Program Analysis with Datalog

Sergey Mechtaev
s.mechtaev@pku.edu.cn
Peking University

GitHub CodeQL – Language For Defining Custom Program Analyses

JavaConverter.java

```
public static Object deserialize (InputStream is)
    throws IOException {
  ObjectInputStream ois = new ObjectInputStream(is);
  return ois.readObject();
}
```

UnsafeDeserialization.ql

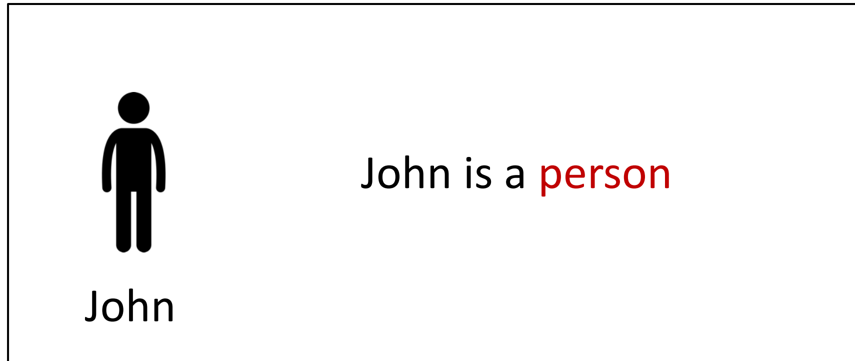
```
from PathNode source, PathNode sink
where flowPath(source, sink)
select sink.getNode().(UnsafeDeserializationSink)
    .getMethodAccess(),
  source, sink, "Unsafe deserialization of $@.",
  source.getNode(), "user input"
```

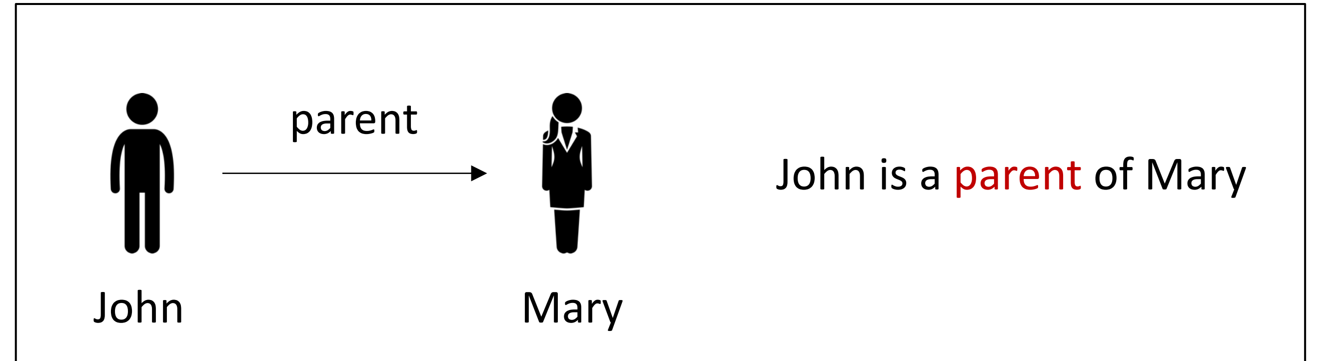
QL Query Results

alerts

- Unsafe deserialization of user input.
- Unsafe deserialization of user input.
  - Path
    1. getContent(...) : InputStream
    2. getContentAsStream(...) : InputStream
    3. toBufferedInputStream(...) : InputStream
    4. getInputStream(...) : InputStream
    5. is : InputStream
    6. ois
  - Path
- Unsafe deserialization of user input.

Introduction to Datalog





John grandparent James ⇐ John parent ? parent James

John is a grandparent of James
if there exists a person who is a parent of James,
and for whom John is a parent

Introduction to Datalog

- A declarative, non-Turing complete logic programming languages
- Some notable applications:
  - Database queries, e.g. Datomic database
  - Program analysis, e.g. Doop, Semmle
  - Network protocol specification
- Formally, a subset of Horn clauses, logical formulas in the form

$$L_0 \leftarrow L_1, \ldots L_n$$

Syntax

- A Datalog program is a set of Horn clauses

$$L_0 \leftarrow L_1, \ldots L_n$$

Head Body

- L_i is a literal symbol in the form $p_i(t_1, \ldots, t_{k_i})$, where p_i is a predicate symbol and t_j are terms
- A term t_j is either a constant or a variable
- Clauses with empty bodies are facts.

