**Приложение Б**

**Пример использования системы**

Запуск системы можно осуществить следующим образом:

java -cp proto-0.0.1.jar;proto-0.0.1-jar-with-dependencies.jar proto.Launcher

Welcome to Proto Compiler 0.0.1! [http://github.com/hisohito/proto-code]

USAGE:

java -cp proto-0.0.1.jar proto.Launcher [--target=<TARGET>] [--output=<OUTPUT\_FILE>] <SOURCE>

<SOURCE> is an proto source file

<TARGET> is one of supported targets (promela by default)

<OUTPUT\_FILE> is an output file name (out.pml by default)

Возьмём в качестве примера описанный ранее пример про имитацию работы автоматической коробки передач:

interface Driver { number press(number boxState);}

class DefaultDriver() < Driver {

number press(number boxState){

return random boxState;

}

}

class AutoGear() << Prototype {

Driver driver = new DefaultDriver();

number cmd;

state finalize() { return "End of work"; }

state sleep() { return neutral(); }

state neutral(){

number cmd = driver.press(3);

if (cmd == 1){ return finalize();

} else if (cmd == 2){ return reverse();

} else { return gear1();}

}

state reverse(){ return neutral(); }

state gear1(){

cmd = driver.press(2);

if (cmd == 1){ return neutral();

} else { return gear2(); }

}

state gear2(){

cmd = driver.press(2);

if (cmd == 1){ return gear1();

} else { return gear3(); }

}

state gear3(){

cmd = driver.press(2);

if (cmd == 1){ return gear2();

} else { return gear4(); }

}

state gear4(){

cmd = driver.press(2);

if (cmd == 1){ return gear3();

} else { return gear5(); }

}

state gear5(){ return gear4(); }

state main(){ return sleep(); }

}

prototype AutoGear() { finally {AutoGear::finalize}; }

Запустим систему, с указанием входного файла autogear.proto:

java -cp proto-0.0.1.jar;proto-0.0.1-jar-with-dependencies.jar proto.Launcher "../examples/autogear.proto"

В результате запуска был создан файл protoOutput0.pml:

int stateA1;

#define AUTOGEAR\_\_GEAR5 0

#define AUTOGEAR\_\_NEUTRAL 1

#define AUTOGEAR\_\_REVERSE 2

#define AUTOGEAR\_\_MAIN 3

#define AUTOGEAR\_\_FINALIZE 4

#define AUTOGEAR\_\_GEAR1 5

#define AUTOGEAR\_\_GEAR2 6

#define AUTOGEAR\_\_SLEEP 7

#define AUTOGEAR\_\_GEAR3 8

#define AUTOGEAR\_\_GEAR4 9

inline A1() {

stateA1 = AUTOGEAR\_\_MAIN;

do

:: ( stateA1 == AUTOGEAR\_\_GEAR5 ) ->

printf("AutoGear::gear5");

if

::stateA1 = AUTOGEAR\_\_GEAR4;

fi;

:: ( stateA1 == AUTOGEAR\_\_NEUTRAL ) ->

printf("AutoGear::neutral");

if

::stateA1 = AUTOGEAR\_\_REVERSE;

::stateA1 = AUTOGEAR\_\_FINALIZE;

::stateA1 = AUTOGEAR\_\_GEAR1;

fi;

:: ( stateA1 == AUTOGEAR\_\_REVERSE ) ->

printf("AutoGear::reverse");

if

::stateA1 = AUTOGEAR\_\_NEUTRAL;

fi;

:: ( stateA1 == AUTOGEAR\_\_MAIN ) ->

printf("AutoGear::main");

if

::stateA1 = AUTOGEAR\_\_SLEEP;

fi;

:: ( stateA1 == AUTOGEAR\_\_FINALIZE ) ->

printf("AutoGear::finalize");

break;

:: ( stateA1 == AUTOGEAR\_\_GEAR1 ) ->

printf("AutoGear::gear1");

if

::stateA1 = AUTOGEAR\_\_NEUTRAL;

::stateA1 = AUTOGEAR\_\_GEAR2;

fi;

:: ( stateA1 == AUTOGEAR\_\_GEAR2 ) ->

printf("AutoGear::gear2");

if

::stateA1 = AUTOGEAR\_\_GEAR1;

::stateA1 = AUTOGEAR\_\_GEAR3;

fi;

:: ( stateA1 == AUTOGEAR\_\_SLEEP ) ->

printf("AutoGear::sleep");

if

::stateA1 = AUTOGEAR\_\_NEUTRAL;

fi;

:: ( stateA1 == AUTOGEAR\_\_GEAR3 ) ->

printf("AutoGear::gear3");

if

::stateA1 = AUTOGEAR\_\_GEAR2;

::stateA1 = AUTOGEAR\_\_GEAR4;

fi;

:: ( stateA1 == AUTOGEAR\_\_GEAR4 ) ->

printf("AutoGear::gear4");

if

::stateA1 = AUTOGEAR\_\_GEAR5;

::stateA1 = AUTOGEAR\_\_GEAR3;

fi;

od;

}

proctype Model() { A1(); }

init { run Model(); }

never { /\* !(<>{stateA1 == AUTOGEAR\_\_FINALIZE}) \*/

accept\_init:

T0\_init: if

:: (! ((stateA1 == AUTOGEAR\_\_FINALIZE))) -> goto T0\_init fi;

