The value dependence graph

Michael Ernst

University of Washington

joint work at Microsoft Research with Daniel Weise, Roger Crew, and Erik Ruf

The value dependence graph (VDG)

Sparse, functional, parallel representation for imperative programs

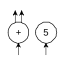
Insights:

- **Only values matter**
- **Original names and control flow are incidental**

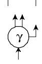
Consequences:

- **All values are explicit**
- **Select values, not control paths**
- **Control flow represented by function calls**

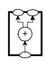
Components of the value dependence graph



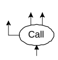
operation nodes



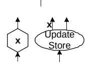
selectors



closures (functions)

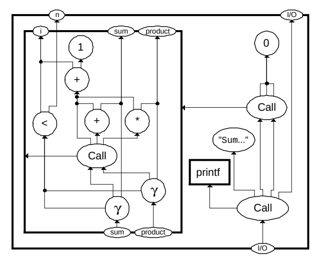


function calls



memory lookups (load instructions)
memory assignments (store instructions)

Example VDG




```
void sum_product(int n)
{
  int sum = 0;
  int product = 0;
  int i = 0;

  while (i++ < n)
    { sum = sum + i;
      product = product * i;
    }
  printf("Sum %d, product %d",
         sum, product);
}
```


The VDG is buzzword-compliant

sparse

functional, explicit

parallel

simple, fine-grained, single graph

demand-driven, dataflow-like

The VDG is sparse

Directly connects producers to consumers

Similar to def-use chains

Analyses take less space and time

The VDG is functional

No hidden side effects

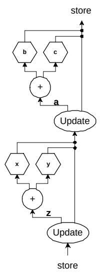
All values are explicit (even the store)

The VDG is parallel

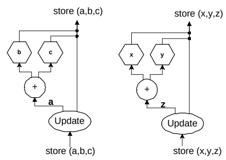
Permits code reordering

The store sequentializes

Store splitting




```
z = ...;
a = b + c;
z = x * y;
```



The VDG is simple

Values, not variables

No special nodes for looping, etc.

Fewer dataflow rules, fewer interactions among nodes

Fine-grained: can refer to any computation, not just those in variables

A single graph: not forced to maintain invariants in other portions of the graph when performing transformations

The VDG is demand-driven

Demand dependences, not control dependences

Only the value computed matters, not the computation producing it

Dataflow-like

Simplifies backward analyses

Analyses and transformations

Dead code elimination

Common subexpression elimination (hash-consing)

Loop invariant code hoisting

Constant propagation

Others are simplified, too

Why do we care?

Any analysis can be performed on any representation.

Which representation makes analyses and transformations easiest, and suggests new ones as well?

Comparing the VDG and PDG

Which PDG?

VDG:

- no implicit quantities
- better integration with programming environment
- easier analysis, transformation
- cleaner interprocedural representation
- single graph
- finer granularity