Logic-based
Software Analysis and Transformation

Dr. Günter Kniesel
gk@cs.uni-bonn.de

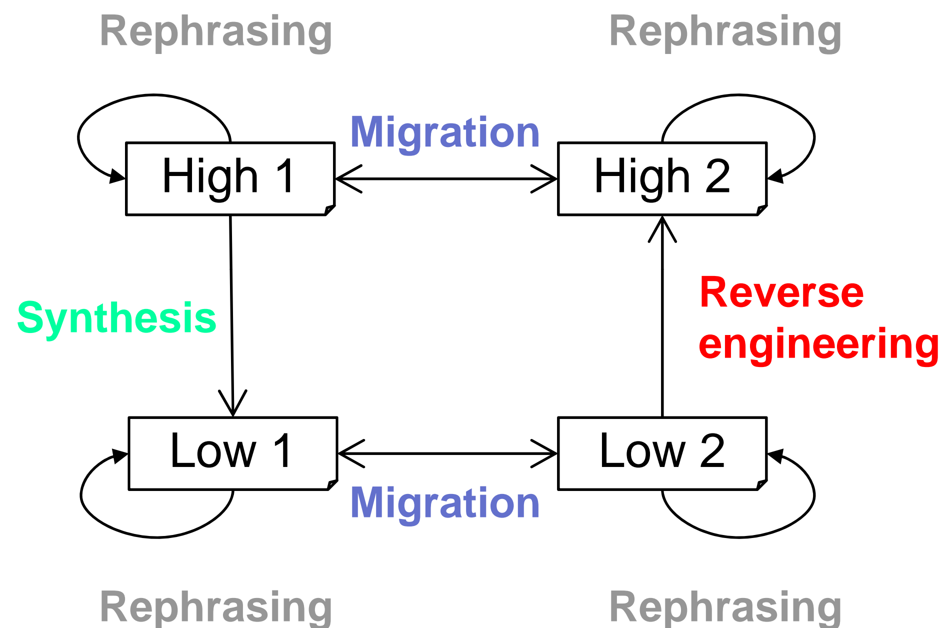
ROOTS Group
Computer Science Department III
University of Bonn

Software Transformation

- Software Transformation
  - Modification (or sequence of modifications) of software artefacts.

Examples

- Program Synthesis
  - derive implementation from specification
- Reverse engineering
  - Extract higher abstraction from lower level artefacts
- Migration
  - translate to other language at same level of abstraction
- Rephrasing
  - transform to same language



Transformation Categories

- Translation:
  Transform program in language A to program in language B
  - Migration
  - Synthesis
  - Reverse engineering
- Rephrasing:
  Transform program in language A to program in the same language
  - Normalization
  - Optimization
  - Refactoring
  - Renovation
  - Enhancement

behaviour-preserving

Software Analysis

- Software Analysis
  - Derivation of implicit information from software artefacts.

Examples

- Control-flow analysis → A calls B
- Data-flow analysis → A is assigned from B
- "Bad smell" detection → "Long method"
- Design Pattern detection → "Pull variant of Observer"
- Concern identification → "Logging"
- ...

Software Analysis and Transformation

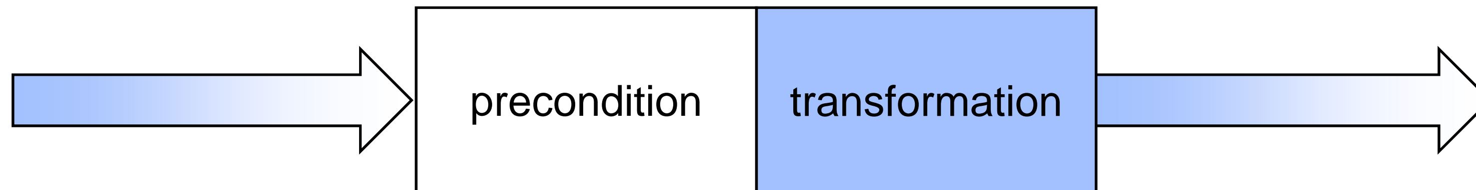
- Transformation needs prior analysis!

- Analysis determines
  - whether a transformation is necessary
    - ➔ bad small detection for refactorings
  - whether a transformation is legal
    - ➔ check behaviour preservation of refactorings
  - the program elements to be transformed
    - ➔ method with long parameter list
  - context information necessary for the refactoring
    - ➔ methods of parent class for which to create forwarding methods in a decorator

➔ Combine transformation and analysis!

