universitätbonn ROOTS

Advanced Logic Programming

Dr. Günter Kniesel

Computer Science Department III
University of Bonn, Germany

gk@cs.uni-bonn.de

universitätbonn ROOTS

Chapter 0. Organizational Issues

Website

At http://sewiki.iai.uni-bonn.de/teaching/lectures/alp/2010 you find

- Course slides
  - updated each Monday before the lecture
- Exercise sheets
  - updated each Monday before the lecture
- Tips about tools and infrastructure
  - Eclipse, Subversion, SWI-Prolog, JTransformer, mailing list registration, …
- Schedule (Google calendar)
  - Lectures, exercises, exams
  - No lecture / exercises on May 18, June 1 – 6 (Whitsun) and July 20 (Exam week)
- References
- News

Regular Schedule

- Monday, 13.30-15.00: ALP lecture (room A121)
- Practical exercises (terminal pool A106)
  - Monday
    - 15:15: Group 1, 2
    - 16:15: Group 3, 4
  - Tuesday
    - 16:15: Group 5, 6
    - 17:15: Group 7, 8
- Time budget per week
  - 1 hour reading / learning + 1 hour practical exercises

Exercise groups

- Small groups
  - 1 tutor , 3-4 students, 45 minutes
- Emphasis on practical skills
  - lots of practical exercises
  - electronic result submission via „Subversion“ repository
  - presentation of results and discussion with tutor in front of your computer
- Teamwork is essential
  - you should solve exercises collaboratively
  - you must be able to present any result of the team

From Exercise Groups to Exams

- To be admitted to the exam
  - you must achieve at least 50% of the points for the exercises
  - you may fail to attend at most two of the exercise group meetings
- Explaining ideas is as important as programming
  - Your tutor will ask each group member questions about any exercise
  - You only get points for the solutions that you are able to explain yourself sensibly
- Regularly doing your exercises and actively participating in the group discussions is the best preparation for your exams!

Exams

- Oral exams
  - 30-40 minutes: 1 student, 1 examiner, 1 minute taker.
- Exam slots
  - Thursday, August 3 -5
- Aims of the exam: Verify that
  - you can discuss confidently any topic of the course
  - you can use the right terms the right way
  - you can asses problems and identify opportunities for using LP techniques
  - you can design a sensible logic programming solution of a problem
  - you can write down short bits of correct code.

Learning

Tell me and I'll forget.

Show me and I'll remember.

Let me do it and I'll know.

Confucius

Chinese Philosopher

551 – 479 B.C.

universitätbonn ROOTS

Chapter 1. Motivation: Logic-based Software Analysis and Transformation

Software Evolution

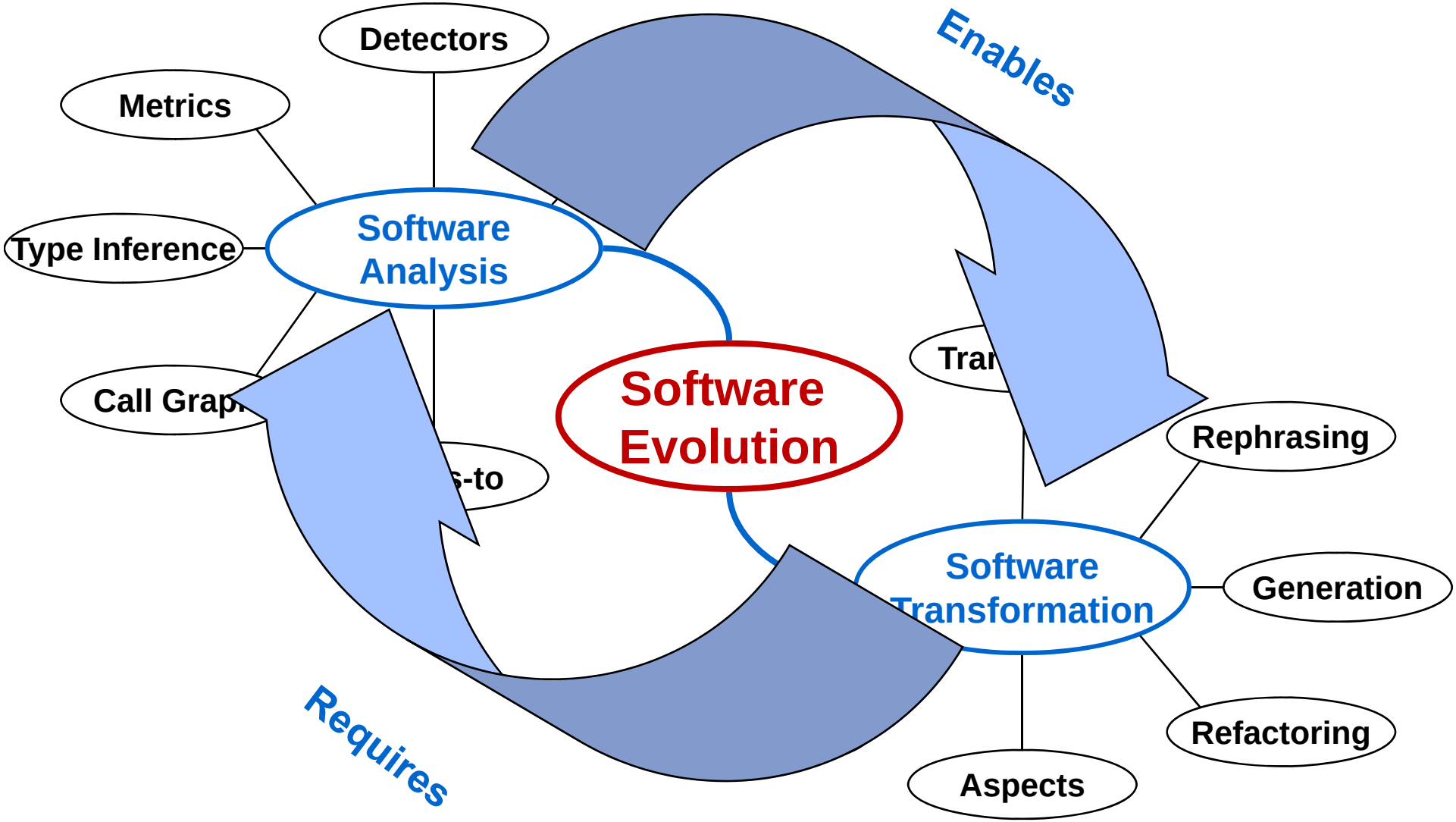
Integrated Environment for Software Analysis and Transformation

