Interprocedural Analysis of Aggregates

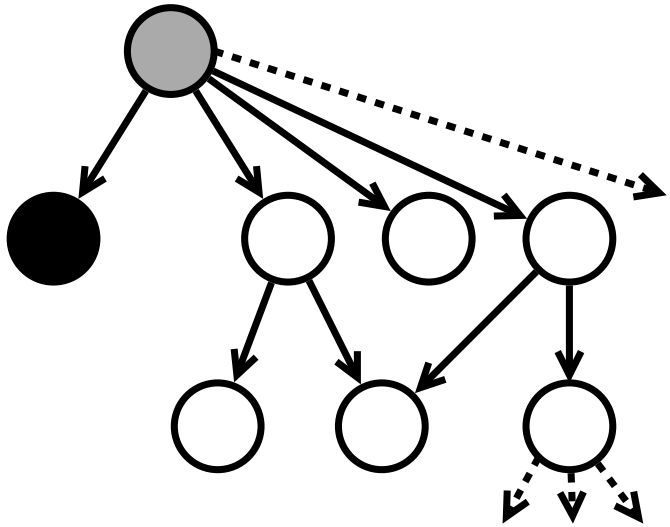
Martin Jambor

Charles University, Prague
Czech Republic

September 16, 2007

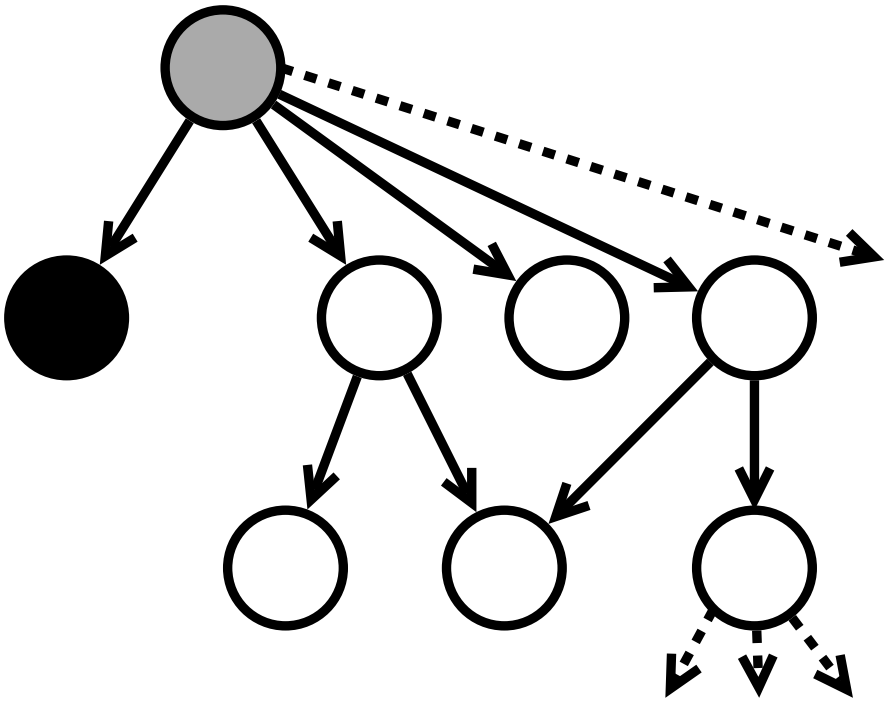
The original idea

- Inspired by analysis of C++ scientific programs.
- Many objects are initialized by the constructor and the same values are used throughout its lifetime.



- Constant sizes, strides etc. . . can allow some transformations, but are hidden in a structure.
- SRA and inlining can only help so much.

Some objects are comprehensible



Many objects are used simply enough to be able to make judgements about their whole lifetime (or at least a part of it).

- Entirely under control.
- Easy to track.

Definition of a *usable* object

A usable object is

1. either allocated (on stack or heap) or passed by reference as a formal parameter,
2. its address does not escape to global variables, fields of structures or element arrays,
3. is not passed to an unknown function (except for deallocation),
4. is never passed to any subroutine in two different arguments, and
5. no references to it take any part in a phi-node

in all procedures it is used in.

Basic data structure: an *item*

Structure `ocp_item` represents

- structures and arrays (`VAR_DECL`,
- pointers and references to structures and arrays (`SSA_NAME`), and
- fields of structures (`FIELD_DECL`).