Safety Conditions

Safety conditions ensure program termination:

1. Each fact in the program should not contain variables (a clause without variables is called a ground clause)
2. Each variable occurring in the head of a rule should also occur in its body

Examples

- Father-son relation

$$father(bob, john)$$

- Grandparent relation:

$$\begin{aligned} grandparent(X, Y) \leftarrow \\ parent(X, Z), \\ parent(Z, Y) \end{aligned}$$

EDB & IDB

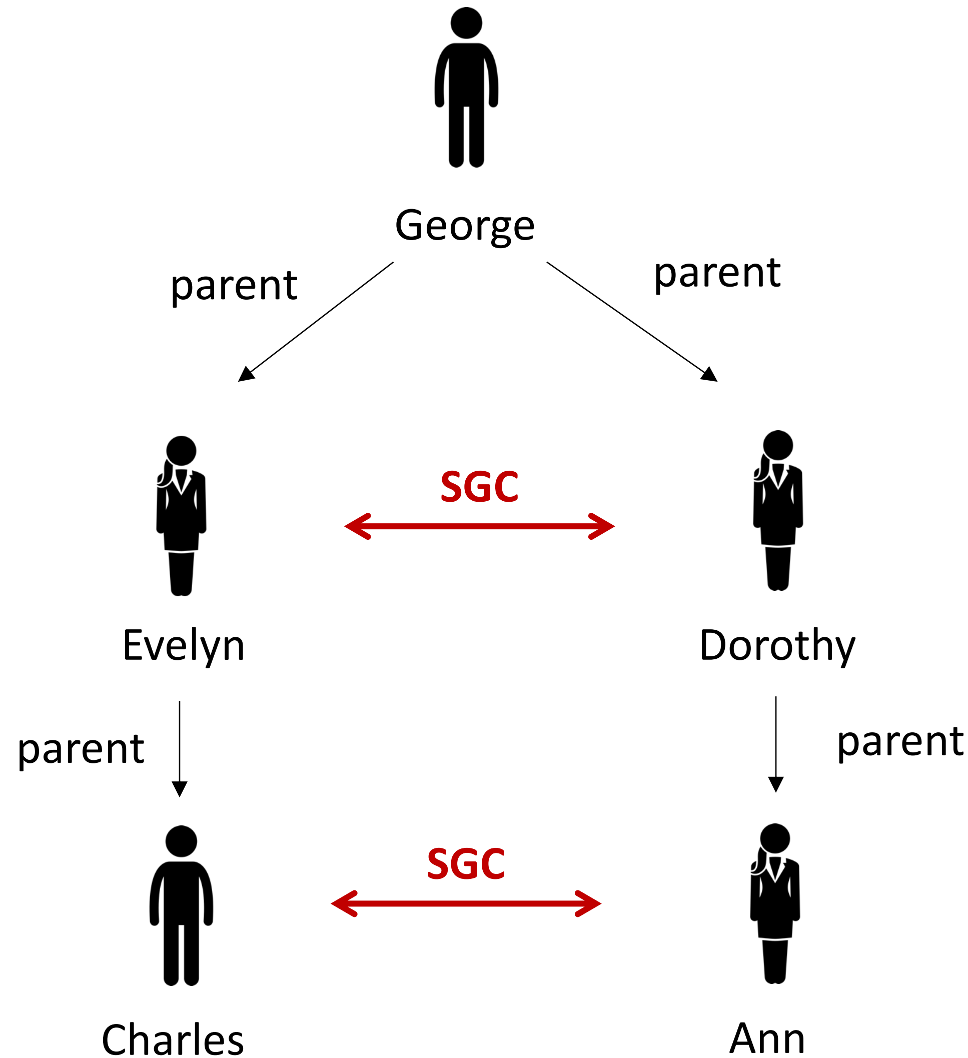
- EDB-predicates are those occurring in EDB
- IDB-predicates are those occurring in IDB, but not EDB
- **EDB** (extentional database) is a set of instances over the EDB-predicates (the input)
- **IDB** (intensional database) is a set of instances over the IDB-predicates (the output)

Example (EDB)