Logic-based Software Representation

Logic-based Software Analysis

Logic-based Software Transformation

Software Evolution



Problem

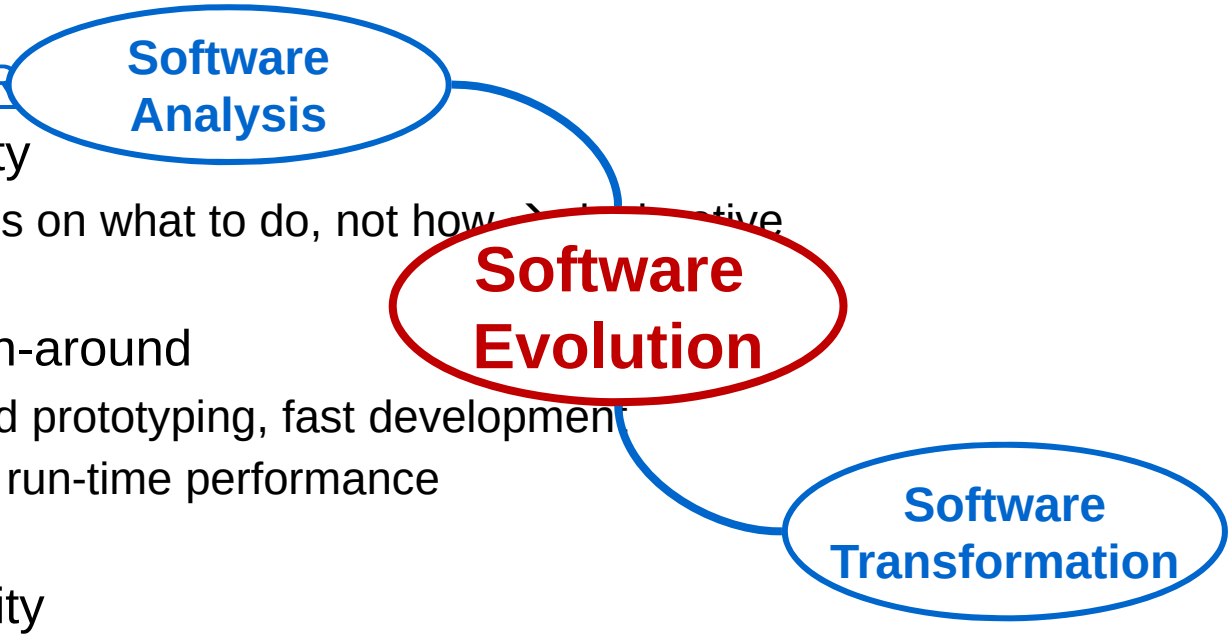
- Many tools – too many
  - Too specialized
  - Too rudimentary
  - Too slow
  - Too expensive
- Difficult to use
  - where to start?
  - learning each tool!!
  - bridging the gaps!!!
- Consequences
  - high costs
  - no wide-spread use

Goal

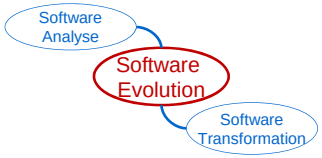
- Integration
  - Free, uniform environment for software analysis and transformation

Additional R

- Simplicity
  - Focus on what to do, not how → declarative
- Fast turn-around
  - Rapid prototyping, fast development
  - High run-time performance
- Scalability
  - Seconds, even on 1.000.000 LOC and beyond
- Usability
  - Smooth integration in development workflows



Overview



Approach

- Logic based Software Artefact Representation
- Logic-based Software Analysis
- Logic based Software Transformation
- JTransformer: Logic-based Analysis and Transformation for Java

