13. Languages for Transformations and Rewriting

1

1) Datentransformation (DTL)
   1) Xcerpt
2) Graph rewriting

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de

Version 13-0.4, 02.01.14

In WS 13/14 for self study

Obligatorische Literatur

2

- http://en.wikipedia.org/wiki/List_of_UML_tools
- http://en.wikipedia.org/wiki/Entity-relationship_model
- http://www.utexas.edu/its/archive/windows/database/datamodeling/index.html
- Sebastian Schaffert, François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt (2004). In Proc. Extreme Markup Languages.
  - http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-7.pdf

http://www.rewerse.net/publications/download/REWERSE-RP-2006-069.pdf

- A Comparison of ATL and Story-Driven Modeling (Fujaba-style GRS)

http://www.es.tu-darmstadt.de/fileadmin/download/publications/spatzina/PP_AGTIVE_2011.pdf

An Old Citation

3

There clearly remains more work to be done in the following areas:

- discovery of other properties of transformations that appear to have relevance to code optimization,
- development of simple tests of these properties, and
- the use of these properties to construct efficient and effective optimization algorithms that apply the transformations involved.

Aho, Sethi, Ullmann in Code Optimization and Finite Church-Rosser Systems, 1972

13.1 Data Transformation Languages (DTL)

4

Text, XML, Term, and Graph Rewriting

DTL und DML

- Mit DML (Datenmanipulationssprachen) formt man Daten um.
- **Deklarative DML (Datentransformationssprachen, DTL)** bestehen aus Regeln, die ein Repositorium ohne die Spezifikation weiteren Steuerflusses transformieren.
  - Termersetzungsregeln, die Bäume oder Dags transformieren (Kap. 35)
  - Graphersetzungsregeln, die Graphen und Modelle transformieren (Kap. 36)
- **Imperative DML (allgemeine DML)** kennen Zustände und Seiteneffekte.
- Beispiele von deklarativen DML (DTL):
  - Xquery
  - Xcerpt als Strom-Manipulationssprache
  - XGRS und Fujaba als Graphtransformationssprachen (eigene Kapitel)

Termersetzungssysteme

- Ersetzungssysteme erlauben die Spezifikation von **transformativer Semantik (reduktiver Semantik)**
  - Sie arbeiten regelbasiert, deklarativ auf einem Heap
- **Term rewrite systems (Termersetzungssysteme)** transformieren baumartige Datenstrukturen und operieren daher meist auf dem abstrakten Syntaxbaum (AST)
  - If pattern is a unordered tree, we speak of **tree rewriting**
  - If pattern is a term (ordered tree), we speak of **term rewriting**
- **XML-Rewrite Systems (XML-Ersetzungssysteme** (Bsp. Xcerpt) work on XML-trees and can treat links
- Einsatz:
  - Identifikation von Baummustern
  - Verschlankungen
  - oder Normalisierungen.
- Strategien:
  - Freies Ersetzen (chaotic rewriting): alle Regeln solange angewendet werden, bis das Repositorium sich nicht mehr ändert
  - Spezialstrategien: Die Spezifikation von Strategien möglich, die die Regeln in

Graphersetzungssysteme

- Graphersetzungssysteme unterliegen der Einschränkung auf Baumdaten nicht.
  - Sie suchen und ersetzen Graphmuster in Graphen.
  - Graphersetzungssysteme beschreiben Strukturen, d.h. können für strukturelle Werkzeuge eingesetzt werden
- Beispiele:
  - Softwareentwicklungsumgebung (Fujaba, MOFLON) reduziert vielfältige Aufgaben im Entwicklungsprozess auf Graphersetzung (und Logik)
    - www.fujaba.de
    - Findet direkte Anwendung in der modellgetriebenen Entwicklung.
    - Übersetzung [Nagl]
    - Verfeinerung im Entwurf [Schürr, Lewerentz]
    - Konfigurationsmanagement [Westfechtel].
  - Graphersetzungssysteme können einfach zum Bau von Interpretierern eingesetzt werden http://groove.cs.utwente.nl/

13.1 The Query and Term Transformation Language Xcerpt

A modern, declarative query and transformation language in the XML technical space

Xcerpt combines a DQL and a DTL

Literature - Modular Xcerpt

Xcerpt prototype compiler: http://sourceforge.net/projects/xcerpt

Sebastian Schaffert. Xcerpt: A Rule-Based Query and Transformation Language for the Web. PhD Thesis,Institute for Informatics, University of Munich, 2004.

Sebastian Schaffert, François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt (2004) In Proc. Extreme Markup Languages

http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-7.pdf

U. Aßmann, S. Berger, F. Bry, T. Furche, J. Henriksson, and J. Johannes. Modular web queries from rules to stores. In 3rd International Workshop On Scalable Semantic Web Knowledge Base Systems.

