



Design and Development of ExTGT an Extensible Test Generation Tool based on the Eclipse Rich Client Platform

Angelo Gargantini

Università degli Studi di Bergamo, Italia
angelo.gargantini@unibg.it

Gordon Fraser,

Saarland University, Saarbrücken, Germany

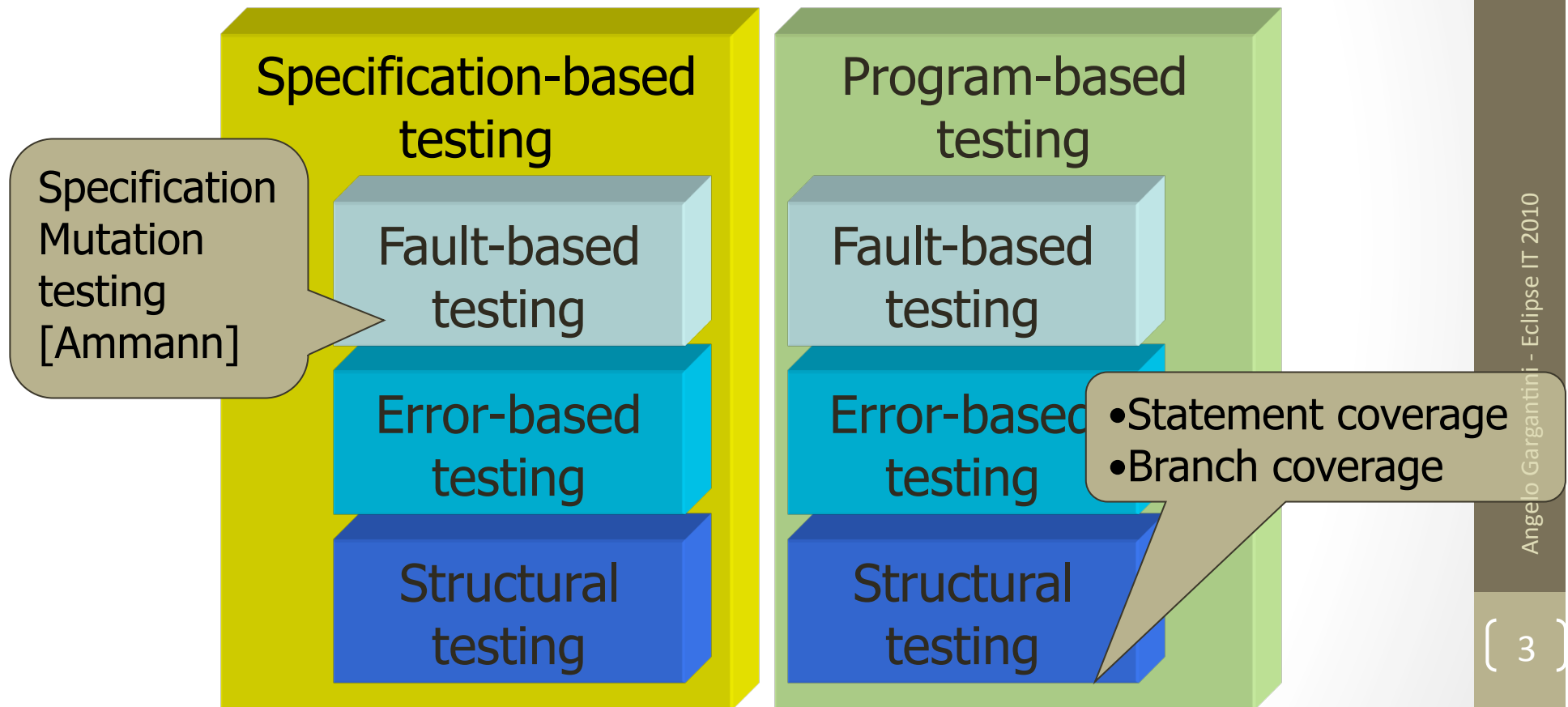


Outline

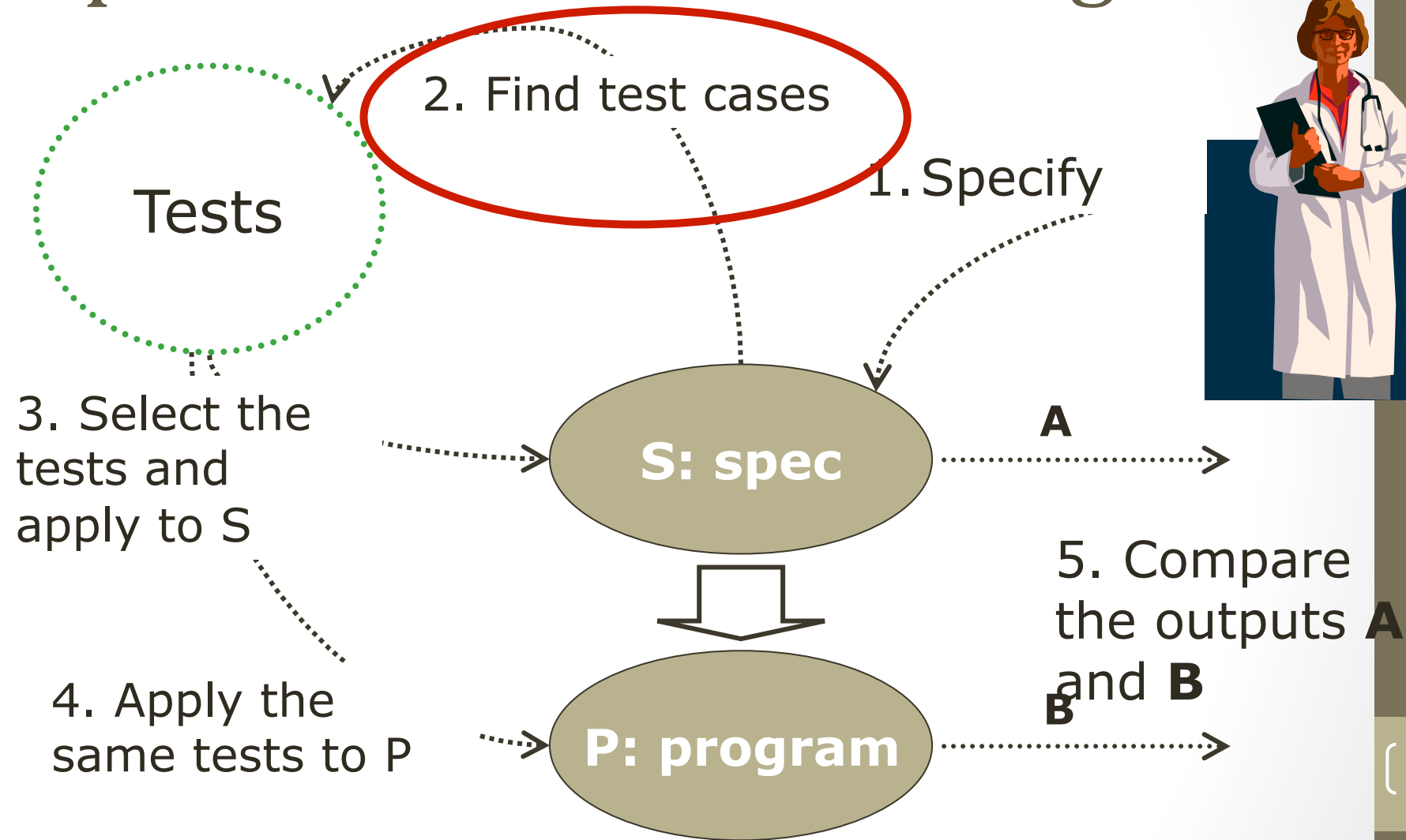
- Model based testing
- Using model checkers for test generation
 - Problems and opportunities
- Exploiting the RCP platform to develop an extensible tool for model-based test generation
 - Extensions and extension points
 - Architecture
 - Validating
- Pro and cons
- Future work

Testing criteria taxonomy

[Zhu, Hall & May]



Specification-Based Testing (2)



Logic approach to the Test Generation Problems

- Testing can be viewed as a logic problem
 1. **Formalize** the testing problem with a suitable logics
 - As a set of logical predicates “**test predicates**”
 2. **Solve** it by a suitable tool
 - Reuse techniques and tools used normally for verification
 - Model checkers, constraint solvers, SMT, SAT and so on
- *TESTS AND PROOFS (TAP) manifesto*

Testing and proving are complementary

Formalize: Test Predicates

- Testing criterion defined by a set of *test predicates*
 - formulas over the state determining if a particular testing goal is reached
- A *test set T is adequate according to a criterion C* if each test predicate of C is true in at least one state of a test sequence of T