Case Studies

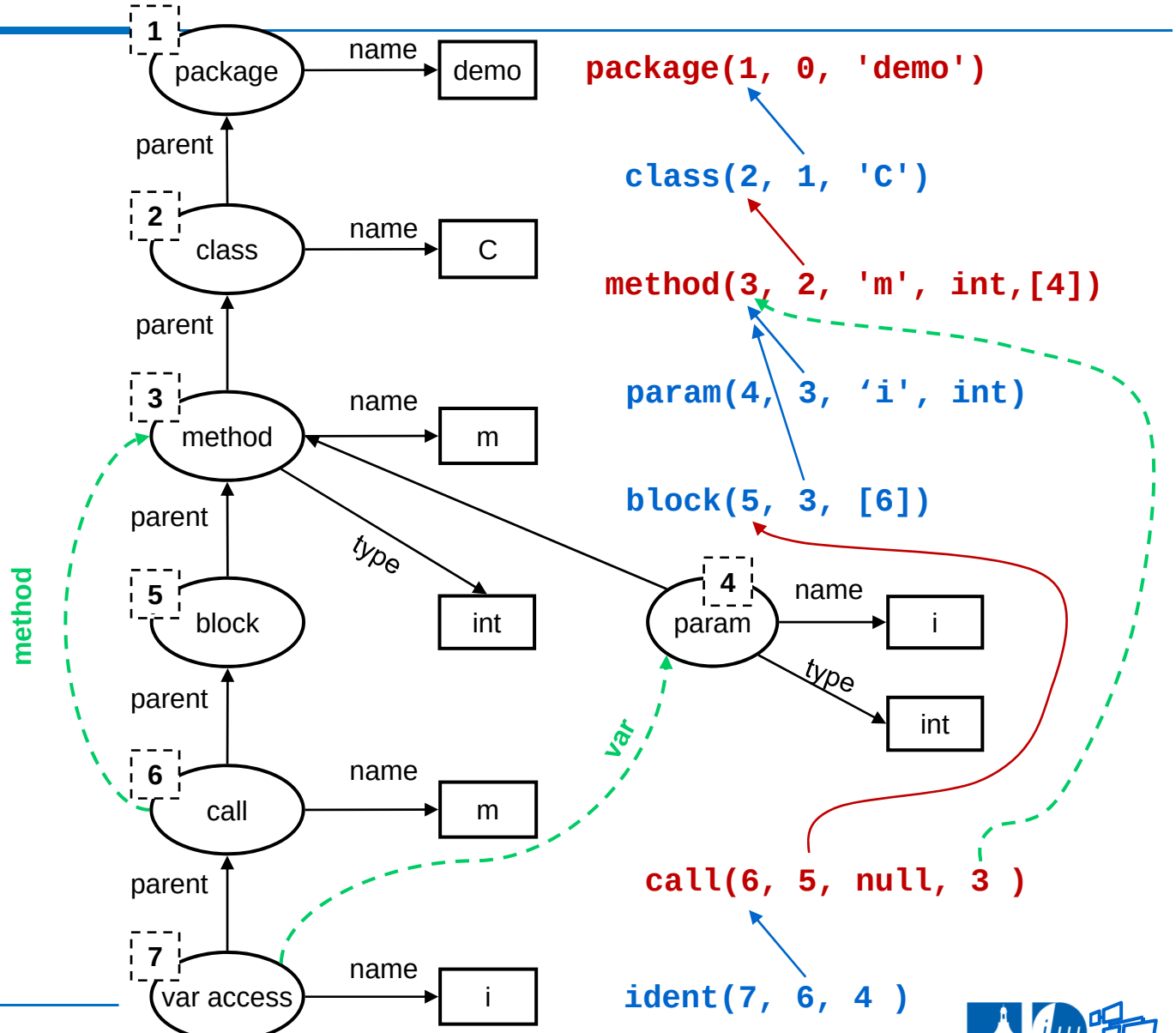
- Design patterns
- Metrics and Smells
- Architecture Analysis
- Performance Analysis
- Smells and Refactorings

