

51. Model and Program Analysis with Graph Reachability

1

Prof. Dr. Uwe Aßmann
 Softwaretechnologie
 Technische Universität Dresden
 Version 12-0.6, 17.01.13

- 1) Model Mapping
- 2) EARS for Reachability
- 3) Regular graph reachability
 - 1) Graph slicing
- 4) Context-free graph reachability
- 5) More on the Graph-Logic Isomorphism
- 6) Implementation in Tools

Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Aßmann

Other References

3

- ▶ 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>

Literature

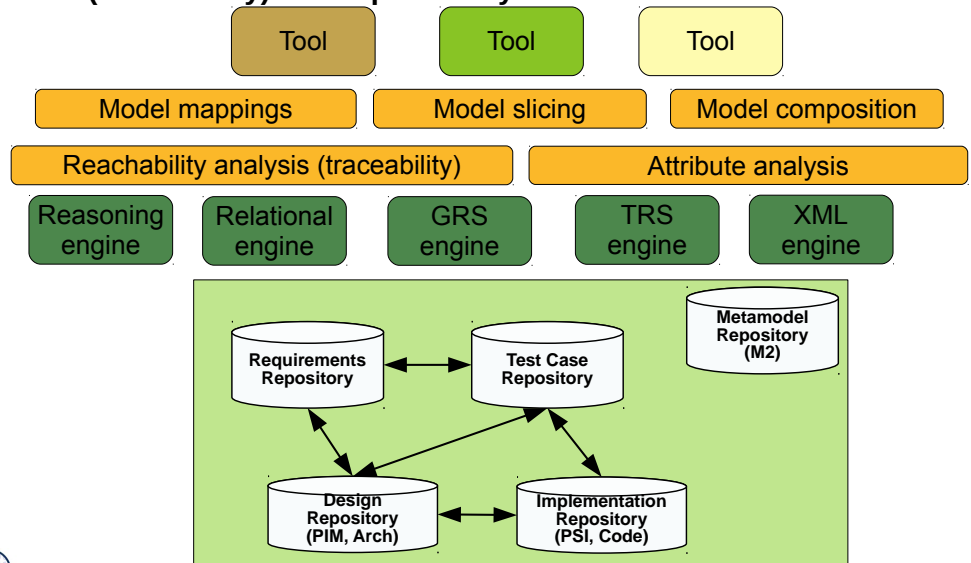
2

- ▶ Hannes Schwarz, Jürgen Ebert, and Andreas Winter. Graph-based traceability: a comprehensive approach. Software and System Modeling, 9 (4):473-492, 2010.
- ▶ 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/>
- ▶ Thomas Reps. Program analysis via graph reachability. Information and Software Technology, 40(11-12):701-726, November 1998. Special issue on program slicing.
- ▶ Mark Weiser. Program slicing. IEEE Transactions on Software Engineering, SE-10(4):352-357, July 1984.
- ▶ Frank Tip. A survey of program slicing techniques. Journal of Programming Languages, 3:121-189, 1995.

Tools in an Integrated Development Environment (IDE)

4

- ▶ **Model mappings** relate different artefacts to enable **traceability (reachability)** and **impact analysis**