- Example

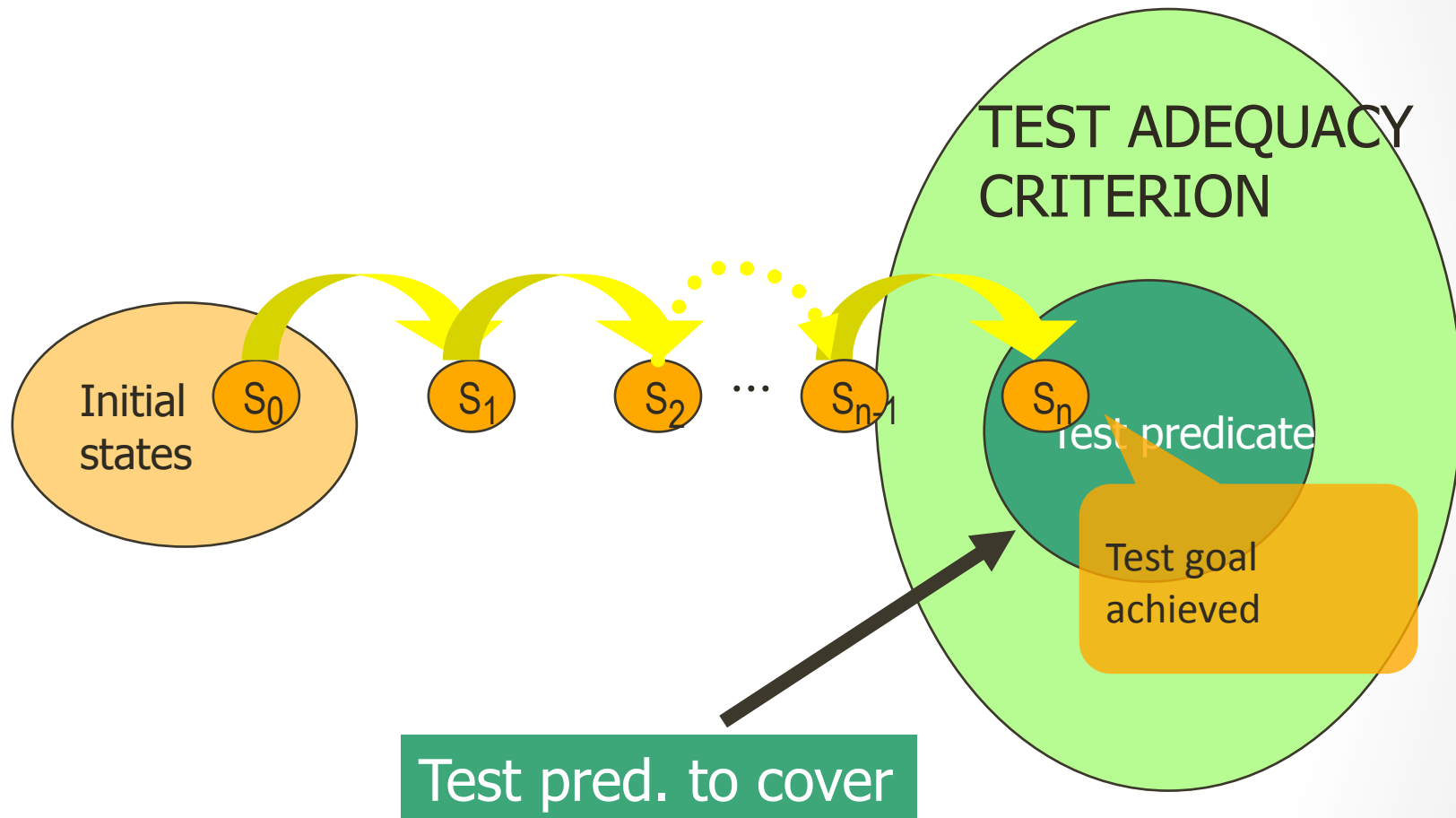
I want to test the case in which x is greater than 0

Test predicate tp: $x > 0$

Solve: find a test

- Find a behavior of the system (as modeled) that satisfies the **model and the test predicate**
 - Test: a **sequence** of inputs with the correct outputs to satisfies the test requiremnt