Conditional Transformations (CTs)

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



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

Overview

- Logic based Software Artefact Representation
- Logic-based Software Analysis

- Introduction of JTransformer
- Analysis Example: Design Pattern Detection

- Logic based Conditional Transformations (CTs)

- Transformation example: Build your own Refactoring

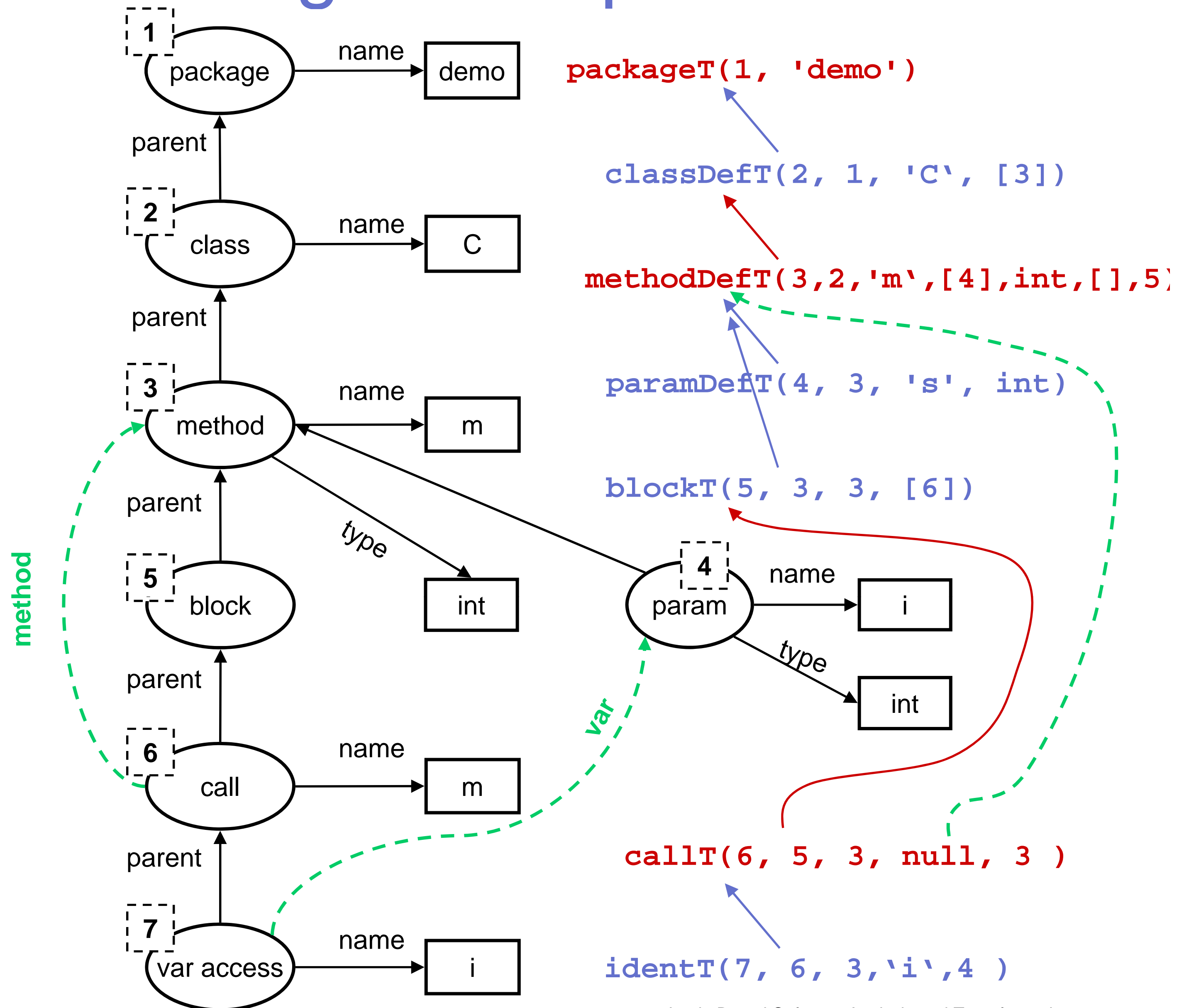
- CTC: A Language-Parametric Transformation System
- How to define own languages

- Exercises (implement yourself)

