Interactive, Scalable, Declarative Program Analysis: From Prototype to Implementation

William C. Benton
University of Wisconsin

willb@cs.wisc.edu
http://pages.cs.wisc.edu/~willb/

Joint work with Charles N. Fischer

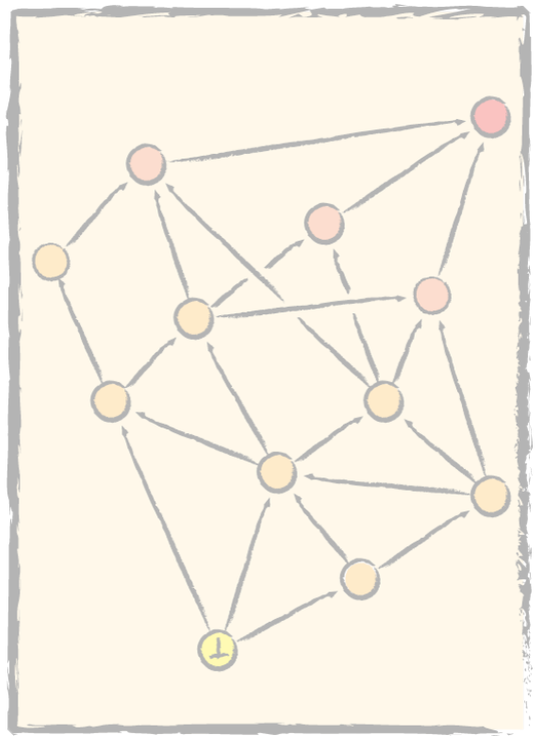
Logic programming is an expressive and effective tool for program analysis.

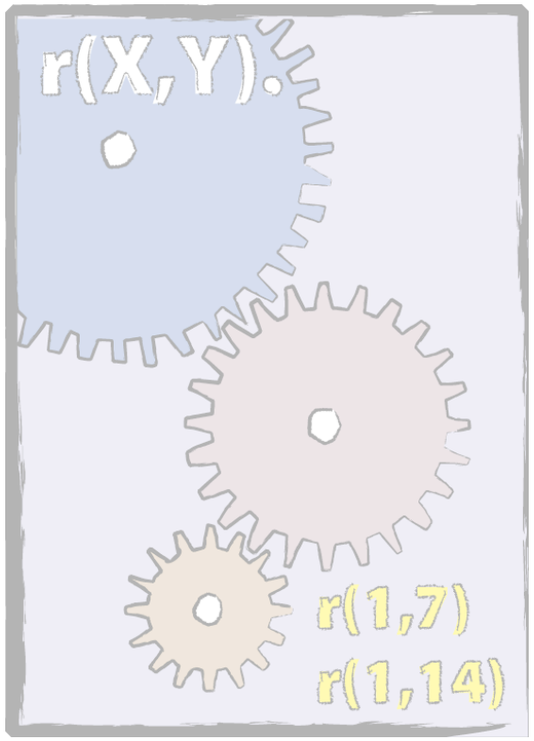
Logic programming is an expressive and effective tool for program analysis.

Logic programming is an expressive and effective tool for program analysis.

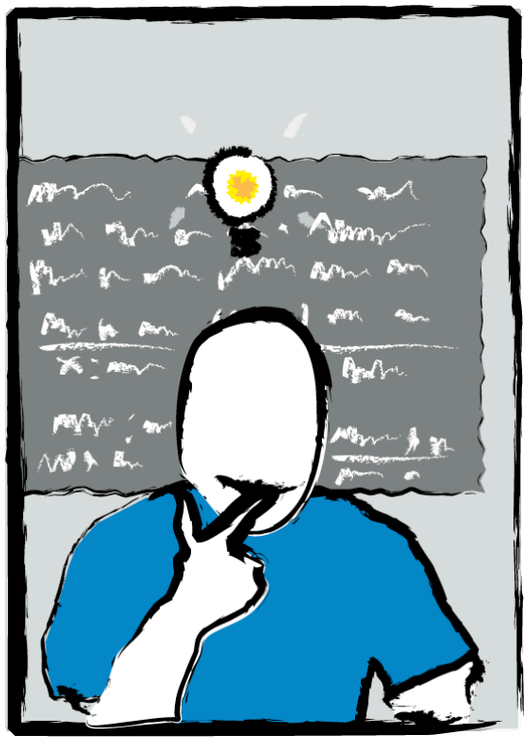
Logic programming is an expressive and effective tool for program analysis.