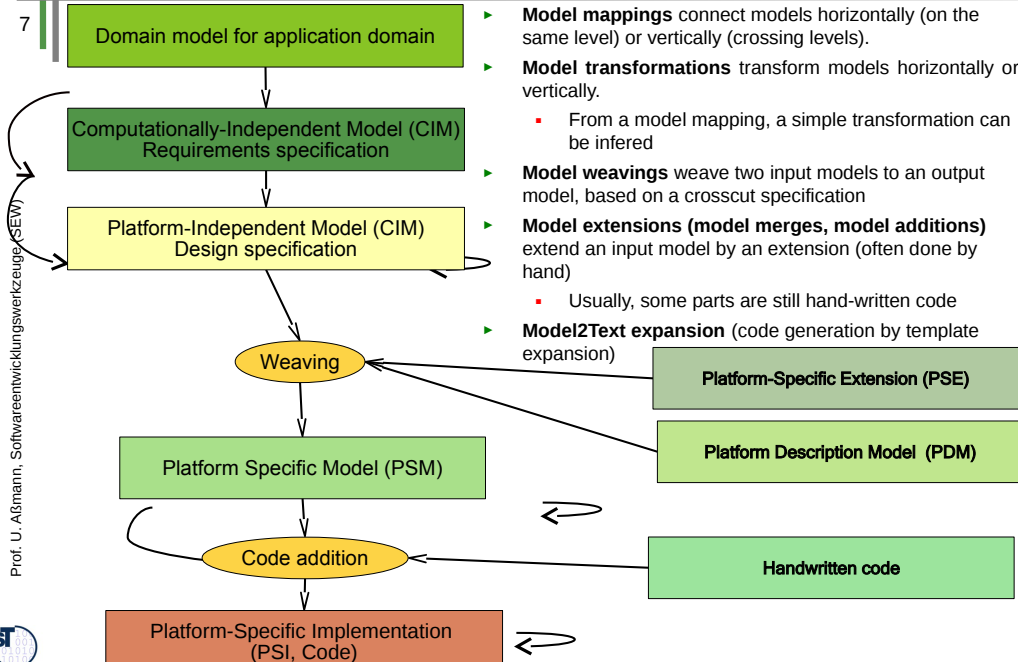
Literature

- ▶ Frédéric Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. In: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06). ACM Press, Dijon, France, chapter Model transformation (MT 2006), pages 1188—1195.
 - <http://atlanmod.emn.fr/bibliography/SAC06a>
- ▶ Tutorial über ATL “Families2Persones”
- ▶ http://www.eclipse.org/m2m/atl/doc/ATLUseCase_Families2Persons.ppt
- ▶ ATL Zoo von Beispielen
 - <http://www.eclipse.org/m2m/atl/atlTransformations>
- ▶ K. Lano. Catalogue of Model Transformations
 - <http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf>
- ▶ Implementation in ATL
 - <http://www.eclipse.org/m2m/atl/atlTransformations/EquivalenceAttributesAssociations/EquivalenceAttributesAssociations.pdf>

Prof. U. Alsmann, Softwareentwicklungswerkzeuge (SEW)

51.1 Model Mapping (Modellverknüpfung)

Model Mappings and Model Weavings



Prof. U. Alsmann, Softwareentwicklungswerkzeuge (SEW)

Model Mapping (Modell-Verknüpfung) with MID INNOVATOR

- ▶ Innovator can be used for requirements models, design models, implementation models, as well as for transformations in between
- ▶ How to relate these models systematically?

The screenshot shows the INNOVATOR software interface. The main window displays a project tree on the left with elements like 'external object', 'Use Case System', 'analysis system', 'Java design system', 'Java implementation system', and 'systemModel management'. On the right, there is a table with columns for Status, Name, Typ, and Änderungsdatum. The table contains several entries, including 'Ausleihe', 'Kunde_anmelden', 'Rückgabe', 'Tonträger_Einkauf', 'Kunden_neu_anlegen', 'AnalysisClassDiagram', 'Verwaltung_AS', 'Tonträger_AS', 'Kunde_AS', 'Kunde_AS', 'Tonträger_AS', 'VerwaltungUI_AS', 'VerwaltungUI_AS', 'Kunde_LUC', 'Bibliothek_LUC', and 'Verwaltung_AS'.

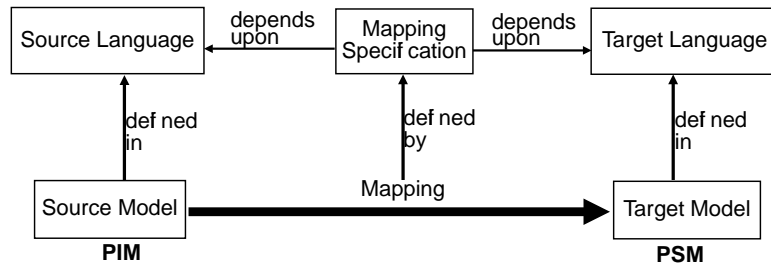
Status	Name	Typ	Änderungsdatum
1 0 A	Ausleihe	Sec...	22.11.2003 00:48:02
2 0 A	Kunde_anmelden	Köll...	10.11.2003 01:21:54
3 0 A	Rückgabe	Sec...	22.11.2003 00:21:47
4 0 A	Tonträger_Einkauf	Sec...	10.11.2003 01:23:59
5 0 A	Kunden_neu_anlegen	Sec...	10.11.2003 01:26:19
6 0 A	AnalysisClassDiagram	Klas...	09.11.2003 15:29:14
7 0 A	Verwaltung_AS	Klas...	09.11.2003 15:25:56
8 0 A	Tonträger_AS	Klas...	09.11.2003 15:20:08
9 0 A	Kunde_AS	Klas...	09.11.2003 15:27:32
...	...	Obj...	09.11.2003 13:20:05
...	...	Obj...	09.11.2003 13:20:16
...	...	Klas...	09.11.2003 15:16:32
...	...	Obj...	09.11.2003 13:23:08
...	...	Obj...	09.11.2003 14:05:54
...	...	Obj...	09.11.2003 15:44:35
...	...	Obj...	09.11.2003 16:14:14