universitätbonn ROOTS

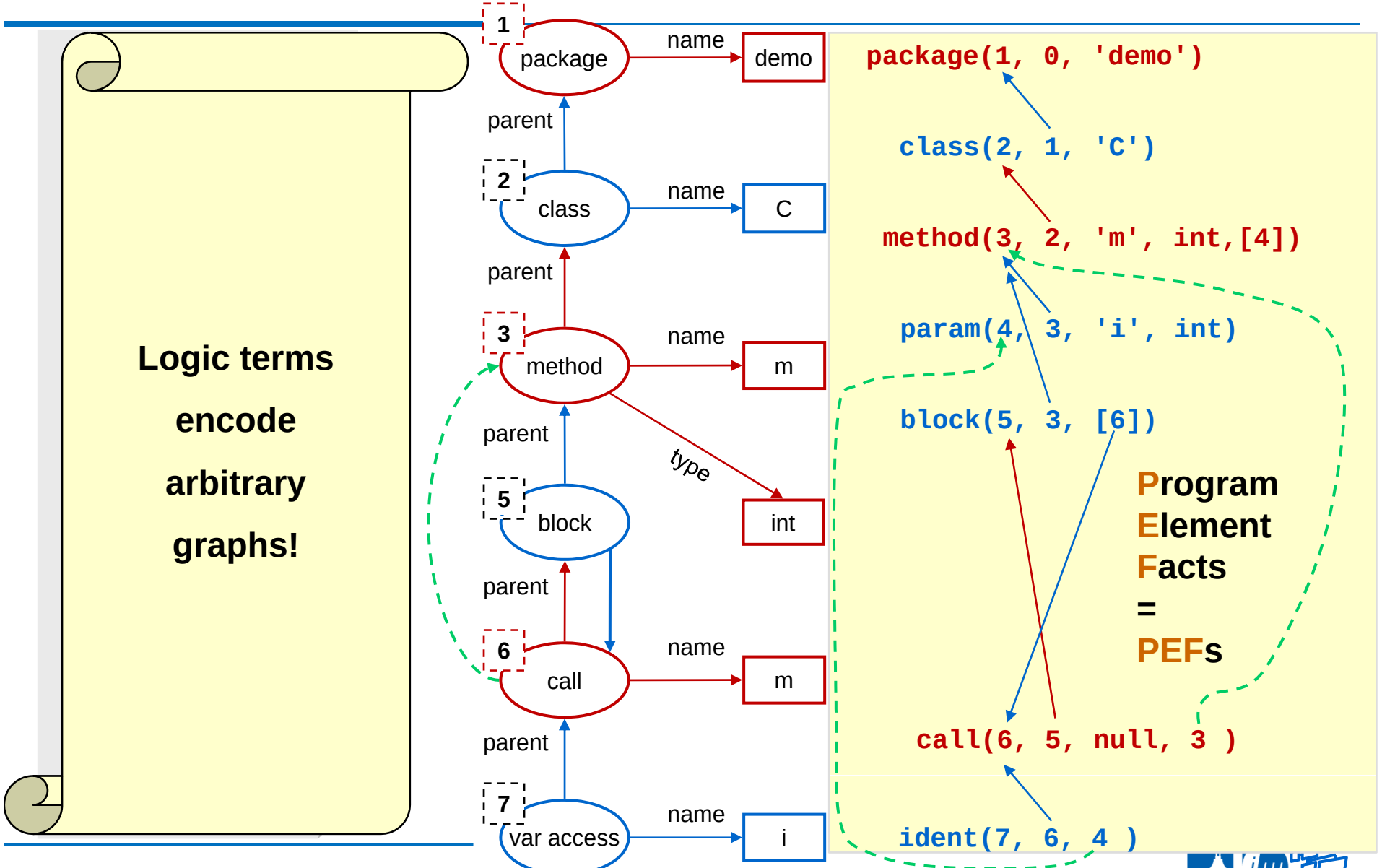
Logic-Based Software Representation

Logic-Based Program Representation

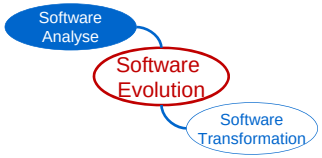
```
package demo;


class C {


   int m(int i) {









      m(i);
   }
}
```



Logic-Based Program Representation



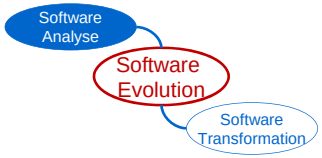
Program Element Facts (PEFs)



- Complete representation of Java 1.4 Abstract Syntax Tree
  - Projects, files and packages
  - Interface elements (types and their members)
  - Code elements (statements and expressions)
  - Comments (javadoc and block comments)

- Representation of Java 1.5 / 1.6 Abstract Syntax Tree
  - Annotations
  - Syntactic suggar (foreach, …)
  - Generics: Work in progress (JTransformer 2.8 ++)
    - ⇨ JT 2.8.0: Programs containing generics can be processed but no PEFs for generic type informations are created

JTransformer



- Eclipse Plug-In
  - Automatic creation of PEFs for Java projects
  - Incremental update of PEFs when source code is changed
  - Program analyses and transformations
    - ⇨ Development environment
    - ⇨ Run-time environment
  - Reverse engineering of Java source from transformed PEFs

- See http://sewiki.iai.uni-bonn.de/research/jtransformer
  - Installation
  - Tutorial
  - PEF Documentation

Learning by Doing

- Install JTransformer
  - See online Installation Guide
- First steps
  - Add „JHotDraw“ project to your workspace (or any project you want)
  - Create factbase for your project
  - Open PEF Documentation
  - Open Prolog Console
- Simple queries
  - Find a class:

```
?- classT(Class, Parent, Name, Members).
```

    - ⇨ Logic variables, backtracking
  - Find a source class:

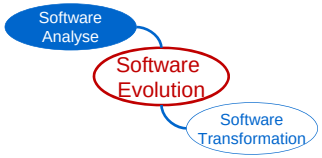
```
?- classT(Class, Parent, Name, Members) , not(externT(Class)).
```