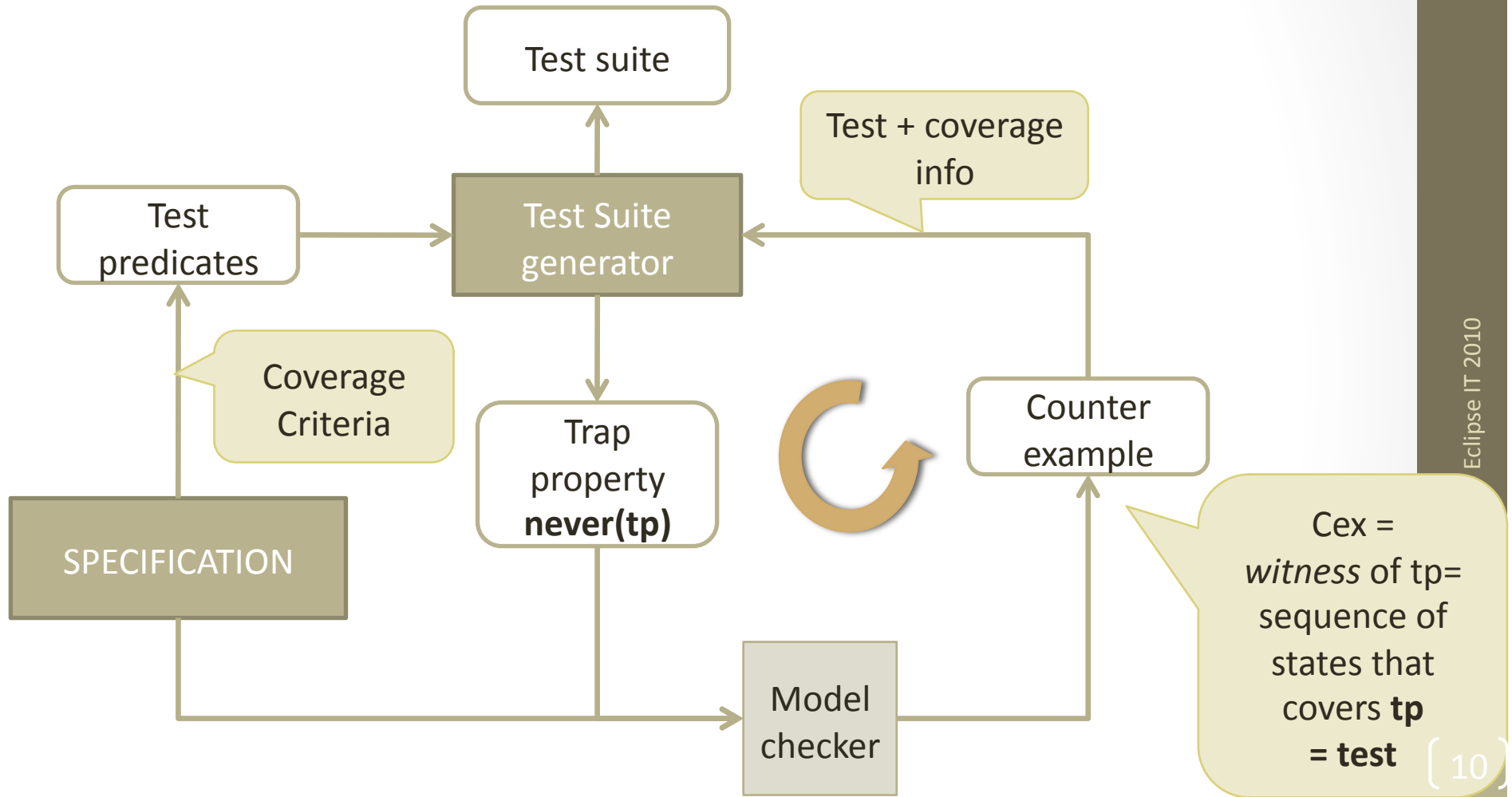
Test predicate \rightarrow Test sequence



