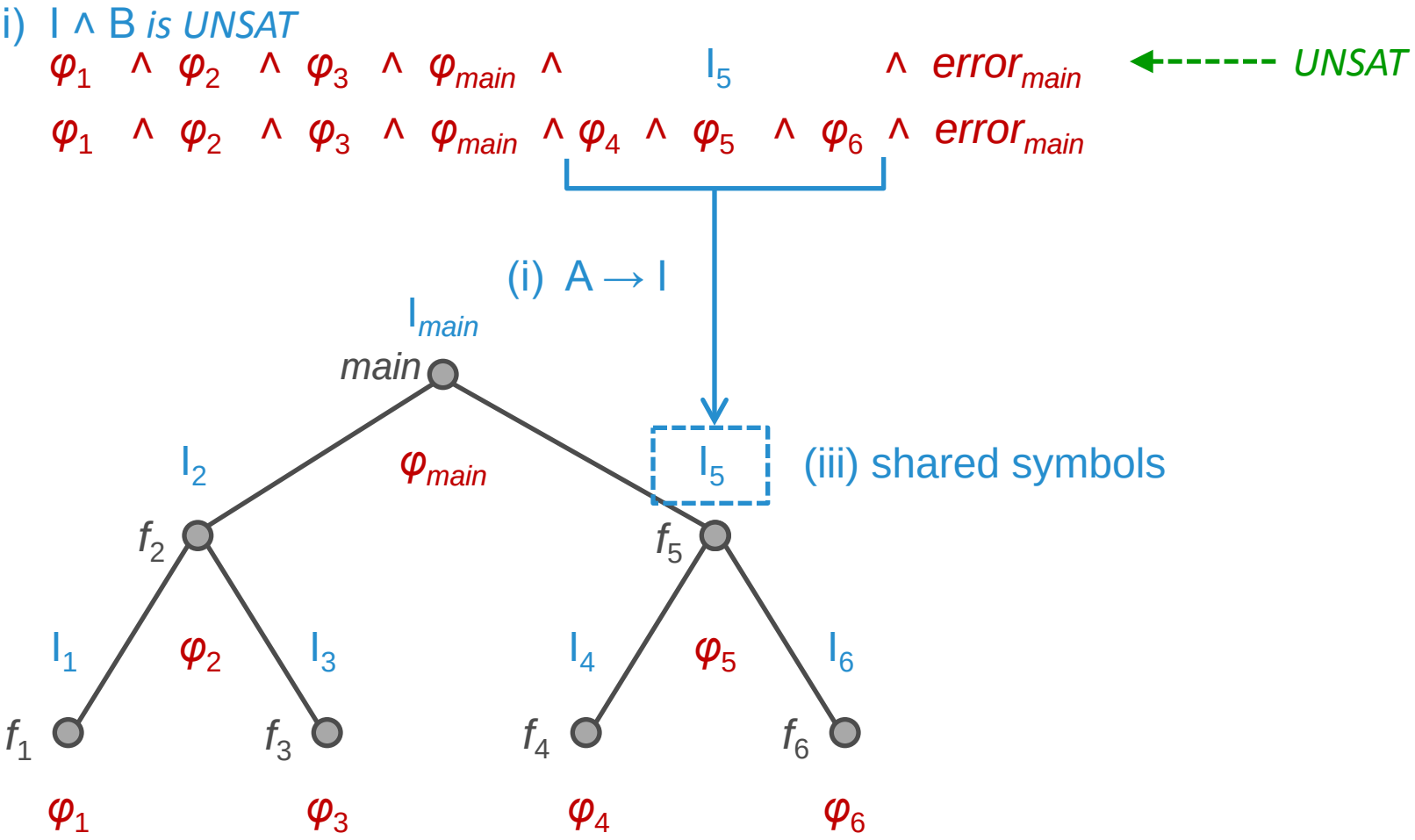
Partitioning BMC

generation of summaries



Partitioning BMC

properties of interpolant-based summaries



Interpolation-based function summaries

- Function summary
  - An over-approximation of the real behavior
    - Considering the given bound
  - Contains only the relevant information
    - Generated from the proof of UNSAT
  - Expressed using function's in/out parameters

Interpolation-based function summaries

- Function summary
  - An over-approximation of the real behavior
    - Considering the given bound
  - Contains only the relevant information
    - Generated from the proof of UNSAT
  - Expressed using function's in/out parameters
- Usage
  1) Same code, different properties
     - To approximate the corresponding functions
  2) Same properties, different code
     - Upgrade checking

Sery O., Fedyukovich G., Sharygina N., *Interpolation-based Function Summaries in Bounded Model Checking*, HVC 2011; *FunFrog tool,* TACAS 2012

Interpolation-based function summaries

- Function summary
  - An over-approximation of the real behavior
    - Considering the given bound
  - Contains only the relevant information
    - Generated from the proof of UNSAT
  - Expressed using function's in/out parameters
- Usage
  1) Same code, different properties
     - To approximate the corresponding functions
  2) Same properties, different code
     - Upgrade checking

Sery O., Fedyukovich G., Sharygina N., *Upgrade Checking*, FMCAD 2012

Upgrades – key idea

Observations:

- *An old summary can remain a valid over-approximation of the new version of a modified function*
- *Old summaries are precise enough to prove the properties of interest*

Idea: Do a **cheap local** check…

The eVolCheck algorithm

overview

0) Verification of the base version of the software (*bootstrap*)
   - function summaries generated and stored

The eVolCheck algorithm

overview

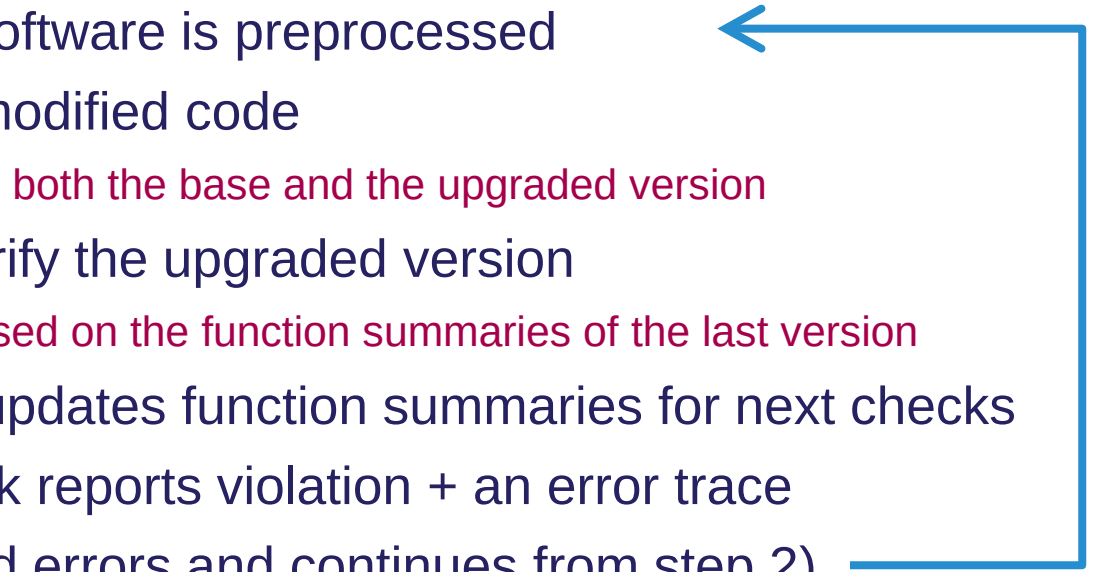
0) Verification of the base version of the software (*bootstrap*)
   - function summaries generated and stored

1) The user upgrades the software
2) Upgraded version of the software is preprocessed
3) eVolCheck identifies the modified code
   - by comparing parse trees for both the base and the upgraded version
4) eVolCheck attempts to verify the upgraded version
   - using cheap local checks based on the function summaries of the last version

5a) If successful, eVolCheck updates function summaries for next checks

5b) If unsuccessful, eVolCheck reports violation + an error trace

6) The user fixes the reported errors and continues from step 2)



The algorithm

overview

0) Verification of the base version of the software (*bootstrap*)
   - function summaries generated and stored

1) The user upgrades the software
2) Upgraded version of the software is parsed by goto-cc
3) eVolCheck identifies the modified code
   - by comparing parse trees for both the base and the upgraded version
4) eVolCheck attempts to verify the upgraded version
   - using cheap local checks based on the function summaries of the last version

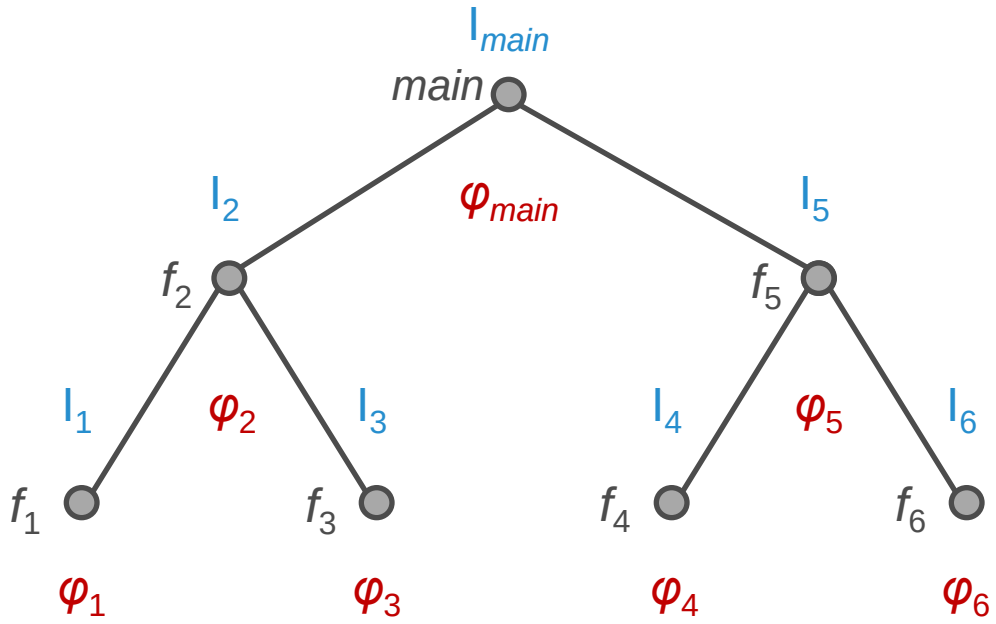
5a) If successful, eVolCheck updates function summaries for next checks

5b) If unsuccessful, eVolCheck reports violation + an error trace

6) The user fixes the reported errors and continues from step 2)