    - ⇨ Comma (',') means conjunction

universitätbonn ROOTS

Logic-Based Software Analysis

Analysis-Examples

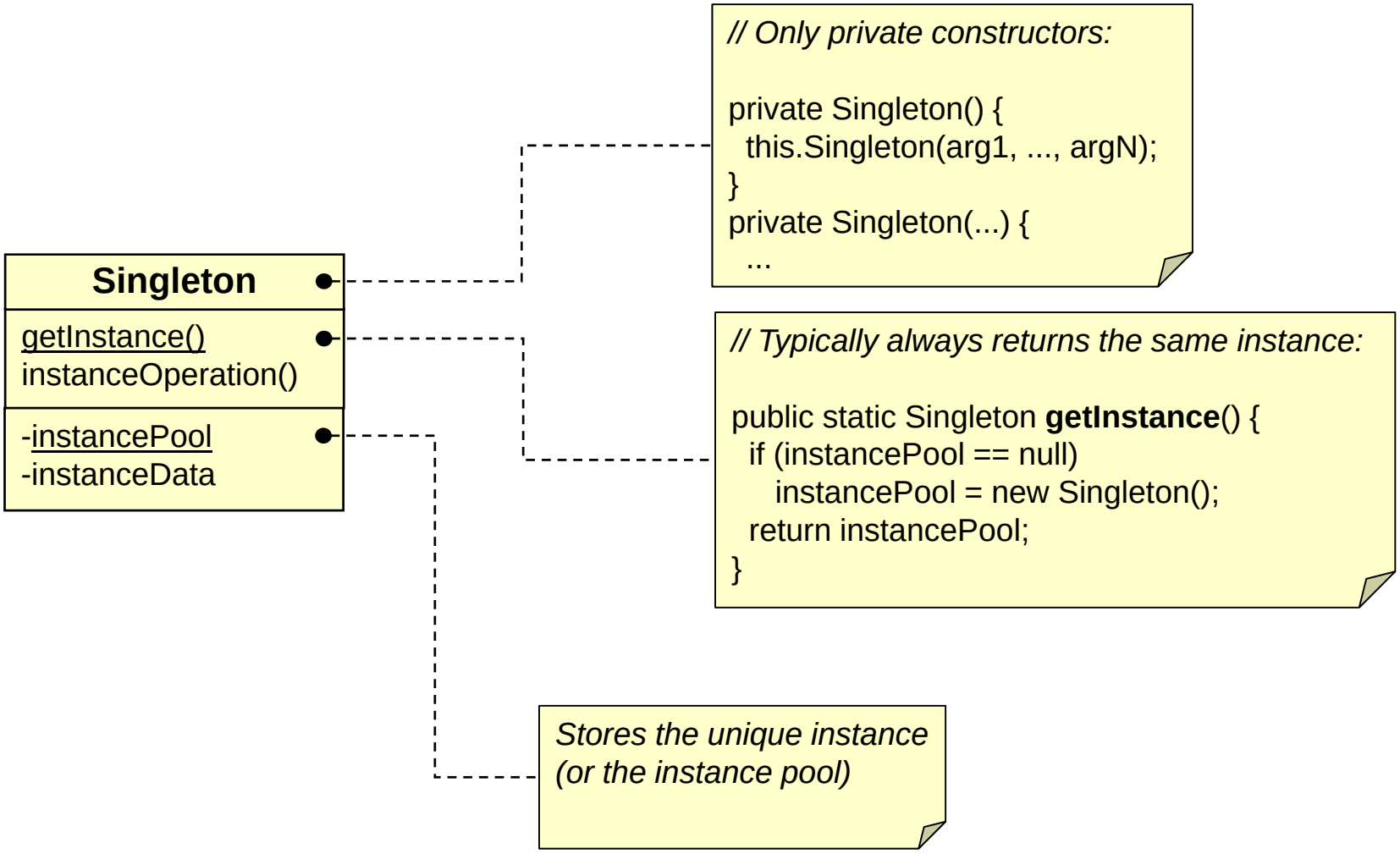


- Metrics
  - Anything you want → „Cultivate Plugin“ (Daniel Speicher)
- Architecture analysis
  - Dependencies, cycles, architectural rule enforcement
- Performance analysis
  - Smelly Database access patterns
- Bad Smells
  - Hints about need for refactorings
- Refactoring Preconditons
  - Type-Constraints, hierarchy structure, ...
- Code comprehension and Mining
  - Design Patterns, Crosscutting Concerns

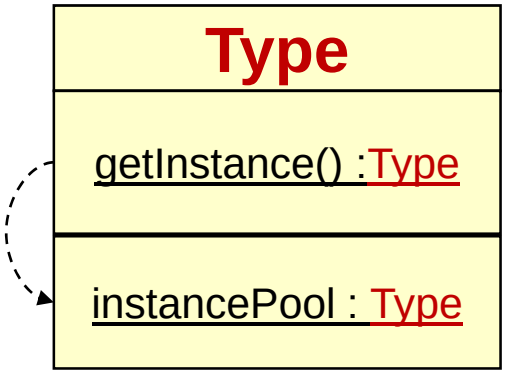
Singleton Pattern - Motivation

- Intention: Limit number of instances of a class
- Typically: just one instance
  - E.g. Facade, Repository, System, Abstract Factory
  - Motivation: Central access point
- Also: fixed number of instances
  - Motivation 1: limited resources.
  - Motivation 2: avoid expensive object creation by „Object Pool“
    → e.g. create 1000 Enterprise Java Beans, use when needed, put each back into pool after use

