ATKINS

Verification by Static Analysis

Intelligent Testing Conference

Bristol, 17th March 2014

Plan Design Enable

Verification overview

ATKINS

- Software Verification is “The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements”, ([IEEE-STD-610)
- There are a number of techniques that can be used, but the two main techniques that are used for software verification are
  - Dynamic Testing
  - Static Analysis

- The presentation talks about the different uses and benefits of static analysis techniques

Limitations of Software Testing

ATKINS

- For any reasonably complex software it is not possible to test all paths through the code or all combinations of inputs
  - Level crossing application – 187 boolean inputs, 10^{56} possible combinations to test!
  - Nuclear Power application – 20 sequential IF statements, > 1 million paths
- Testing is expensive relative to the coverage achieved
- Latent bugs will remain in the software after testing

Types of Static Analysis

ATKINS

Static Analysis is any verification technique that verifies the software without executing it.

There are two main objectives:

- **Integrity Checking**, checking for an absence of run-time errors
- **Functional Analysis**, checking that the code and specification are consistent

Analysis is normally tool supported, and can include a wide range of approaches

- Compiler Checks (e.g. Variables initialised before use)
- Heuristic Tools (that find some instances of faults but not all)
- Formal Verification (that constructs a proof that the software is free of certain classes of error)

Benefits of Static Analysis

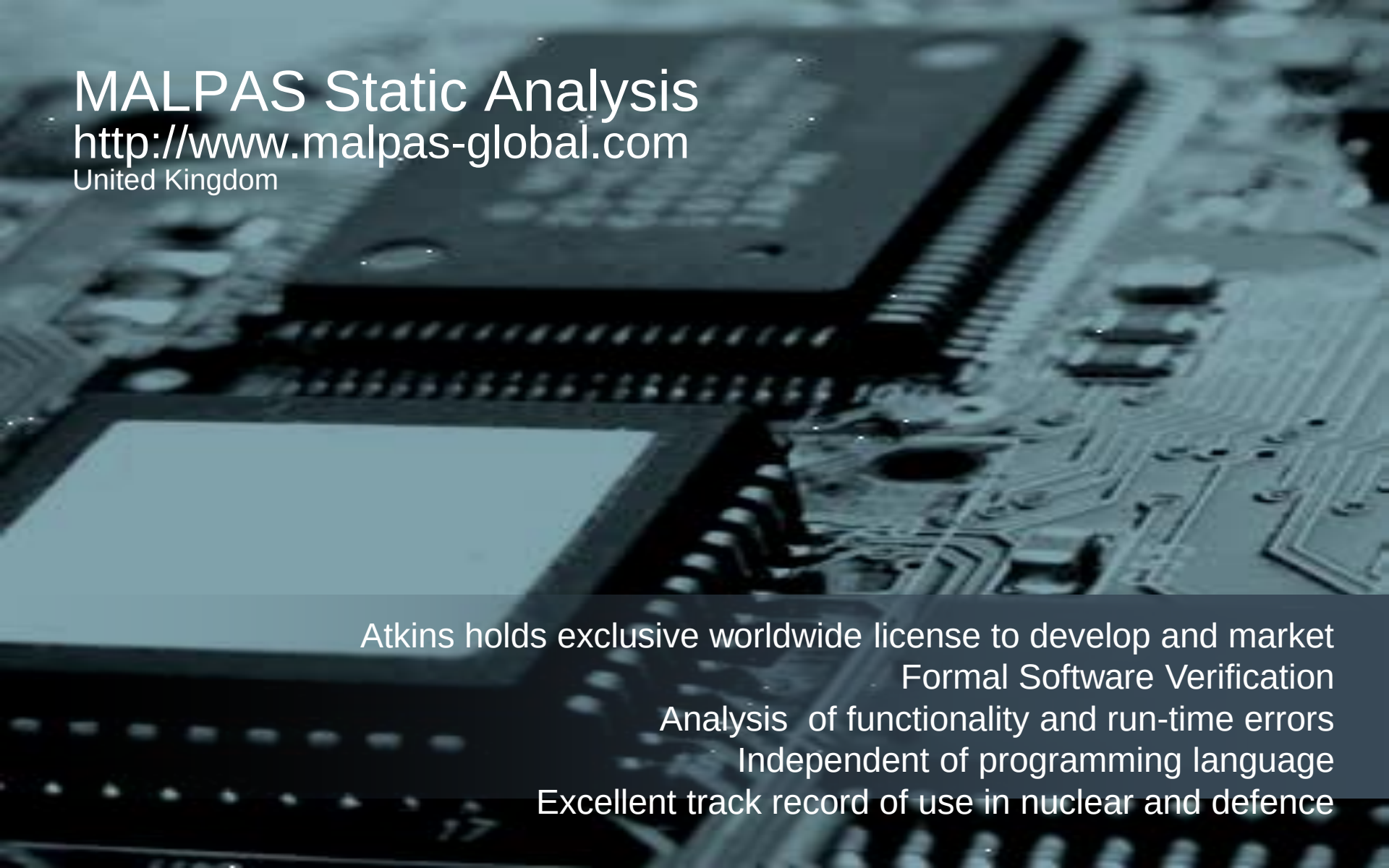
ATKINS

- **Quick**. The tool checks are automated and run quickly
- **Objective.** The analysis is not dependent on the skills and bias of the tester
- **Coverage.** Analysis will cover all paths through the code and all possible combinations of input
- **Repeatable and Auditable.** The analysis meets the good practice of being repeatable, and will generate auditable evidence

Common uses

ATKINS

- Trusted Software Initiative
  - Cabinet Office initiative to improve software quality
  - Mainly aimed at Cyber risks
  - Recommend use of static analysis, e.g. “turn on compiler warnings”
- Cyber Security – to check for buffer overruns, etc
- Safety Critical Systems
  - Nuclear Power regulator requires formal verification for critical systems
  - Railway standard recommends static analysis techniques for safety critical systems



MALPAS Static Analysis

http://www.malpas-global.com

United Kingdom

Atkins holds exclusive worldwide license to develop and market
Formal Software Verification
Analysis of functionality and run-time errors
Independent of programming language
Excellent track record of use in nuclear and defence

MALPAS Background

ATKINS

Originally developed by UK Ministry of Defence in early 1980s

Rights acquired by Atkins in 1986 to sell licenses and develop the tool

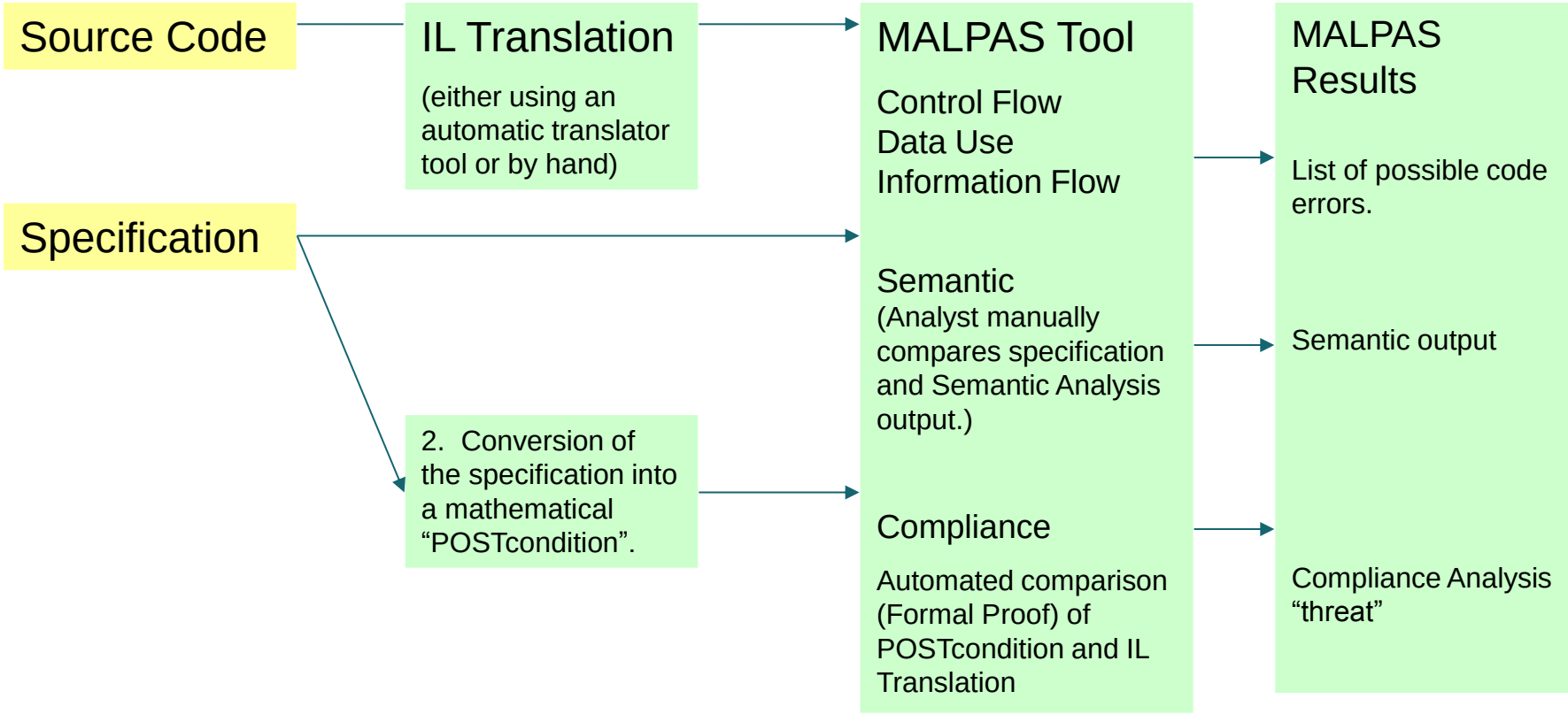
De facto tool for verification of safety critical systems in UK Nuclear Power Plants