Uwe Aßmann, Andreas Bartho, Wlodek Drabent, Jakob Henriksson and Artur Wilk

Composition Framework and Typing Technology tutorial In Rewerse I3-d14

http://rewerse.net/deliverables/m48/i3-d14.pdf

Jakob Henriksson and Uwe Aßmann. Component Models for Semantic Web Languages. In Semantic Techniques for the Web. Lecture Notes in Computer Science 5500. Springer Berlin / Heidelberg, ISSN 0302-9743, 2009

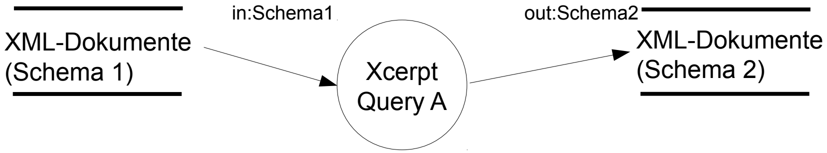
http://springerlink.metapress.com/content/x8q1m87165873127/?p=edfdbbaec29743d59da1cd6f1ea50826&pi=4

Artur Wilk. Xcerpt web site with example queries.

http://www.ida.liu.se/~artwi/XcerptT

Xcerpt: A Modern Web Query Language

- Xcerpt is a modern, pattern-based query language for XML formatted data
  - Terms, trees, and XML terms
  - Patterns match data w.r.t. document structure
  - Fully declarative, in contrast to Xquery
  - Rule-based, declarative Style of Logic Programming (LP)
  - Much more flexible than XPath, which supports only path-based selection
- Xcerpt is also a transformation language in form of a term rewrite system (Termersetzungssystem):
  - Separate query terms (left-hand side) and construct terms (right-hand side) not like in XQuery)
  - it has “Construct terms” to simplify creation of new documents
- Xcerpt is stream-based: processes read and write streams
  - Xcerpt can be used as generator and transformer in DFD



Xcerpt Data Terms (XML Terms)

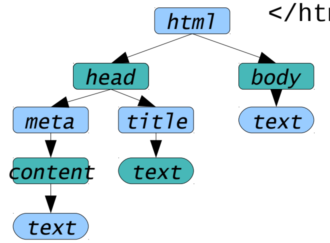
- Xcerpt data terms represent XML Terms with nice syntax
- Basic constructors for data terms:
  - **exact description of a collection of children:**
    - ordered list [...],
    - unordered set {...}
  - **partial description of a collection of children:**
    - ordered partial list [[...]]
    - unordered partial set {{...}}
  - **references/links:**
    - key id@, keyref ^id

```
<book><title>The Last Nizam</title></book>
equivalent to:
book [ title [ “The Last Nizam” ] ]
```


Xcerpt Data Terms

```
html [
      head [
            meta [
              content {“text/html”}
            ]
            title [“Website”]
      ],
      body [“content”]
]
```

```
<html>
      <head>
      <meta content=”text/html”/>
            <title>
                  Website
            </title>
      </head>
      <body>
            content
      </body>
</html>
```



Xcerpt Query Terms

- A **query term** is a pattern containing variables (noted in uppercase letters) over XML data, underspecified data terms:
  - Ordered matching: `data [ term [ … X ] ]`
  - No order matching: `data { term { … X } }`
  - Ordered partial matching: `[[ X ... ]]`
  - Unordered partial matching `{{ … X }}`
  - Queries connect query terms with logical expressions:

    `and { ... }, or { ... }`
  - Variables can unify to subterms
- Query terms are data terms with variables prefixed by keyword “var”

```
title [ book [ var X ] ]
```

```
// the data base
bib [
   book [ title [ “The Last Nizam” ], author [ “John Zubrzycki” ] ],
   book [ title [ “In Spite of the Gods” ], author [ “Edward Luce” ] ]
]
```


Xcerpt Query Terms

- **Query terms** are data terms with variables prefixed by keyword “var”
- Construct Terms also use variables

```
// the result
“The Last Nizam”,
“In Spite of the Gods”
```

```
book [[ title [ var X ] ]]
```

```
// the data base
book [ title [ “The Last Nizam” ],  author [ “John Zubrzycki” ] ],
book [ title [ “In Spite of the Gods” ], author [ “Edward Luce” ] ]
```


Xcerpt Query Terms

- A **query term** (match expression, left-hand side of a rule) is a data term with variables and partial matching

```
library {{
        book {{
              var Author -> author {{
                                          surname {“Aßmann”}
                                    }},
              title [ var Title ]
        }}
}}
```

- Query matches all books with at least one author “Aßmann”
  - assigns the matched authors to variable Author
  - assigns the matched book titles to variable Title
- Produces a stream of a pair of variables (Author, Title)

Xcerpt Transformation Rules

- Combine Query and Construct Terms via common variables

```
// the result
title [ book [ “The Last Nizam” ] ],
title [ book [ “In Spite of the Gods” ] ]
```

```
FROM book [[ title [ var X ] ]]
CONSTRUCT title [ book [ var X ] ]
```

```
// the data base
book [ title [ “The Last Nizam” ],  author [ “John Zubrzycki” ] ],
book [ title [ “In Spite of the Gods” ], author [ “Edward Luce” ] ]
```