Logic-Based Software Representation

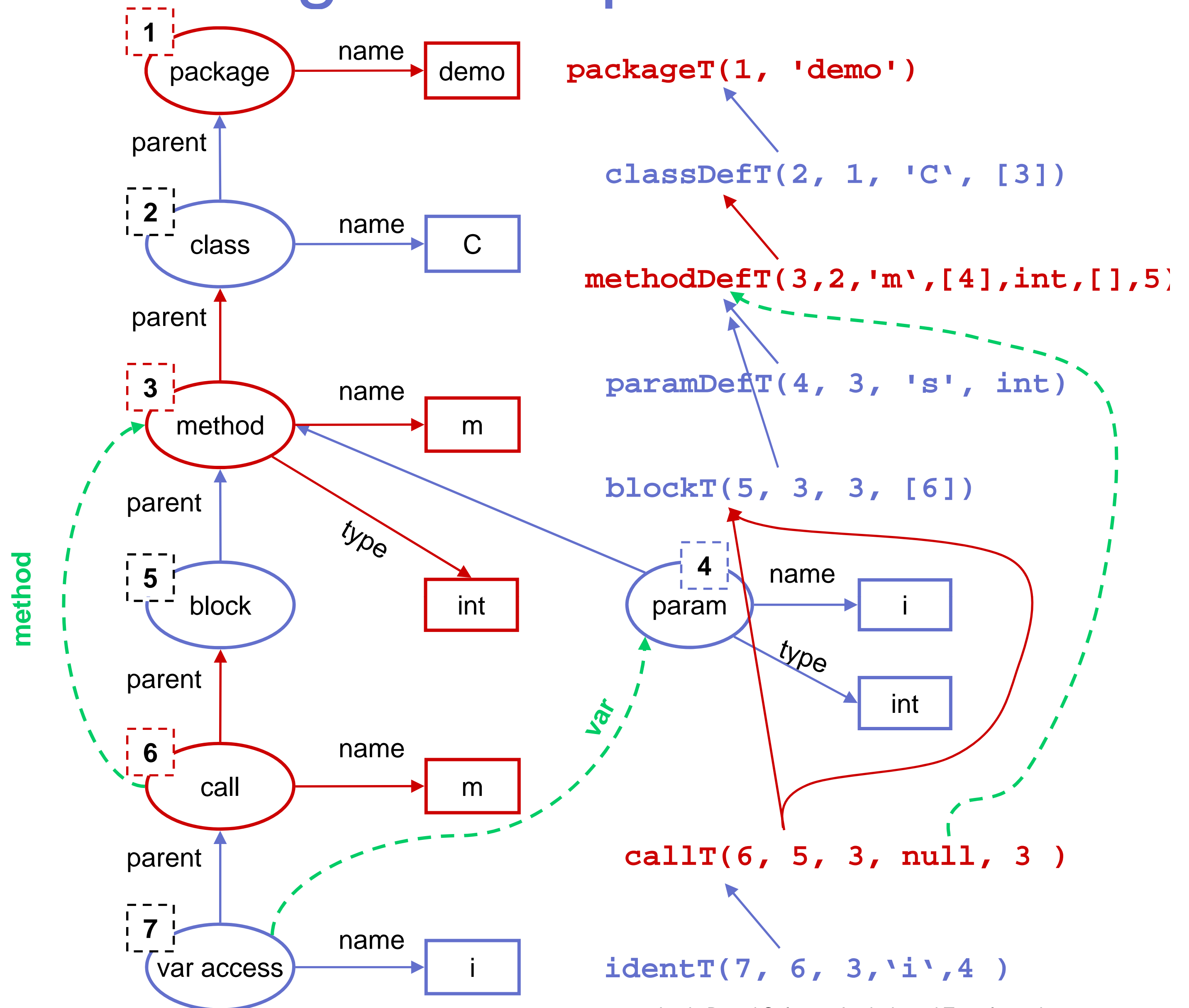
Logic-Based Program Representation

```
package demo;

class C {

    int m(int i) {

        m(i);
    }
}
```