Singleton: Structure + Implementation



Logic-based Program Analysis Example: Pattern Mining



The Singleton Pattern

A static method in *Type* with 0 arguments
returns an instance of *Type*
by accessing a static field in *Type* that has type *Type*!

```
classMethodReturnsOwnInstance(Type, Method, Field) :-

   methodT(Method, Type, _, [], type(_, Type, 0), _, _),
   modifierT(Method, static),

   fieldT(Field, Type, type(_, Type, 0),_,_),
   modifierT(Field, static),

   getFieldT(_, _, Method, _, _, Field).
```

Logic-based Program Analysis Example: Pattern Mining

- Query ?- classMethodReturnsOwnInstance(Type, Method, Field).
  - Returns tuples of values for <Type, Method, Field> that represent singletons.
  - Generates all results via backtracking.

```
classMethodReturnsOwnInstance(Type, Method, Field) :-

   methodT(Method, Type, _, [], type(_, Type, 0), _, _),
   modifierT(Method, static),

   fieldT(Field, Type, type(_, Type, 0),_,_),
   modifierT(Field, static),

   getFieldT(_, _, Method, _, _, Field).
```

Learning by Doing

- Open JT-Tutorial/patterns/singleton.pl
- Run it
  - ?- `classMethodReturnsOwnInstance(Type, Method, Field).`
- Inspect the results using the multi-way linking feature
- Multi-way linking (Query – Source – Factbase)
  - Link console → editor
  - Link console → factbase
  - Link factbase → editor
  - Link editor → factbase

Learning by Doing

- Open JT-Tutorial/patterns/singleton.pl
  - „consult“ the file (menu item or F9)
- Run the Query
  - ?- `classMethodReturnsOwnInstance(Type, Method, Field).`
- Inspect the results using the multi-way linking feature (Query – Source – Factbase)
  - Link console → editor
  - Link console → factbase
  - Link factbase → editor
  - Link editor → factbase

Demo

```
120     */
121    protected JPanel createAttributesPanel() {
122        JPanel panel = new JPanel();
123        panel.setLayout(new PaletteLayout(2, new Point(2,2), false));
124        return panel;
125    }
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140        panel.add(new JLabel("Pen"));
141        fFrameColor = createColorChoice("F
142        panel.add(fFrameColor);
143
```

JTransformer - Src for id 52465

```
protected javax.swing.JPanel createAttributesPanel() {
  javax.swing.JPanel panel = new javax.swing.JPanel();
  panel.setLayout(new CH.ifa.draw.util.PaletteLayout(2, new java.awt.Point(2,
2), false));
  return panel;
}
```

OK

Double klick on PEF in the Factbase Inspector shows reverse engineered source code

Context menu shows Java source in editor or internal representation in Factbase Inspector (FBI)

Prolog Console

```
JHotDraw
count( classDefT(_,_,_,_), AllClasses).

AllClasses = 1350 ;

No
28 ?- classDefT(Id,Pkg,'DrawApplet',Members).

Id = 31081
Pkg = 31073
Members = [31078, 52445, 52446, 52447, 52448, 52449, 52450, 52451, 52452, 52453, 52454, 52455,
52456, 52457, 52458, 52459, 52460, 52461, 31114, 52462, 52463, 52464, 52465, 52466, 52467, 52468,
52469, 32826, 52470, 31097, 52471, 31111, 52472, 52473, 52474, 52475, 32861, 52476, 32876, 52477,
52478,

No
29 ?-
```

Query factbase or run CTs

PEF Navigator

```
methodDefT(52465, 31081, createAttributesPanel, [], type(class, 15313, 0)
  BODY: blockT(52574, 52465, 52465, [52575, 52576, 52577])
    ENCL: methodDefT(52465, 31081, createAttributesPanel, [], type(c
    PARENT: methodDefT(52465, 31081, createAttributesPanel, [], typ
    stmts
      execT(52576, 52574, 52465, 52581)
      localDefT(52575, 52574, 52465, type(class, 15313, 0), panel, 5
      returnT(52577, 52574, 52465, 52593)
  PARENT: classDefT(31081, 31073, 'DrawApplet', [31078, 52445, 52446,
methodDefT(52466, 31081, createAttributeChoices, [52594], type(basic, v
methodDefT(52467, 31081, createColorChoice, [52736], type(class, 15624,
```

Screenshot and explanation of behaviour TO BE UPDATED

Metrics: Depth of Inheritance (DOI)

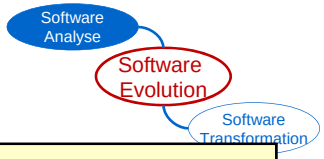
Find out the inheritance depth of any class.