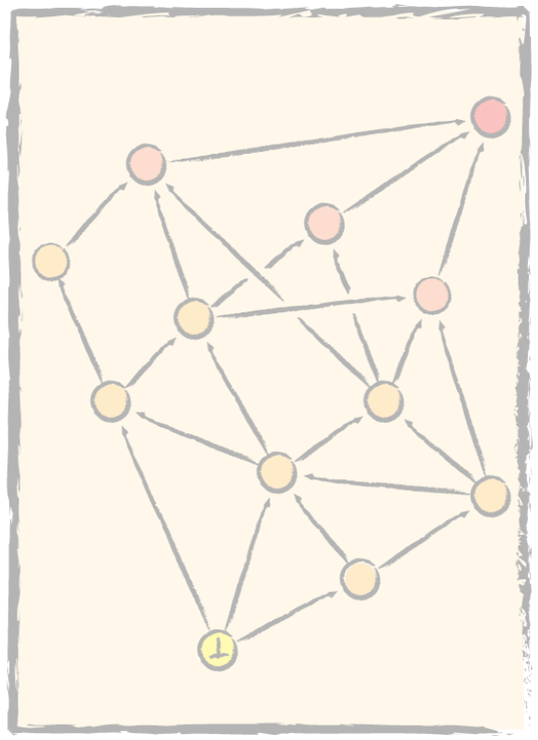
Four activities in analysis design

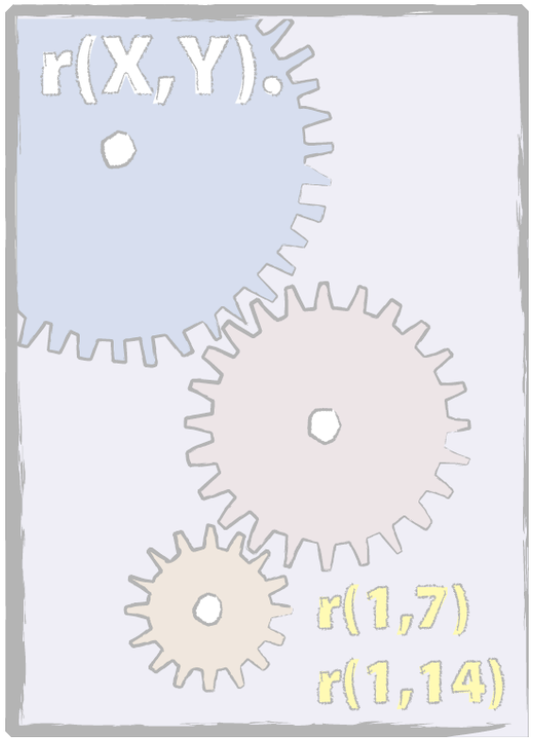




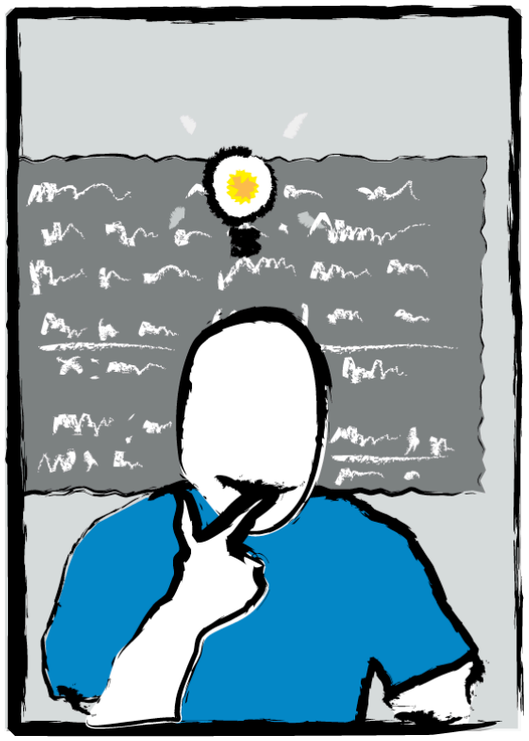
Four activities in analysis design

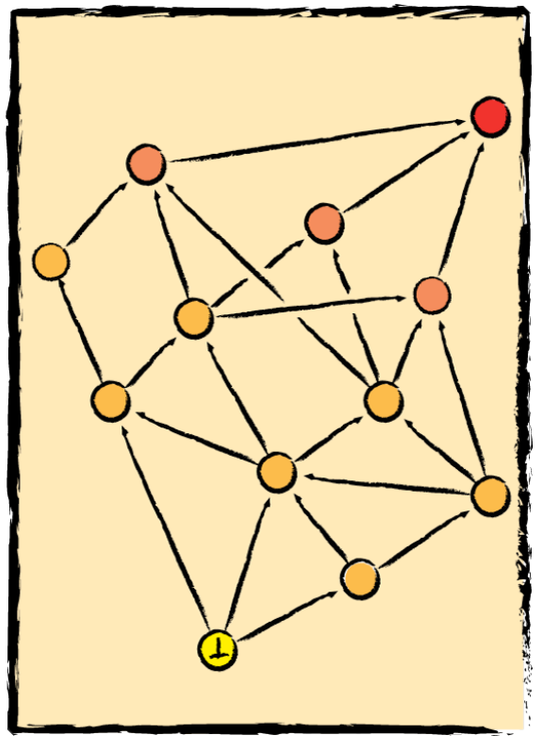


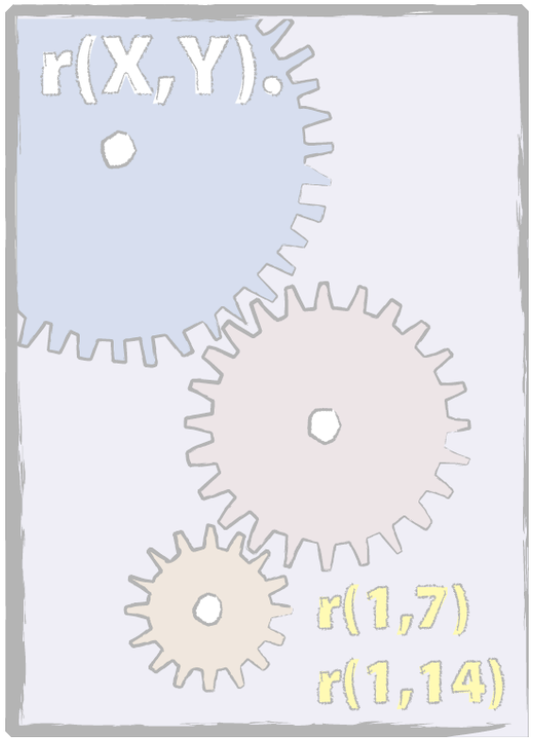




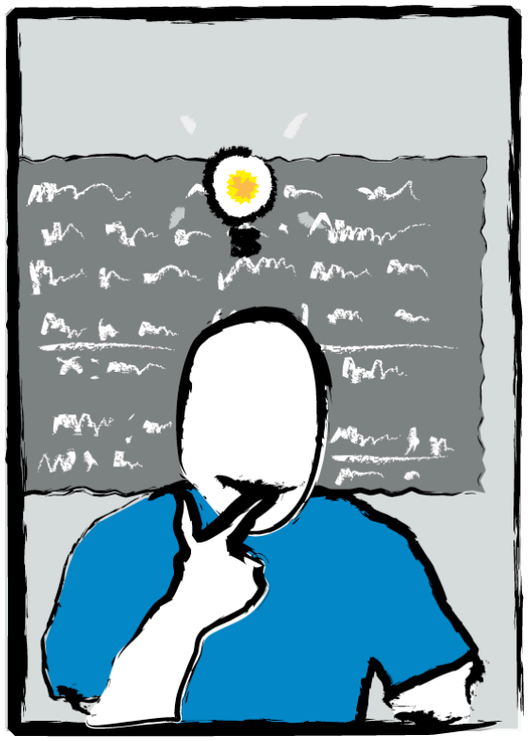
Four activities in analysis design

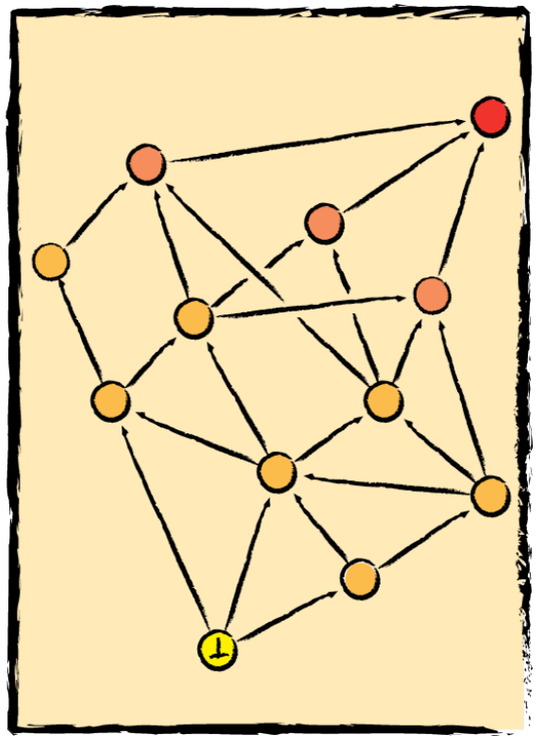


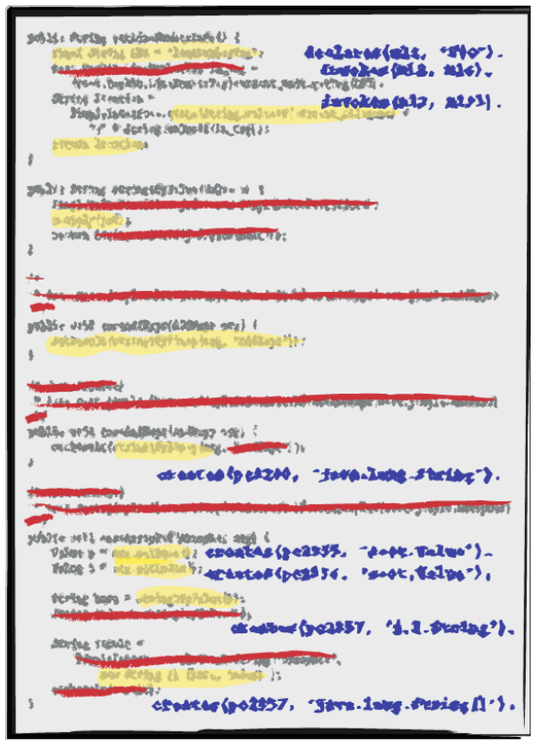


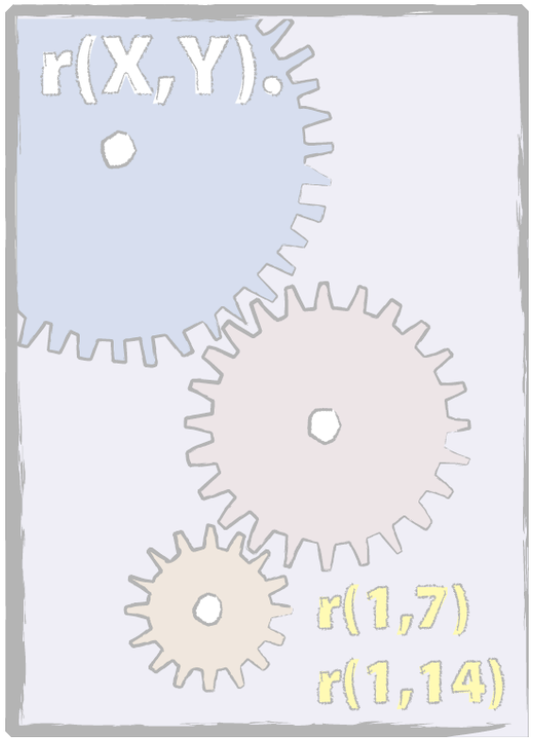