}

Используя полученную модель, можем запустить SPIN на верификацию:

spin -a protoOutput0.pml

SPIN генерирует файл pan.c, представляющий собой верификатор для данной модели на языке C. Скомпилируем полученный файл с помощью компилятора gcc:

gcc –o”autogear.exe” pan.c

Запускаем полученный файл с ключом a (полная верификация):

autogear.exe –a

warning: for p.o. reduction to be valid the never claim must be stutter-invariant

(never claims generated from LTL formulae are stutter-invariant)

pan:1: acceptance cycle (at depth 16)

pan: wrote protoOutput0.pml.trail

(Spin Version 6.1.0 -- 4 May 2011)

Warning: Search not completed

+ Partial Order Reduction

Full statespace search for:

never claim + (never\_0)

assertion violations + (if within scope of claim)

acceptance cycles + (fairness disabled)

invalid end states - (disabled by never claim)

State-vector 24 byte, depth reached 27, **errors: 1**

14 states, stored

0 states, matched

14 transitions (= stored+matched)

0 atomic steps

hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):

0.001 equivalent memory usage for states (stored\*(State-vector + overhead))

0.286 actual memory usage for states (unsuccessful compression: 53575.00%)

state-vector as stored = 21414 byte + 16 byte overhead

2.000 memory used for hash table (-w19)

0.343 memory used for DFS stack (-m10000)

2.539 total actual memory usage

pan: elapsed time 0.001 seconds

Если ошибки найдены, то строится trail-файл. В данном случае trail-файл выглядит так:

-2:2:-2

-4:-4:-4

1:0:62

2:1:60

3:0:62

4:2:0

5:0:62

6:2:18

7:0:62

8:2:19

9:0:62

10:2:20

11:0:62

12:2:38

13:0:62

14:2:39

15:0:62

16:2:40

-1:-1:-1

17:0:62

18:2:6

19:0:62

20:2:7

21:0:62

22:2:8

23:0:62

24:2:13

25:0:62

26:2:14

27:0:62

28:2:15

Верификатор SPIN обрабатывает этот файл и строит контрпример:

spin -t -p protoOutput0.pml

Never claim moves to line 79 [(!((stateA1==4)))]

Starting Model with pid 2

2: proc 0 (:init:) protoOutput0.pml:77 (state 1) [(run Model())]

4: proc 1 (Model) protoOutput0.pml:15 (state 1) [stateA1 = 3]

6: proc 1 (Model) protoOutput0.pml:34 (state 19) [((stateA1==3))]

AutoGear::main 8: proc 1 (Model) protoOutput0.pml:35 (state 20) [printf('AutoGear::main')]

10: proc 1 (Model) protoOutput0.pml:37 (state 21) [stateA1 = 7]

12: proc 1 (Model) protoOutput0.pml:54 (state 39) [((stateA1==7))]

AutoGear::sleep 14: proc 1 (Model) protoOutput0.pml:55 (state 40) [printf('AutoGear::sleep')]

16: proc 1 (Model) protoOutput0.pml:57 (state 41) [stateA1 = 1]

<<<<<START OF CYCLE>>>>>

18: proc 1 (Model) protoOutput0.pml:22 (state 7) [((stateA1==1))]

AutoGear::neutral 20: proc 1 (Model) protoOutput0.pml:23 (state 8) [printf('AutoGear::neutral')]

22: proc 1 (Model) protoOutput0.pml:25 (state 9) [stateA1 = 2]

24: proc 1 (Model) protoOutput0.pml:29 (state 14) [((stateA1==2))]

AutoGear::reverse 26: proc 1 (Model) protoOutput0.pml:30 (state 15) [printf('AutoGear::reverse')]

28: proc 1 (Model) protoOutput0.pml:32 (state 16) [stateA1 = 1]

spin: trail ends after 28 steps

#processes: 2

stateA1 = 1

lastEvent = 0

28: proc 1 (Model) protoOutput0.pml:16 (state 56)

28: proc 0 (:init:) protoOutput0.pml:78 (state 2) <valid end state>

28: proc - (never\_0) protoOutput0.pml:79 (state 3)

2 processes created

Исправим требования, проделаем весь процесс сначала и получим следующее:

warning: for p.o. reduction to be valid the never claim must be stutter-invariant

(never claims generated from LTL formulae are stutter-invariant)

(Spin Version 6.1.0 -- 4 May 2011)

+ Partial Order Reduction

Full statespace search for:

never claim + (never\_0)

assertion violations + (if within scope of claim)

acceptance cycles + (fairness disabled)

invalid end states - (disabled by never claim)

State-vector 20 byte, depth reached 0, **errors: 0**

1 states, stored

0 states, matched

1 transitions (= stored+matched)

0 atomic steps

hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):

0.000 equivalent memory usage for states (stored\*(State-vector + overhead))

0.286 actual memory usage for states (unsuccessful compression: 832500.00%)

state-vector as stored = 299684 byte + 16 byte overhead

2.000 memory used for hash table (-w19)

0.343 memory used for DFS stack (-m10000)

2.539 total actual memory usage

unreached in proctype Model

(0 of 60 states)

unreached in init

protoOutput0.pml:78, state 2, "-end-"

(1 of 2 states)

unreached in claim never\_0

protoOutput0.pml:88, state 8, "-end-"

(1 of 8 states)

pan: elapsed time 0 seconds