Xcerpt Programs

17

- Xcerpt programs consist of rules and data-terms
  - 1+ goal rules
  - 0+ construct-query rules
  - 0+ data-terms

- *Construct rules (transformation rules*: produce intermediate results (transformation FROM pattern match)

```
CONSTRUCT <head> FROM <body> END
FROM <body> CONSTRUCT <head> END
```

Result schema of a rule

Goal schema

- *Goal rules*: final output

```
GOAL <head> FROM <body> END
```

  - Where <head>: *construct term*; <body>: *query*

```
CONSTRUCT
  construct term
FROM
  query term
END
```


Simple Xcerpt program

18

- Matching query → variable bindings
  → apply bindings to construct term

```
CONSTRUCT
   titles [
      all title [ var Title ]
   ]
FROM
   bib {{
      book {{
         title [ var Title ],
   }} }}
END
```

produce

```
titles [
   title [
      “The Last Nizam” ],
   title [
      “In Spite of the Gods” ]
]
```

query

```
// the data base
bib [
   book [ title [ “The Last Nizam” ], author [ “John Zubrzycki” ] ],
   book [ title [ “In Spite of the Gods” ], author [ “Edward Luce” ] ]
]
```


Xcerpt Construct Terms

19

- **Construct Terms** (transformation expressions, right-hand sides of a rule) construct arbitrary structured XML data
  - access data from variables bound by query terms
  - aggregate/re-group data
  - can only have single brackets (no optional content)
- constructing one title/author pair in a result tag

```
result {
        var Title, var Author
}
```

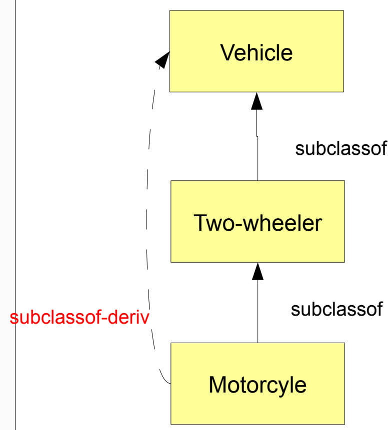
- constructing a complete books result list grouped by full author name

```
booklist {
        all books {
                all var Author,
                var Title
        }
}
```


Rule Dependencies in a Set of Rules (here: Transitive Closure)

21

```
CONSTRUCT
    subclassof-deriv [ var Cls, var Cls ]
FROM
    or { subclassof [ var Z, var Cls ],
         subclassof [ var Cls, var Z ] }
END

CONSTRUCT
    subclassof-deriv [ var Sub, var Sup ]
FROM
    or {  subclassof [ var Sub, var Sup ],
       and {
           subclassof [ var Sub, var Z ],
           subclassof-deriv [ var Z, var Sup ]
    } }
END

CONSTRUCT subclassof [ var Sub, var Sup ]
FROM
    in { resource { “file:...”, “xml” },
         <query> }
END
```

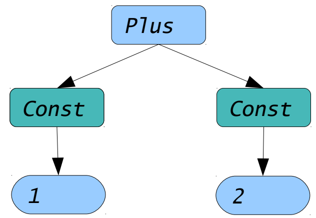


13.1.2 Code Transformations with Term Rewriting

Xcerpt Data Terms for Representing Expression Code

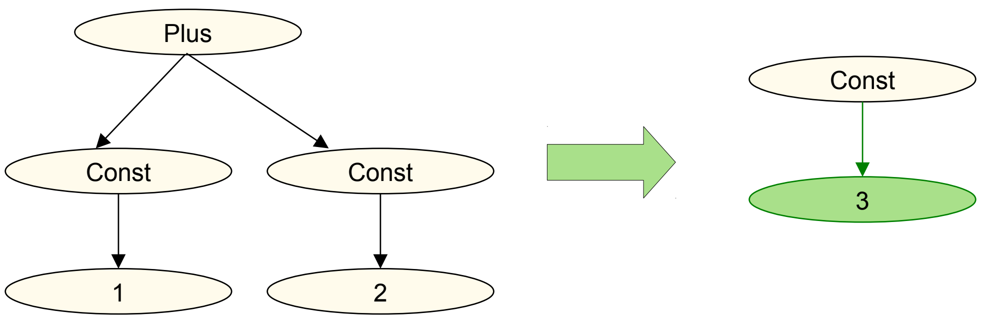

```
Plus [
        Const [ 1 ],
        Const [ 2 ]
]
```

```
<Plus>
        <Const>
                1
        </Const>
        <Const>
                2
        </Const>
</Plus>
```



Constant Folding

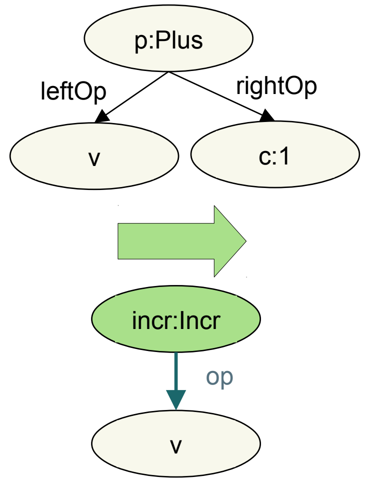
- A **local rewriting (context-free rewriting)** matches a weakly connected left-hand side graph with a redex.
  - Matching of one redex can be done in constant time
- Subtractive because redexes are destroyed




```
FROM Plus [ Const [ 1 ], Const [ 2 ] ] CONSTUCT Const [3]
```