Four activities in analysis design

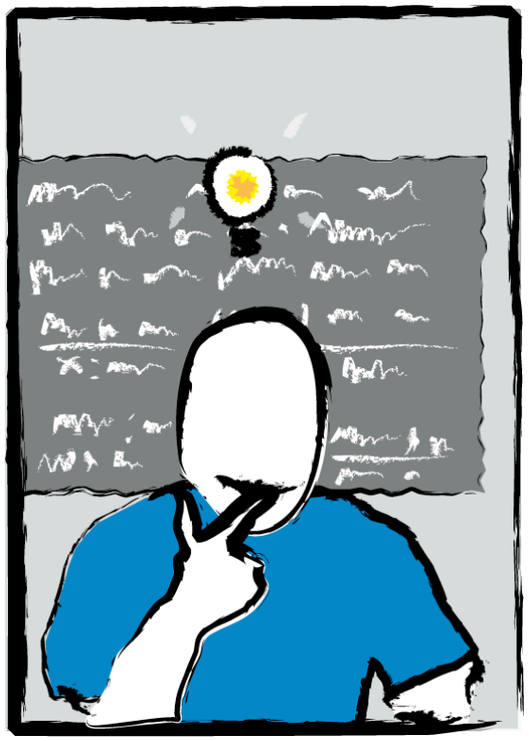


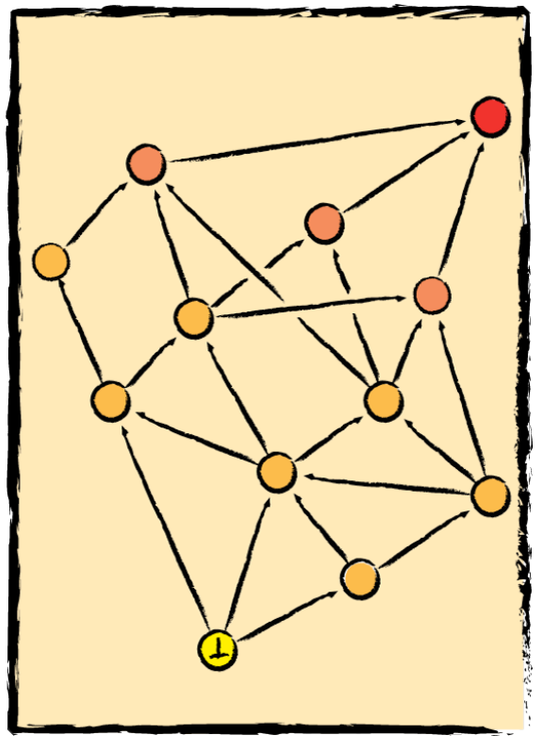


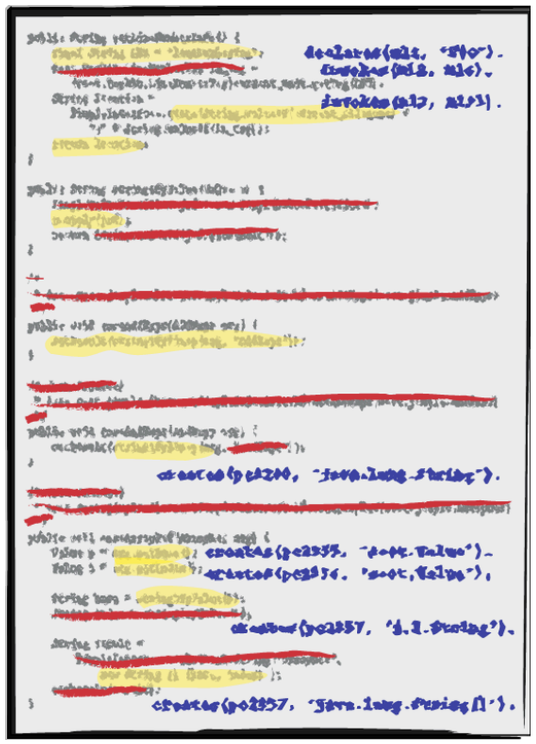


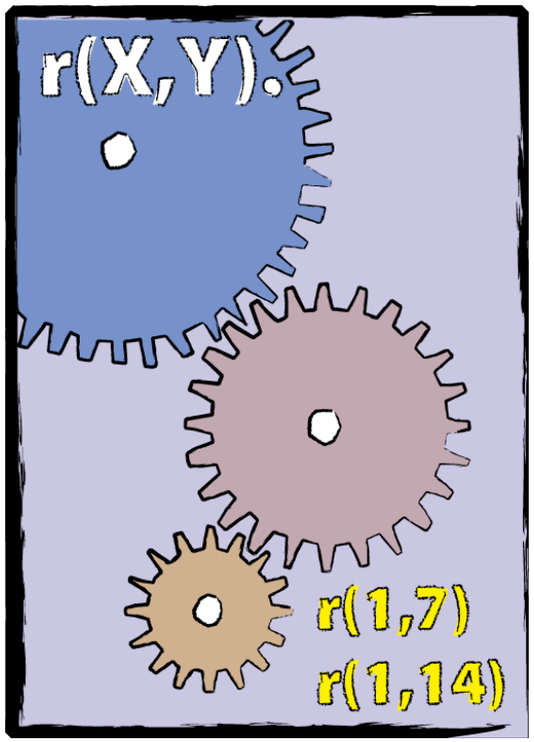


Four activities in analysis design

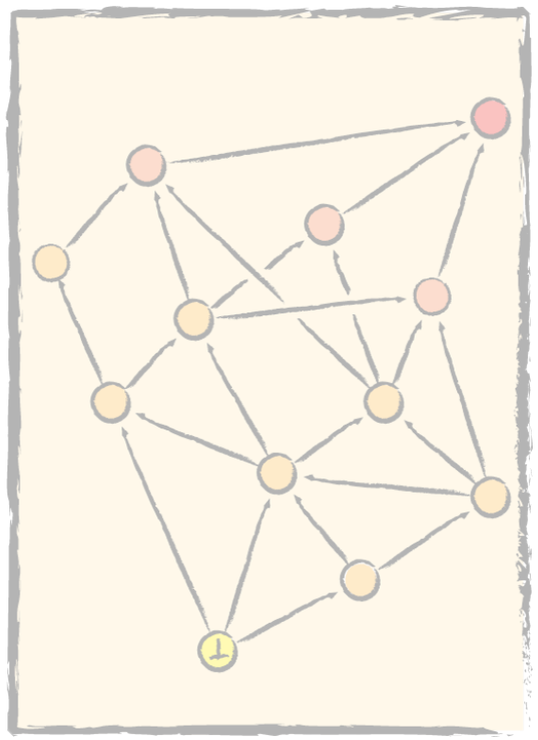


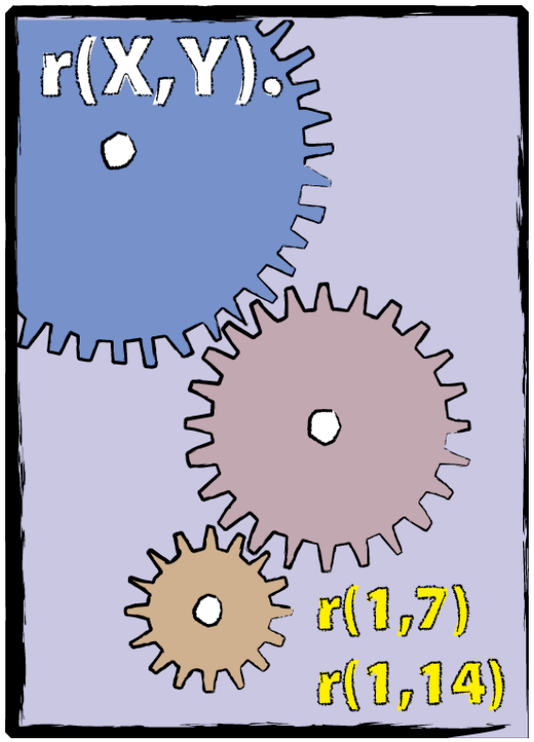


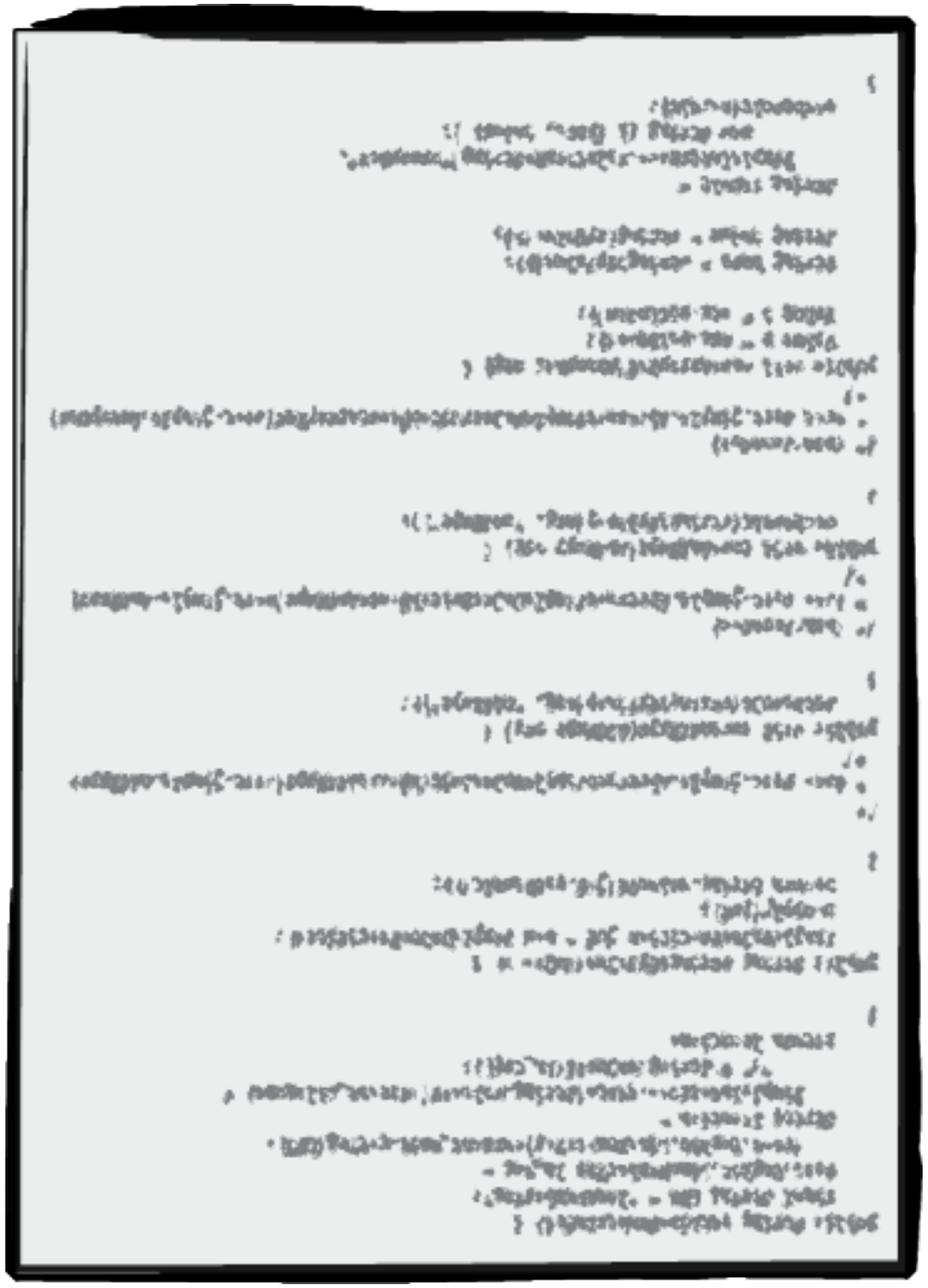


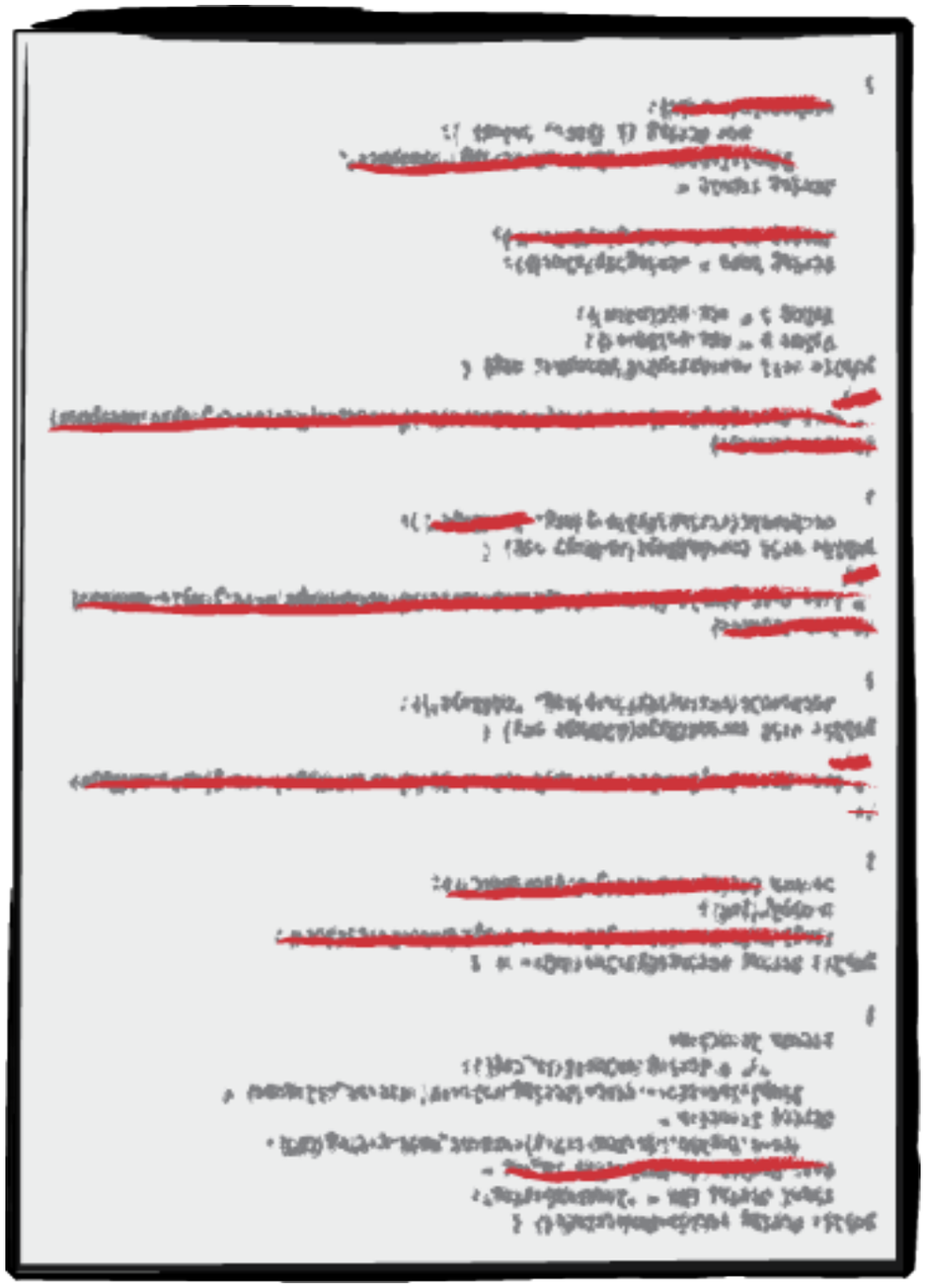


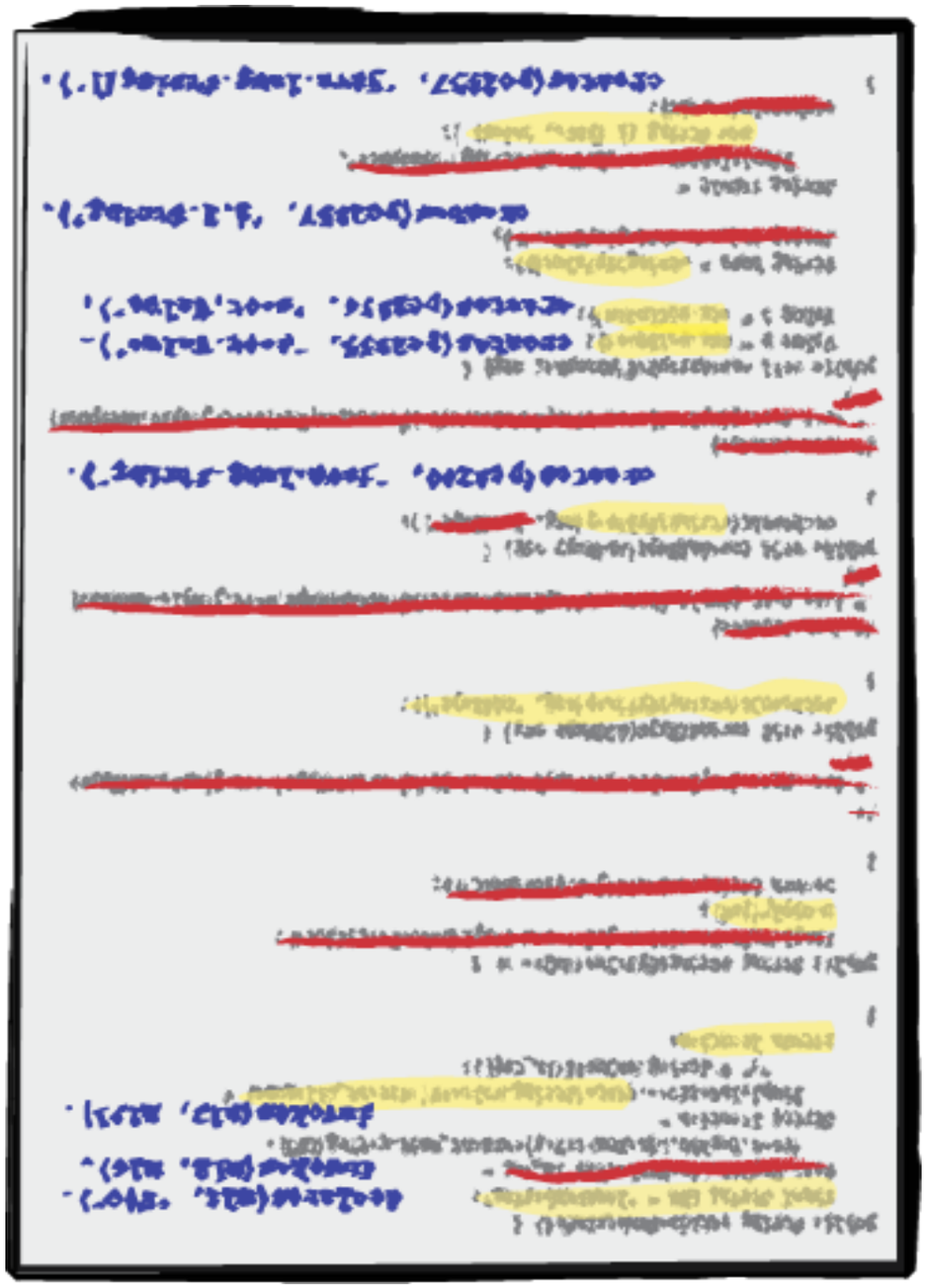