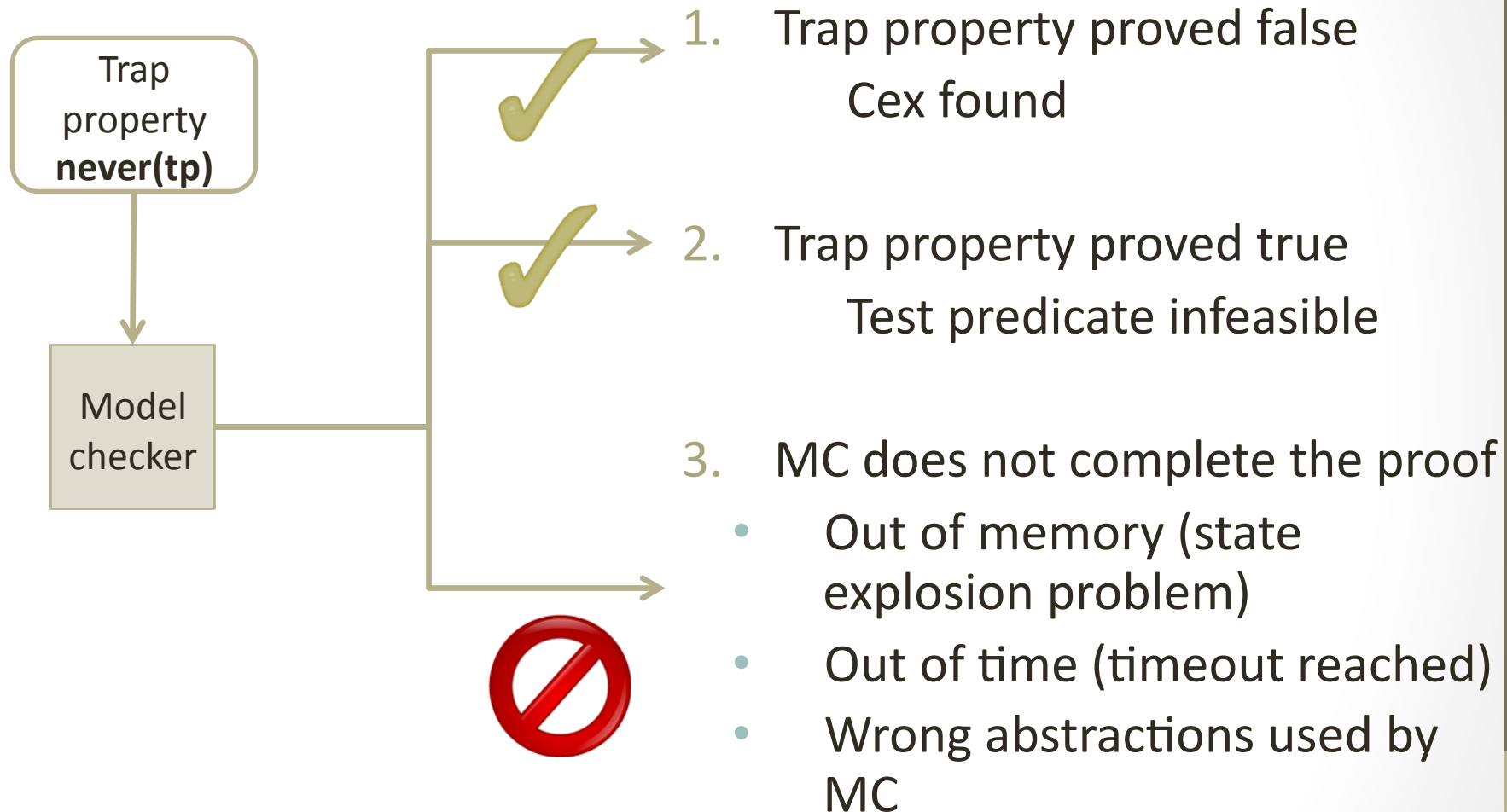
Test generation by model checking

- Model checking can be used for model-based tests generation
 - [FAW09] reviewed 140 papers
 - [GH99] and [ABM98] have more than 260 citations
 - several notations, systems, coverage criteria (data flow, structural, mutation, ...) and using several model checkers
 - [FAW09] Fraser, Ammann, and Wotawa. *Testing with Model Checkers: A Survey*. *Journal for Software Testing, Verification and Reliability*, 2009
 - [GH99] Gargantini, Heitmeyer. *Using model checking to generate tests from requirements specifications*. *FSE/ESEC*, 1999
 - [ABM98] Ammann, Black, Majurski. *Using model checking to generate tests from specifications*. *Formal Engineering Methods*, 1998
- Several commercial tools exploits similar techniques: Conformiq, Matlab DV toolbox, ...