Context-Free Local Rewritings: Operator Strength Reduction

// if-then rules:
if leftOp(p:Plus,v),
  rightOp(p,c:1),
then
  Delete p,
  Delete c,
  Add incr:Incr,
  op(incr,v);

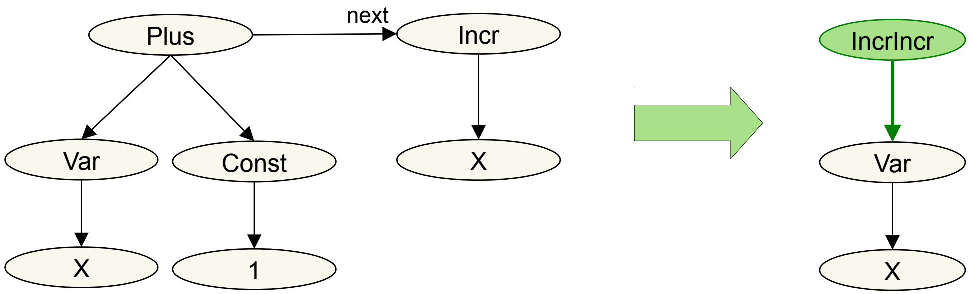



```
FROM Plus [ leftOp [ var v ], rightOp [ c:1 ] ]
CONSTUCT incr:Incr [ var v]
```

Peephole Optimization

27

- Peephole optimization is done on statement lists or trees
- Subtractive problem, because redexes are destroyed



13.1.3 Larger Xcerpt Programs

28

Modular Xcerpt

29

- *Modular Xcerpt* = Xcerpt + Module support
- http://www.reuseware.org/index.php/Screencast_LoadMXcerptProject_0.5.1
- Declaring a module in Modular Xcerpt

```
MODULE module-id
  module-imports
  xcerpt-rules
```

- Declaring a module's interface
  - Modular Xcerpt programs importing a module can reuse public construct terms

```
public construct term
```

  - Modular Xcerpt programs can provide data to an imported module's public query terms

```
public query term
```


Modular Xcerpt

30

- Modular Xcerpt is an Extension of Xcerpt for larger programs
-
- A query can be reused via a module's interface

```
IMPORT module AS name
```

  - reuses public construct terms as a data provider for the given query term

```
in module ( query term )
```

  - provides the given construct term to public query terms of an imported module

```
to module ( construct term )
```

32

```
IMPORT file:subclassof.mx AS mod

GOAL    vehicles [ all var Sub ]
FROM
    IN mod (
        output [[
            subclassof [ var Sub,
                         "Vehicle" ]
    ]] )
END

CONSTRUCT
    TO mod (
        input [
            subclassof [
                var Sub, var Sup ]
        ] )
FROM
    in { resource {
            "file:...", "xml" },
            <query> }
END
```

Result:

```
vehicles [
"Vehicle", "Two-wheeler", "Motorcycle"
]
```

→ = *data flow*

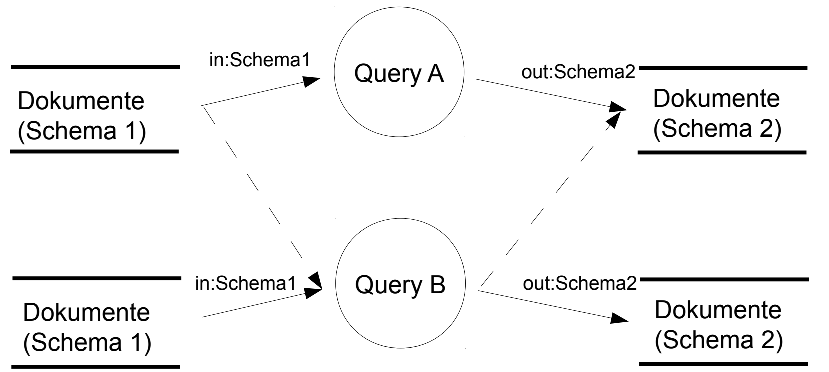
Reusable module, in file:subclassof.mx

```
MODULE subclassof-reasoner

CONSTRUCT
    public output [
        all subclassof [ var Sub, var Sup ] ]
FROM  subclassof-deriv [ var Sub, var Sup ]
END

CONSTRUCT
    subclassof-deriv [ var Cls, var Cls ]
FROM
    or { subclassof [ var Z, var Cls ],
     Subclassof [ var Cls, var Z ] }
END

CONSTRUCT
    subclassof-deriv [ var Sub, var Sup ]
FROM
 or { subclassof [ var Sub, var Sup ],
  and { subclassof [ var Sub, var Z ],
    subclassof-deriv [ var Z, var Sup ]
 } }
END

CONSTRUCT     subclassof [ var Sub, var Sub ]
FROM
    public input [[
   subclassof [ var Sub, var Sup ] ]]
END
```


Einsatz von QL in Werkzeugen

33

- Stromverarbeitende QL sind sehr gut geeignet für Werkzeugkomposition



Zwei stromtransformierende Werkzeuge können komponiert werden, falls ihre Ein- und Ausgabetypen übereinstimmen und keine Reihenfolge im Ausgabespeicher vorliegt.