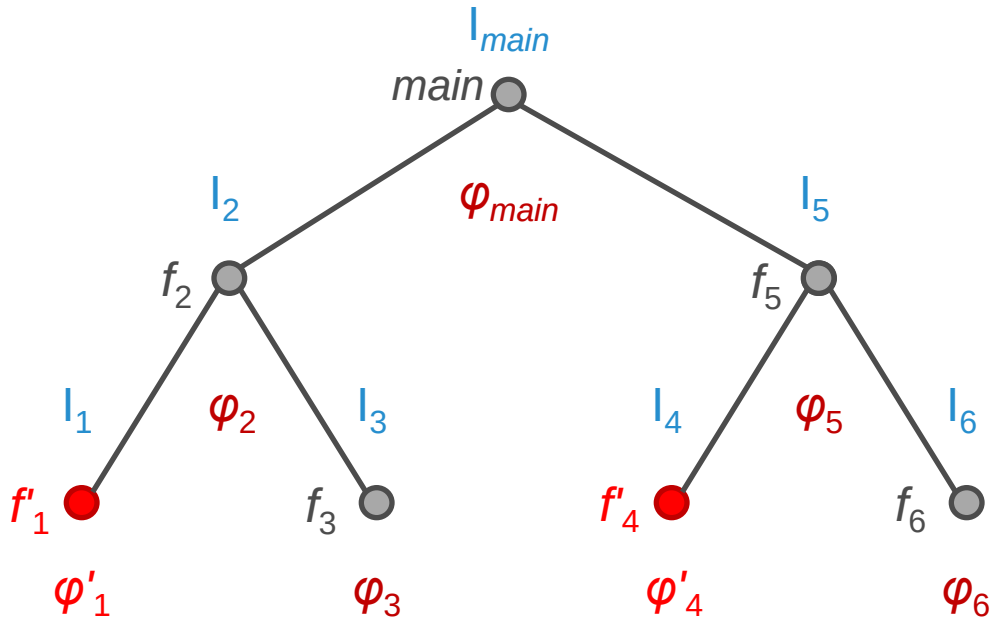
Incremental upgrade check

Functions f_1 and f_4 upgraded…



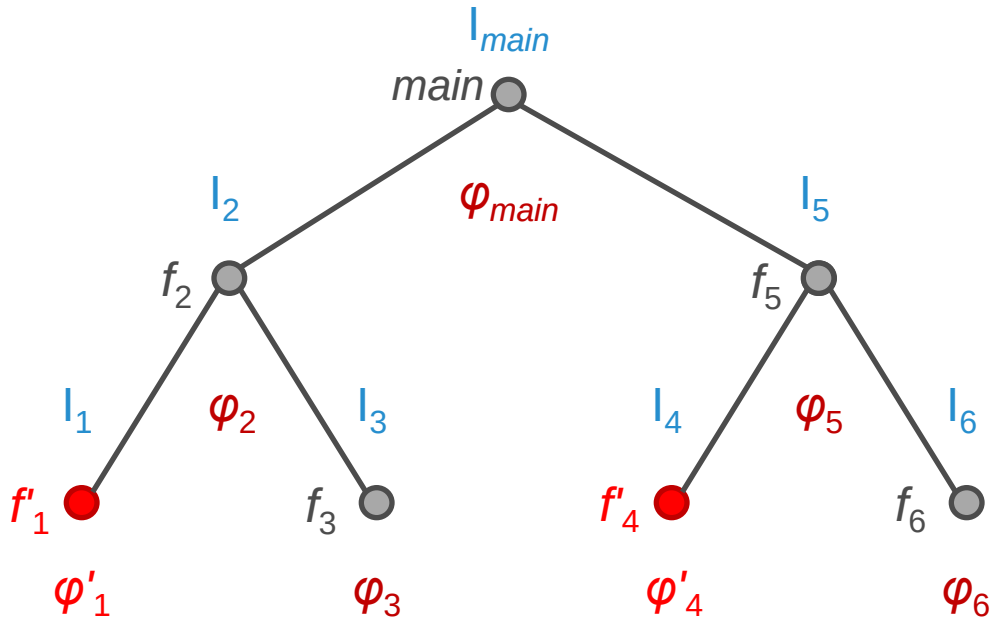
Incremental upgrade check

Functions f_1 and f_4 upgraded…



Incremental upgrade check

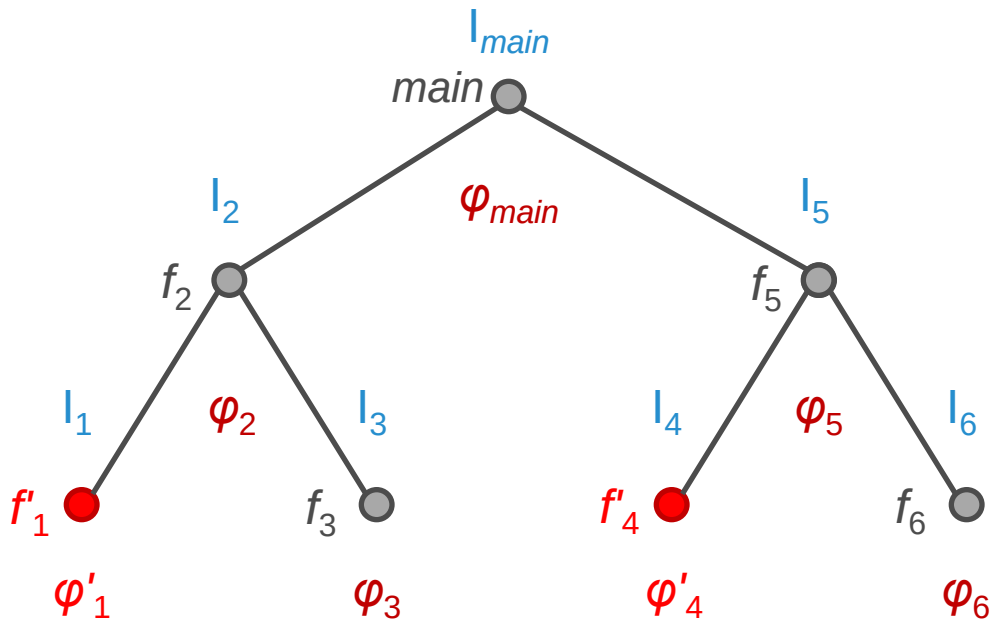
We attempt to verify that summaries I_1 and I_4 are still valid over-approximations.



Incremental upgrade check

Check for I_1:

$\varphi'_1 \rightarrow I_1$ ✓ *upgrade is safe*



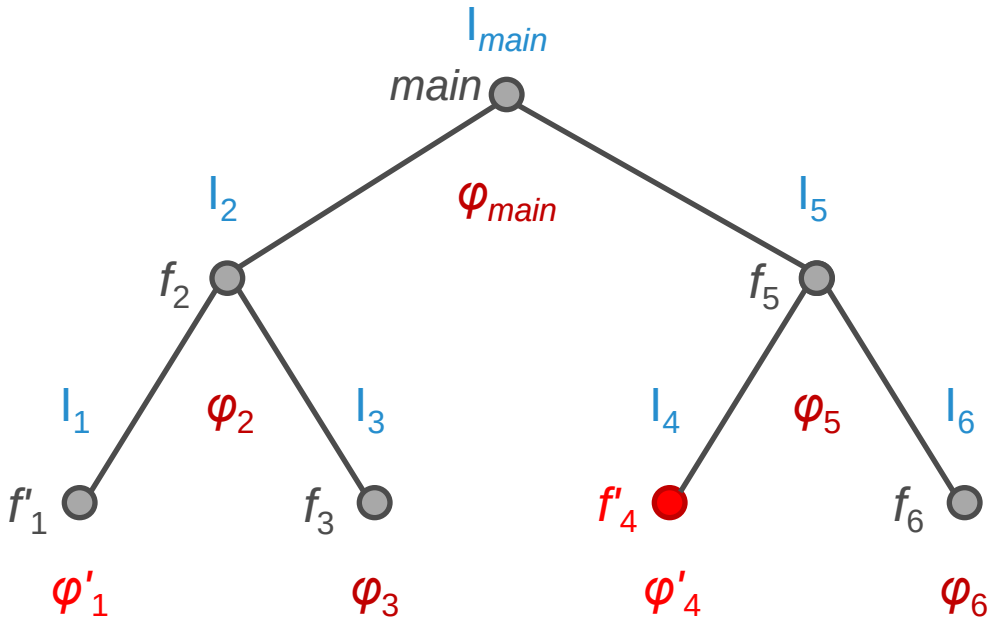
Incremental upgrade check

Check for I_1:

$\varphi'_1 \rightarrow I_1$ ✓ *upgrade is safe*

Check for I_4:

$\varphi'_4 \rightarrow I_4$ ✗ *propagate upwards*



Incremental upgrade check

Check for I_1:

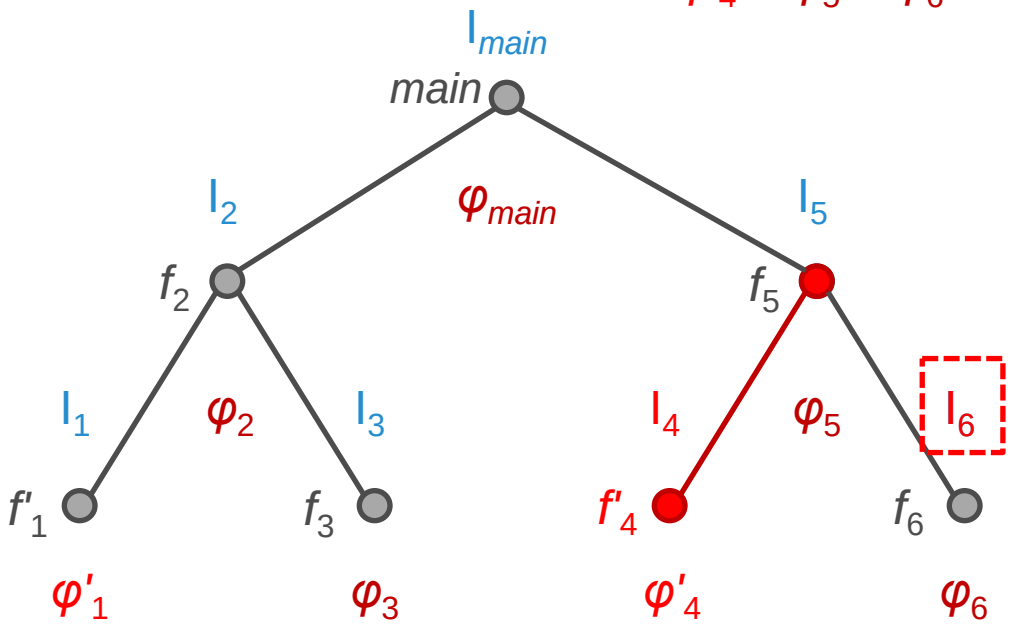
$\varphi'_1 \rightarrow I_1$ ✓ *upgrade is safe*