Mc for test generation



Some limits



Open issues – state of the art

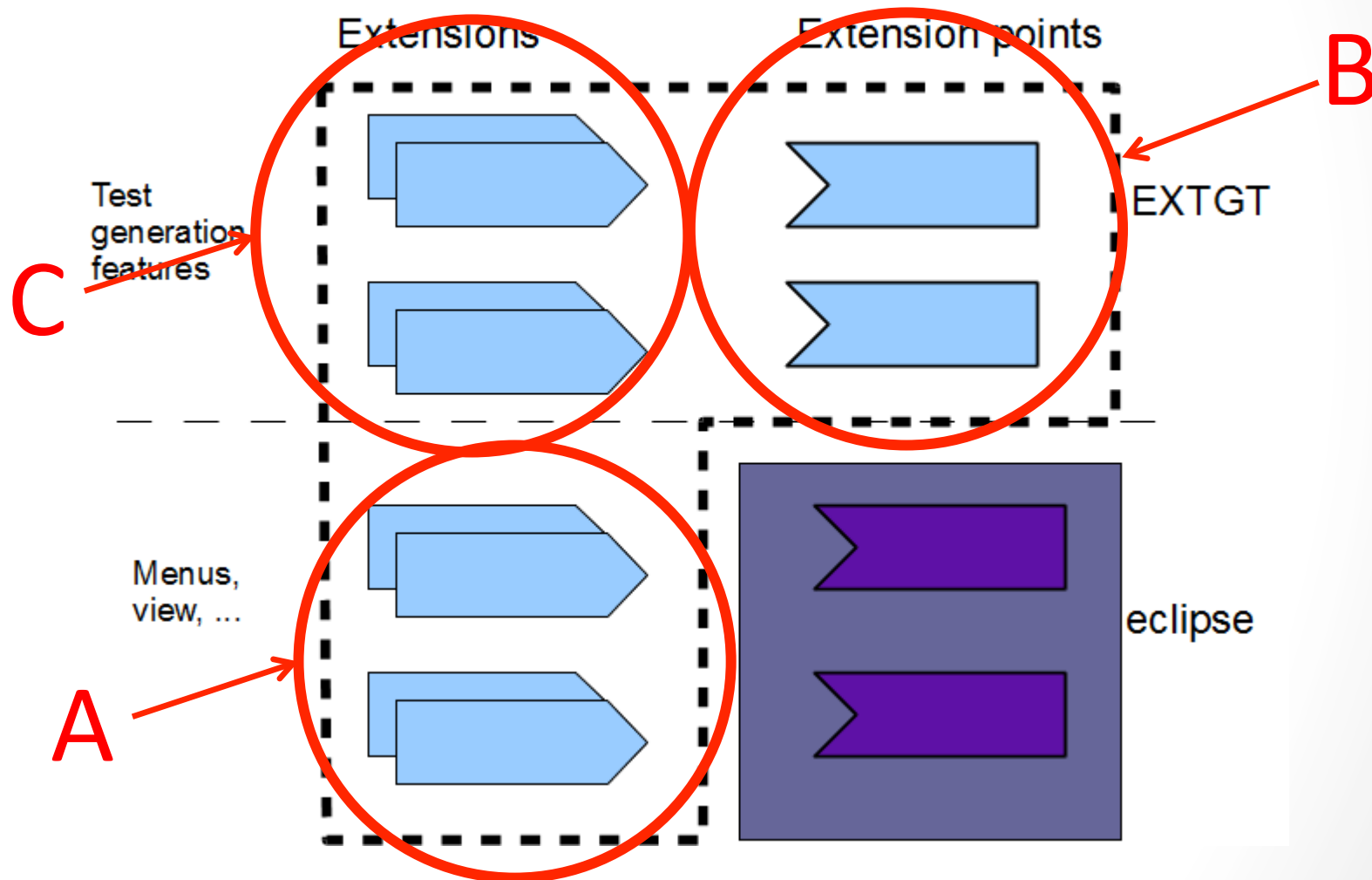
- Which testing criteria work better?
 - Structural, combinatorial, fault based ...
- Which model checker?
 - Explicit state ...
- Code reuse – existing tool ATGT
 - Tool for generation of tests starting from Abstract State Machine
 - Experience with SCR, ASM, Boolean spec
 - Interested in defining a base framework in which several components can be added later
 - Coverage criteria
 - Model checkers
 - ...



