Points-to Analysis

Amira Mensi

October 20, 2021

1 Introduction

The data structures defined here are used to compute, store and use points-to information. A points-to analysis is one of the pointer analyses. It associates some abstract address to every known pointer at any control point in the code. The description of the first points-to analysis implemented in PIPS is in Amira Mensi's PhD dissertation.

The points-to data structures depend on the internal representation and on the memory effects.

```
import descriptor from "ri.newgen"

import approximation from "ri.newgen"

import statement from "ri.newgen"

import cell from "effects.newgen"
```

2 Points-to arcs

A points-to data structure implements an arc from an abstract memory location called *source* to another abstract memory location called *sink*. The arc between the two locations may either always exist, and the approximation is EXACT, or possibly exist, and the approximation is MAY. The abstract memory locations may contain program variables or PIPS entities in their subscript expressions, and their values are constrained by the *descriptor*.

```
points_to = source:cell x sink:cell x approximation x descriptor
```

Note: the domains *approximation* and *descriptor* are documented in ri.tex.

Approximations are not yet well semantically defined because they can be related to the number of memory locations abstracted by the source, the number of memory locations abstracted by the sink, or to the number of arcs leaving a source. The current definition of an exact arc is an arc starting from a unique memory location with outdegree one and ending on a unique memory location.

Different kinds of points-to abstractions may be built. They depend on the subsets of cells that are used, as well as on the descriptor kind.

The initial implementation is based on *constant memory paths*, that are references independent of the current store and that are called points_to_cells in the source code. Such references are extensions of the references defined by `ri.tex`. Field entities can be used as subscript as well the special entity `UNBOUNDED_DIMENSION`. This is convenient for the array-based data dependence tests and the parallelization passes of PIPS because they can handle C `struct` variables without modifications. The API and the source code dealing with points_to_cells is located in library effects-util. No descriptor is used, and hence all descriptor field should contain the kind `none`.

Descriptors could be sets of affine constraints and be used to represent the relation existing between arrays of pointers and locations. For instance, `a[i]->malloc[i]` $\forall ii n[0, 10[$. They could be used to represent multidimensional arrays allocated dynamically and to parallelize loops using them.

3 Points-to relation

The points-to relation can be represented by a list of points-to arcs:

points_to_list = bottom:bool x list : points_to*

The points-to graph domain is used to represent the points-to relation, from a set of cells to a set of cells, as a set of arcs. For a given statement, this relation may be empty, for instance because no pointers are used, but it may also not exist at all because the statement is unreachable with respect to the points-to information.

points_to_graph = bottom:bool x set:points_to{}

The double representation of the points-to graph, as a list of arcs and as a set of arcs, is linked to the implementation. The lists are fine to store and exploit the points-to information. The set if more efficient when the points-to information is computed. Not that the hash function used to implement the sets of points-to arcs is fragile. No side-effects are permitted on arcs belonging to a points_to_set.

4 Mapping from statements to their points-to relations

This last domain is used to associate its points-to information to each statement for storage or use of the points-to information:

statement_points_to = persistant statement -> points_to_list

The points-to information is a precondition. It holds in the memory state preexsiting to the statement execution.