Check for I_4:

$\varphi'_4 \rightarrow I_4$ ✗ *propagate upwards*

$\varphi'_4 \wedge \varphi_5 \wedge I_6 \rightarrow I_5$ ✗ *refine downwards*

$\varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \rightarrow I_5$ ✓ *upgrade is safe*



Incremental upgrade check

Check for I_1:

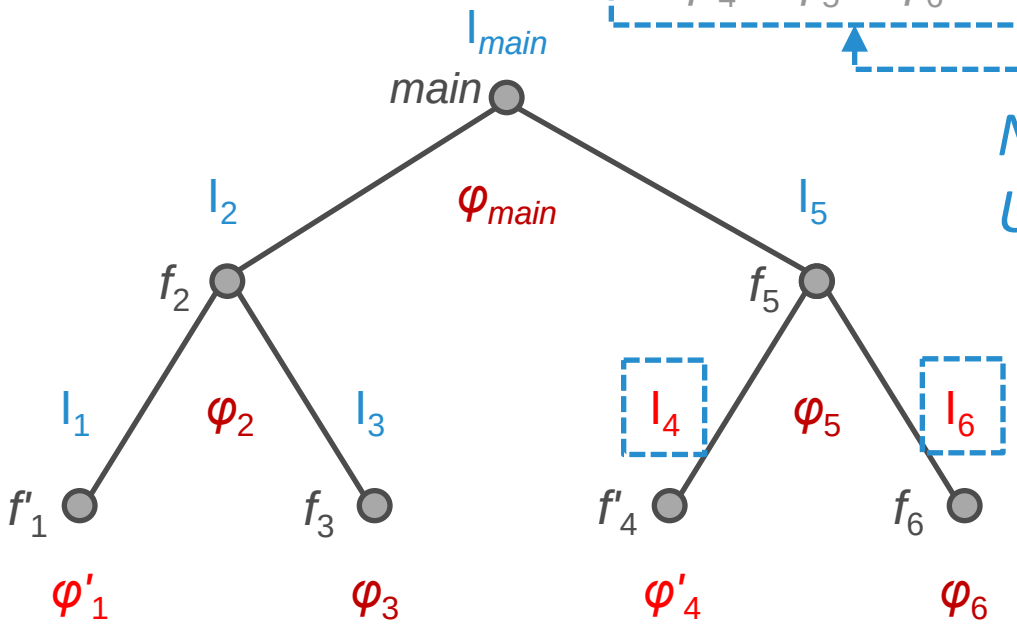
$\varphi'_1 \rightarrow I_1$ ✓ *upgrade is safe*

Check for I_4:

$\varphi'_4 \rightarrow I_4$ ✗ *propagate upwards*

$\varphi'_4 \wedge \varphi_5 \wedge I_6 \rightarrow I_5$ ✗ *refine downwards*

$\varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \rightarrow I_5$ ✓ *upgrade is safe*



Note that • is checked as UNSAT of:

$\varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \neg I_5$

… we can regenerate summaries

Incremental upgrade check

Check for I_1:

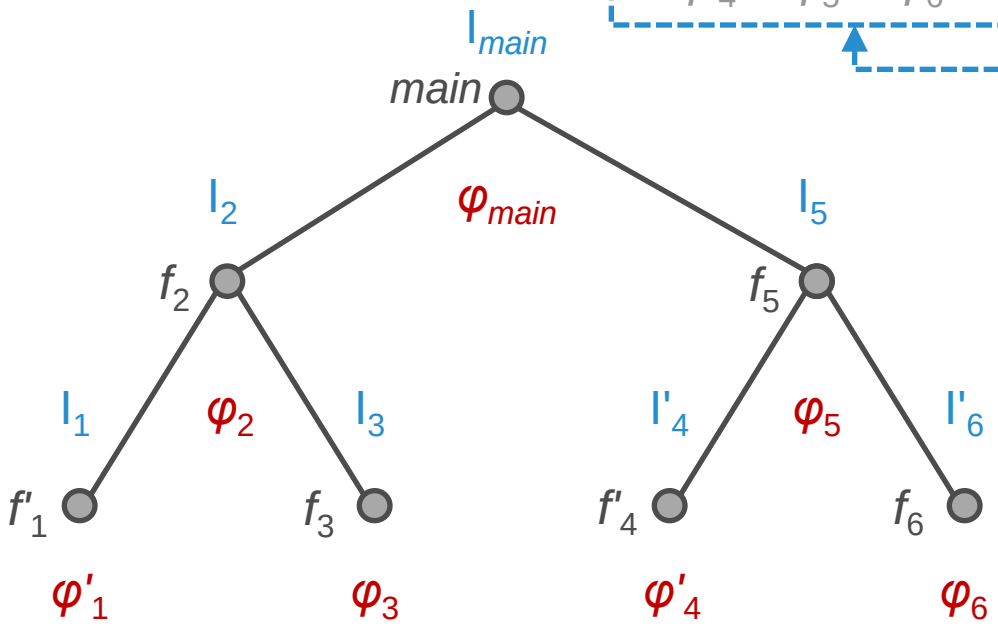
$\varphi'_1 \rightarrow I_1$ ✓ *upgrade is safe*

Check for I_4:

$\varphi'_4 \rightarrow I_4$ ✗ *propagate upwards*

$\varphi'_4 \wedge \varphi_5 \wedge I_6 \rightarrow I_5$ ✗ *refine downwards*

$\varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \rightarrow I_5$ ✓ *upgrade is safe*



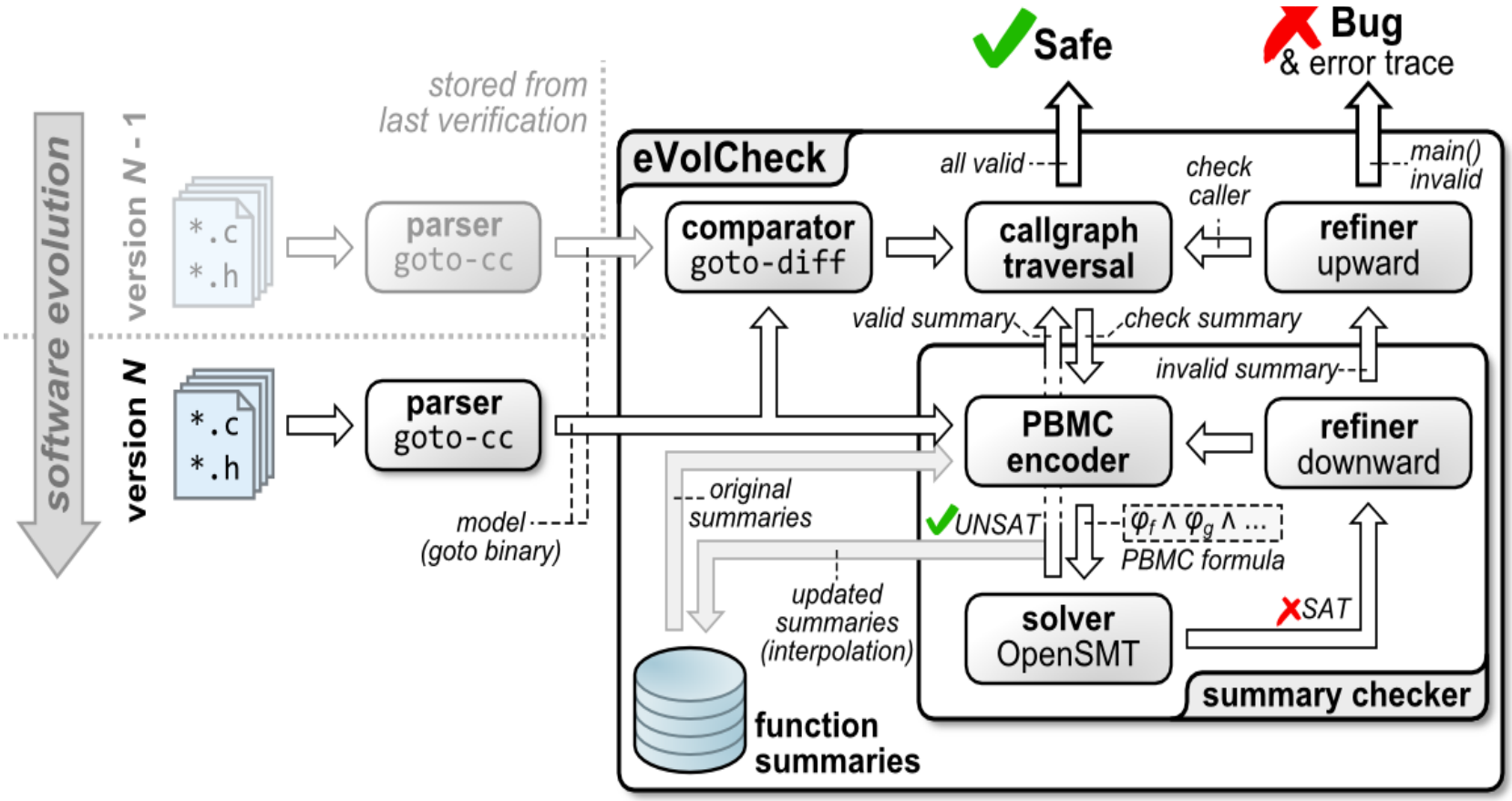
Note that • is checked as UNSAT of:

$\varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \neg I_5$

… we can regenerate summaries

eVolCheck

architecture



eVolCheck

experimentation

Benchmark	Bootstrap		Upgrade check					
Name	Total [s]	Itp [s]	Total [s]	Diff [s]	Itp [s]	**Speedup**	Result	ISR
ABB_A	8.644	0.008	0.04	0.009	0.003	**220**x	SAFE	0/7
ABB_B	6.236	0.009	0.006	0.006	—	**935**x	SAFE	0/9
ABB_C	8.532	0.015	0.059	0.008	0.003	**157**x	SAFE	0/8
VTT_A	0.512	0.001	0.006	0.006	—	**85.5**x	SAFE	0/9
VTT_B	0.514	0.001	0.031	0.006	—	**0.7**x	BUG	1/9
euler_A	12.56	0.099	0.179	0.001	0.016	**70.4**x	SAFE	1/6
euler_B	12.547	0.095	2.622	0.001	0.031	**4.74**x	SAFE	3/5
life_A	13.911	1.366	0.181	0.001	<0.001	**77.0**x	SAFE	0/5
life_B	13.891	1.357	6.774	0.001	—	**0.31**x	BUG	5/5
arithm_A	0.147	0.007	0.355	0.001	—	**0.39**x	BUG	3/3
diskperf_A	0.167	0.001	0.024	0.008	<0.001	**5.79**x	SAFE	0/21
diskperf_B	0.137	0.001	0.062	0.009	—	**2.25**x	BUG	3/21
floppy_A	2.146	0.229	0.422	0.202	<0.001	**5.02**x	SAFE	0/226
floppy_B	2.183	0.237	2.277	0.206	—	**0.82**x	BUG	79/226
kbfiltr_A	0.288	0.011	0.081	0.023	0.001	**3.40**x	SAFE	1/63
kbfiltr_B	0.320	0.009	0.088	0.023	0.001	**1.85**x	SAFE	3/63