$person(ann)$
$person(bertrand)$
$person(charles)$
$person(dorothy)$
$person(evelyn)$
$person(fred)$
$person(george)$
$person(hilary)$

$parent(george, dorothy)$
$parent(george, evelyn)$
$parent(dorothy, bertrand)$
$parent(dorothy, ann)$
$parent(hilary, ann)$
$parent(evelyn, charles)$

Example: Same Generation Cousins (SGC)



Example (IDB)

Same generation cousins (sgc):

Rule 1: $sgc(X,X) \leftarrow person(X)$

Rule 2: $sgc(X,Y) \leftarrow parent(X1,X),$
$parent(Y1,Y),$
$sgc(X1,Y1).$

Some facts belonging to sgc:

$$sgc(dorothy, evelyn)$$
$$sgc(charles, ann)$$

Elementary production principle

The meaning of Datalog program is defined through fact inference.

Elementary production principle (EPP): for a rule $L_0 \leftarrow L_1, \dots L_n$ and a list of facts $F_1, \dots F_n$, if there is a substitution θ (of variables with values) such that $L_i\theta = F_i$, then from the facts $F_1, \dots F_n$ we can infer the fact $L_0\theta$.

Example: from $sgc(X, X) \leftarrow person(X)$ and $person(ann)$, we can infer $sgc(ann, ann)$.

Proof-theoretic Semantics of Datalog

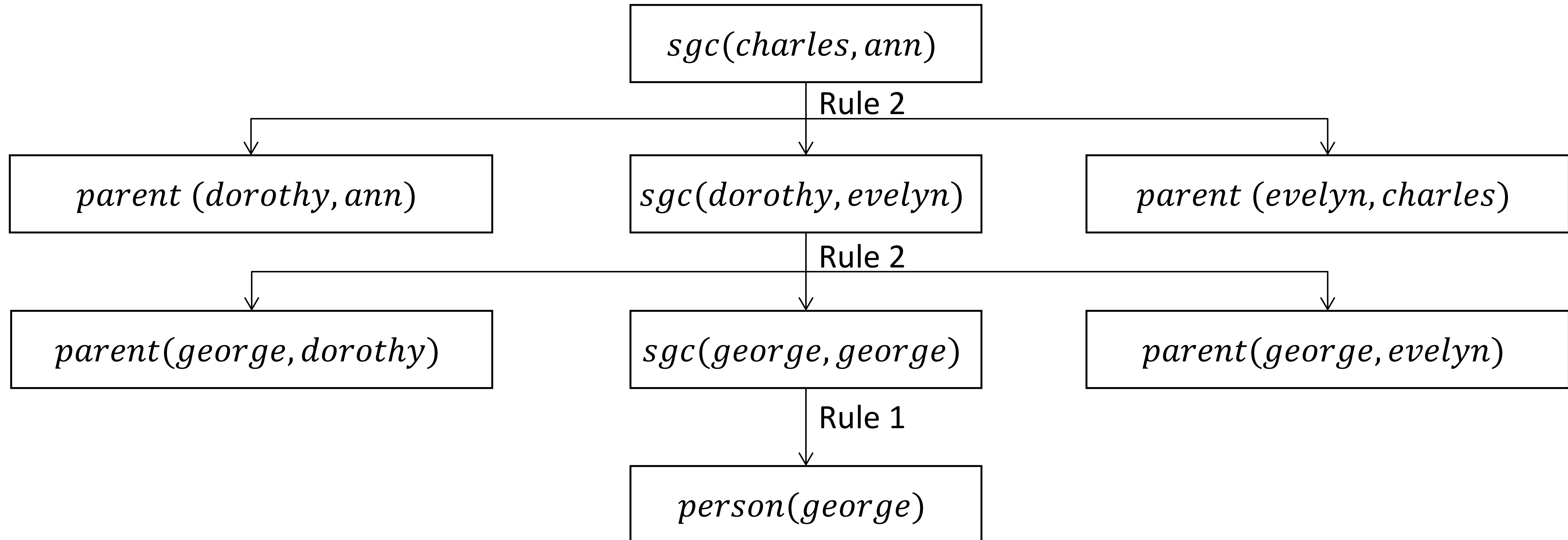
Given a set of clauses S, we define the inference relation $\vdash$ for fact F:

1. $S \vdash F$ if $F \in S$
2. $S \vdash F$ if a rule $R \in S$ and facts $F_1, \ldots, F_n$ exists such that $S \vdash F_i$ for all $i \in [1..n]$ and F is inferred by applying EPP to R and $F_1, \ldots, F_n$.