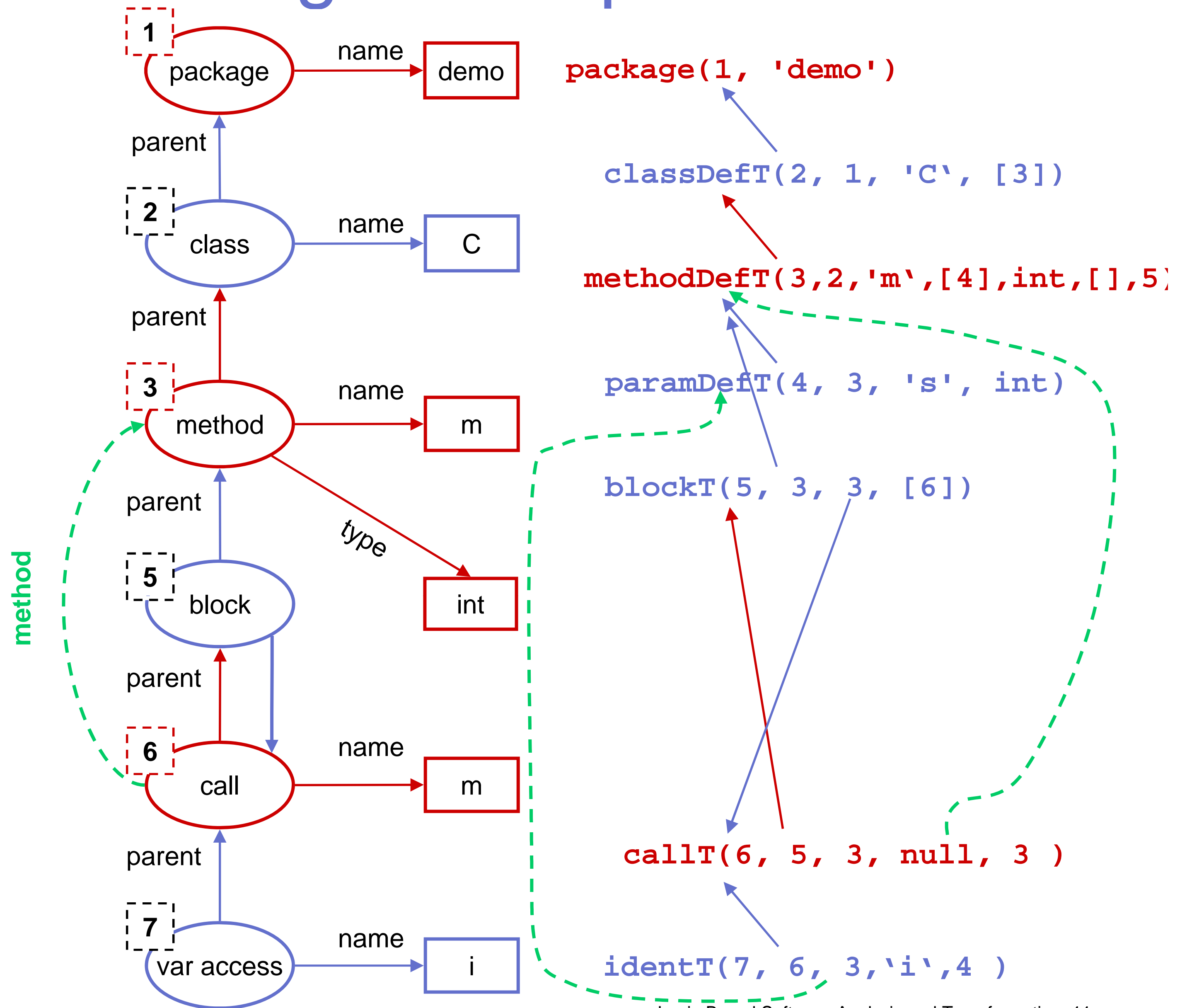
Logic-Based Program Representation

```
package demo;

class C {

    int m(int i) {

        m(i);

    }
}
```



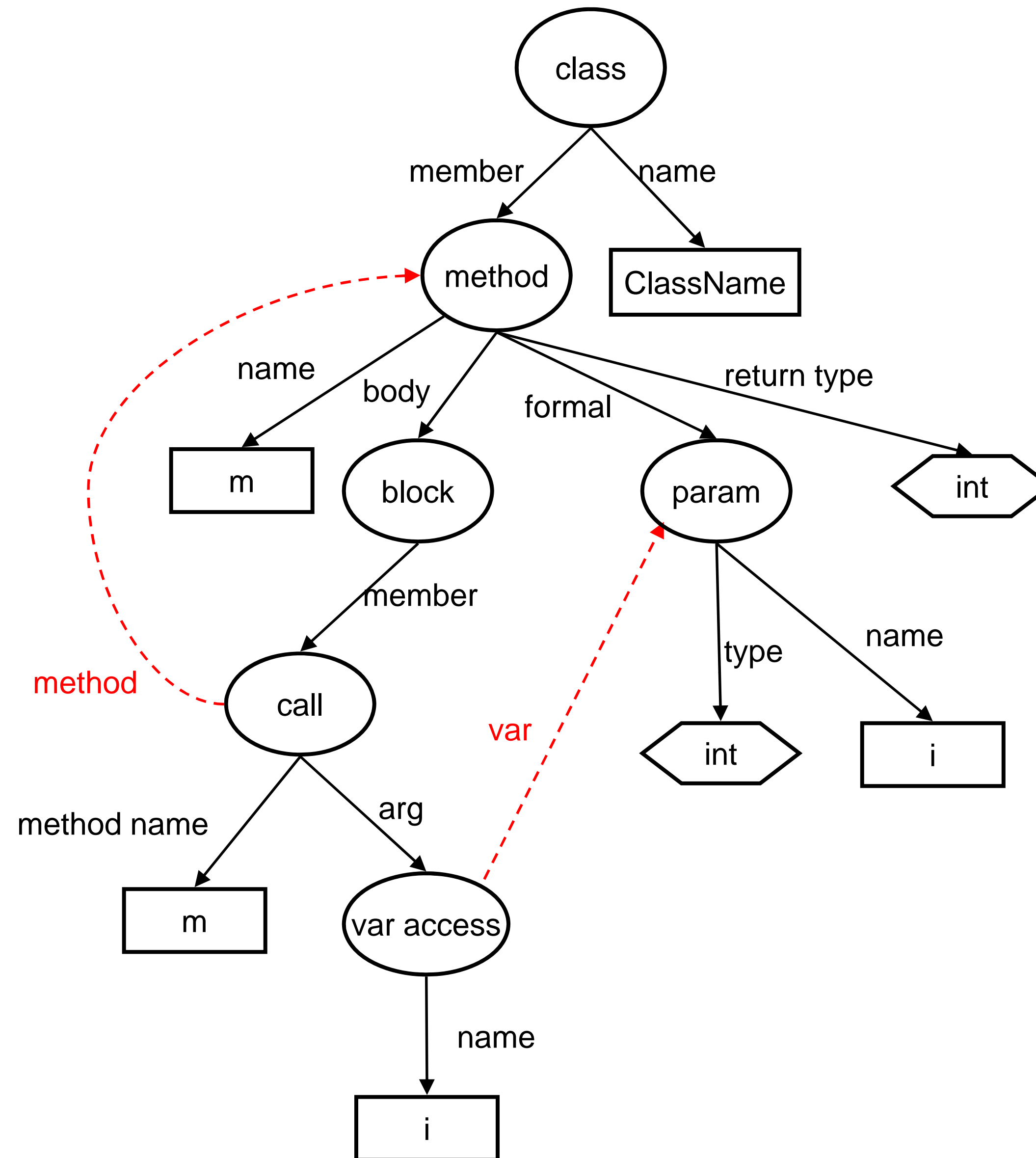
Logic-Based Program Representation

```
package demo;

class C {

    int m(int i) {

        m(i);
    }
}
```