Exploiting the plugin architecture

- Eclipse RCP for flexible and extensible desktop applications.
- Eclipse RCP applications can decide to use parts of the components provided by Eclipse, like editor, menus and so on.
- Eclipse provides the concept of extension points and extensions
 - An **extension point** defines a contract how other plug-ins can contribute
 - An **extension** contributes to the defined extension point
- The **Extensible Test Generator Tool** - ExTGT :
 - Is itself a plug-in which provides several **extensions** to standard Eclipse components, like view, perspectives, menus, editors ...
 - defines several **extension points** and **extensions**

ExTGT extensions [points]



A Extensions of the eclipse/RCP

- ExtTGT is itself a plug-in to Eclipse and provides several extensions to standard Eclipse components such as views, perspectives, menus, actions ...
- **Example**

```
<extension point="org.eclipse.ui.perspectives">
    <perspective
        class="xtgt.Perspective"
        icon="icons/icona.gif"
    </perspective>
</extension>
<extension point="org.eclipse.ui.views">
    .....
<extension point="org.eclipse.ui.editors">
```

B Extension points of ExTGT

- `<extension point="xtgt.AsmSpecReader">`
to introduce several parsers for Abstract State Machine specifications.
- `<extension point="xtgt.coverageBuilders">`
to define new testing criteria building a list of test predicates.
- `<extension point="xtgt.faultexpression">`
to introduce new mutation operators for expressions
- `<extension point="xtgt.generatorMethod">`
to introduce new test generator methods: given a test predicate return a test that satisfies it

C Extensions

<AsmSpecReader

```
point="xtgt.AsmetaLoader">
```

To read specs in the Asmeta format

```
point="xtgt.AsmGoferLoader">
```

To read AsmSpec in ASMGofer format

<TestGeneratorMethod

```
point="xtgt.TestGeneratorMethodSpinFlat">
```

Spin (JPL) – explicit state model checker

```
point="xtgt.TestGenMethodSalCombinatorial">
```

SAL (SRI) – BDD/BMC model checker

```
point="xtgt.TestGeneratorMethodHysat">
```

HySAT – SAT solver for non linear constraints

```
point="xtgt.TestGeneratorMethodYices">
```

Yices – SMT Solver

Coverage criteria

```
<CoverageBuilder
```

```
    point="xtgt.BasicRuleVisitor"/>
```

```
    point="xtgt.CompleteRuleVisitor"/>
```

```
    point="xtgt.RuleUpdateVisitor"/>
```

Structural based testing criteria

```
    point="xtgt.MCDCCoverage"/>
```

Modified Condition and Decision Coverage

```
    point="xtgt.PairwiseCovBuild">
```

Combinatorial testing

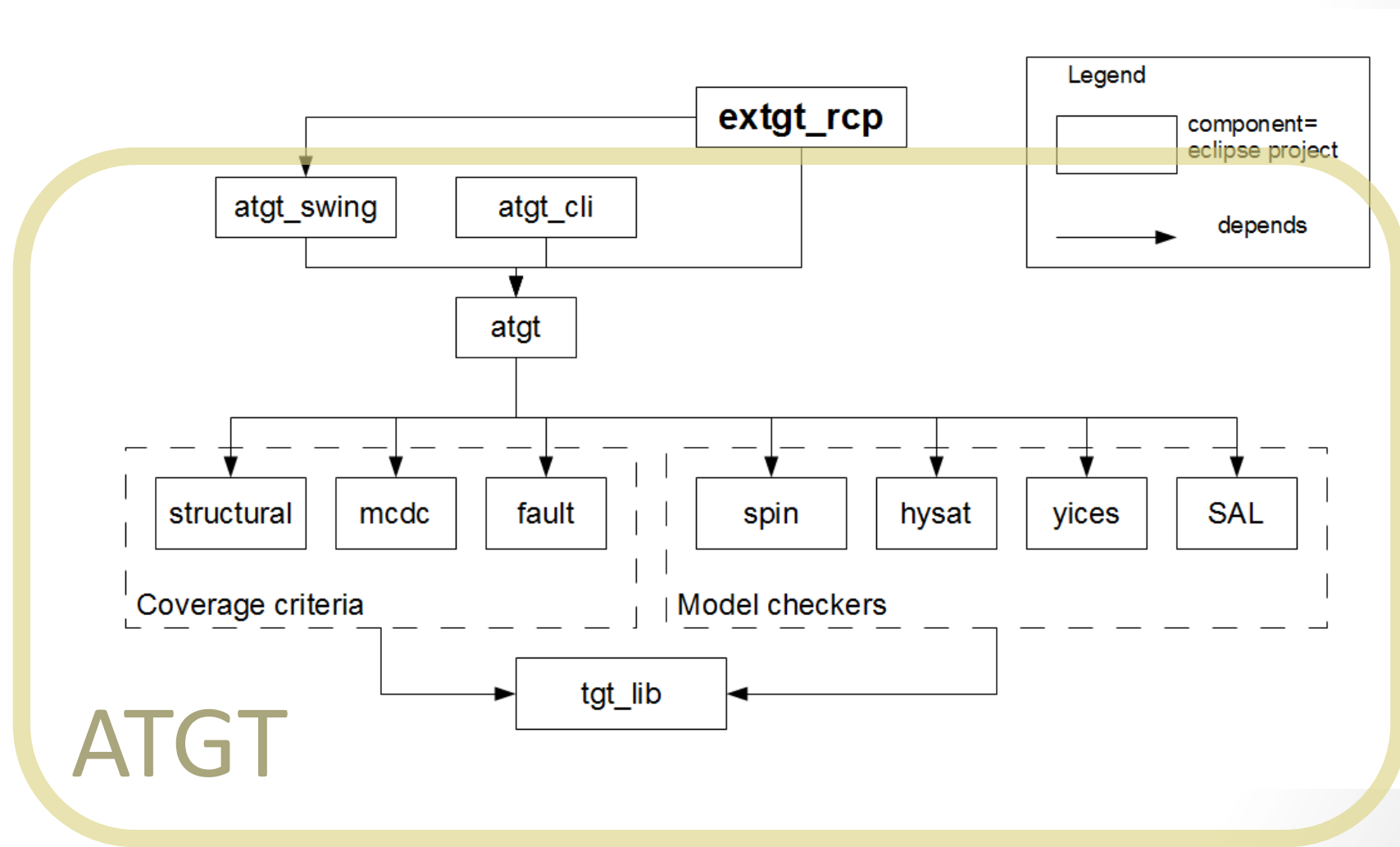
```
    point="xtgt.FaultBasedCovBuilder">
```