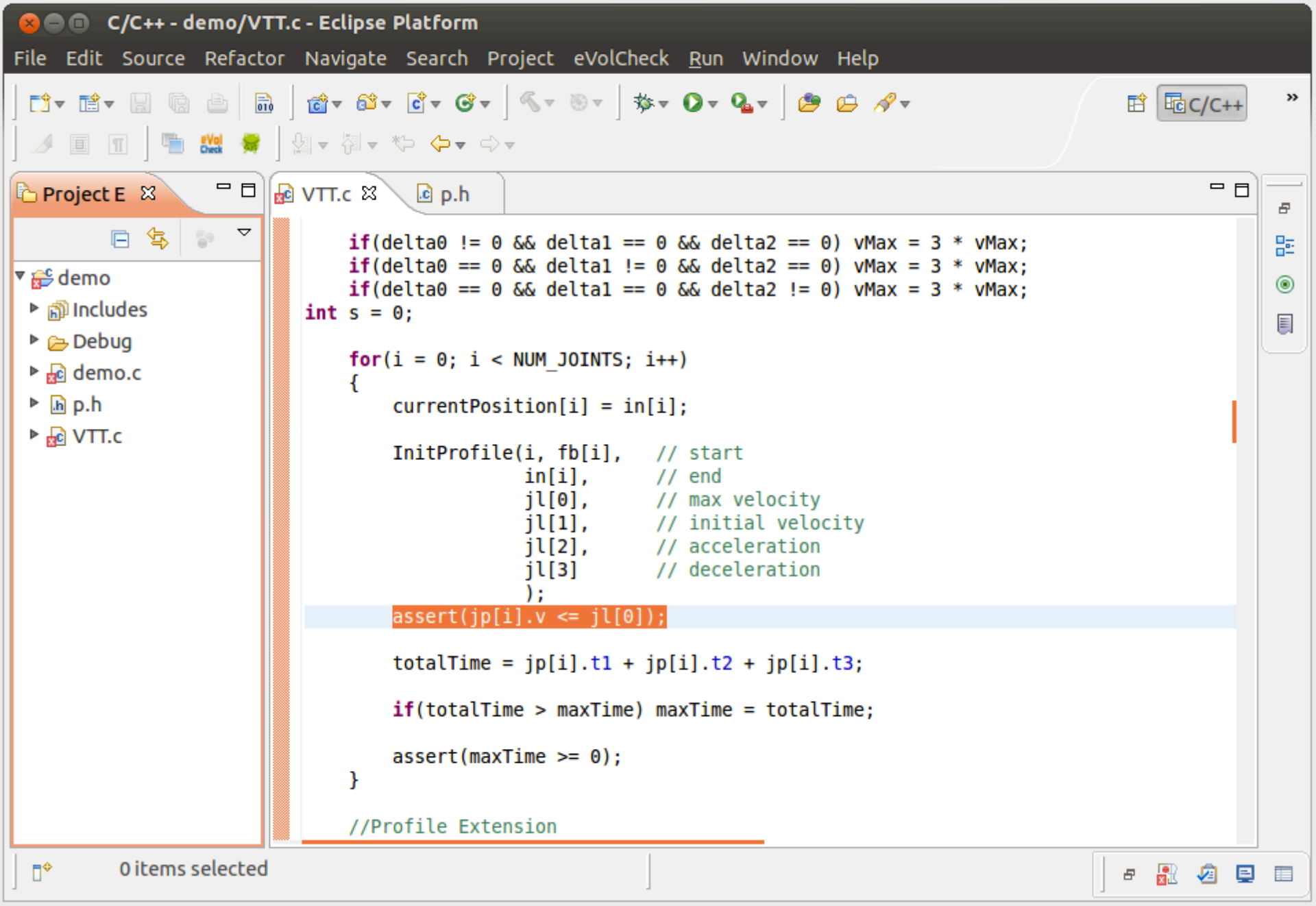
Linux, x64, Intel i-7, 3.4GHz, 16GB

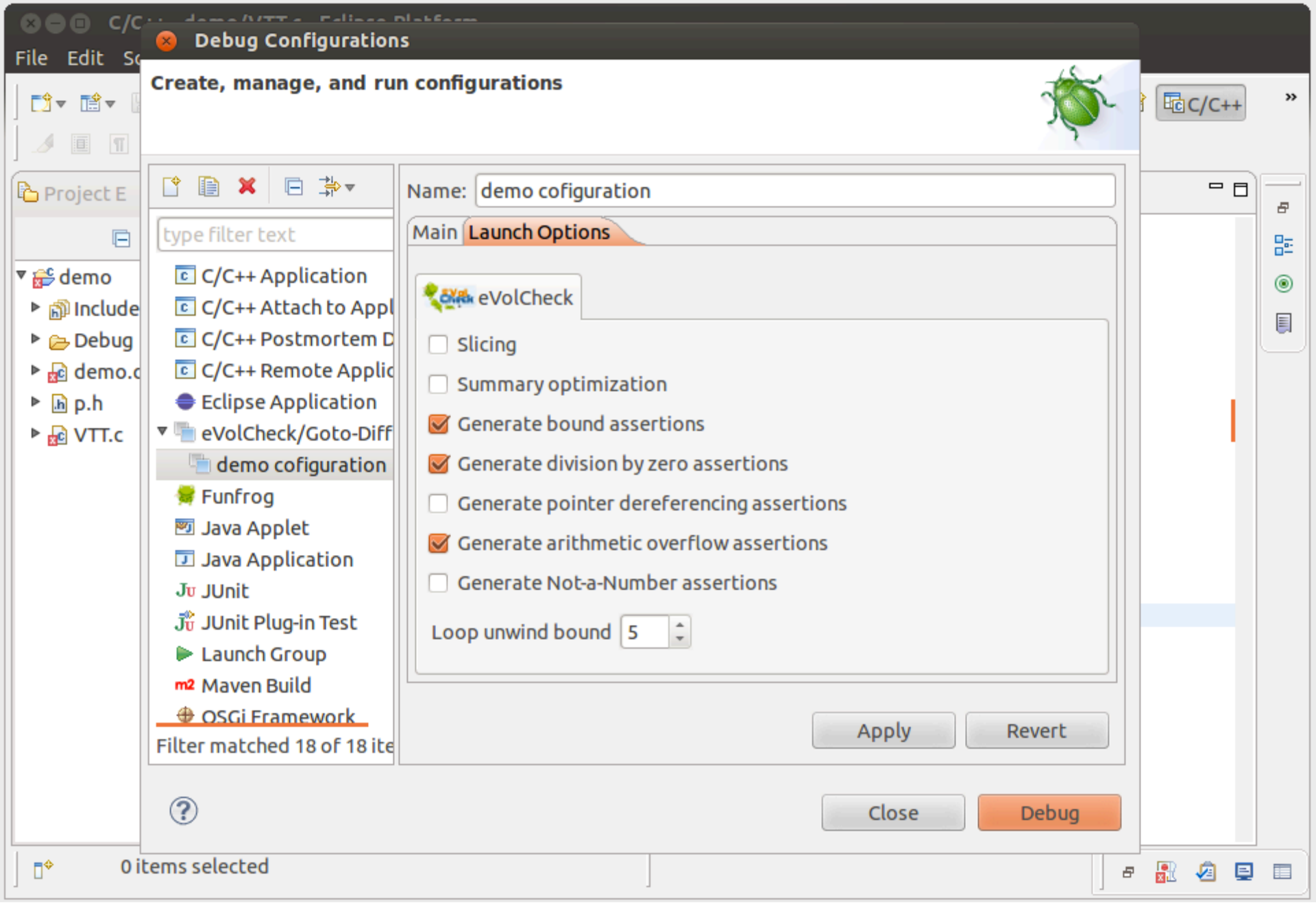
eVolCheck

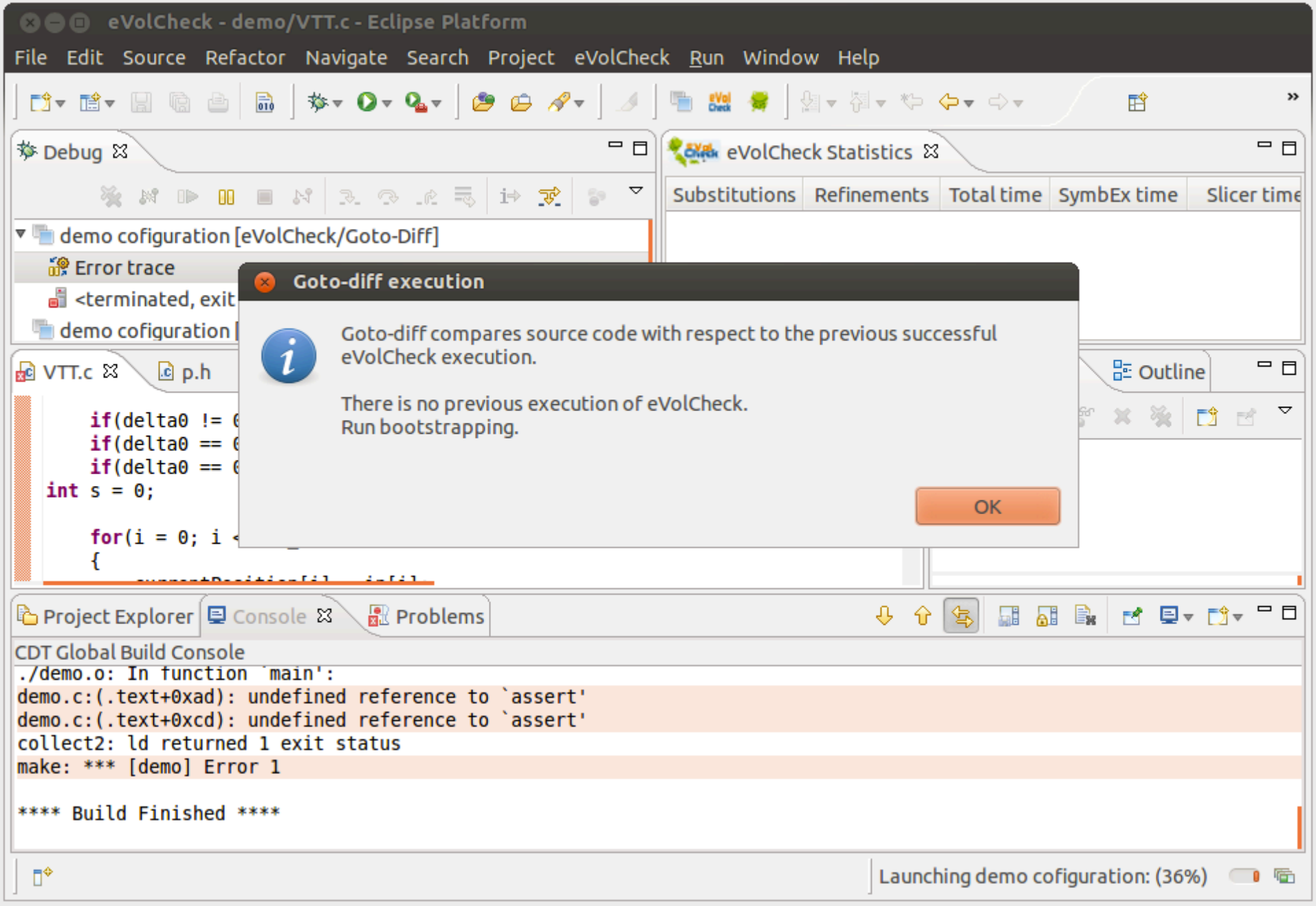
experimentation

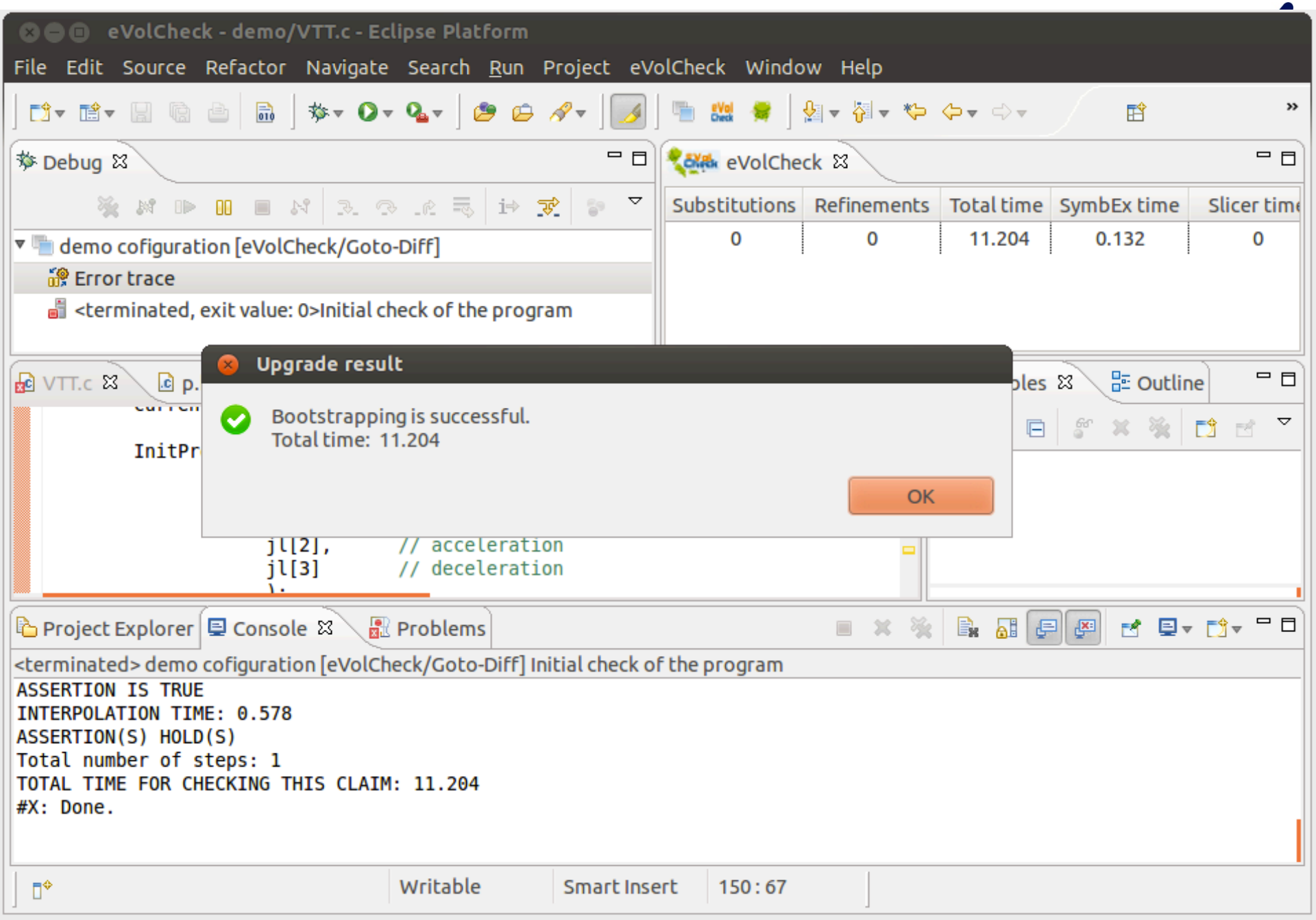
Benchmark	Bootstrap		Upgrade check					
Name	Total [s]	Itp [s]	Total [s]	Diff [s]	Itp [s]	**Speedup**	Result	ISR
ABB_A	8.644	0.008	0.04	0.009	0.003	**220**x	SAFE	0/7
ABB_B	6.236	0.009	0.006	0.006	—	**935**x	SAFE	0/9
ABB_C	8.532	0.015	0.059	0.008	0.003	**157**x	SAFE	0/8
VTT_A	0.512	0.001	0.006	0.006	—	**85.5**x	SAFE	0/9
VTT_B	0.514	0.001	0.031	0.006	—	**0.7**x	BUG	1/9
euler_A	12.56	0.099	0.179	0.001	0.016	**70.4**x	SAFE	1/6
euler_B	12.547	0.095	2.622	0.001	0.031	**4.74**x	SAFE	3/5
life_A	13.911	1.366	0.181	0.001	<0.001	**77.0**x	SAFE	0/5
life_B	13.891	1.357	6.774	0.001	—	**0.31**x	BUG	5/5
arithm_A	0.147	0.007	0.355	0.001	—	**0.39**x	BUG	3/3
diskperf_A	0.167	0.001	0.024	0.008	<0.001	**5.79**x	SAFE	0/21
diskperf_B	0.137	0.001	0.062	0.009	—	**2.25**x	BUG	3/21
floppy_A	2.146	0.229	0.422	0.202	<0.001	**5.02**x	SAFE	0/226
floppy_B	2.183	0.237	2.277	0.206	—	**0.82**x	BUG	79/226
kbfiltr_A	0.288	0.011	0.081	0.023	0.001	**3.40**x	SAFE	1/63
kbfiltr_B	0.320	0.009	0.088	0.023	0.001	**1.85**x	SAFE	3/63