Also used widely in UK Defence projects and Rail projects

MALPAS can

- Improve the integrity of systems
- Demonstrate to regulatory bodies and customers that programs are free of errors
- Support software safety cases that require strong analytical evidence at a high integrity level
- Create critical software more cost-effectively (i.e. delivered on time and within budget)
- Formal proof of malware absence

MALPAS Static Analysis



Example Compliance Analysis

ATKINS

```
int *a;
int b, c;
proc1 ()
{
a = &b;
b = 3;
c = *a;   /* final value of c is 3 */
}

proc2 () {
proc1();  /* prove that c = 3 */
}
```

Translated C

```
TITLE ex1;
_INCLUDE/NOLIST/NOWARN "EX1.PRE"
PROCSPEC [proc1] proc1(
       OUT   result_proc1 : int)
IMPLICIT (
   INOUT c : int
   INOUT b : int
   INOUT a : int-pointer)
_INCLUDE "EX1_proc1.0"
PRE  true
POST c = 3;

PROCSPEC [proc2] proc2(
       OUT   result_proc2 : int)
IMPLICIT (
   INOUT c : int
   INOUT b : int
   INOUT a : int-pointer)
_INCLUDE "EX1_proc2.0"
PRE  true
POST c = 3;
```

```
PROC proc1;

   a := POINTER $int_at_b;
   b := 3;
   $malpas_check(a EQ POINTER $int_at_b);
   c := b
ENDPROC [proc1]


PROC proc2;
   VAR temp_1 : int;
   proc1(temp_1) ASSUME POST
ENDPROC [proc2]
FINISH
```

Results

```
--------------------------------------------------------------------------------
Semantic/Compliance Release 8.4.34 17 March 2014 09:32:41 - Compliance Analysis
ex1 - Procedure proc1
--------------------------------------------------------------------------------
After STRICT ONE-ONE
From node #START to node #END   (0 semantically impossible paths detected)
All threats are false
[------------------------------------------------------------------------------]

--------------------------------------------------------------------------------
Semantic/Compliance Release 8.4.34 17 March 2014 09:32:41 - Compliance Analysis
ex1 - Procedure proc2
--------------------------------------------------------------------------------
After STRICT ONE-ONE
From node #START to node #END   (0 semantically impossible paths detected)
All threats are false
[------------------------------------------------------------------------------]
Date time = 17/03/14 09:32:41.59  Semantic/Compliance finished
```

.

www.atkinsglobal.com