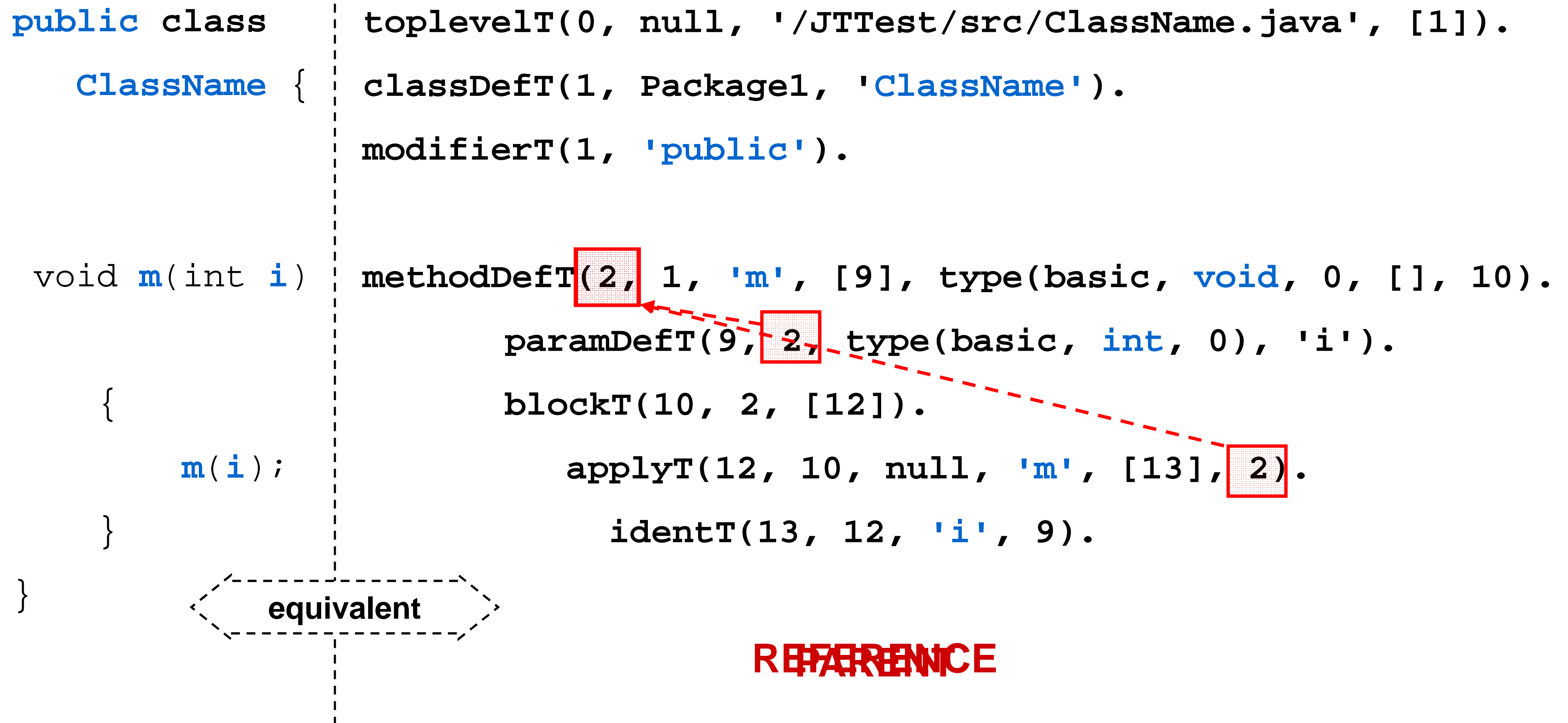


Generalised Abstract Syntax Tree

```
class ClassName {

 int m(int i) {

        m(i);
    }
}
```



Logic Fact Representation of AST



Logic-based Program Analysis

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

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

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

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

A static method in Type
returns an instance of Type
by accessing a static field that has type Type

→ Singleton Pattern

Program Element Facts (PEFs) for Java

- Complete representation of Java 1.4 Abstract Syntax Tree
  - Files and packages
  - Interface elements (types and their members)
  - Code elements (Statements and expressions)
- See http://roots.iai.uni-bonn.de/research/jtransformer

- JTransformer
  - Eclipse Plug-In
  - Automatic creation and incremental update of PEFs for Java project
  - Development environment for program analyses and transformations
  - Reverse engineering of Java code from PEF representation

JTransformer Tutorial

– Program Representation and Analysis –

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
...
140      panel.add(new JLabel("Pen"));
141      fFrameColor = createColorChoice("FrameColor");
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 shows reverse engineered source code

Context menu shows Java source in editor or internal representation in PEF navigator

Query factbase or run CTs

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, 52479, 52480, 52481, 52482, 52483, 52484, 52485, 52486|...] ;

No
29 ?-
```

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, vo
methodDefT(52467, 31081, createColorChoice, [52736], type(class, 15624,
methodDefT(52468, 31081, createFontChoice, [], type(class, 15624, 0), [],
methodDefT(52469, 31081, createButtonPanel, [], type(class, 15313, 0), []
methodDefT(52470, 31081, createToolPalette, [], type(class, 15313, 0), [],
methodDefT(52471, 31081, createSelectionTool, [], type(class, 10454, 0), [
methodDefT(52472, 31081, createDrawing, [], type(class, 10740, 0), [], 53
methodDefT(52473, 31081, createDrawingView, [], type(class, 10612, 0), [
```