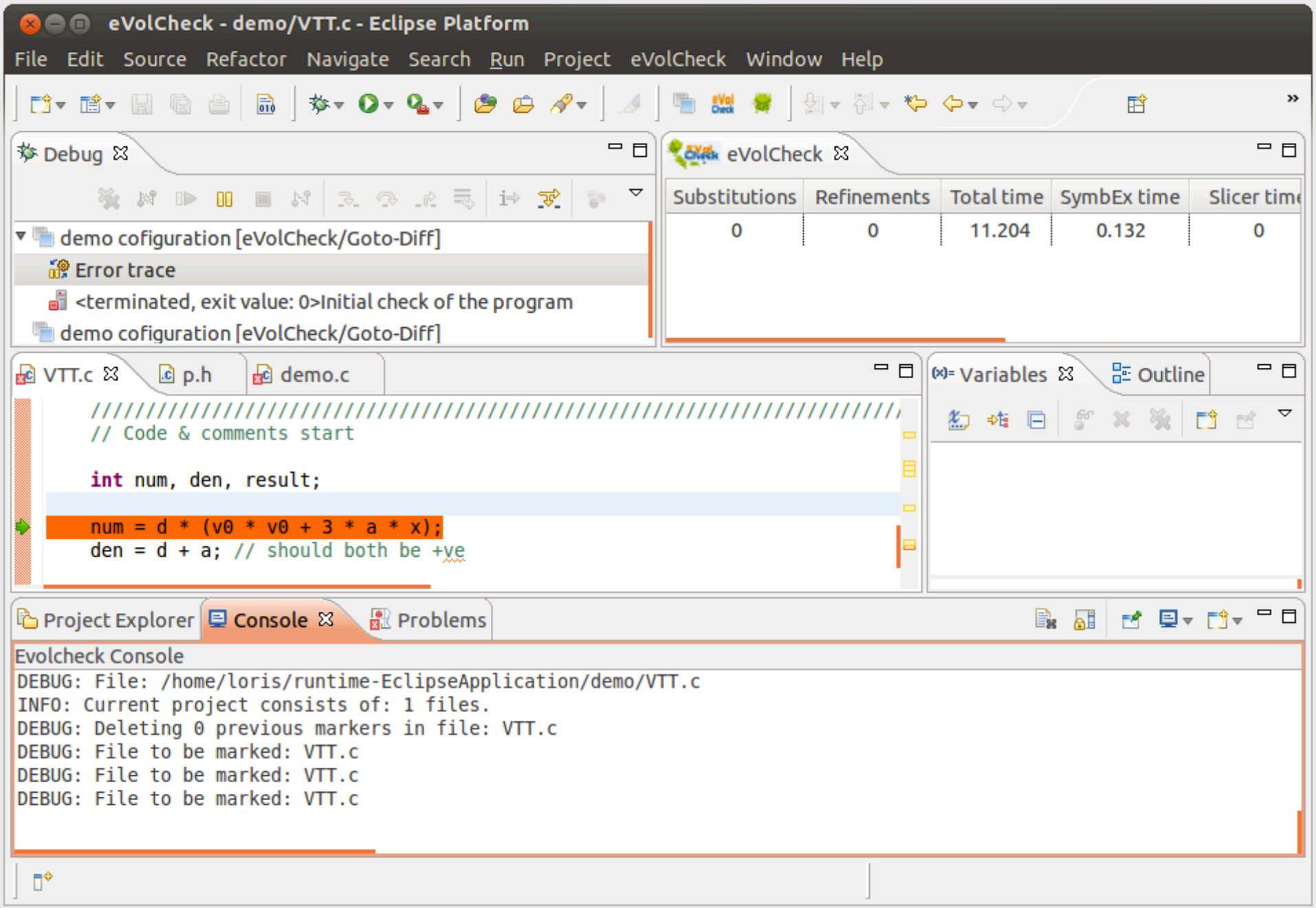
Linux, x64, Intel i-7, 3.4GHz, 16GB

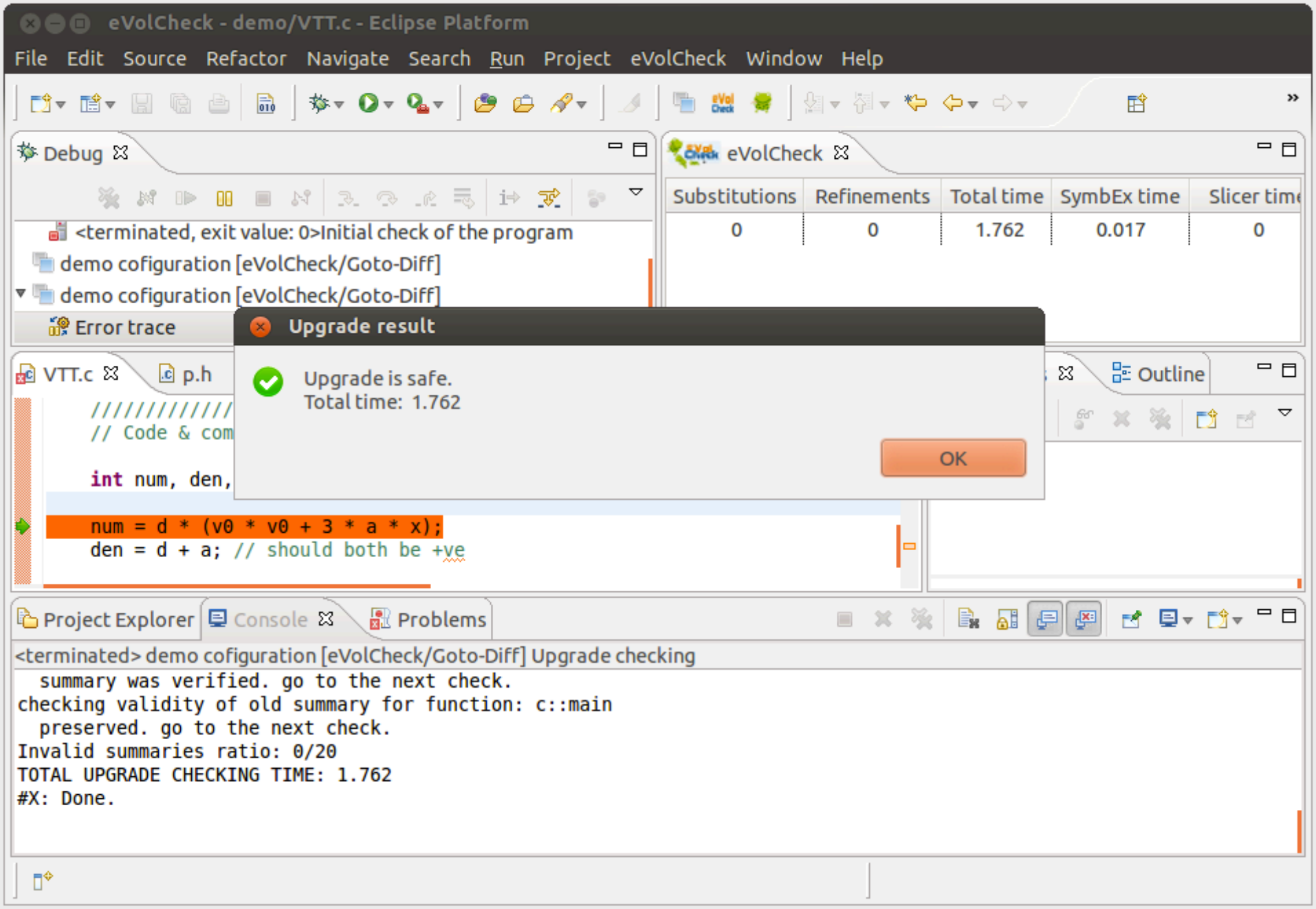

C/C++ - demo/VTT.c - Eclipse Platform
File Edit Source Refactor Navigate Search Project eVolCheck Run Window Help
C/C++
Project E
demo
Includes
Debug
demo.c
p.h
VTT.c
VTT.c
p.h
if(delta0 != 0 && delta1 == 0 && delta2 == 0) vMax = 3 * vMax;
if(delta0 == 0 && delta1 != 0 && delta2 == 0) vMax = 3 * vMax;
if(delta0 == 0 && delta1 == 0 && delta2 != 0) vMax = 3 * vMax;
int s = 0;
for(i = 0; i < NUM_JOINTS; i++)
{
currentPosition[i] = in[i];
InitProfile(i, fb[i], // start
in[i], // end
jl[0], // max velocity
jl[1], // initial velocity
jl[2], // acceleration
jl[3] // deceleration
);
assert(jp[i].v <= jl[0]);
totalTime = jp[i].t1 + jp[i].t2 + jp[i].t3;
if(totalTime > maxTime) maxTime = totalTime;
assert(maxTime >= 0);
}
//Profile Extension
0 items selected

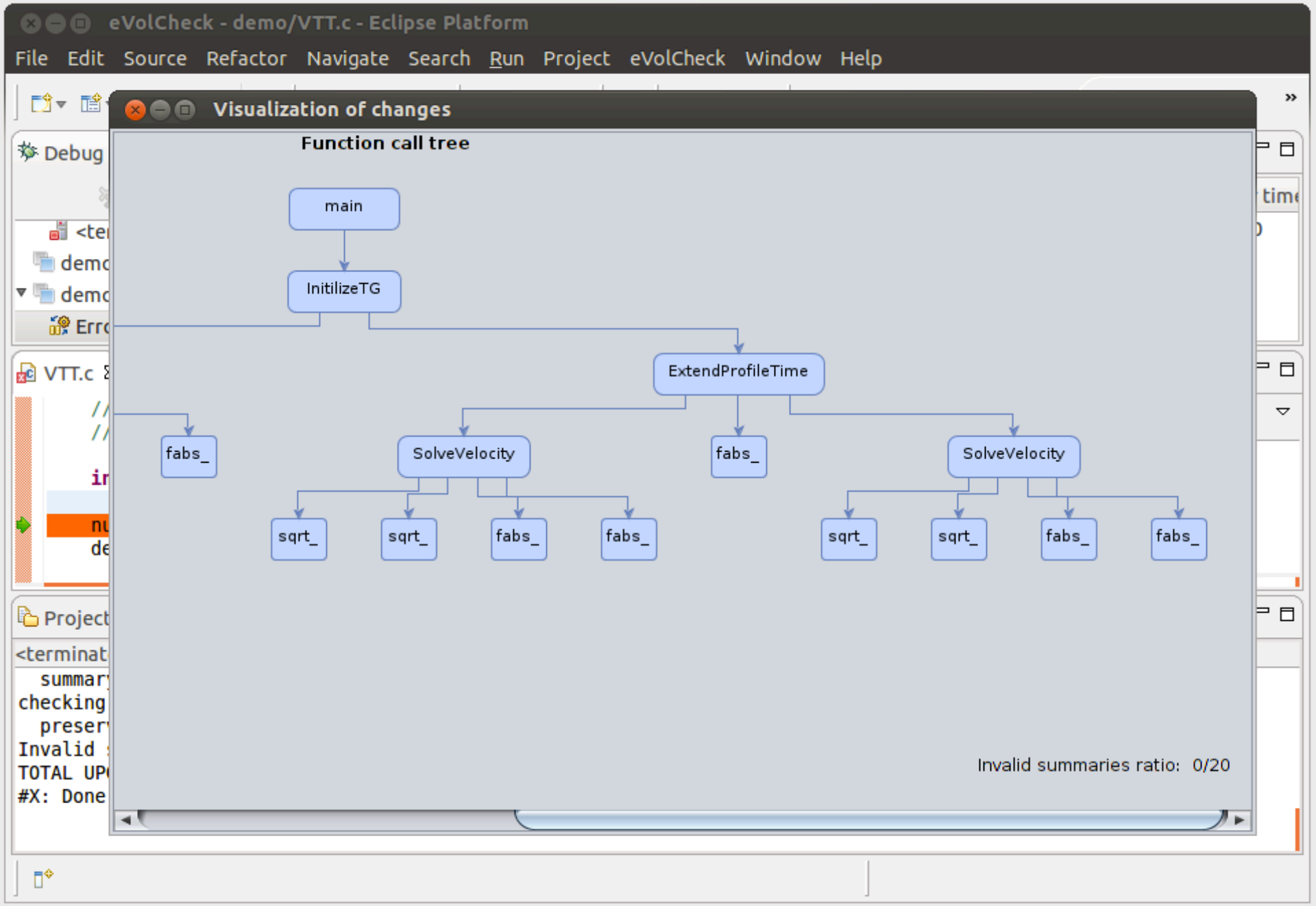

Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Appl
C/C++ Postmortem D
C/C++ Remote Applic
Eclipse Application
eVolCheck/Goto-Diff
demo cofiguration
Funfrog
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Launch Group
Maven Build
OSGi Framework
Filter matched 18 of 18 ite
Name: demo cofiguration
Main
Launch Options
eVolCheck
Slicing
Summary optimization
Generate bound assertions
Generate division by zero assertions
Generate pointer dereferencing assertions
Generate arithmetic overflow assertions
Generate Not-a-Number assertions
Loop unwind bound 5
Apply
Revert
Close
Debug
0 items selected