Four activities in analysis design

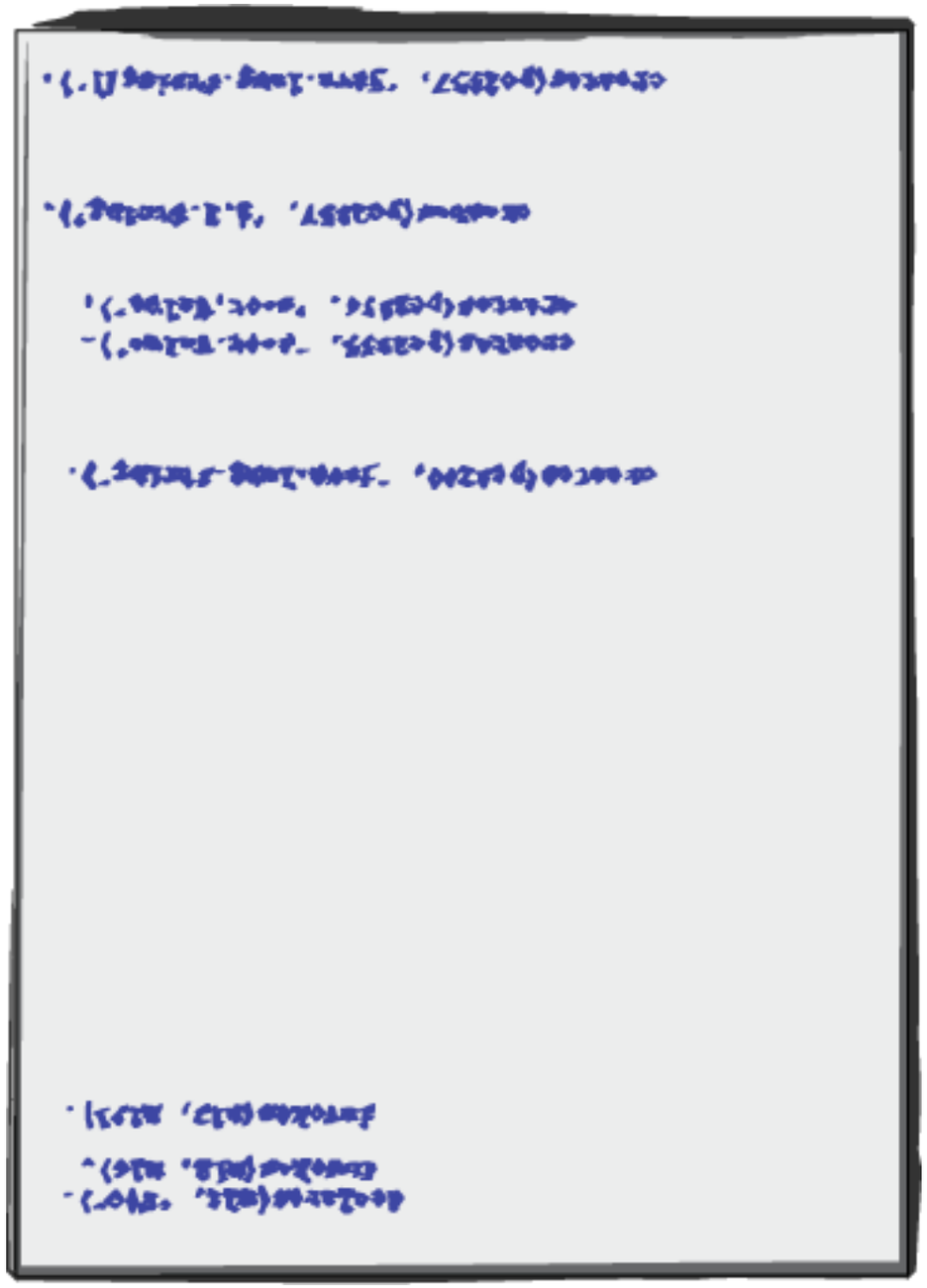








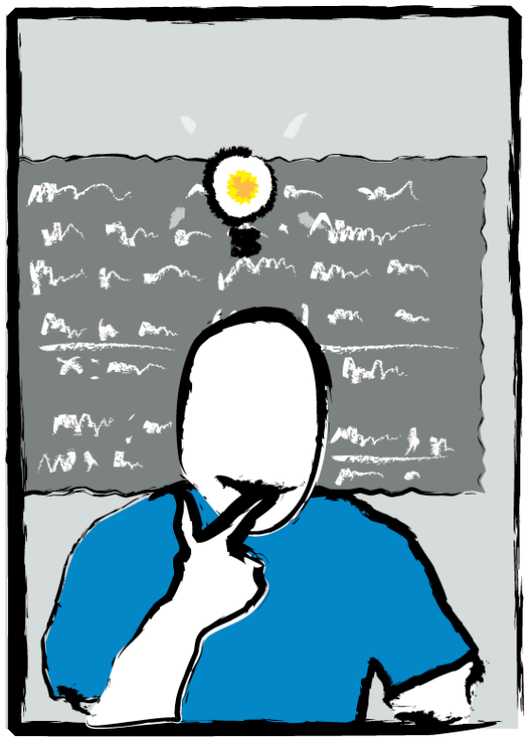


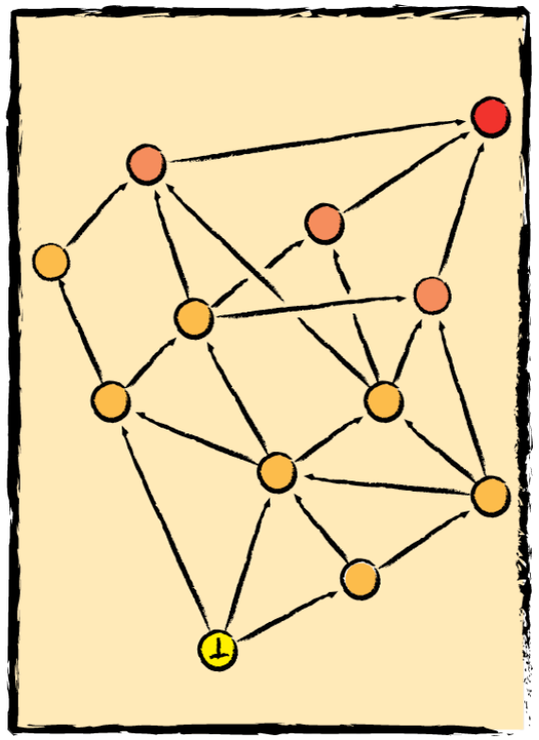


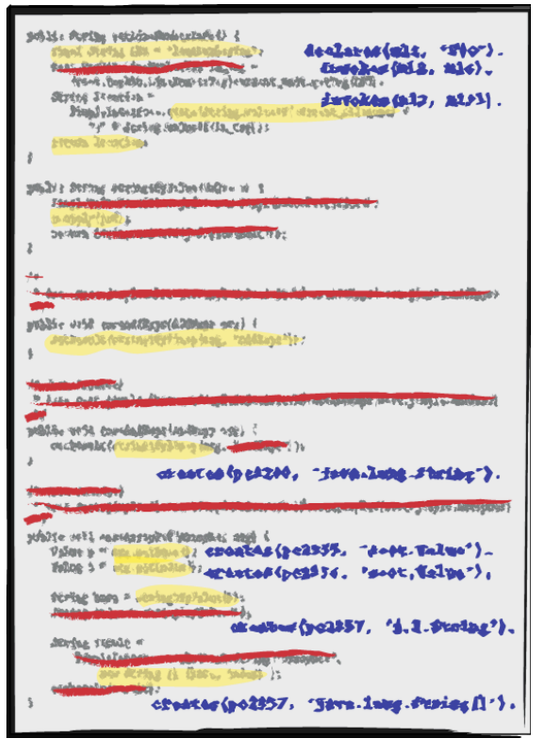
Our contribution: DIMPLE

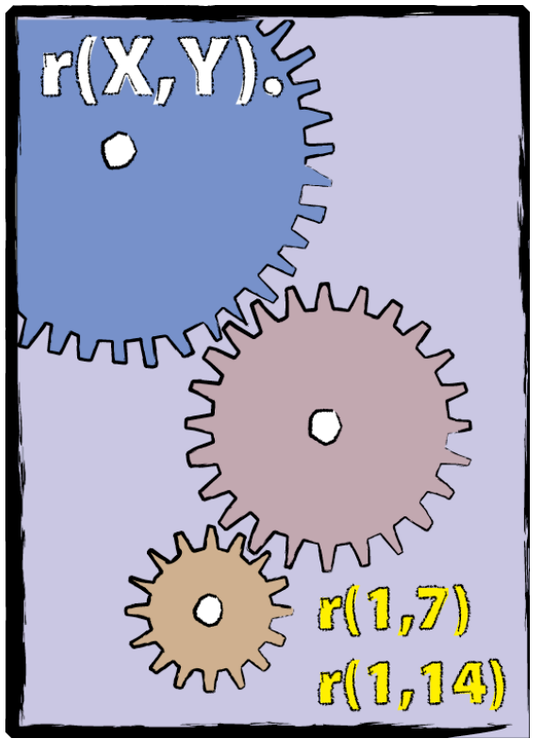
- Analysis framework for Java bytecodes
- Flexible enough for interactive use
- Fast enough for production
- Techniques scale to realistic programs

DIMPLE supports each activity!