13.2. Model Transformation and Program Optimization with Graph Rewrite Systems

34

Prof. Dr. Uwe Aßmann
Softwaretechnologie
Technische Universität Dresden
Version 12-0.2, 02.01.14

1) Basic Setting
2) Examples
3) More on the Graph-Logic Isomorphism
4) Implementation in Tools
5) ATL

Obligatory Literature

35

- Kevin Lano. Catalogue of Model Transformations
  - http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf
- Uwe Aßmann. Graph rewrite systems for program optimization. ACM Transactions on Programming Languages and Systems (TOPLAS), 22(4):583-637, June 2000.
  - http://portal.acm.org/citation.cfm?id=363914
- Tom Mens. On the Use of Graph Transformations for Model Refactorings. GTTSE 2005, Springer, LNCS 4143
  - http://www.springerlink.com/content/5742246115107431/

Other References

36

- Uwe Aßmann. OPTIMIX, A Tool for Rewriting and Optimizing Programs. In Graph Grammar Handbook, Vol. II. Chapman-Hall, 1999.
- K. Lano. Catalogue of Model Transformations
  - http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf

13.2.1 Using GRS for Analysis and Transformation of Models and Code

37

Problem and Goal

38

- MDSD tools need model transformations
- Compilers need program transformations

Model Transformation and Optimization with Graph Rewriting

39

- Use the graph-logic-isomorphism: Represent everything in a program or a model as directed graphs
  - Program code (control flow, statements, procedures, classes)
  - Model elements (states, transitions, ...)
  - Analysis information (abstract domains, flow info ...)
- Directed graphs with node and edge types, node attributes
  - one-edge condition (no multi-graphs)
- Use edge addition rewrite systems (EARS) to
  - Query the graphs
  - Analyze the graphs
  - Map the graphs to each other
- Use graph rewrite systems (GRS) to
  - Construct and augment the graphs
  - Transform the graphs
- Preferably, the GRS should terminate (XGRS, exhaustive GRS)
- Use the graph-logic isomorphism to encode
  - Facts in graphs

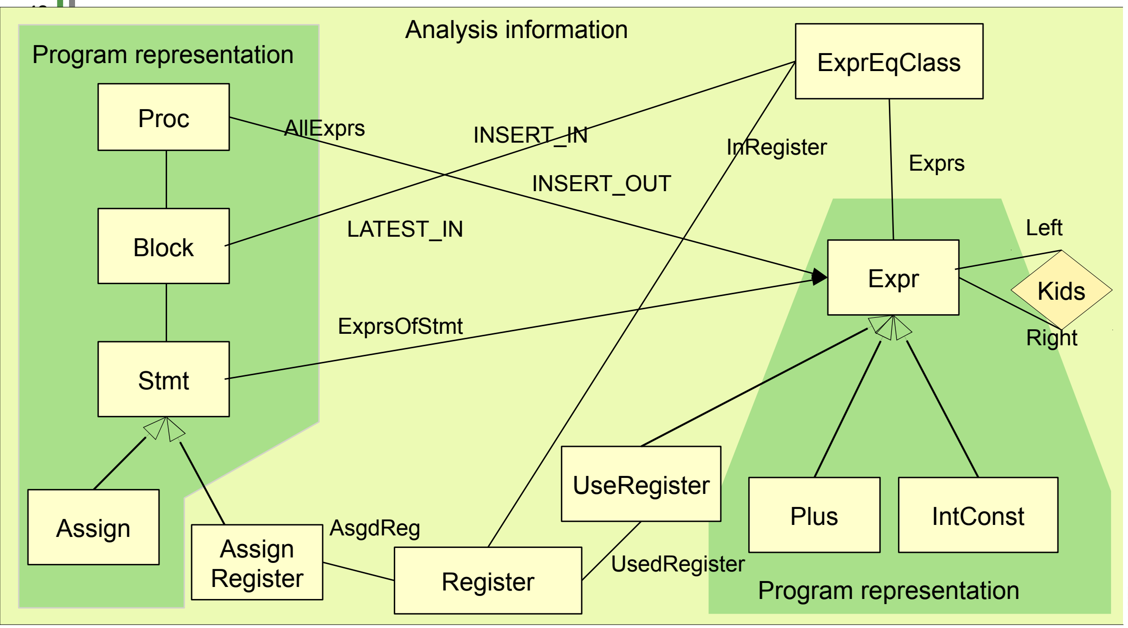
Terminology for Automated Graph Rewriting