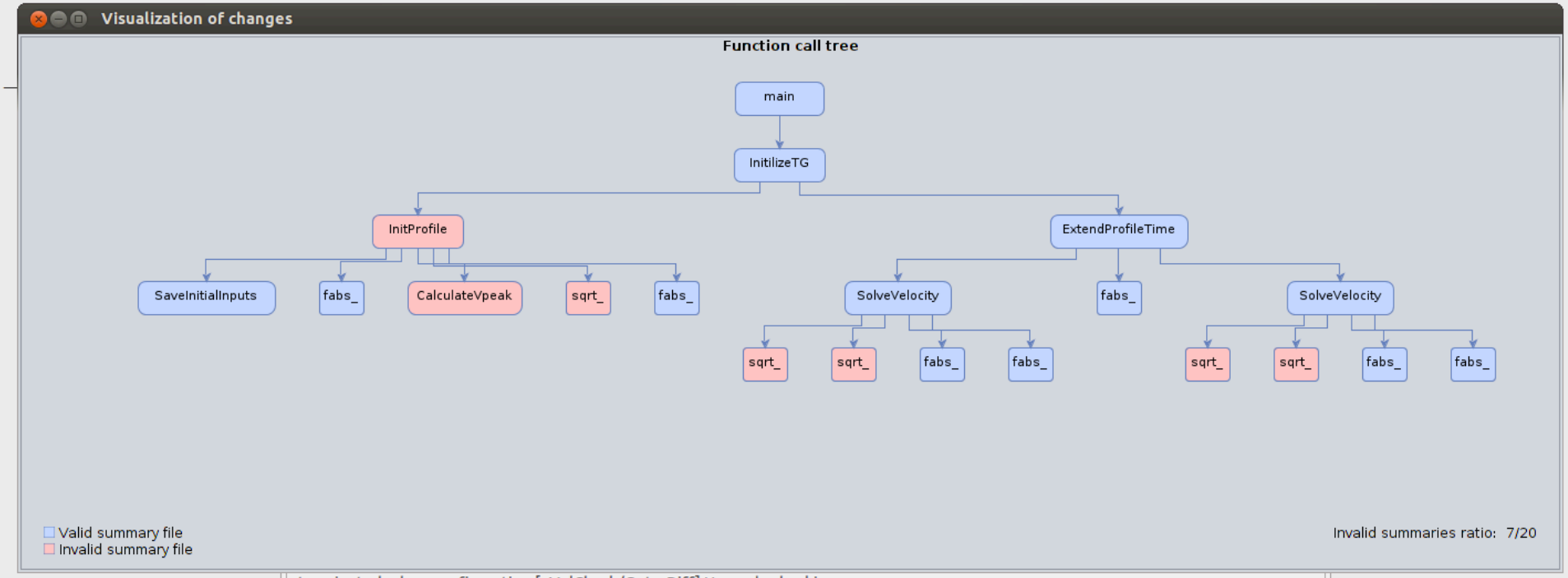

eVolCheck - demo/VTT.c - Eclipse Platform
File Edit Source Refactor Navigate Search Project eVolCheck Run Window Help
Debug
demo cofiguration [eVolCheck/Goto-Diff]
Error trace
<terminated, exit
demo cofiguration
eVolCheck Statistics
Substitutions
Refinements
Total time
SymbEx time
Slicer time
Goto-diff execution
Goto-diff compares source code with respect to the previous successful eVolCheck execution.
There is no previous execution of eVolCheck.
Run bootstrapping.
OK
VTT.c
p.h
Outline
if(delta0 != 0
if(delta0 == 0
if(delta0 == 0
int s = 0;
for(i = 0; i <
{
Project Explorer
Console
Problems
CDT Global Build Console
./demo.o: In function `main':
demo.c:(.text+0xad): undefined reference to `assert'
demo.c:(.text+0xcd): undefined reference to `assert'
collect2: ld returned 1 exit status
make: *** [demo] Error 1
**** Build Finished ****
Launching demo cofiguration: (36%)

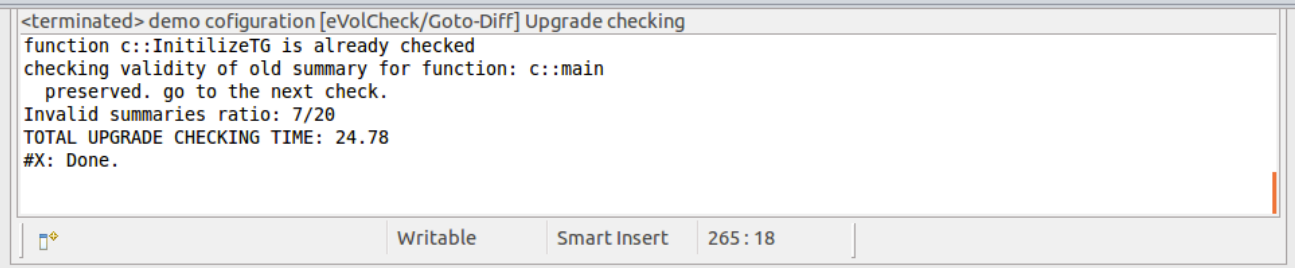

eVolCheck - demo/VTT.c - Eclipse Platform
File Edit Source Refactor Navigate Search Run Project eVolCheck Window Help
Debug
demo cofiguration [eVolCheck/Goto-Diff]
Error trace
<terminated, exit value: 0>Initial check of the program
eVolCheck
Substitutions Refinements Total time SymbEx time Slicer time
0 0 11.204 0.132 0
Upgrade result
Bootstrapping is successful.
Total time: 11.204
OK
VTT.c
Outline
InitPr
jl[2], // acceleration
jl[3] // deceleration
Project Explorer
Console
Problems
<terminated> demo cofiguration [eVolCheck/Goto-Diff] Initial check of the program
ASSERTION IS TRUE
INTERPOLATION TIME: 0.578
ASSERTION(S) HOLD(S)
Total number of steps: 1
TOTAL TIME FOR CHECKING THIS CLAIM: 11.204
#X: Done.
Writable
Smart Insert
150 : 67


eVolCheck - demo/VTT.c - Eclipse Platform
File Edit Source Refactor Navigate Search Run Project eVolCheck Window Help
Debug
demo cofiguration [eVolCheck/Goto-Diff]
Error trace
<terminated, exit value: 0>Initial check of the program
demo cofiguration [eVolCheck/Goto-Diff]
eVolCheck
Substitutions Refinements Total time SymbEx time Slicer time
0 0 11.204 0.132 0
VTT.c p.h demo.c
// Code & comments start
int num, den, result;
num = d * (v0 * v0 + 3 * a * x);
den = d + a; // should both be +ve
Variables Outline
Project Explorer Console Problems
Evolcheck Console
DEBUG: File: /home/loris/runtime-EclipseApplication/demo/VTT.c
INFO: Current project consists of: 1 files.
DEBUG: Deleting 0 previous markers in file: VTT.c
DEBUG: File to be marked: VTT.c
DEBUG: File to be marked: VTT.c
DEBUG: File to be marked: VTT.c


eVolCheck - demo/VTT.c - Eclipse Platform
File Edit Source Refactor Navigate Search Run Project eVolCheck Window Help
Debug
<terminated, exit value: 0>Initial check of the program
demo cofiguration [eVolCheck/Goto-Diff]
demo cofiguration [eVolCheck/Goto-Diff]
Error trace
eVolCheck
Substitutions Refinements Total time SymbEx time Slicer time
0 0 1.762 0.017 0
Upgrade result
Upgrade is safe.
Total time: 1.762
OK
VTT.c
p.h
Outline
// Code & com
int num, den,
num = d * (v0 * v0 + 3 * a * x);
den = d + a; // should both be +ve
Project Explorer
Console
Problems
<terminated> demo cofiguration [eVolCheck/Goto-Diff] Upgrade checking
summary was verified. go to the next check.
checking validity of old summary for function: c::main
preserved. go to the next check.
Invalid summaries ratio: 0/20
TOTAL UPGRADE CHECKING TIME: 1.762
#X: Done.


eVolCheck - demo/VTT.c - Eclipse Platform
File Edit Source Refactor Navigate Search Run Project eVolCheck Window Help
Visualization of changes
Function call tree
main
InitilizeTG
ExtendProfileTime
fabs_
SolveVelocity
fabs_
SolveVelocity
sqrt_
sqrt_
fabs_
fabs_
sqrt_
sqrt_
fabs_
fabs_
Invalid summaries ratio: 0/20


Visualization of changes
Function call tree
main
InitilizeTG
InitProfile
ExtendProfileTime
SaveInitialInputs
fabs_
CalculateVpeak
sqrt_
fabs_
SolveVelocity
fabs_
SolveVelocity
sqrt_
sqrt_
fabs_
fabs_
sqrt_
sqrt_
fabs_
fabs_
Valid summary file
Invalid summary file
Invalid summaries ratio: 7/20


<terminated> demo cofiguration [eVolCheck/Goto-Diff] Upgrade checking
function c::InitilizeTG is already checked
checking validity of old summary for function: c::main
preserved. go to the next check.
Invalid summaries ratio: 7/20
TOTAL UPGRADE CHECKING TIME: 24.78
#X: Done.
Writable
Smart Insert
265 : 18

Summary

- Technology for incremental upgrade checking
  - Local check → cheap check
  - Support for incremental SW development
- Tool: ***eVolCheck***
  - **http://www.verify.inf.usi.ch/evolcheck.html**

Thank You!

Thank You!

257647 PINCETTE

Thank you for your attention!