Tabled Prolog

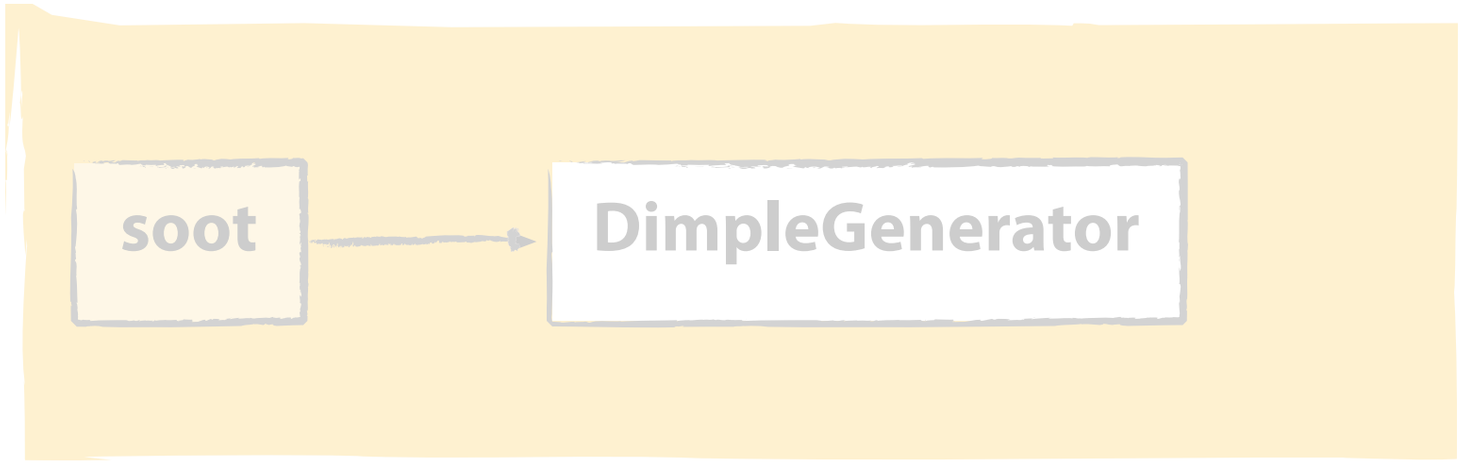
- Some predicates use Prolog-style search
- Some predicates are *tabled*
  - Stores subgoal results in a trie
  - Identifies SCCs, finds finite fixed points
  - Bounded term-size implies termination

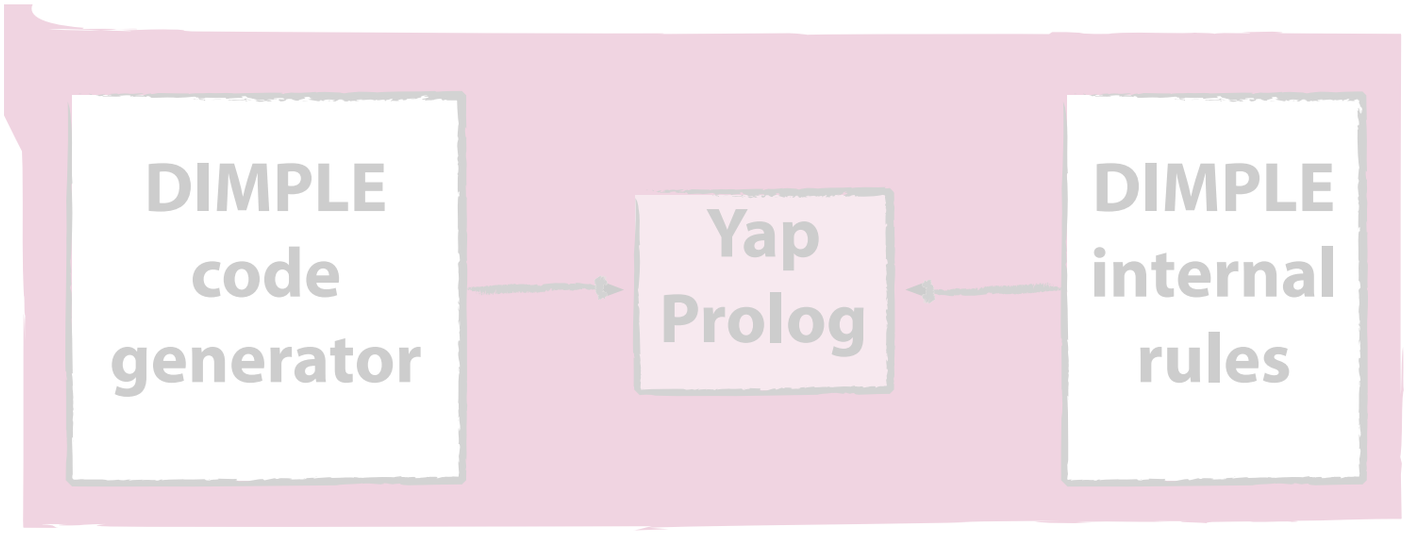
Forecast

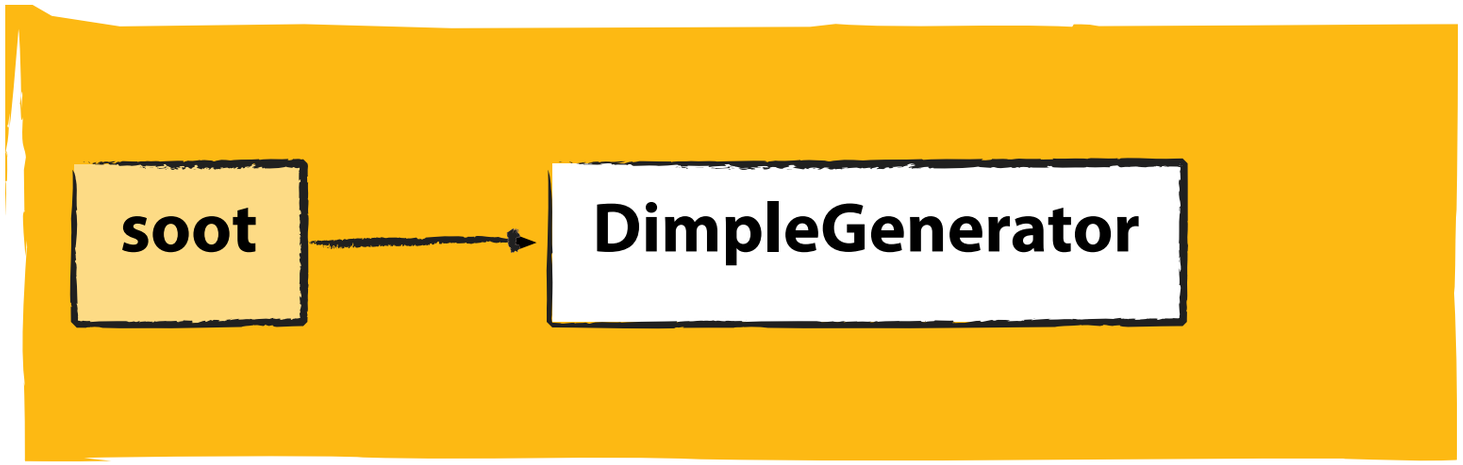
- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen's analysis

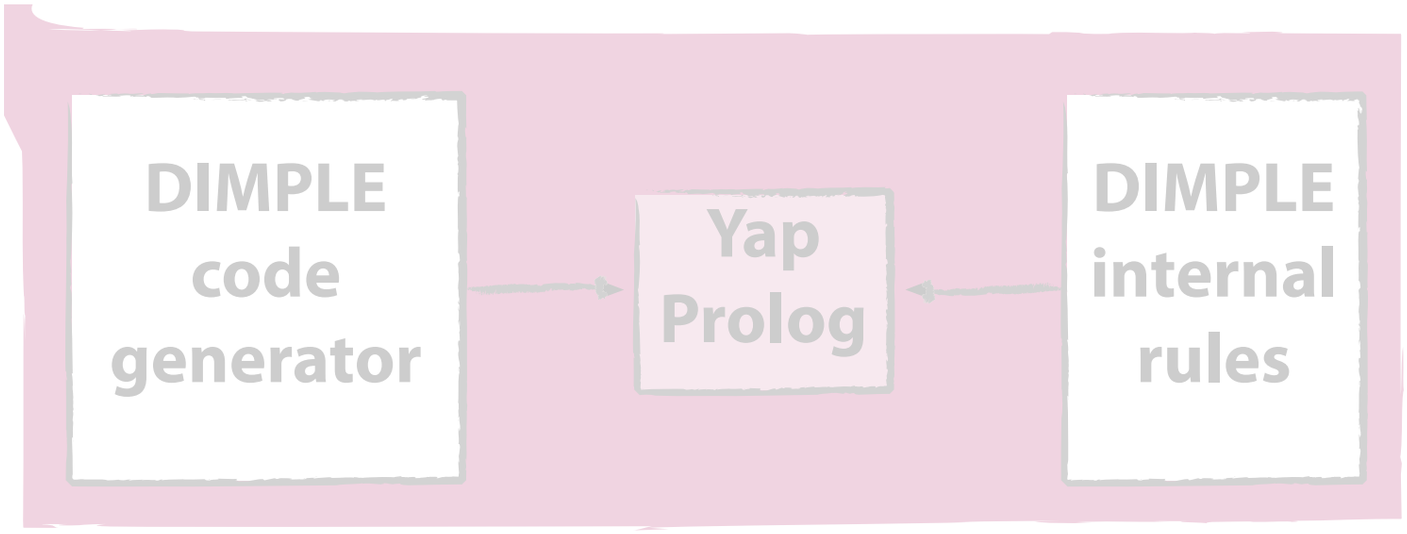
Forecast

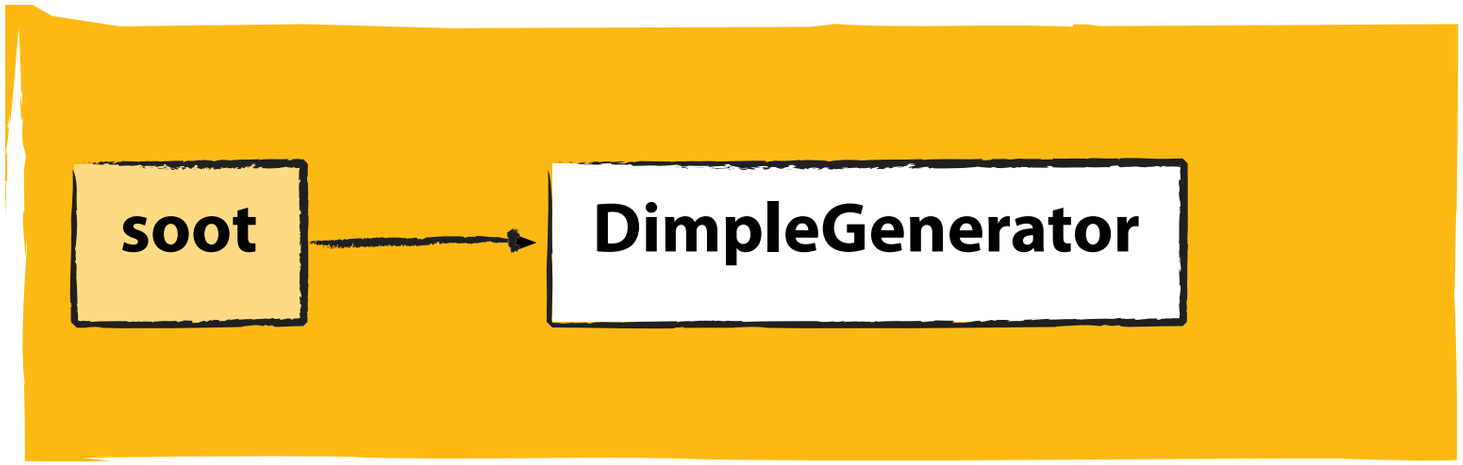
- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen's analysis