Conditional Transformations

Conditional Transformations with JT

Fixed set of Program Element Facts for Java 1.4:

PEF $\in$ { classDefT(Id,Pkg,Name,Members),
fieldDefT(Id,Class, Name,Init), ...,
applyT(...), ... }

Condition Language

$C \rightarrow EC \wedge C \mid C \vee C \mid \text{not}(C)$
$EC \rightarrow \text{true} \mid \text{false} \mid \text{newId(Var)} \mid pef \in PEF$

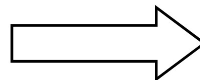
Transformation Language

$T \rightarrow ET \mid ET\ ,\ T$
$ET \rightarrow \text{add}(pef) \mid \text{delete}(pef) \mid \text{replace}(pef_1,pef_2)\ :\ pef \in PEF$

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;
  }

   ...

}
```

AddGetter CT

```
ct(addGetter(Class,Field,Type,Getter), (
    classDefT(Class, , , ),
    not(externT(Class)),
    fieldDefT(Field,Class,Type,Name,_),
    % No method with signature "<Type> get<Name>()" :
    concat(get, Name, Getter),
    not( methodDefT(Method,Class,Getter,[],Type,_,_) ),
    % Identities of elements to be created:
    new_id(Method),...,new_id(Get)
),(
    % Create Method "<Type> get<Name>() { return <Field>}":
    add( methodDefT(Method,Class,Getter,[],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)
)).
```

CT are Parametric

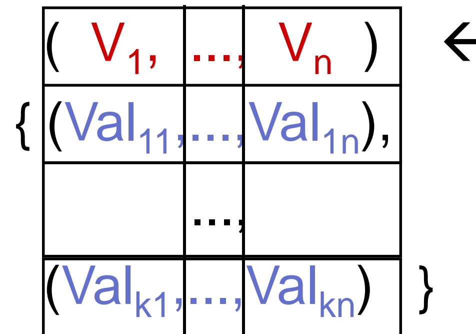
- $CT(V_1,...,V_n) = condition(V_1,...,V_n) \rightarrow transformation(V_1,...,V_n)$

- Variables describe program elements
  - to be changed
  - needed to determine that the change is legal
  - needed to express context-dependent transformations

- Application of a CT to a program
  - $\forall$ substitutions that make the precondition true:
    $\Rightarrow$ perform the transformations subject to these substitutions

CT are Parametric

- Application of a CT to a program
  - ∀ substitutions that make the precondition true:
    ⇒ perform the transformations subject to these substitutions

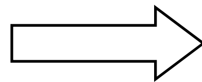


- $CT(V_1,\ldots,V_n)$ = $condition(V_1,\ldots,V_n) \rightarrow transformation(V_1,\ldots,V_n)$

Replace Read Accesses

- ReplaceReadAccesses
  - for all fields that have a getter method and for all read accesses to the field outside of its getter method...
  - ... replace the read access by the getter invocation