Prof. U. Alsmann, Softwareentwicklungswerkzeuge (SEW)

Rpt. from ST-II: Model Mapping, Transformation and Synchronization in the MDA

9 The MDA architecture derives from a *platform-independent model (PIM) by hand, by rules, by transformations, by metaprograms platform-specific models (PSM)*

- ▶ Model mapping connects systematically all elements of a **source model** to the elements of a **target model**.
- ▶ From the mappings, a translation, transformation, or synchronization can be automatically inferred.



Quelle: Kleppe, A., Warmer, J., Bast, W.: MDA Explained - Practice and Promise of the Model Driven Architecture; Addison Wesley 2003 (Draft 25.10.02)

Prof. U. Alsmann, Softwareentwicklungswerkzeuge (SEW)

Problem: Analysis and Reachability

- 10
- ▶ We need graph reachability analyzers
 - to create trace graphs for reachability and traceability
 - to create model mappings, model slicings
 - to prepare refactorings, transformers, and optimizers
 - For models: For model refactoring, adaptation and specialization, weaving and composition
 - For code: Portability to new processor types and memory hierarchies
 - For optimization (time, memory, energy consumption)
 - ▶ However, reachability analyzers are big beasts
 - Current implementation techniques are hard to understand and to a large extent unsystematic
 - ▶ Idea: Use graph-logic isomorphism

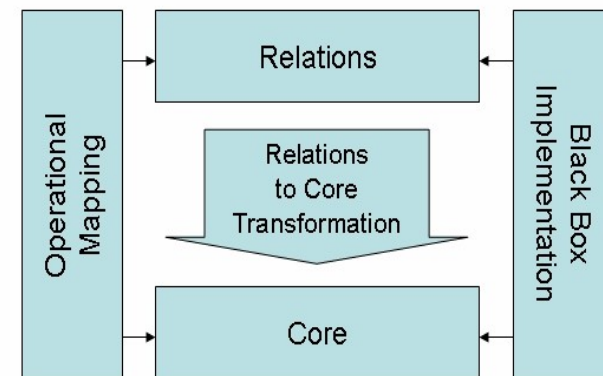
Prof. U. Alsmann, Softwareentwicklungswerkzeuge (SEW)

51.1.1 Query-View-Transformations (QVT)

The language of the OMG for model transformations within MDA

QVT Dialects

12



Prof. U. Alsmann, Softwareentwicklungswerkzeuge (SEW)

Transitive Closure with QVT Relations

- 13
- ▶ QVT relations uses logic expressions on base and derived relations (graph-logic isomorphism)

```
// Transitive Closure in QVT relations,  
// Modeled with recursive relation "transitiverelation"  
relation transitiverelation {  
  domain node:Node {  
    // matching attributes  
    name = sameName;  
  }  
  domain node2:Node {  
    // node2 must have the  
    // same name as node  
    name = sameName;  
  }  
  domain node3:Node {  
    // node3 must also  
    // have the same name  
    name = sameName;  
  }  
  when {  
    // conditions: base relation must exist  
    baserelation(node,node2) or  
    // or a transitive relation to a base relation  
    (transitiverelation(node,neighbor)  
    and baserelation(neighbor,node2));  
  }  
  where { // Aufruf einer Transformation  
    makeNodeSound(node);  
  }  
}
```

Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)



QVT Tools

14

Tool			
Eclipse M2M Project	Operational	http://www.eclipse.org/m2m/	
Magic Draw	Operational		
MediniQVT	Relational	http://projects.ikv.de/qvt/wiki	

Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)



OCL for Model Search, Query, and Mapping

- 15
- ▶ OCL is a graph-query language, similar to EARS and .QL
 - ▶ OCL can be called within QVT scripts
 - Two different DQL are combined within a single language

```
// this is QVT  
rule checkNoDoubleFeatureInSuperClasses(name:String) {  
  from node: Class (  
    node->TransitiveClosure()->collect().exists(s |  
s.name() = name);  
  )  
  to  
    System.out.println("Error: super class has doubly  
defined feature: "+s.name());  
}
```

Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)



51.2 Using EARS for Analysis and Mappings of Models and Code

16

- Graph reachability engines are A-tools answering questions about structure of models and programs
- EARS can be employed for regular graph reachability, context-free graph reachability, slicing, data-flow analysis



EARS for Model Mapping

- 17
- ▶ QVT Relational is a language for **Edge addition rewrite systems (EARS)**
 - ▶ EARS can be used for model mapping:
 - Transitive closure
 - Regular path reachability
 - Context-free path reachability

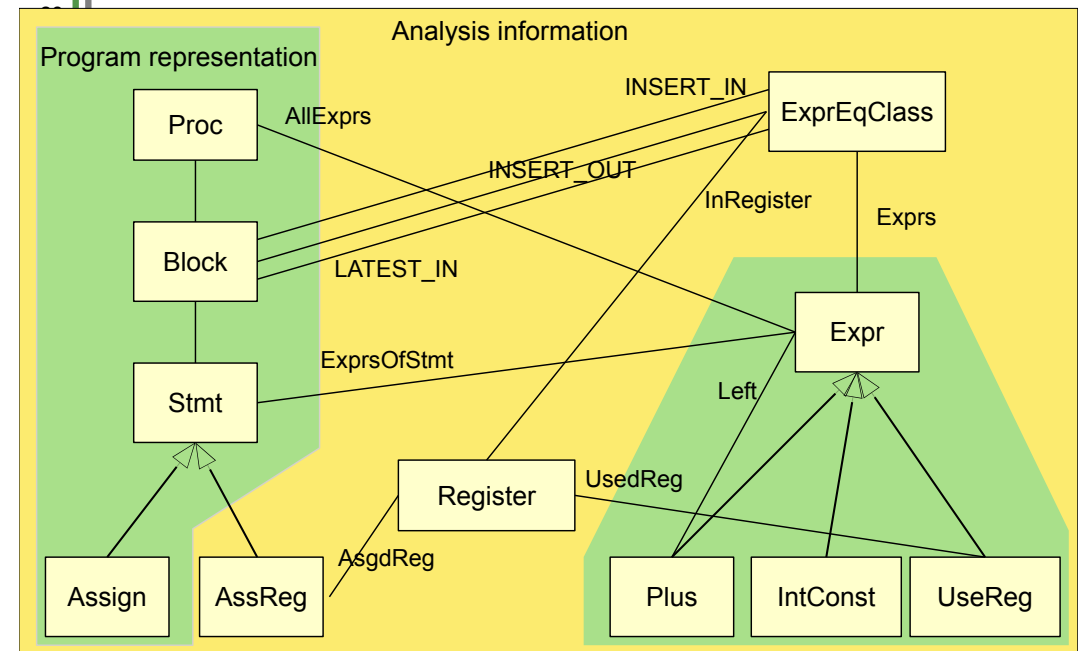
Model Analysis with Graph Reachability

- 18
- ▶ 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) and other graph reachability specification languages to
 - Query the graphs (on values and patterns)
 - Analyze the graphs (on reachability of nodes)
 - Map the graphs to each other (model mapping)
 - ▶ Later: Use graph rewrite systems (GRS) to construct and augment the graphs, transform the graphs
 - ▶ Use the graph-logic isomorphism to encode
 - Facts in graphs
 - Logic queries in graph rewrite systems

Specification Process

- 19
- 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 inferred 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
 - Materializing *model mappings*
 - 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) Model and Program transformation (optimization)
 - Transforming the program representation

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



51.2. Simple Reachability: Path Abbreviations in Graph Analysis

21

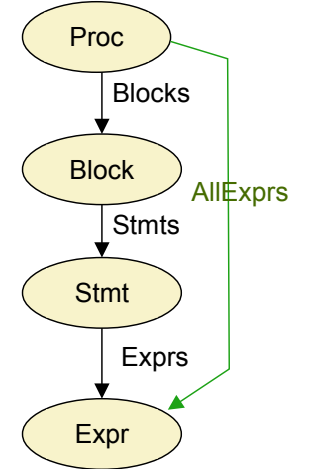
- With edge addition rewrite systems

22

Path Abbreviations

- Path abbreviations shorten paths in the manipulated graph.
- They may collect nodes into the neighborhood of other nodes.
- Ex.: Collection of Expressions for a procedure: edge addition

```
-- Datalog notation:
AllExprs(Proc,Expr) :-
  Blocks(Proc,Block),
  Stmts(Block,Stmt),
  Exprs(Stmt,Expr).
-- if-then rules:
if  Blocks(Proc,Block),
    Stmts(Block,Stmt),
    Exprs(Stmt,Expr)
then
  AllExprs(Proc,Expr);
- regular expression notation (TGreQL):
AllExprs := Proc Blocks.Stmts.Exprs Expr
```