Theorem. All facts that can be inferred from S are exactly the facts that logically follow from S.

Proof Tree

A **proof tree** for $sgc(charles, ann)$ visualises $\vdash$ relationship:



Negation

$\neg sgc(bertrand, X)$ – X is not a same generation cousin of Bertrand

Negation is incorporated based on **Closed World Assumption** (CWA): if a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.

Safety: every variable appearing in a negated literal must also appear in a positive literal of the same rule. Negation in the head of a rule is not allowed.

Problem of Negation with Recursion

IDB:

$$r(X) \leftarrow s(X), \neg t(X)$$
$$t(X) \leftarrow s(X), \neg r(X)$$

EDB:

$$s(1)$$

In this case, we have two solutions according to the fixpoint semantics:

1. $cons(S) \coloneqq \{r(1), \neg t(1)\}$
2. $cons(S) \coloneqq \{\neg r(1), t(1)\}$

Stratified Datalog

A stratified Datalog program can be partitioned into disjoint sets of clauses $P = P^1 \cup \cdots \cup P^n$ called strata, so that

1. Each IDB predicate has its defining clauses within one stratum
2. P^1 can contains only negation of EDB predicates
3. P^i can only contain negation of IDB predication from P^j, $j < i$

Ensures that the semantics with negation is well-defined.

Stratified Datalog Example

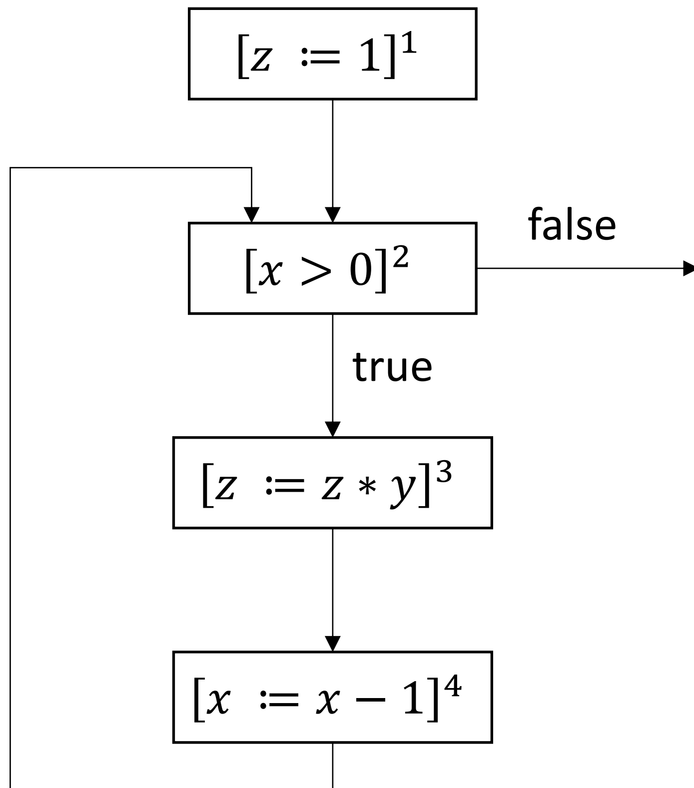
$r_1: p(X,Y) \leftarrow \neg q(X,Y), s(X,Y)$
$r_2: q(X,Y) \leftarrow q(X,Z), q(Z,Y)$
$r_3: q(X,Y) \leftarrow d(X,Y), \neg r(X,Y)$
$r_4: r(X,Y) \leftarrow d(Y,X)$
$r_5: s(X,Y) \leftarrow q(X,Y), q(Y,T), X \neq T$

Strata:

$P^1 = \{r_4\}$
$P^2 = \{r_2, r_3, r_5\}$
$P^3 = \{r_1\}$

EDB predicates: d

CFG As EDB



$label(1)$
$label(2)$
$label(3)$
$label(4)$
$init(1)$
$final(2)$
$flow(1,2)$
$flow(2,3)$
$flow(3,4)$
$flow(4,2)$