- **Graph rewrite rule:** rule (left, right hand side) to match left-hand side in the graph and to transform it to the right-hand side
- **Graph rewrite system:** set of graph rewrite rules
- **Start graph (axiom):** input graph to rewriting
- **Graph rewrite problem:** a graph rewrite system applied to a start graph
- **Manipulated graph (host graph):** graph which is rewritten in graph rewrite problem
- **Redex**: (reducible expression) application place of a rule in the manipulated graph
- **Derivation:** a sequence of rewrite steps on the manipulated graph, starting from the start graph and ending in the normal form
- **Normal form:** result graph of rewriting; manipulated graphs without further redex
- **Unique normal form:** unique result of a rewrite system, applied to one start graph
- **Terminating GRS:** rewrite system that stops after finite number of rewrites
- **Confluent GRS:** two derivations always can be commuted, resp. joined together to one result
- **Convergent GRS:** rewrite system that always yields unique results (terminating and confluent)

Specification Process

1) Specification of the data model (graph schema)
   - Specification of the graph schema with a graph-like DDL (ERD, MOF, GXL, UML or similar):
     - **Schema of the program representation**: program code as objects and basic relationships. This data, i.e., the start graph, is provided as result of the parser
     - **Schema of analysis information** (the infered predicates over the program objects) as objects or relationships
2) Program analysis (preparing the abstract interpretation)
   - Querying graphs, enlarging graphs
   - Materializing implicit knowledge to explicit knowledge
3) Abstract Interpretation (program analysis as interpretation)
   - Specifying the transfer functions of an abstract interpretation of the program with graph rewrite rules on the analysis information
4) Program transformation (optimization)
   - Transforming the program representation

A Simple Program (Code) Model (Schema) in UML



13.2.2 Examples

13.2.1 Local Rewritings

44

Code Optimizations Expressible by Local Graph Rewritings

45

- Local transformations of the program representation
  - copy propagation, constant propagation
  - loop optimizations (unrolling etc.)
  - branch optimization, strength reduction
  - idiom recognition
  - dead code elimination

13.2.2. Complex Local Rewritings

46

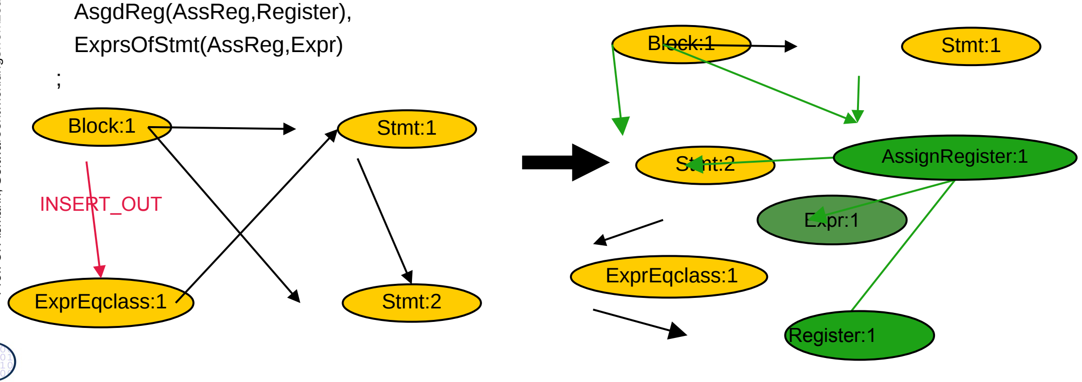
- On Dags (with joins) and Graphs (with cycles)

Example: Lazy Code Motion Transformation

47

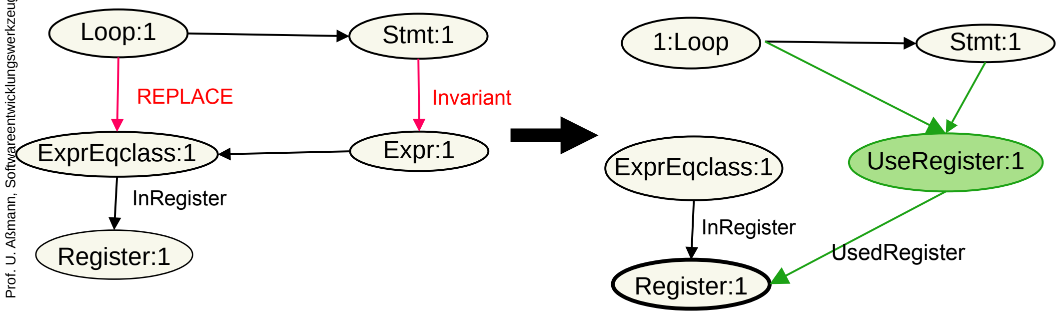
```
if Stmts.last(Block,Stmt),
  INSERT_OUT(Block,ExprEqclass)
then
  new Register:Register;
  new Expr:Expr;
  new AssReg:AssReg;
  InRegister(ExprEqclass,Register),
  AsgdReg(AssReg,Register),
  ExprsOfStmt(AssReg,Expr)
;
```