```
public class C {

  B b = new B();

  B getB() {...}

  foo() {
      ...
      b.m();
  }
}
```



```
public class C {

  B b = new B();

  B getB() {...}

  foo() {
      ...
      getB().m();
  }
}
```

Full „Encapsulate Field“ Refactoring

- Five CT definitions
  - AddGetter
  - AddSetter
  - ReplaceReadAccesses
  - ReplaceWriteAccesses
  - MakeFieldPrivate

- A CT sequence definition
  - invoke all of the above in the specified order

- Syntax of CT invocation in JTransformer
  - apply_ct(Head): invoke a CT
  - apply_ctlist([Head1, ..., HeadN]): invoke a sequence of CTs

Example: Refactorings as CTs

→ Demo

See „encapsulateField.pl“ file provided for the exercise

at

Language-Parametric System

Is it possible to do everything for arbitrary languages...

... with the same system?

CTC: Language-Independent CTs

Alphabet of Program Element Terms (choose your own)

Σ = { classDefT(Id,Pkg,Name,Members), fieldDefT(Id,Class,Type,Name,Init),
methodDefT(Id,Class,Name,...), getFieldT(Id,...,Meth,Field) }

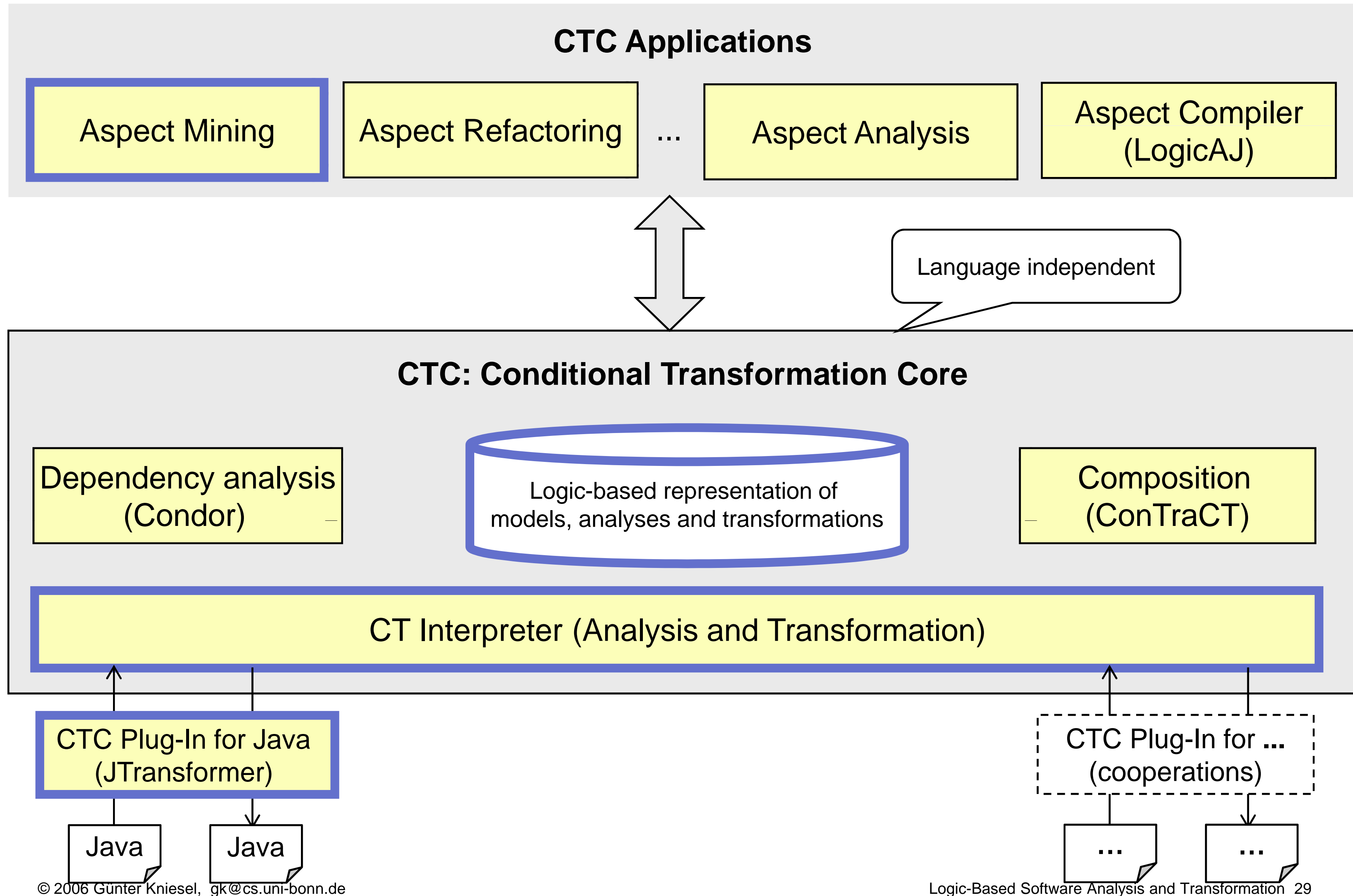
Condition Language

$C \rightarrow EC \wedge C \mid C \vee C \mid \text{not}(C)$
$EC \rightarrow \text{true} \mid \text{false} \mid \text{exists}(\text{elem}) : \text{elem} \in \Sigma$

Transformation Language

$\text{T} \rightarrow \text{ET} \mid \text{ET} , \text{T}$
$\text{ET} \rightarrow \text{add}(\text{elem}) \mid \text{delete}(\text{elem}) \mid \text{replace}(\text{elem}_1,\text{elem}_2) : \text{elem} \in \Sigma$
| new_node_id(Var)

The Big Picture: Conditional Transformations



Defining Your Own Language

- 1. Define Program Element Terms
  - Define the AST of your language
- 2. Specify Program Element Term structure in a standard form
  - Use ast_node_def(Language, AST_Node_Type, Argument_Descriptions)
- 3. Implement a "Reader"
  - A parser for your language that generates Prolog facts corresponding to the PET structure specified in 2.
- 4. Implement a "Writer"
  - A generator that converts PETs into source code.
- 5. (Optional): Possibly provide a set of predefined conditions and CTs for working on Programs in your new language

- 2 - 5 are the elements of a plugin for the CTC that enables it to work with your language.

1. Defining Program Element Terms

- Syntax → Program Element Terms
  - Non-terminals are good candidates for PET types
- Determine attributes of PET types
  - Names, ...
- Determine relations between elements
  - e.g. extends(Sub,Super)
- Avoid mutual references
- Consider good database schema design rules
  - functional dependencies
  - normal forms (4NF)

2. Specification of PET structure

- PET structure: fieldDefT(id#, parent#, type, name, expr#)
- Specified by:

language PET type (= AST node type)