An important property:

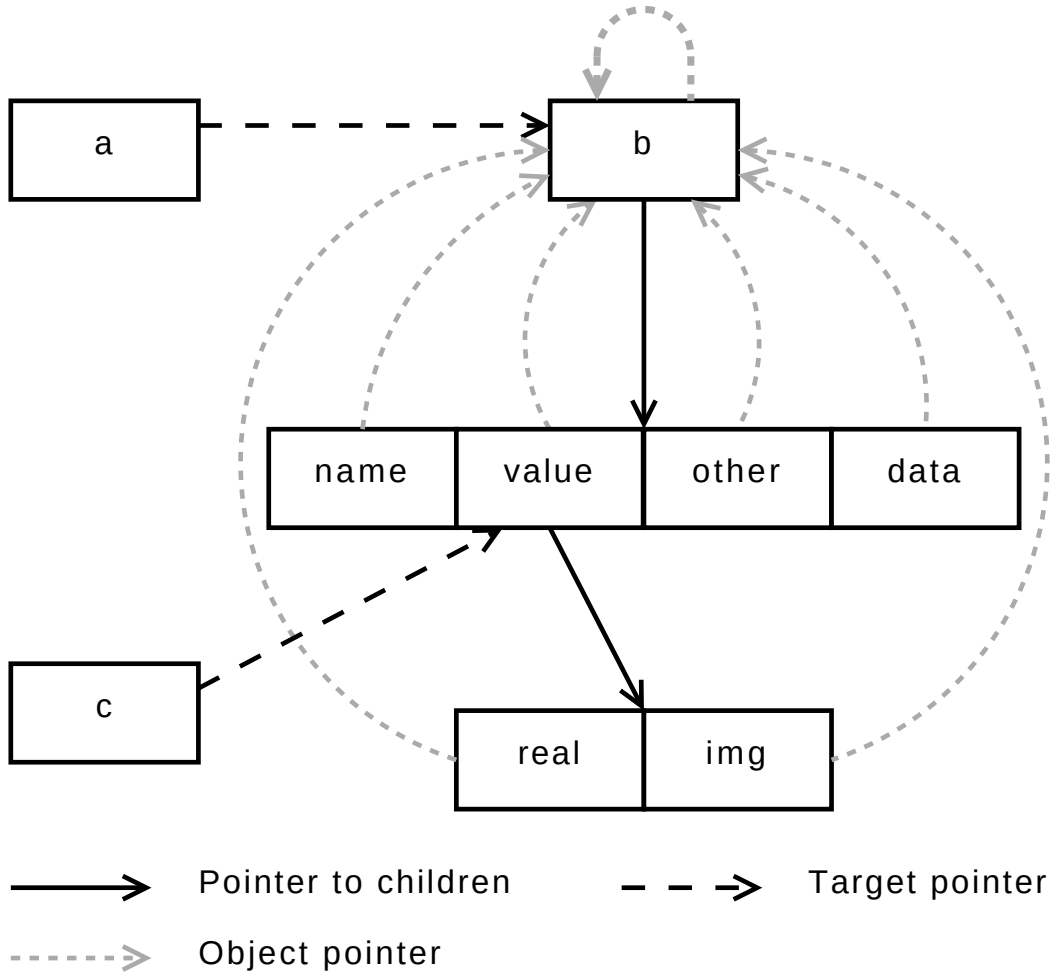
- When multiple functions share an object, each has a corresponding item for it.

Example representation by items

```
struct complex {
        double real;
        double img;
};

struct example {
        char *name;
        struct complex value;
        struct complex *other;
        int data[10];
};

struct example b, *a = &b;
struct complex *c = &b.value;
```