- Insert expressions at an optimally early place
- INSERT_OUT indicates, at which block-exit an expression should be made available



Loop Invariant Code Motion

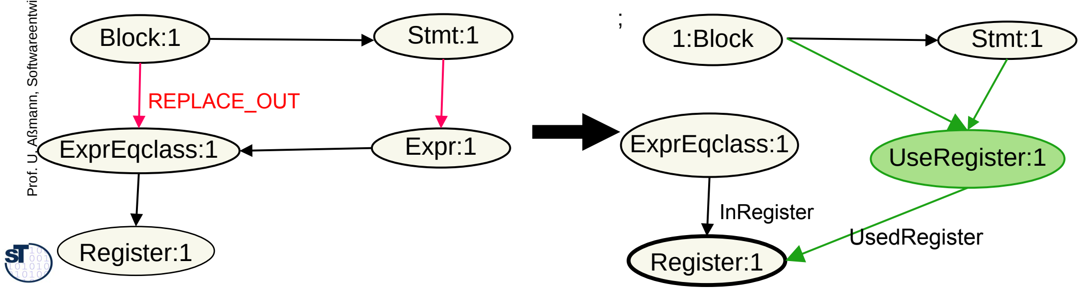
- Loop-invariant code motion moves code before loops which is over and over computed again in the loop (loop-invariant)



Lazy Code Motion Transformation

- REPLACE_OUT indicates at which block-exist an expression should no longer be computed, but its result should be re-used from a register

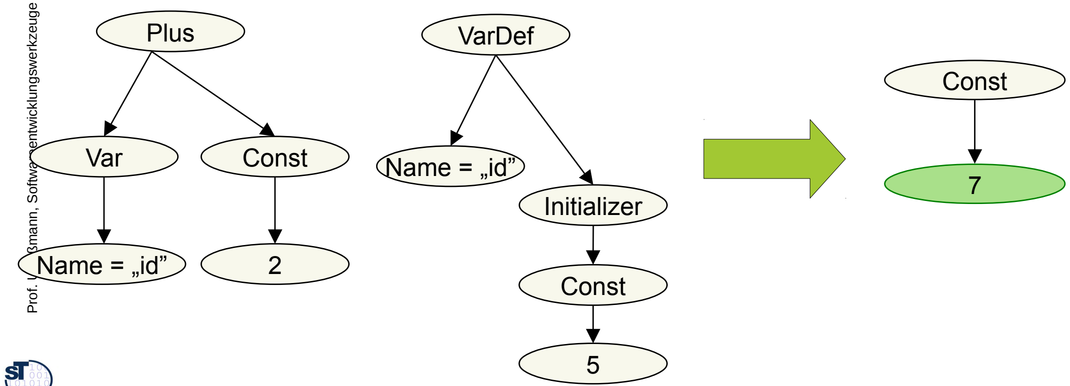
```
if  Stmts(Block,Stmt),
    ExprsOfStmt(Stmt,Expr),
    REPLACE_OUT(Block,ExprEqclass),
    InRegister(ExprEqclass,Register),
    Computes(Expr,ExprEqclass)
then
    new UseReg:UseReg;
    delete Expr;
    ExprsOfStmt(Stmt,UseReg),
    UsedReg(UseReg,Register)
;
```



60.3. Context-Sensitive Rewritings

Extended Constant Folding as Subtractive GRS

- A term rewrite system usually works context-free, i.e., matches and rewrites only one term.
- A **context-sensitive rewriting** matches a non-connected left-hand side graph with a redex.
  - Matching of one redex can be done in quadratic time, because non-connected nodes have to be pairwise compared



Covered Code Optimizations

- Global transformations
  - lazy and busy code motion (loop invariant code motion)
  - message optimization

13.4. Model Transformations with ATL

ATLAS Transformation Language

Statt QVT ist in der Praxis auch ATL sehr beliebt

http://www.eclipse.org/atl/

ATL integrates OCL as Query Language

```
// Transitive Hülle in ATL, mit Verwendung einer rekursiven OCL Query
rule computeTransitiveClosureBaseCase {
  from node: Node (
    // possible to call OCL expressions
    node->baserelation.collect( e | e.baserelation)->flatten() );
  )
  to newNode mapsTo node (
    // set new transitive relation
    newNode->transitiverelation <- node->baserelation
  )
}
rule computeTransitiveClosureRecursiveCase {
  from node: Node (
    node->transitiverelation.collect( e | e.baserelation)->flatten() );
  )
  to newNode mapsTo node (
    // set new transitive relation
    newNode->transitiverelation <- node->transitiverelation
  )
}
```


Tools for Model-Driven Software Development

- In MDSD and MDA, horizontal and vertical model transformations should be specified with graph rewrite systems
- Example tools:
  - **Grgen** (Karlsruhe)
  - **Fujaba** (Kassel, Paderborn)
  - **MOFLON** (Darmstadt)
  - **VIATRA2** on EMF http://eclipse.org/gmt/VIATRA2/