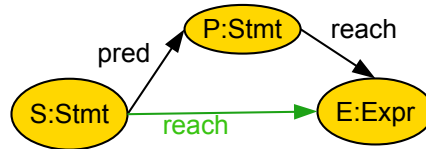
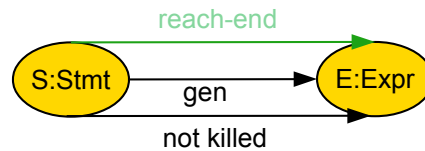


Transitive Closure

23

- Reachability most often can be reduced to transitive closure
- “Does an Stmt S reach a expression E?”
- Left or right recursion in F-Datalog
- Kleene * in TgreQL
- Thick arrow in Fujaba

```
// TGreQL
reach*(S:Stmt, E:Expr)
```



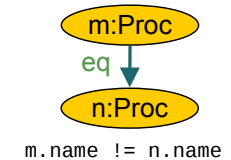
```
// F-Datalog
reach(S:Stmt,E:Expr) :- gen(S:Stmt,E:Expr), not killed(S:Stmt,E:Expr).
reach(S:Stmt,E:Expr) :- pred(S:Stmt,P), reach(P,E:Expr).
```

Relating Nodes into Equivalence Classes

24

- Ex.: Computing equivalent nodes
- Context-sensitive problem, because m is not in the context of n

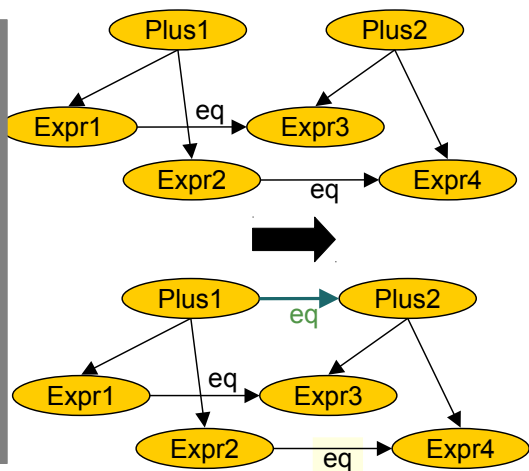
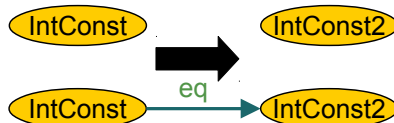
```
baserule:
eq(m:Proc,n:Proc) :-
  m.name != n.name.
-
If (m:Proc, n:Proc) and m.name !=
  n.name
  eq(m,n)
}
- TgreQL regular expression:
m:Proc eq n:Proc if
m.name != n.name
```



Relating Nodes into Equivalence Classes (Here: Value Numbering, Synt. Expression Equivalence)

25

- Ex.: Computing structurally equivalent expressions



```

baserule:
eq(IntConst1,IntConst2) :-
    IntConst1 ~ IntConst(Value),
    IntConst2 ~ IntConst(Value).
recursive_rule:
eq(Plus1,Plus2) :-
    Plus1 ~ Plus(Type),
    Plus2 ~ Plus(Type),
    Left(Plus1,Expr1),
    Right(Plus1,Expr2),
    Left(Plus2,Expr3),
    Right(Plus2,Expr4),
    eq(Expr1,Expr3),
    eq(Expr2,Expr4).
    
```

51.3. Data-Flow Analysis as Graph Reachability

26

- with edge additions

Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Aßmann

Data-flow Analysis for Reachability and Traceability

27

- Data-flow analysis** is a specific form of abstract interpretation asking **reachability questions**, i.e., computing the flow of data through the program, from variable assignments to variable uses
 - Result: the **value-flow graph (data-flow graph)**
- Examples of reachability problems:**
- AllSuperClasses:** find out for a class transitively all superclasses
- AllEnclosingScopes:** find out for a scope all enclosing scopes
- Reaching Definitions Analysis:** Which Definitions (Assignments) of a variable can reach which statement?
- Live Variable Analysis:** At which statement is a variable live, i.e., will further be used?
- Busy Expression Analysis:** Which expression will be used on all outgoing paths?
 - Central part: 1 recursive system