- Software evolves
  - Small frequent upgrades
  - Complete re-verification impractical / infeasible
- Incremental verification
  - Store information from previous verification runs
  - Speed-up consecutive runs
- Our take
  - Bounded model checking
  - Interpolation-based function summaries

Function summary

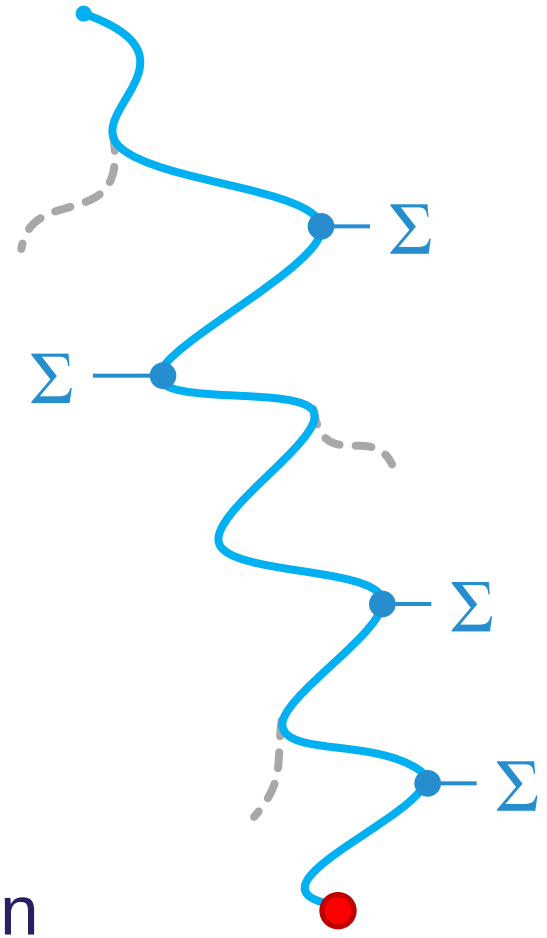
- Over-approximation of real behavior
  - Considering the given bound
- Contains relevant information
  - Derived from UNSAT proof
- Expressed using function's in/out parameters

Usage

1) Same code, different prop.
   - To approximate functions
2) Same prop., different code
   - In upgrade checking

Check with summaries

1) No error is reachable
   → OK, program is safe
2) Error is reachable
   A) due to over-approximation of summaries
   B) real error

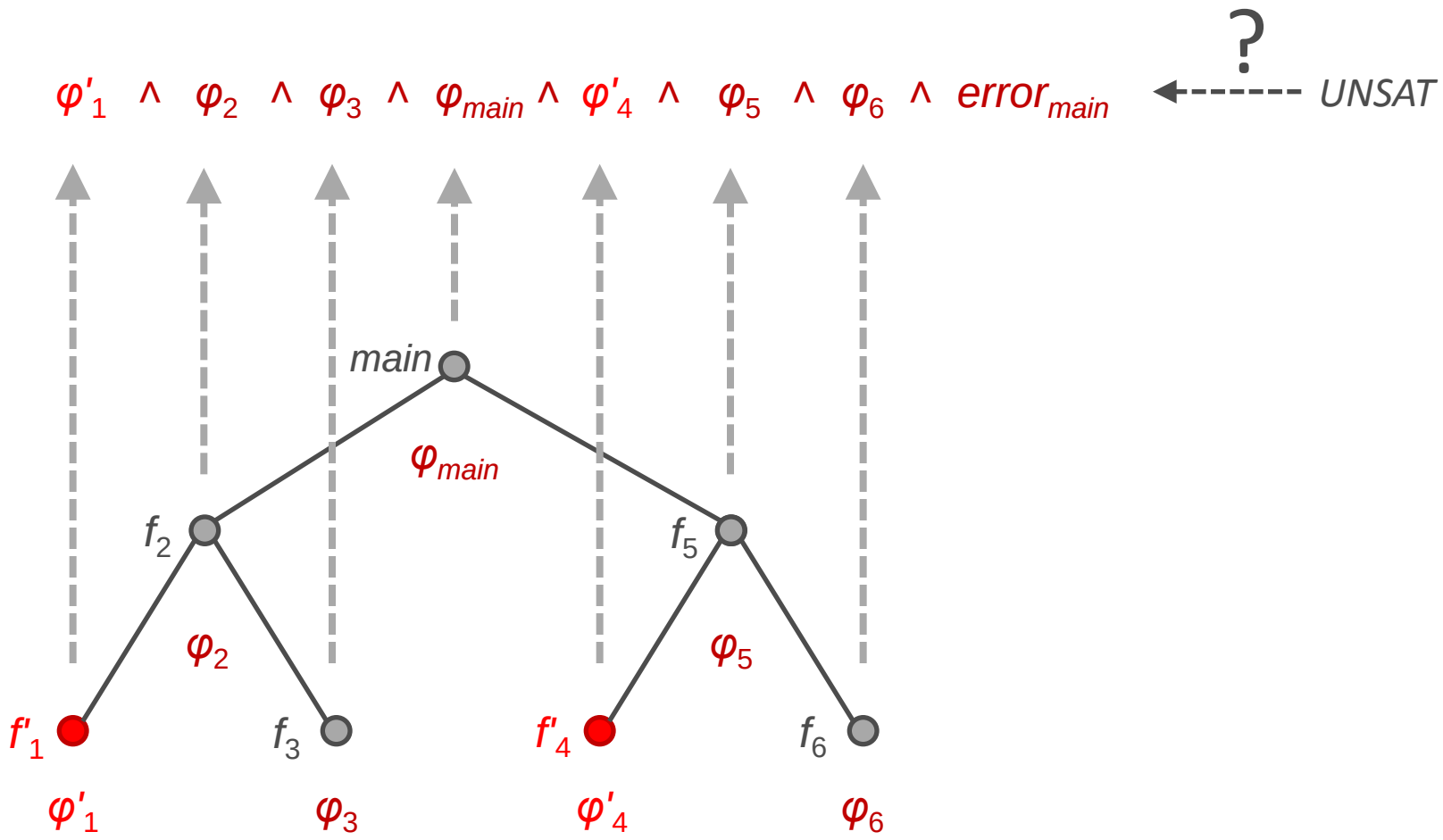


Solution: Refine the abstraction

- Identify victim summaries
  - Error trace analysis
  - Dependency analysis
- Replace them by precise representation

Emergency slides

Correctness

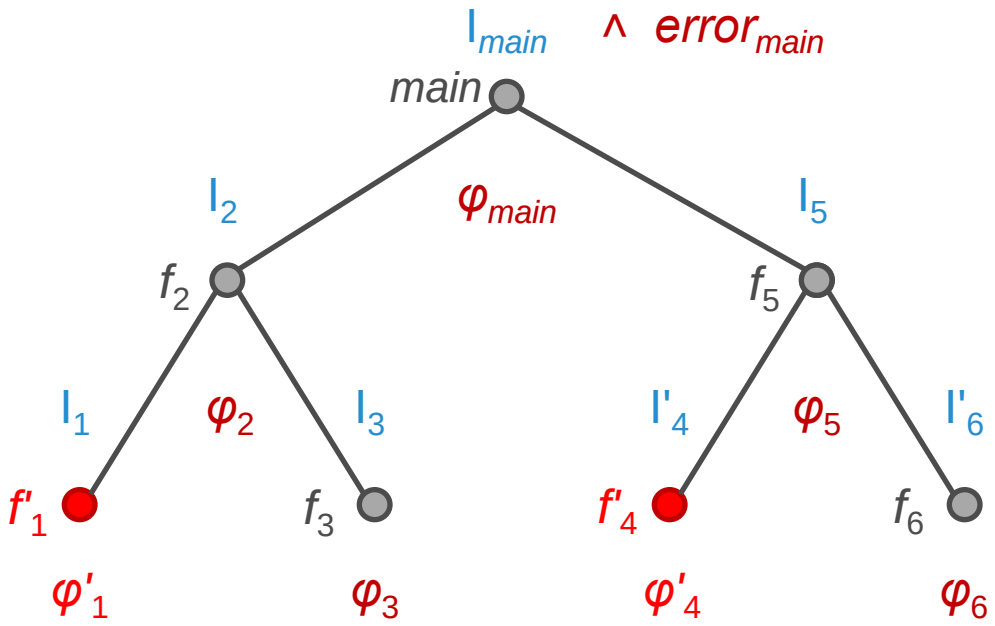


Correctness

(iv) *tree interpolant property:*

$\varphi_f \wedge I_{child_1} \wedge \ldots \wedge I_{child_n} \rightarrow I_f$

$\varphi'_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{main} \wedge \varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \wedge error_{main}$?⟵ UNSAT

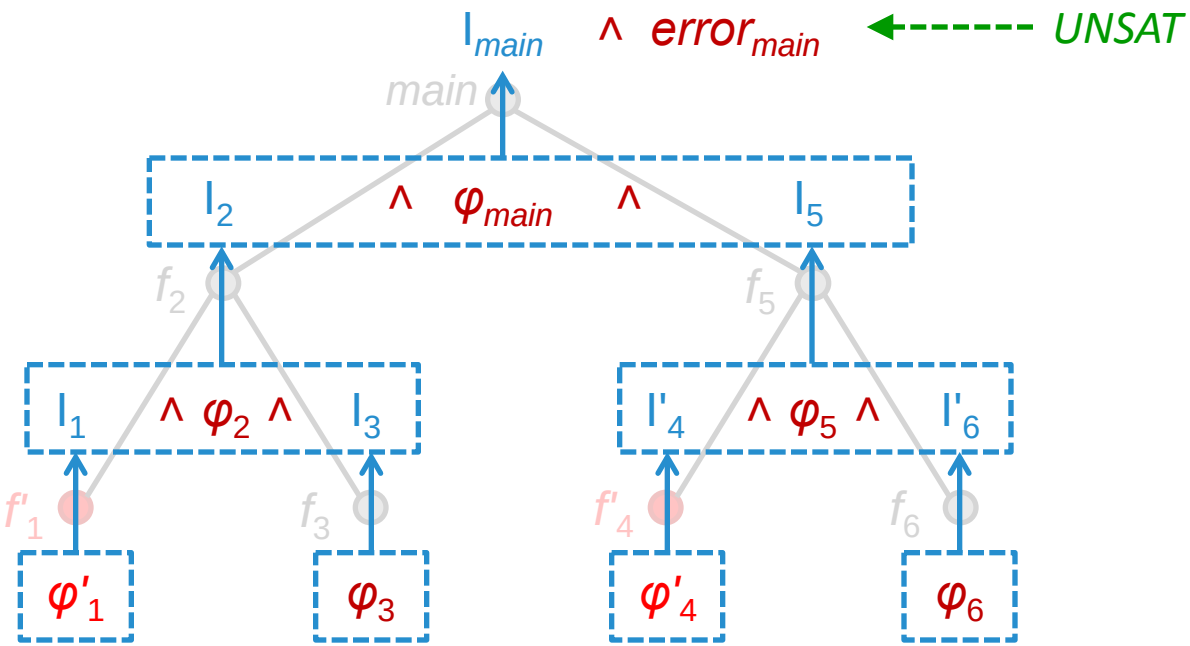


Correctness

(iv) *tree interpolant property:*

$\varphi_f \wedge I_{child_1} \wedge \ldots \wedge I_{child_n} \rightarrow I_f$

$\varphi'_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{main} \wedge \varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \wedge error_{main}$? ⟵ UNSAT



Correctness

(iv) *tree interpolant property:*

$\varphi_f \wedge I_{child_1} \wedge \ldots \wedge I_{child_n} \rightarrow I_f$

$\varphi'_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{main} \wedge \varphi'_4 \wedge \varphi_5 \wedge \varphi_6 \wedge error_{main}$ ✓ ⟵ UNSAT

$I_{main} \wedge error_{main}$ ⟵ UNSAT