Learn to write recursive predicates.

```
metric_doi(Class, 0) :-
   classT(Class, _, _, _),
   not(extendsT(Class,_Super)).

metric_doi(Class, DOI) :-
   classT(Class, _, _, _),
   extendsT(Class, Super),
   metric_doi(Super, DOISuper),    // recursive call
   DOI is DOISuper + 1.            // evaluation of arithm. expr.
```

- Predicate defined by multiple clauses → Disjunction
- Recursion in the second clause.

Metrics: Depth of Inheritance



Write a smell detector that uses the previous metric.

Should only report values above a certain limit for source classes.

```
smell_doi(Limit, DOI, FQN):-
   metric_doi(Class,DOI),          % use the metric
   not(externT(Class))             % for source classes only
   DOI >= Limit,                   % metric has critical value
   fullQualifiedName(Class,FQN).   % get full name of class (JT)


?- smell_doi(5, DOI, FQN).         % Find classes at depth >= 5.
```

- Try it out on JHotDraw.

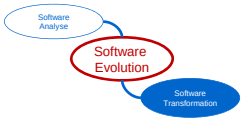
Bad Smells: Indicators of Refactoring needs

```
non_private_field(Class,Field,FieldType,FieldName,Modif) :-
    fieldT(Field,Class,FieldType,FieldName,_),
    modifierT(Field,Modif),
    ( Modif = public
    ; Modif = package
    ; Modif = protected
    ).


field_without_getter(Field,Class,Type,Name,Getter) :-
    non_private_field(Class,Field,Type,Name,_Modif),
    concat(get, Name, Getter),
    % No method with signature "Type Getter()" :
    not( methodT(_Meth,Class,Getter,[],Type,_,_) ).
```

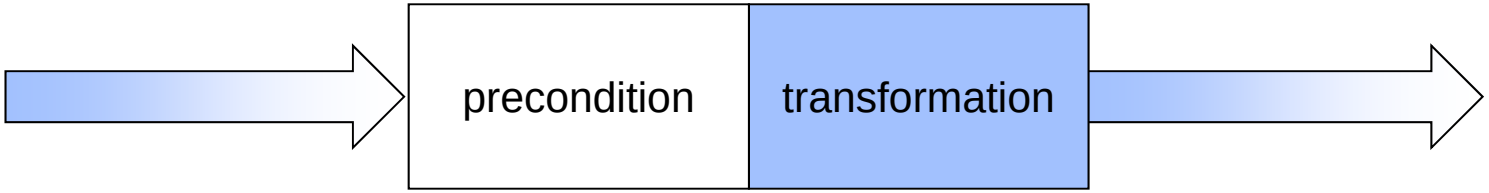
- Suppose, we find 231 non-encapsulated fields!
- Would you bother to encapsulate them one by one?

Analysis and Transformation

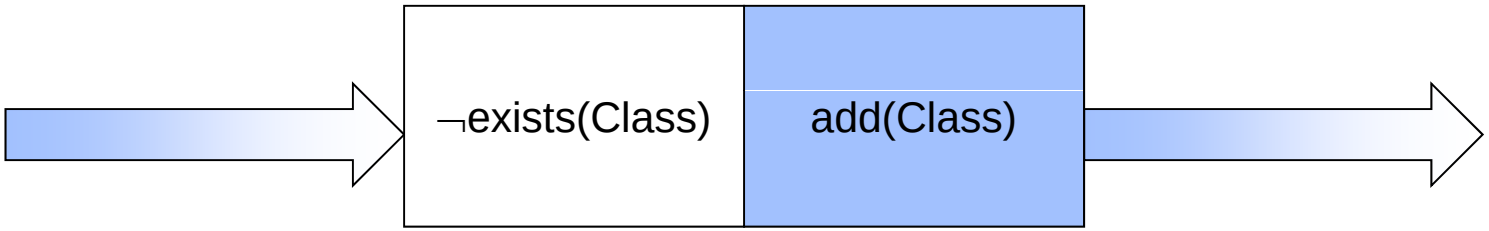
Conditional Transformations (CTs)



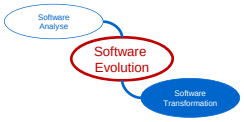
- CT = Condition + Transformation
  - Condition is true ⇒ Transformation may be executed