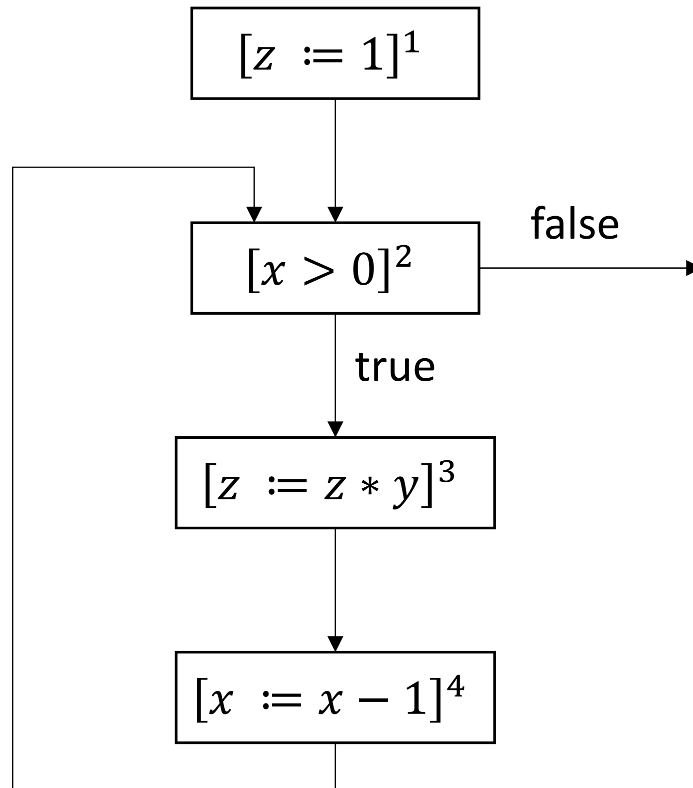
CFG Reachability

The relation $reach(L_1, L_2)$ states that there is a path from L_1 to L_2 in CFG:

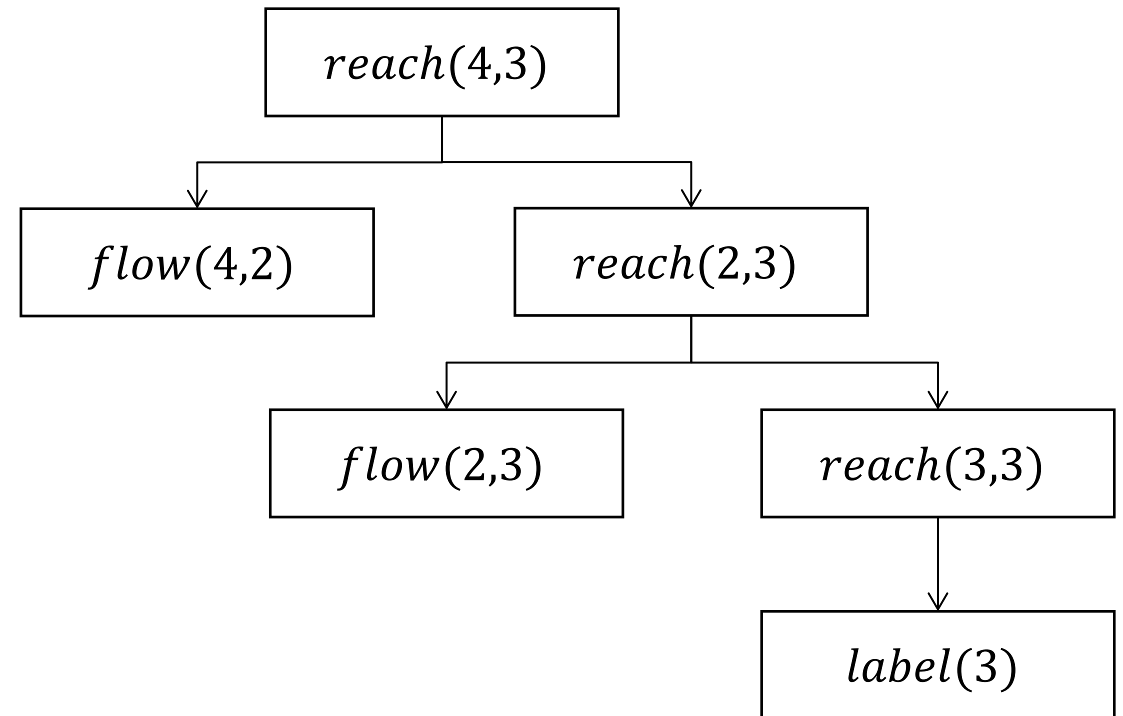
$$reach(X, X) \leftarrow label(X)$$
$$reach(X, Y) \leftarrow flow(X, Z), reach(Z, Y)$$