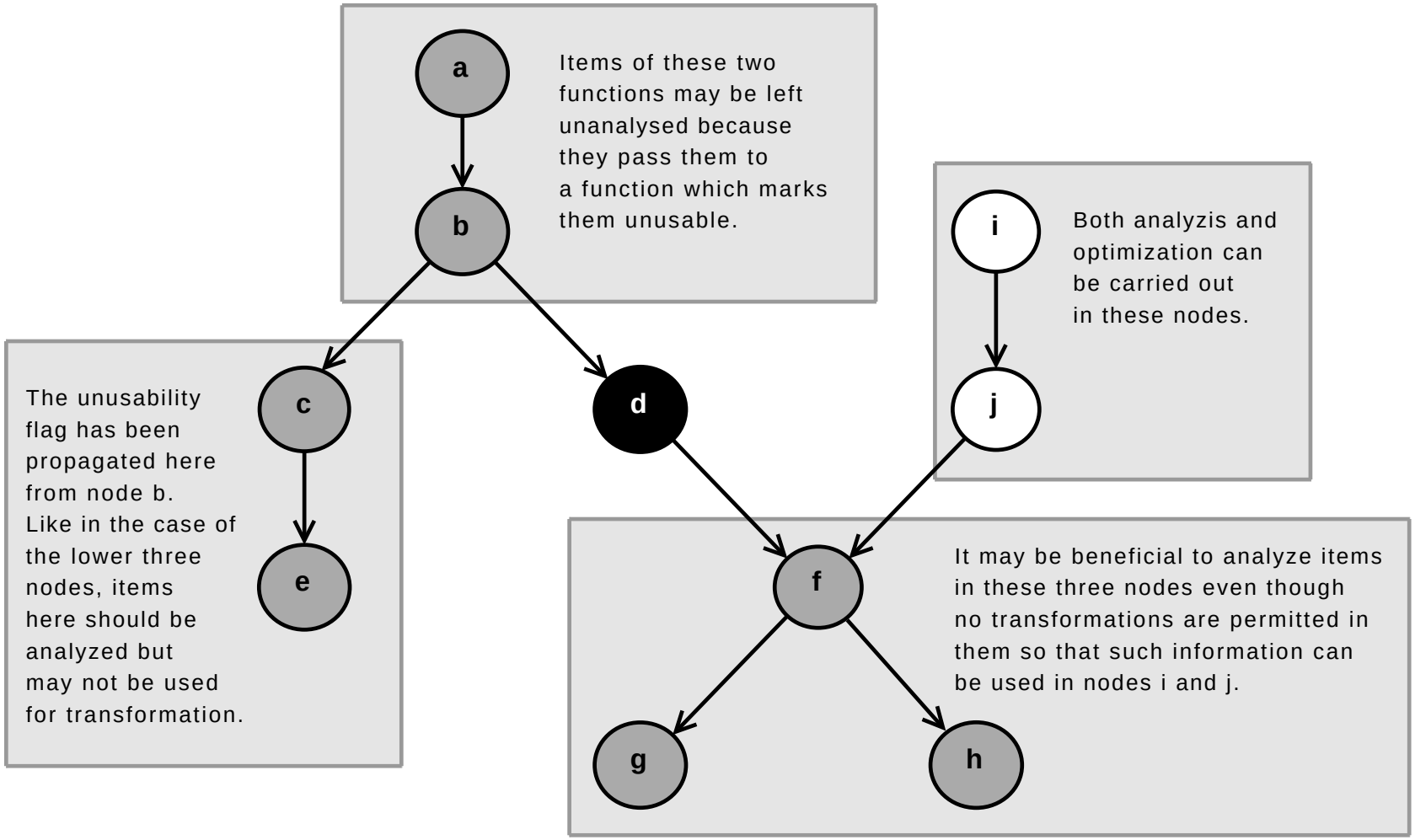


Intraprocedural stage

1. Checking SSA definitions
   - safe origin?
   - alias of another item or sub-item?
2. Scrutinizing SSA uses
   - leaking reference?
3. Function scan
   - function calls
   - further dangerous `ADDR_EXPR`'s

Violation of rules $\Rightarrow$ item marked as unusable

Interprocedural unusability propagation



Interprocedural propagation of constants within aggregates

Fairly similiar to existing scalar IPA-CP:

0. Based on traditional lattices
1. Jump and return function building stage
   - Return functions also required
2. Interprocedural lattice analysis
3. Replacement stage
   - Substitution of constants
   - Clones created as necessary

Usable items and performed replacements

Numbers of different items:

Benchmark	Aggregates	Usable Items	Unusable Items
Tramp3D	2,809	9,174 (27%)	26,143 (73%)
FreePOOMA test suite	67,311	162,845 (23%)	558,497 (77%)
Xpdf	1,134	1,719 (15%)	9,495 (85%)

Number of replacements by the ipa constant propagation:

Benchmark	Scalar	Array	Total
GCC bootstrap	189	0	189
GCC test suite	675	12	687
FreePOOMA test suite	3,367	12	3,379
Tramp3D	2	0	2
DLV	118	0	118
Blitz++ test suite	96	0	96

Current Plan

- Relax unusability rules
- Decoupling analysis and constant propagation some more
- Allow for incremental updates
- Integrating into gcc-ipa branch so that using the analysis is as convenient as possible.

Potential applications

- Interprocedural propagation of constants
- Field modification flag propagation
- Procedure cloning
- Type information propagation (i.e. for devirtualization)
- Removal of unused fields
- Alias analysis
- Representation of interprocedural dataflow
- Possibly more...