Fault based testing

Fault classes/Mutation operators

```
• <faultexpression
•   class="extgt.coverage.fault.mutators.AssociativeShiftFault"
•   group="FaultExpression"
•   id="AssociativeShiftFault"
•   point="xtgt.AssociativeShiftFault"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.ExpressionNegationFault"
•   group="FaultExpression"
•   id="ExpressionNegationFault"
•   point="xtgt.ExpressionNegationFault"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.LiteralNegationFault"
•   group="FaultExpression"
•   id="LiteralNegationFault"
•   point="xtgt.LiteralNegationFault"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.MissingLiteralFault"
•   group="FaultExpression"
•   id="MissingLiteralFault"
•   point="xtgt.MissingLiteralFault"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.OperatorReferenceFault"
•   group="FaultExpression"
•   id="OperatorReferenceFault"
•   point="xtgt.OperatorReferenceFault"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.RelationalOperatorFault"
•   group="FaultCoverage"
•   id="RelationalOperatorFault"
•   point="xtgt.RelationalOperatorFault"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.StuckAt0"
•   group="FaultExpression"
•   id="StuckAt0"
•   point="xtgt.StuckAt0"/>
• <faultexpression
•   class="extgt.coverage.fault.mutators.StuckAt1"
•   group="FaultExpression"
•   id="StuckAt1"
•   point="xtgt.StuckAt1">
• </faultexpression>
```

Architecture



Validation with SWTBot

- We used SWTBot for validating ExtGT
- It allows the test of RCP applications together with their GUI

```
@RunWith(SWTBotJUnit4ClassRunner.class)
public class OpenFileAnalyzeAndRun {

    private SWTWorkbenchBot bot;

    @Test
    public void canCreateAMessage() throws Exception {
        SWTBotMenu filemenu = bot.menu("File");
        assertNotNull(filemenu);
        SWTBotMenu openm = filemenu.menu("Open");
        assertNotNull(openm);

        ...
    }
}
```

Lesson learned

- Easy to reuse the existing code
 - Small modifications
 - public constructors to allow the use of reflection
 - Embedded Swing into SWT
 - But slow and not native look & feel
- Easy to develop a new plugin to add functionalities by external groups
- Not easy to validate
 - Debugging and testing not easy
- Too many extension points in eclipse itself
 - E.g. a button can be done in **n** ways
- New extension points are continuously added and the documentation becomes soon old
 - E.g. the books become useless
- Use of XML together with Java can be problematic
 - Not clear what is programmable

Future work & Conclusions

- Eclipse RCP suitable to build an extensible tool
 - To foster code reuse
- Add new extension points /extensions
 - Notation: - not only abstract state machines
 - New model checkers (like NuSMV)
 - Test predicate ordering,