CFG Reachability

CFG



Proof Tree of $\boldsymbol{reach(4, 3)}$



Dominators (Part 1)

$dominate(D, N)$: A node D **dominates** a node N if every path from the entry node to N must go through D.

$$
\begin{aligned}
&reach_without(L, L, D) \leftarrow \\
&\quad label(L), \\
&\quad label(D), \\
&\quad L \neq D \\
&reach_without(L_1, L_2, D) \leftarrow \\
&\quad L_1 \neq D, \\
&\quad flow(L_1, L_3), \\
&\quad reach_without(L_3, L_2)
\end{aligned}
$$

Dominators (Part 2)

$dominate(D, N)$: A node D **dominates** a node N if every path from the entry node to N must go through D.

$$
\begin{aligned}
¬_dominate(D, N) \leftarrow \\
&\quad init(L), \\
&\quad reach_without(L, N, D) \\
&dominate(D, N) \leftarrow \\
&\quad D \; \neq N, \\
&\quad init(L), \\
&\quad reach(L, N), \\
&\quad label(D), \\
&\quad !\,not_dominate(D, N)
\end{aligned}
$$

The role of negation

Negation helps to express a property that hold for a set of paths.

$$
\begin{aligned}
&dominate(D, N) \leftarrow \\
&\quad D \neq N, \\
&\quad init(L), \\
&\quad reach(L, N), \\
&\quad label(D), \\
&\quad !not_dominate(D, N)
\end{aligned}
$$

D dominates N if there **exists** a path from an init node to N, and it is **not** true that there **exists** a path from an init node to N that does **not** visit D.

Reaching Definitions

Reaching definitions analysis determines for each program point, which assignments may have been made and not overwritten, when program execution reaches this point along some path.

$[x \coloneqq 5]^1;$
$[y \coloneqq 1]^2;$
$while\ [x > 1]^3\ do$
$\quad [y \coloneqq x * y]^4;$
$\quad [x \coloneqq x - 1]^5;$

All assignments reach the entry of 4; only the assignments 1,4,5 reach the entry of 5.

EDB for Reaching Definitions

Domains

- program variables (V_i)
- labels (L_i)

Relations

$label(L)$
$flow(L_1, L_2)$
$init(L)$
$final(L)$
$variable(V)$
$defined(V, L)$

Killed Assignments

$$kill_{RD}([x := a]^l) = \{(x, ?)\} \cup \{(x, l') \mid B^{l'} \textit{is an assignment to } x\}$$
$$kill_{RD}([skip]^l) = \emptyset$$
$$kill_{RD}([b]^l) = \emptyset$$

where ? is the label for uninitialised variables.

$$kill(L, V, D) \leftarrow$$
$$defined(V, L),$$
$$defined(V, D),$$
$$L \neq D$$
$$kill(L, V, ?) \leftarrow$$
$$defined(V, L)$$

Generated Assignments

$$gen_{RD}([x := a]^l) = \{(x, l)\}$$
$$gen_{RD}([skip]^l) = \emptyset$$
$$gen_{RD}([b]^l) = \emptyset$$

$$gen(L, V, L) \leftarrow defined(V, L)$$

Analysis

$$RD_{entry}(l) = \begin{cases} \{(x,?) \mid x \in Vars\} & if\ l = init(program) \\ \cup\, \{RD_{exit}(l') \mid (l',l) \in flow\ (program)\} & otherwise \end{cases}$$

$$RD_{exit}(l) = \left(RD_{entry}(l) \setminus kill_{RD}(B^l)\right) \cup gen_{RD}(B^l)$$

$$rd_entry(L,V,?) \leftarrow init(L)$$
$$rd_entry(L,V,D) \leftarrow flow(L',L), rd_exit(L',V,D)$$
$$rd_exit(L,V,D) \leftarrow rd_entry(L,V,D), !\,kill(L,V,D)$$
$$rd_exit(L,V,D) \leftarrow gen(L,V,D)$$