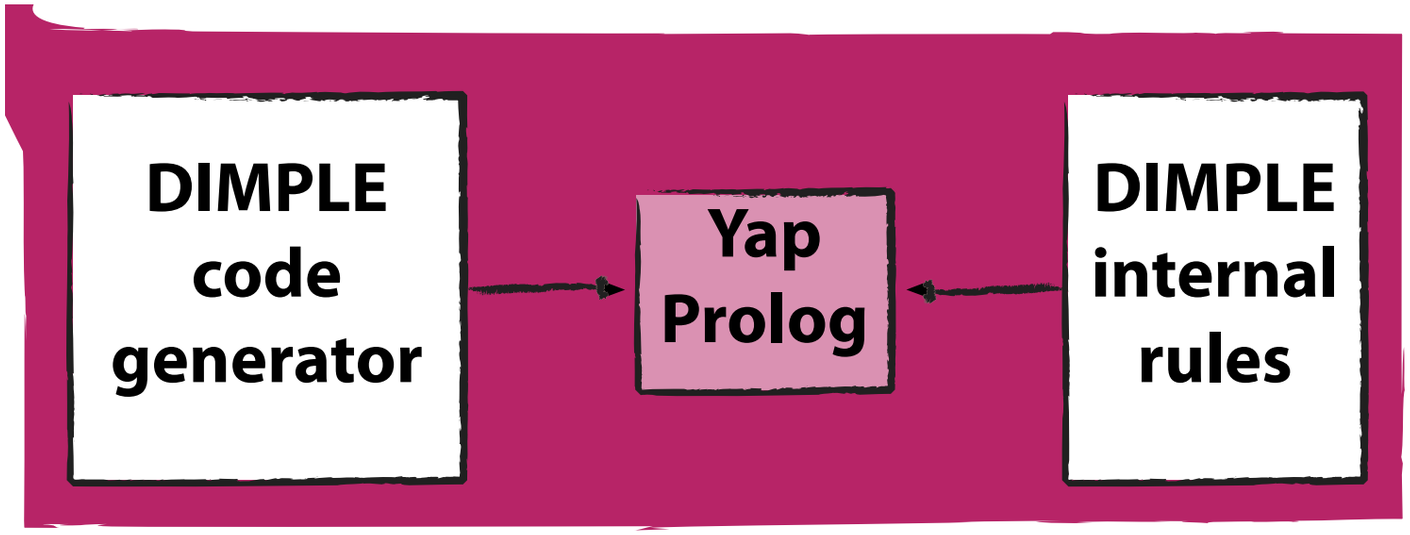

soot
DimpleGenerator

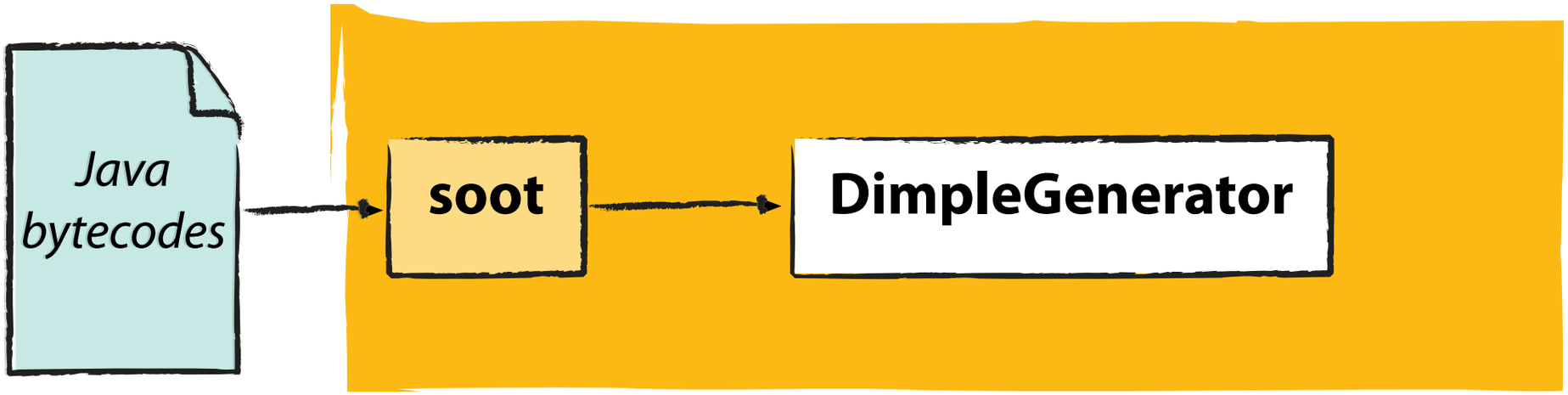

DIMPLE code generator
Yap Prolog
DIMPLE internal rules

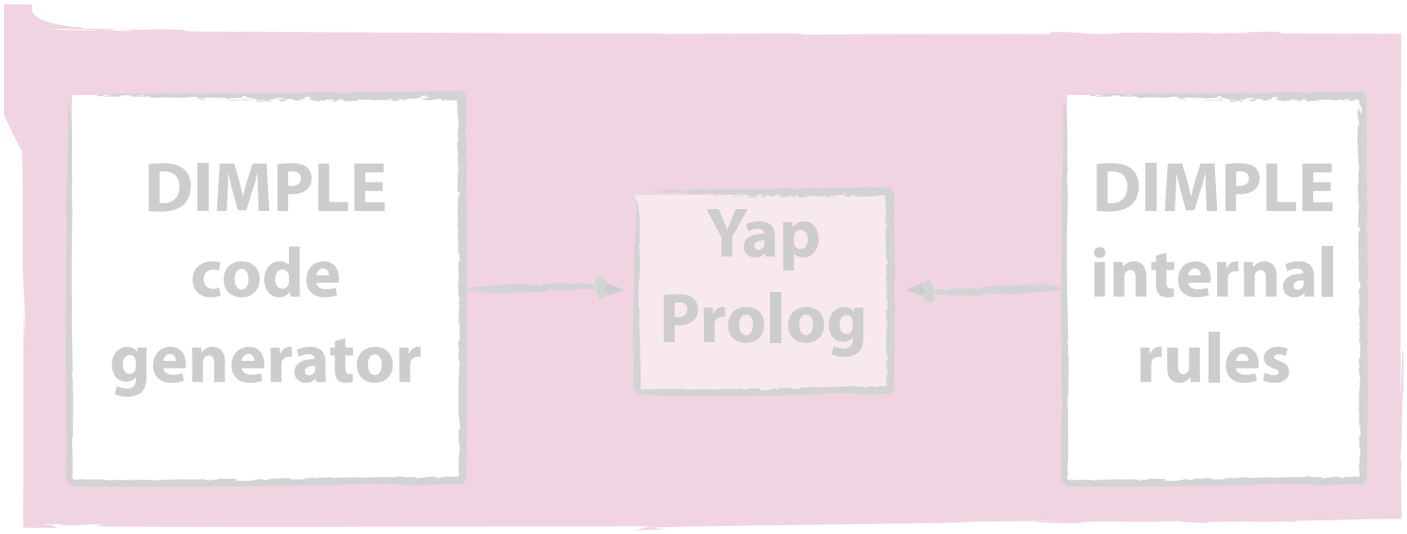

soot
DimpleGenerator

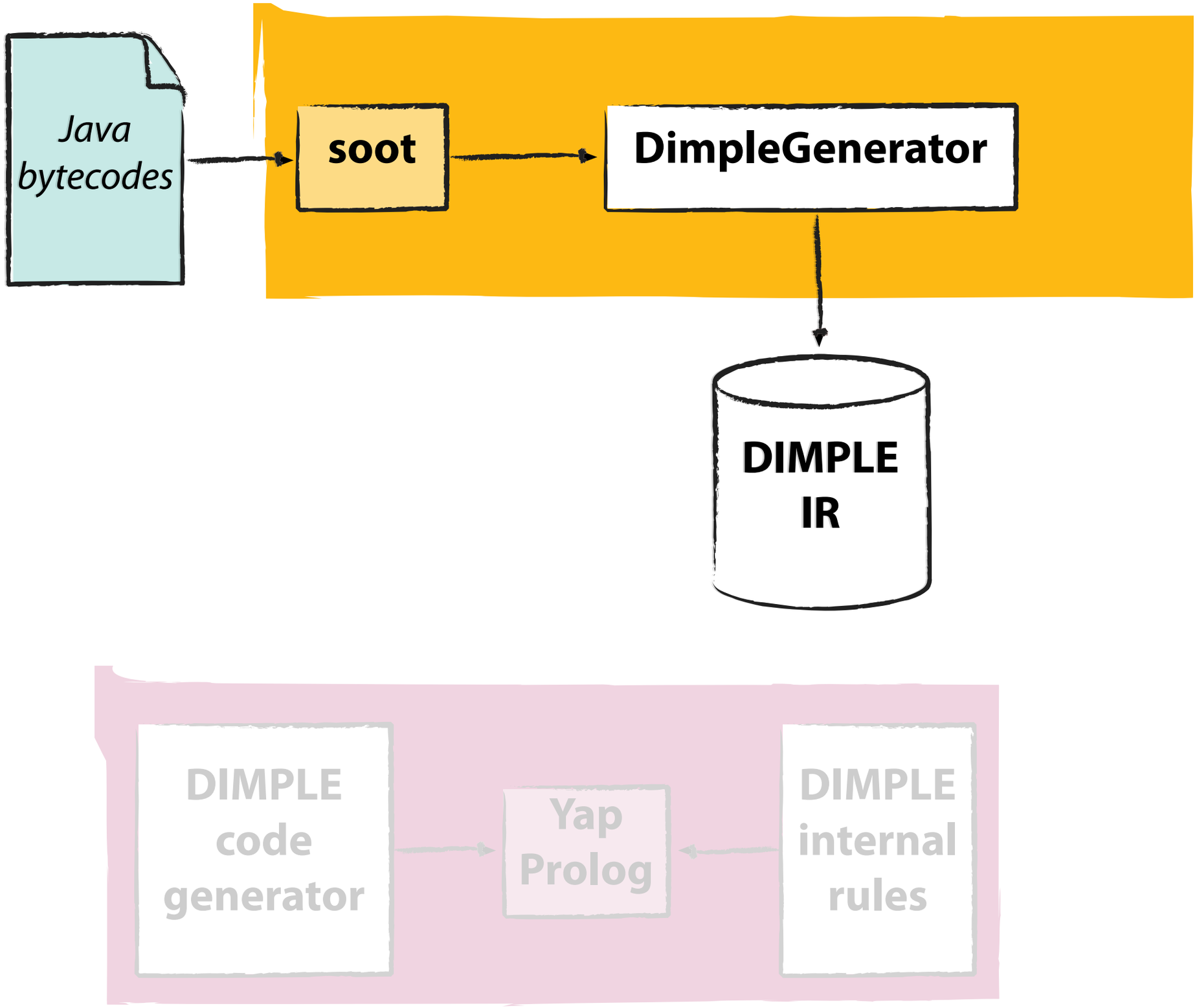

DIMPLE code generator
Yap Prolog
DIMPLE internal rules

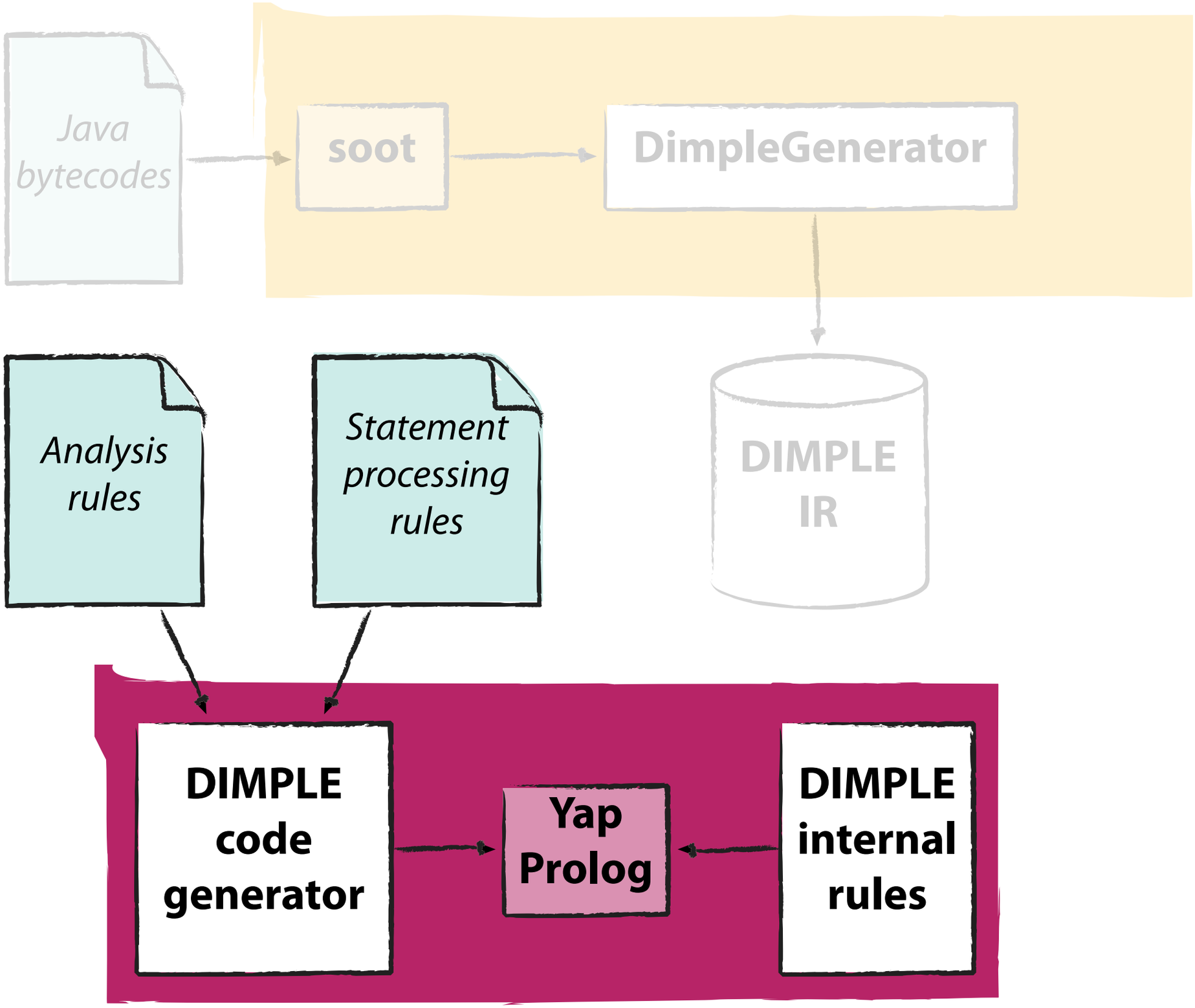

soot
DimpleGenerator

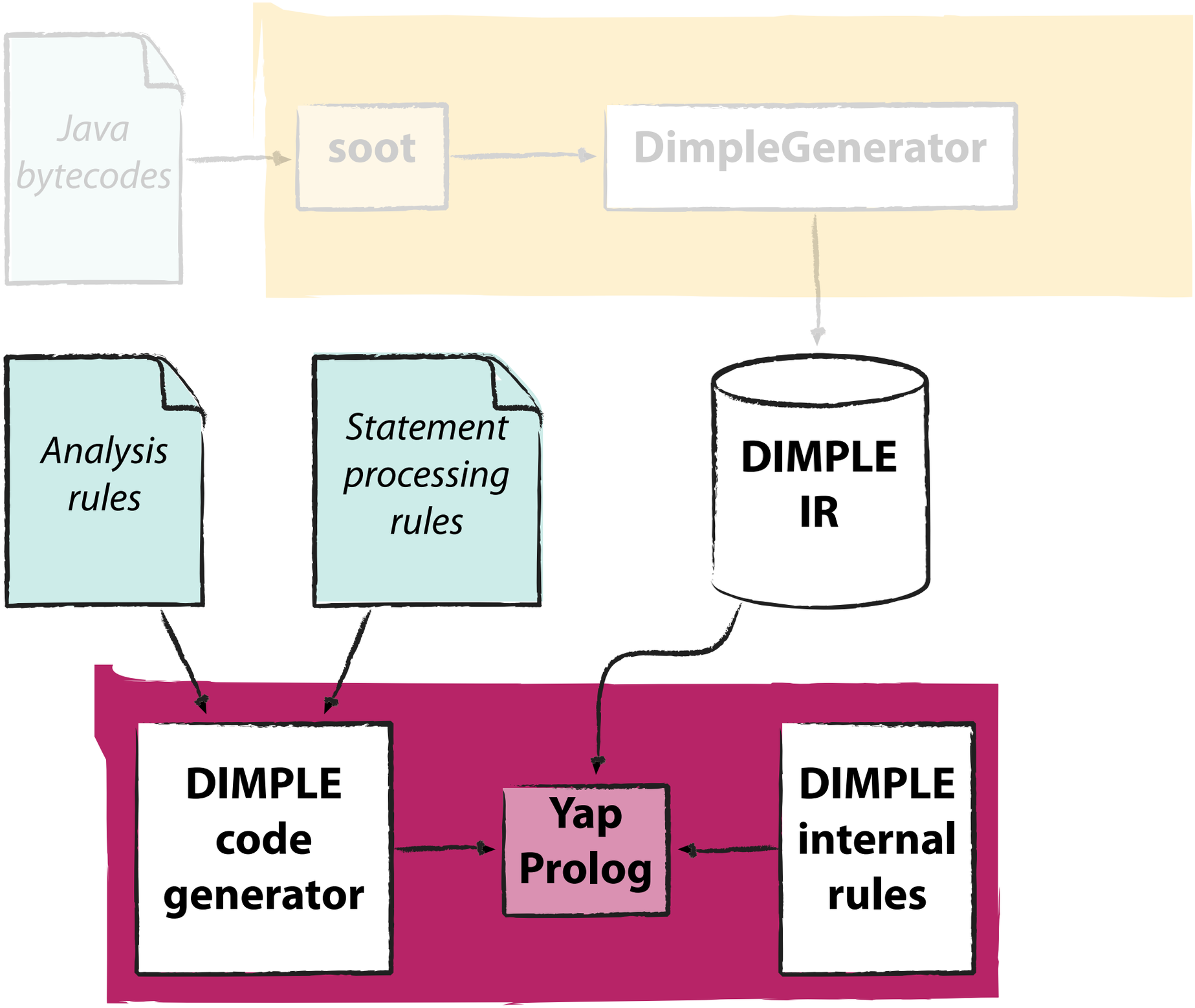

DIMPLE code generator
Yap Prolog
DIMPLE internal rules

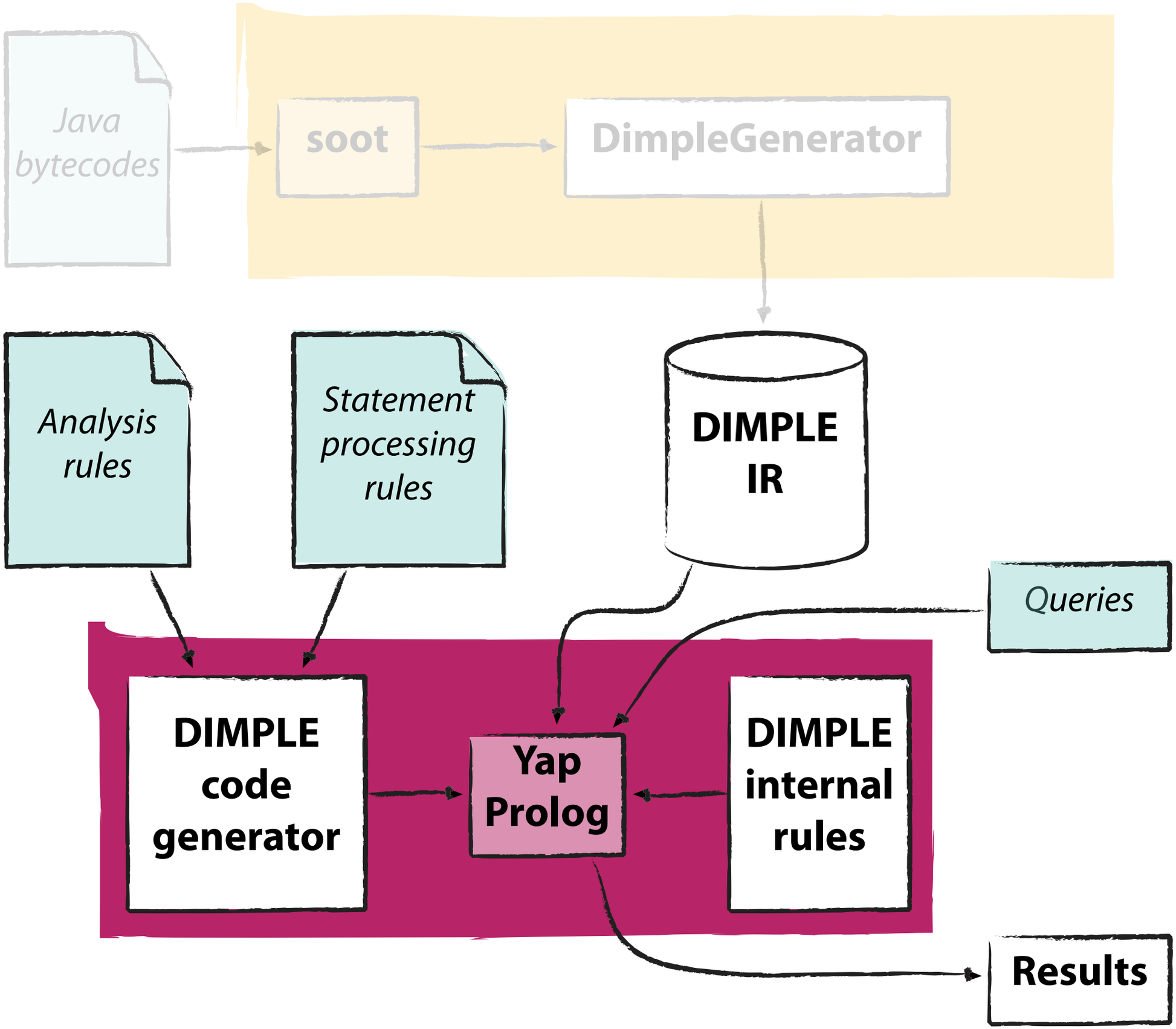

Java bytecodes
soot
DimpleGenerator


DIMPLE code generator
Yap Prolog
DIMPLE internal rules


Java bytecodes
soot
DimpleGenerator
DIMPLE IR
DIMPLE code generator
Yap Prolog
DIMPLE internal rules


Java bytecodes
soot
DimpleGenerator
DIMPLE IR
Analysis rules
Statement processing rules
DIMPLE code generator
Yap Prolog
DIMPLE internal rules


Java bytecodes
soot
DimpleGenerator
Analysis rules
Statement processing rules
DIMPLE IR
DIMPLE code generator
Yap Prolog
DIMPLE internal rules


Java bytecodes
soot
DimpleGenerator
Analysis rules
Statement processing rules
DIMPLE IR
Queries
DIMPLE code generator
Yap Prolog
DIMPLE internal rules
Results