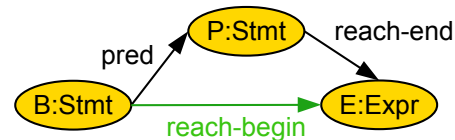
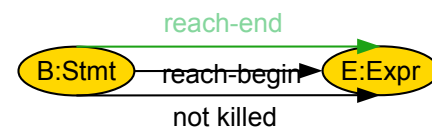
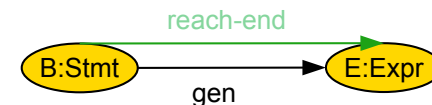
Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

Reaching Definition Analysis

28

Reaching Definitions Analysis:

- Which Assignments of a variable can reach which using statement?
- Which variable definitions reach which expression?
- Graph rewrite rules implement an abstract interpreter
 - On instructions or on blocks of instructions
 - Flow information is expressed with edges of relations "reach-*
- Recursive system (via edge reach-begin)
 - B reach-end E == E reaches end of block B



Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

```

reach-end(B,E) :- gen(B,E).
reach-end(B,E) :- reach-begin(B,E), not killed(B,E).
reach-begin(B,E) :- pred(B,P), reach-end(P,E).
    
```

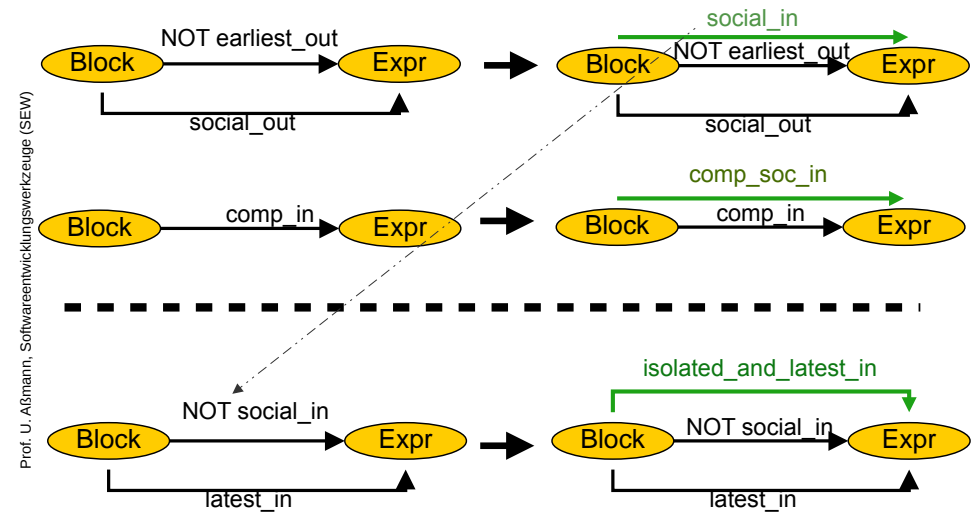


Code Motion Analysis

- 29
- ▶ **Code motion** is an essential transformation to speed up the generated code. However, it is a complex transformation:
 - Discovering loop-invariant expressions by data-flow analysis
 - Moving loop-invariant expressions out of loops upward
 - Code motion needs complex data-flow analysis
 - ▶ **Busy Code Motion (BCM)** moves expressions as upward (early) as possible
 - ▶ **Lazy Code Motion (LCM)**
 - Moving expressions out of loops to the front of the loop, upward, but carefully:
 - Moving expressions to an optimal place so that register lifetimes are shorter and not too long (optimally early)
 - LCM analysis computes this optimal early place of an expression [Knoop/Steffen]
 - Analyze an optimally early place for the placement of an expression
 - About 6 equation systems similar to reaching-definitions
 - Every equation system is an EARS

Excerpt from LCM Analysis with Overlaps

- 30
- ▶ Compute an optimally early block for an expression (out of a loop)



