ARAL: A LANGUAGE FOR INFORMATION EXCHANGE BETWEEN PROGRAM ANALYSIS TOOLS

15. Kolloquium Programmiersprachen
Austria, October 12-14, 2009

Dr. Markus Schordan

Deputy Program Director of Multimedia and Software Engineering

Deputy Program Director of Game Engineering and Simulation

UAS Technikum Wien

> So spannend kann Technik sein.

FACHHOCHSCHULE
TECHNIKUM WIEN

Motivation

- Combination and comparison of analyzers
  - Using different intermediate representations
- Unified analysis results format
  - SATIrE, ROSE [R&D 100 AWARD 2009]
  - Connection SATIrE <-> PAG/aiT( ), SWEET(MDH)

SATIrE/ROSE Analyses

Analysis Name	Implementation Language	Input	Flow Sensitive	Context Sensitive
„classic analyses“ (RD, AE, LV, CP)	FULA (PAG)	ICFG	Yes	Yes
Shape	FULA (PAG)	ICFG	yes	Yes
Points-to	C++	ROSE-AST	No	Yes
Type-Based Alias	C++	ROSE-AST	No	No
Interval	FULA (PAG)	ICFG	Yes	Yes
Loop-Bound	Prolog (+Constraints)	Intervals	No	No

Scope of ARAL
Analysis Results Annotation Language

- Program summaries
- Function summaries
- Flow-sensitive analysis results
- Context-sensitive analysis results
- Constraints

Example: RD Analysis

```
/* input program */
int main() {
  int a,b,c;
  a=3;
  b=a;
  while(a<10) {
    if(a<b) {
      a=a+1;
    } else {
      b=b+1;
    }
    c=a+b;
  }
  a=c;
  return 0;
}
```

1. Front End: reads program (EDG)
2. Generate AST (ROSE)
3. Generate ICFG & Mappings (SATIrE)
4. Run RD analysis on ICFG (PAG)
5. Annotate AST with ARAL (SATIrE)
6. Back End: generates C/C++ program (ROSE)

ARAL: Source-Code Annotations

RD

Reaching Definitions

```
int compute_sum() {
…
#pragma ARAL RD@5 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
while(a < 10){
  #pragma ARAL RD@8 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
  if (a < b) {
    #pragma ARAL RD@10 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
    a = (a + 1);
    #pragma ARAL RD@10 post {(b,@12),(c,@-1),(b,@11),(a,@10),(c,@7)}
  } else {
    #pragma ARAL RD@11 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
    b = (b + 1);
    #pragma ARAL RD@11 post {(c,@-1),(b,@11),(c,@7),(a,@10),(a,@13)}
  }
  #pragma ARAL RD@8 post {(b,@12),(c,@-1),(b,@11),(a,@13),(a,@10),(c,@7)}
  #pragma ARAL RD@7 pre {(b,@12),(c,@-1),(b,@11),(a,@13),(a,@10),(c,@7)}
  c = (a + b);
  #pragma ARAL RD@7 post {(b,@12),(b,@11),(c,@7),(a,@13),(a,@10)}
}
#pragma ARAL RD@5 post {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
... }
```

ARAL: External File

```
ANALYSIS
    MAPPING
            abstract_loc = tuple(string,string),
            map(label,abstract_loc) = { …
            (@5,(”compute_sum”,”L”)),
            (@8,(”compute_sum”,”L(B)”)),
            (@10,(”compute_sum”,”L(B(S,_))”)),
            (@11,(“compute_sum”,”L(B(_,S))”)),
            (@7,(“compute_sum”,”L(BS)”)), …
            };
    RESULT
            NAME RD
            TYPE set(tuple(identifier,label))
            DATA
                { … …. }

    END
END
```

Abstract Source Code Locations (path-expressions)

RD

Reaching Definitions

```
@5 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
@8 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
@10 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
@10 post {(b,@12),(c,@-1),(b,@11),(a,@10),(c,@7)}
@11 pre {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
@11 post {(c,@-1),(b,@11),(c,@7),(a,@10),(a,@13)}
@8 post {(b,@12),(c,@-1),(b,@11),(a,@13),(a,@10),(c,@7)}
@7 pre {(b,@12),(c,@-1),(b,@11),(a,@13),(a,@10),(c,@7)}
@7 post {(b,@12),(b,@11),(c,@7),(a,@13),(a,@10)}
@5 post {(b,@12),(c,@-1),(b,@11),(c,@7),(a,@13),(a,@10)}
```

ARAL Language Constructs

- Mappings
  - From Labels to abstract source code locations
  - From IDs to syntactic constructs of the analyzed language
- Labels ‘@‘
- IDs ‘#‘
  - functions, statements, expressions, variables
  - Type-based equivalence-relation for expressions
- Collection types: set, list, map, tuple
- Basic types: number(s), string, identifier
- Constraints

Compactness of Data

- Three different cases of client:
  1) Sole use of analysis results (by other tool)
  2) Transfer-functions are available
  3) Analyzer is available
- For each case 1 to 3, the analysis results can be more compact/smaller
  - e.g. single iteration analysis (K. Klohs, COCV'08)

Transforming ARAL into ASSERTIONS

- Example (Interval-Analysis, Constraints, Assert)
  - #pragma ARAL interval {(x,(5,10)),…}
  - #pragma ARAL constraint ${ x>=5 and x<=10,..}$
  - assert(x>=5 && x<=10)
- Allows testing of analyzers
- Formal verification engine to prove assertions (e.g. software model checkers Blast,CBMC)

Implementation

- Independent from other tools
- Follows source-to-source approach
- ARAL Front End
- Intermediate representation (IR)
  - Object-oriented AST
  - Visitor design pattern, deepCopy, operators
- Back End (implemented with Visitor)
  - ARAL file can be generated from IR
  - Any other format can be easily generated

Conclusions

- ARAL can be used for
  - annotating source-code
  - in a separate file for analysis results exchange between tools
- Level of analysis information representation
  - Mapping capabilites of tools
  - Extension: operators for post-analysis computions in ARAL
  - Adaptors between analysis components