The DIMPLE IR

- Describes whole program & libraries
  - Class hierarchy
  - Control flow
  - Conservative call graph
  - Every statement

DIMPLE IR statement example

```
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```

DIMPLE IR statement example

Program counter

```
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```

DIMPLE IR statement example

Program counter

Statement structure

```
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```

DIMPLE IR statement example

Program counter

```
stmt(units_736,
  assignStmt(
    local(locals_730, method(SIG), primtype('boolean')),
    intConstant(1))).
```

Statement structure

Local

DIMPLE IR statement example

Program counter

```
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```

Statement structure

Local

DIMPLE IR statement example

Program counter

```
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```

Statement structure

Local

Method signature

e.g. '<java.lang.Object: equals(Ljava/lang/Object;)Z>'

DIMPLE IR statement example

Program counter

```
stmt(units_736,
  assignStmt(
    local(locals_730, method(SIG), primtype('boolean')),
    intConstant(1))).
```

Statement structure

Type

Local

Method signature

e.g. '<java.lang.Object: equals(Ljava/lang/Object;)Z>'

Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen's analysis

Statement processing rules

Head <-- Body

Statement processing rules

```
allocSite(PC, Type) <--
    stmt(PC, assignStmt(local(L,M,T), newExpr(Loc,Type))).


reference(L) <--
    local(L,M,T), T \= primtype(P).


explicitlyThrows(Sig,T) <--
    containsStmt(Sig, PC),
    stmt(PC, throwStmt(local(L,M,T))).
```

Statement processing rules

```
allocSite(PC, Type) <--
    stmt(PC, assignStmt(local(L,M,T), newExpr(Loc,Type))).


reference(L) <--
    local(L,M,T), T \= primtype(P).


explicitlyThrows(Sig,T) <--
    containsStmt(Sig, PC),
    stmt(PC, throwStmt(local(L,M,T))).
```

Statement processing rules

```
allocSite(PC, Type) <--
    stmt(PC, assignStmt(local(L,M,T), newExpr(Loc,Type))).


reference(L) <--
    local(L,M,T), T \= primtype(P).


explicitlyThrows(Sig,T) <--
    containsStmt(Sig, PC),
    stmt(PC, throwStmt(local(L,M,T))).
```

Statement processing rules

```
allocSite(PC, Type) <--
    stmt(PC, assignStmt(local(L,M,T), newExpr(Loc,Type))).


reference(L) <--
    local(L,M,T), T \= primtype(P).


explicitlyThrows(Sig,T) <--
    containsStmt(Sig, PC),
    stmt(PC, throwStmt(local(L,M,T))).
```

Statement processing rules

- Act like filters on program database
- Body may contain arbitrary Prolog code
- Generate extra debugging information

Analysis rules

$$\mathit{Head} \Leftarrow\!= \mathit{Body}$$

Analysis rules

Analysis rules

x_r(a,b) <== r(a,b).

x_r(a,b) <== x_r(a,i), r(i,b).

Analysis rules

- Act like predicates over relations
- Typically execute as tabled code
  - User-directed option to disable
- Support tracing and debugging

Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen's analysis

Interactive development

- Whole program is available for queries
- From intuition to hypothesis to analysis
- Test and refine ideas, define new rules

Query example

*"Most field accesses are to/from **this**."*

```
stmt(Id, assignStmt(local(A,B,C),
  instanceFieldRef(local(L,M,T), F))),
isthis(L).

stmt(Id, assignStmt(local(A,B,C),
  instanceFieldRef(local(L,M,T), F))),
\+ isthis(L).
```

Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen's analysis

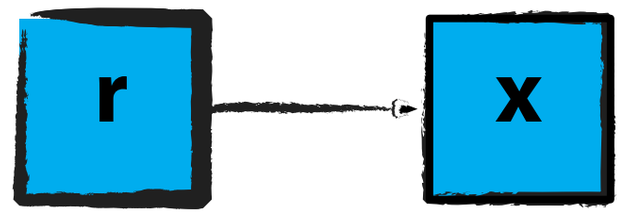
Andersen's analysis

- Points-to analysis
- Flow- and context-insensitive
- Subset-based

Andersen's analysis

```
pc: r = new C();
```

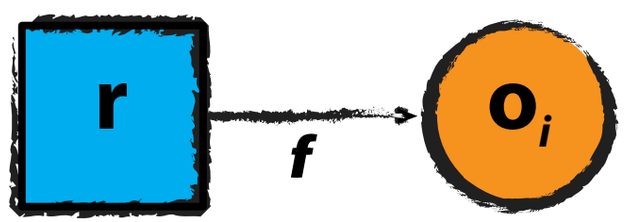

```
pc: r = x;
```




```
pc: r.f = x;
```

$\forall\, i : pt(r, o_i)$

```
pc: r = x.f;
```



$\forall\, i : pt(x, o_i)$

```
v_pt(Ref,Id) <== pt_immed(Ref,Id).

v_pt(Ref,Id) <== assign(Ref, Refl), v_pt(Refl, Id).

v_pt(Ref, Id) <== s_load(Ref, F), s_store(F, Refl), v_pt(Refl, Id).

v_pt(Ref, Id) <== formal(Ref, I, M), actual(Refl, I, M), v_pt(Refl, Id).

h_pt(Obj1, F, Obj2) <==
  store(Ref1, F, Ref2), v_pt(Ref1, Obj1), v_pt(Ref2, Obj2).

v_pt(Ref, Id) <==
  load(Ref1, F, Ref), v_pt(Ref1, Id1), h_pt(Id1, F, Id).

v_pt(Ref, Id) <==
  ret_caller(Ref, Method), ret_callee(Refl, Method), v_pt(Refl, Id).
```

(Rules adapted from Whaley and Lam, 2004.)

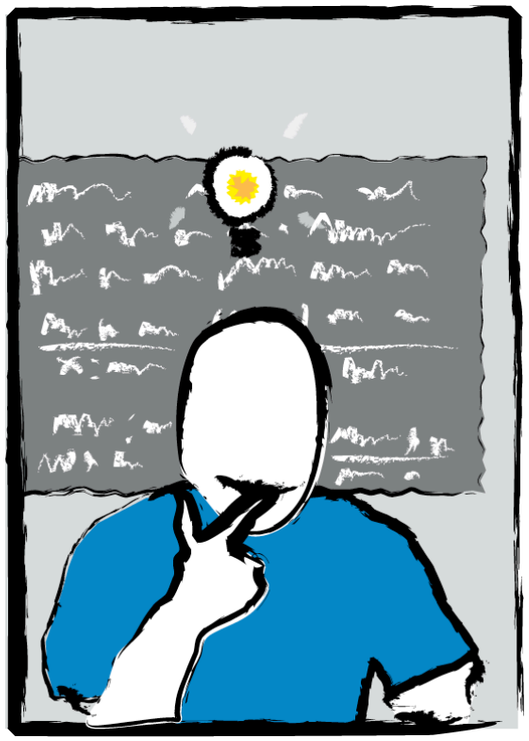
Andersen's analysis: performance results

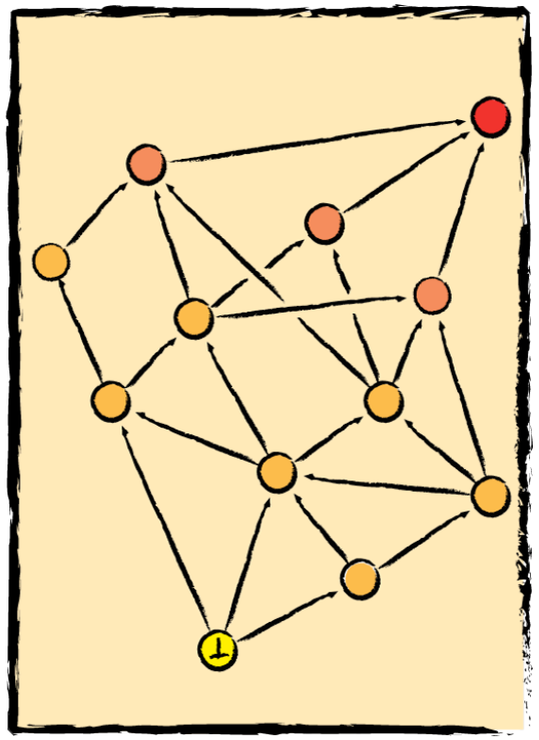
input	relations	outputs ($\times 10^6$)	CPU time (s.)
antlr	347,290	12.31	8.13
bloat	476,635	19.41	13.54
chart	586,337	4.74	3.88
eclipse	401,474	12.06	11.91
hsqldb	304,451	6.64	4.56
luindex	322,634	7.32	7.42
pmd	427,279	10.67	7.01
xalan	302,150	6.18	4.13

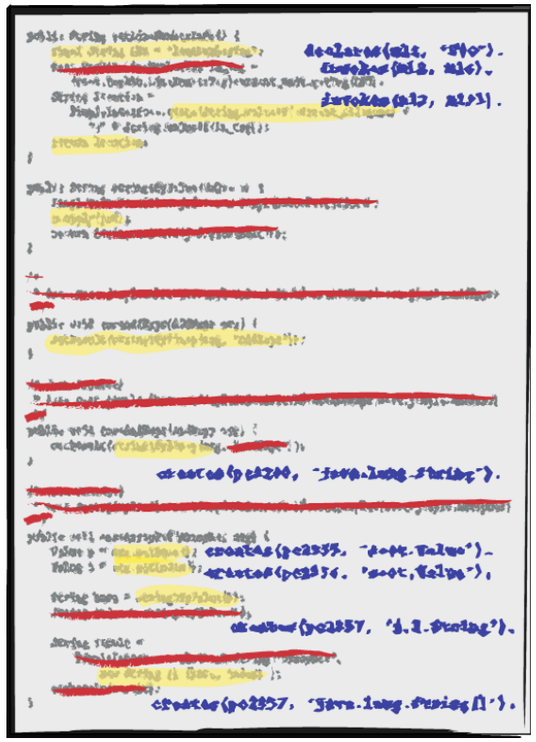
Summary

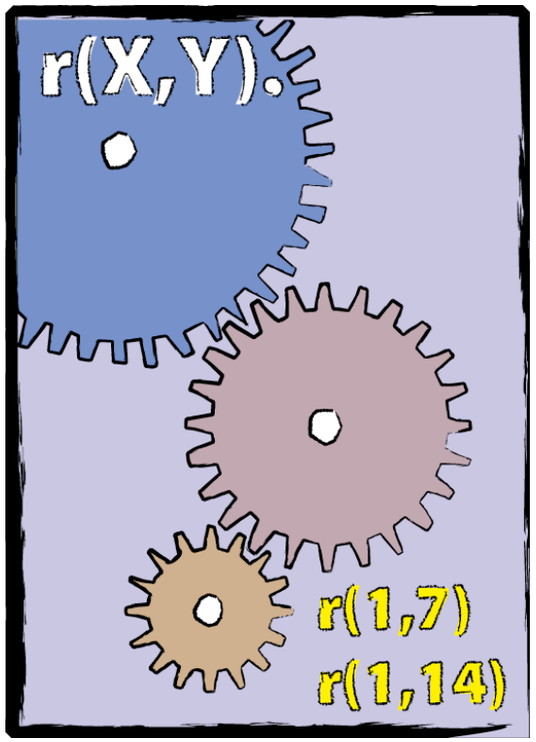
- Tabled Prolog is useful for analysis tasks
- DIMPLE: a framework for analysis
  - Logic programming used throughout
  - Suitable from prototype to production

Any questions?









http://pages.cs.wisc.edu/~willb/dimple/