51.3 Regular Graph Reachability

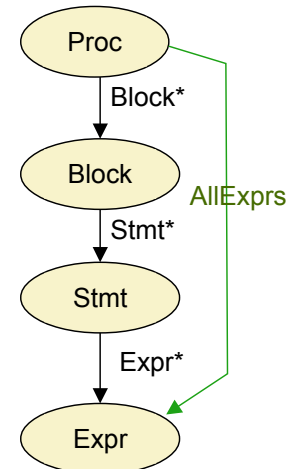
31

Regular Graph Reachability

- 32
- ▶ If the query can be expressed as a regular expression, the query is a **regular graph reachability problem**
 - ▶ Kleene star is used as transitive closure operator
 - ▶ TqreQL and Fujaba are languages offering Kleene *

```

-- F-Datalog notation:
AllExprs(Proc, Expr) :-
  Block*(Proc, Block),
  Stmt*(Block, Stmt),
  Expr*(Stmt, Expr).
-- if-then rules:
if Block*(Proc, Block),
  Stmt*(Block, Stmt),
  Expr*(Stmt, Expr)
then
  AllExprs(Proc, Expr);
-- regular expression notation (TGreQL):
AllExprs := Proc Block*.Stmt*.Expr* Expr
    
```



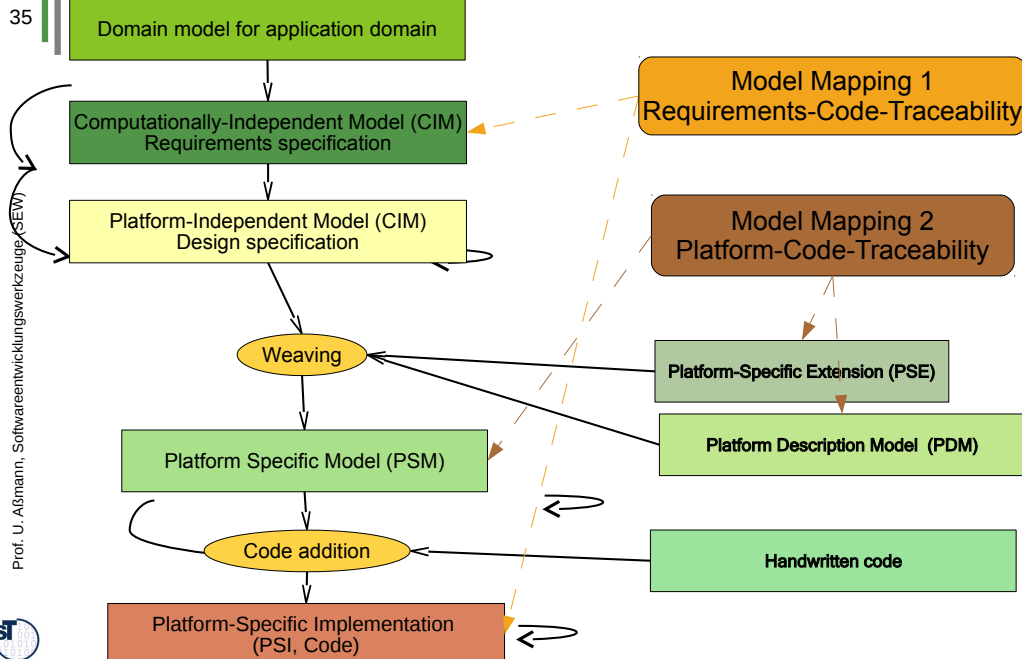
51.3.1 Static Slicing: Single-Source-Multiple-Target Regular Reachability

- 33
- ▶ [Weiser] [Tip]
 - ▶ A **static slice** is the region of a program or model reached from *one source* node by a regular reachability query
 - ▶ A **forward slice** is a region in forward direction of the program
 - The uses of a variable
 - The callees of a call
 - The uses of a type
 - ▶ A **backward slice** is a region in backward direction of the program
 - The assignments which can influence the value of a variable
 - The callers of a method
 - The type of a variable
 - ▶ A static slice introduces path abbreviations from one entity to a region
 - ▶ Slicing can map arbitrary entities in programs and models to other entities, based on a regular graph expression

Traceability between Models

- 34
- ▶ Data-flow analysis (graph reachability, slicing) can be done
 - intraprocedurally
 - Interprocedurally (program-wide)
 - intermodel: then it creates trace relations
 - interspecification: between requirements models, design models, and code models
 - Inter-MDA
 - ▶ **Traceability** is intermodel slicing and graph reachability
 - ▶ A **model mapping** is an intermodel trace graph

Application of Traceability: Model Mappings and Model Weavings



51.3.2 Context-Free Graph Reachability

36

- ▶ F-Datalog and EARS can describe other recursions than regular ones (linear recursions)
 - Context-free recursions
 - Cross-recursions
- ▶ Then, we speak of **context-free graph reachability**
 - A context-free language describes graph reachability
- ▶ Application: interprocedural, whole-program analysis (see separate optional chapter)
 - Interprocedural IDFS framework (Reps)