```
ast_node_def('Java',fieldDefT,[
    ast_arg( id,        mult(1,1,no ),  id,    [fieldDefT]   ),
    ast_arg( parent,    mult(1,1,no ),  id,    [classDefT]   ),
    ast_arg( type,      mult(1,1,no ),  attr,  [typeTerm]    ),
    ast_arg( name,      mult(1,1,no ),  attr,  [atom]        ),
    ast_arg( expr,      mult(0,1,no ),  id,    [...]  ),
    ast_arg( modifier,  mult(0,*,no ),  attr,  [atom]        )
]).
```

specification of PET arguments

argument name

multiplicity

order
(no = not ordered,
ord = ordered)

kind of value
(id = identity,
attr = primitve)

legal syntactic type(s) of argument values
(multiple types are allowed,
e.g. an expr, can have many)

3. Reader & 4. Writer

- Reader
  - Implement a parser
  - Possibly use an existing one and just implement a visitor on its internal AST that creates appropriate Prolog facts

- Writer
  - Can be implemented in Prolog, immediately creating source in your language
    - ➔ look at file 'java_writer.pl' for an example how this was done for Java
  - Alternatively you can write out the factbase to a Prolog text file
    - ➔ using the call 'writeTreeFacts(FileToWhichToWrite)'
  - ... and then use any tool you like to convert the generated text to your language syntax

State of Affairs

Done

- jTransformer: CT-based transformation engine for Java
- LogicAJ: Translation of a generic aspect language to CTs
- Condor: CT-based dependency analysis
- CT-Core (CTC): Core for language-parametric system

Ongoing

- *Transformer: language-parametric transformation tool
  - CTC with language-independent GUI tool à la JTransformer
- Integrate the subsystems
- Demonstrate their expressiveness on applications

http://roots.iai.uni-bonn.de/research