13.5 More on the Logic-Graph Isomorphism

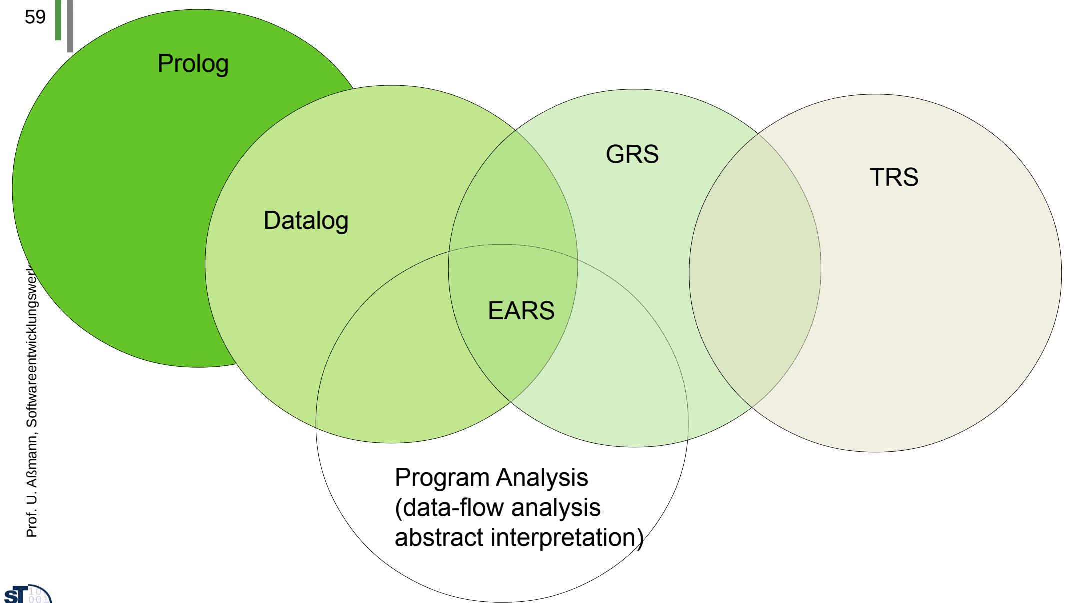
Results

- Theory:
  - If a termination graph can be identified, a graph rewrite systems terminates.
  - Graph rewriting, DATALOG and data-flow analysis have a common core: EARS
- Program optimization:
  - Spezification of program optimizations is possible with graph rewrite systems. Short specifications, fewer effort.
  - Practically usable optimizer components can be generated.
- Uniform Specification of Analysis and Transformation
  - If the program analysis (including abstract interpretation) is specified with GRS
  - It can be unified with program transformation

Limitations

- Several optimizations can be specified with GRS which are not exhaustive (peephole optimization, constant propagation with partial evaluation).
- As general rule embedding is not allowed, a rule only matches a fixed number of nodes.
  - Thus those transformations, which refer to an arbitrary set of nodes, cannot be specified.

The Common Core of Logic, Rewriting and Program Analysis



13.A

60

Process: How to Build an Optimizer or Model Transformer

61

- Specify the optimizer/model transformer in steps:
  - Preprocessing steps with XGRS and EARS
    - that convert the abstract syntax tree to an abstract syntax graph with definition-use relations
    - that diminish the domains of the analyses (e.g., equivalence classing)
    - that build summary information for procedures
    - that build indices for faster (constant) access
  - Analyses: specify abstract interpretations with EARS
    - reaching-definition information, value flow information
    - SSA
  - Transformation: apply XGRS and stratifiable XGRS

Efficient Evaluation Algorithms from Logic Programming

62

- „Order algorithm“ scheme [Aßmann00]
  - Variant of nested loop join
  - Easy to generate into code of a programming language
  - Works effectively on very sparse directed graphs
  - Sometimes fixpoint evaluations can be avoided
  - Use of index structures possible
  - Linear bitvector union operations can be used
- DATALOG optimization techniques can be employed
  - Bottom-up evaluation is normal, as in Datalog
  - Top-down evaluation as in Prolog possible, with resolution
  - semi-naive evaluation
  - index structures
  - magic set transformation
  - transitive closure optimizations

Practical Features

63

- Short specifications
  - expression equivalence classes 30 rules
  - DFA reaching definitions 20-40
  - copy propagation 5
  - lazy code motion 5
- Velocity:
  - Tool Optimix generates the Order algorithm for a GRS
  - Compiler with generated components is slower, but ..
  - important algorithms run as fast as hand-written algorithms (DFA)
- Flexibility:
  - intermediate language CCMIR for C (CoSy), Modula-2, Fortran (Aßmann)
  - Model transformations (Alexander Christoph)
  - Aspect weaving (Aßmann, Heidenreich, many others)
  - Refactorings (Aßmann, Mens)
- OPTIMIX 2.5 on optimix.sourceforge.net
  - Works with CoSy, Cocktail, or plain C
  - A prototype code generator for Java exists