- Example: „Add Class" Transformation
  - Condition: Class does not exist

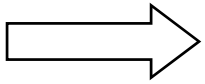


Example: Create Accessor Method



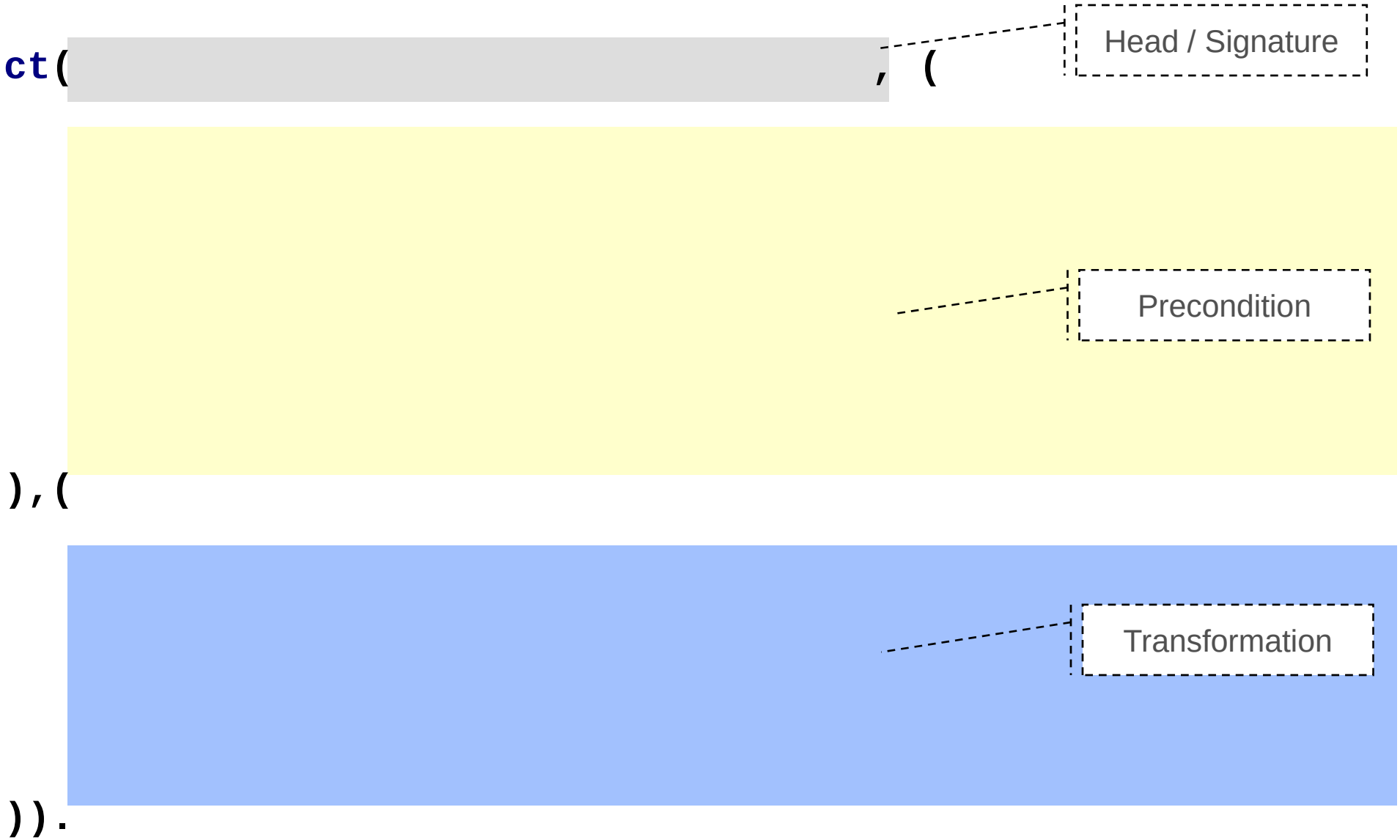
- „AddGetter“ CT
  - for all fields that have no getter method ...
  - ... add method that returns the field's value

```
public class C {

  B b = new B();




  ...

}
```

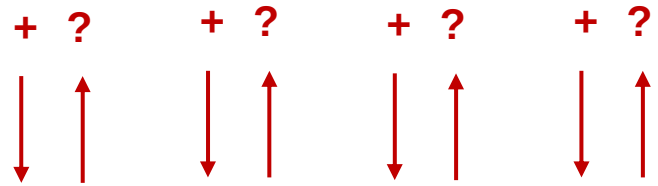


```
public class C {

  B b = new B();

  B getB() {
     return b;
  }

   ...

}
```

The Structure of a CT



The CT



No method with signature "<Type> get<Name>()" exists.

```
ct( addGetter(Class,Field,Type,GName), (
    classT(Class,_,_,_),
    fieldT(Field,Class,Type,Name,_),

    concat(get, Name, GName),
    not( methodT(Getter,Class, GName,[],Type,_,_),
         getFieldT(_, _, Getter, _, _, Field) ),
    new_id(Method),...,new_id(Get)
),(
    add( methodT(Method,Class, GName,[],Type,[],Block) ),
    add( blockT(Block,Method,Method,[Return]) ),
    add( returnT(Return,Block,Method,Get) ),
    add( getFieldT(Get,Return,Method,null,Name,Field) ),
    add_to_class(Class,Method)
)).
```

Create new identities for new elements:

Create method "<Type> get<Name>() { return <Field>}":

Summary

- Prolog is a
  - quick and easy
  - efficient (if you know how)
  - an fun

  way to solve many hard problems in many domains

- We shall apply it in the domain of software analysis and transformation
  - “Think different!” … about model driven software engineering (MDSE)

- Mastering Prolog also means opportunities for diploma / master theses in the software engineering group (“ROOTS”)
  - … starting in the winter semester or later

Course Preview

1. Syntax and operational semantics
2. Declarative semantics, (in)completeness, correctness
3. Programming idioms
4. Beyond declarative programming
    1. I/O
    2. Side-effects
    3. State representation
    4. …
5. Metaprogramming
6. Partial Evaluation
7. Abstract Interpretation
8. Conditional Transformations
9. Applications