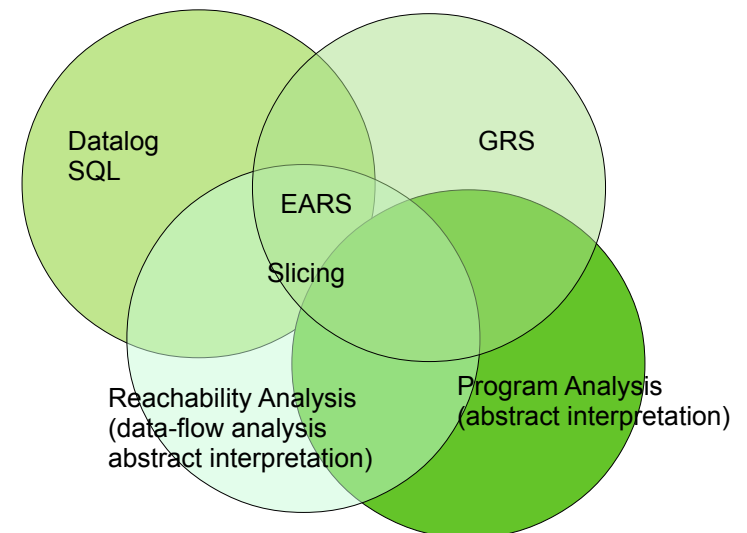
Program and Model Analyses Covered by Graph Reachability

- ▶ Reachability Analysis is a simple form of abstract interpretation
 - Slicing is a Single-Source-Multiple-Target reachability analysis
- ▶ Every abstract interpretation where a mapping of the abstract domains to graphs can be found.
 - monotone and distributive data-flow analysis
 - control flow analysis
 - SSA construction
 - Interprocedural IDFS framework (Reps)

51.4 More on the Logic-Graph Isomorphism

The Common Core of Logic, Graph Rewriting and Program Analysis

- Graph rewriting, DATALOG and data-flow analysis have a common core: EARS



Relation DFA/DATALOG/GRS

- 41
- ▶ Abstract interpretation (Data-flow analysis), DATALOG and graph rewrite systems have a common kernel: EARS
 - As DATALOG, graph rewrite systems can be used to query the graph.
 - ▶ Contrary to DATALOG graph rewrite systems materialize their results instantly.
 - Graph rewriting is restricted to binary predicates and always yields all solutions.
 - ▶ Graph rewriting can do transformation, i.e. is much more powerful than DATALOG.
 - Graph rewriting enables a uniform view of the entire optimization process
 - There is no methodology on how to specify general abstract interpretations with graph rewrite systems
 - In interprocedural analysis, instead of chaotic iteration special evaluation strategies must be used [Reps95] [Knoop92].
 - Currently strategies have to be modeled in the rewrite specifications explicitly.



51.5 Implementation in Tools

43



Relation DFA/DATALOG/GRS

- 42
- ▶ Uniform Specification of Analysis and Transformation
 - If the program analysis (including abstract interpretation) is specified with GRS
 - It can be unified with program transformation



Efficient Evaluation Algorithms from Logic Programming

- 44
- ▶ Tool OPTIMIX uses the „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
 - ▶ F-DATALOG optimization techniques can be employed
 - Bottom-up evaluation is normal, as in F-Datalog
 - Top-down evaluation as in Prolog possible, with resolution
 - semi-naive evaluation
 - index structures
 - magic set transformation
 - transitive closure optimizations



Related Tools

45

- ▶ Fujaba and MOFLON graph rewrite systems
 - TGG for Model Mapping
 - QVT Relational is very similar to TGG
 - See chapter MOFLON and course ST-II
- ▶ AGG graph rewrite system (From Berlin)
- ▶ VIATRA2 graph rewrite system
- ▶ Program Analysis Generators
 - PAG (Alt, Martin)
 - Sharlit (Tijang)
 - MetaFrame with modal logic (Knoop, Steffen)
 - Slicing-Tools (Reps, Field/Tip, Kamkar)



Terminology for Automated Graph Rewriting and Graph Reachability

47

- ▶ **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)



The End - Appendix Comprehension Questions

46

- ▶ Explain program slicing
- ▶ Why is regular graph reachability “regular”?
- ▶ How do you create a model mapping with regular graph reachability?
- ▶ Explain a typical data-flow analysis

