PoTATo: Points-to Analysis via Domain-Specific MLIR Dialect

Robert Končar, and Henrich Lauko

Goals of Points-to Analysis MLIR Dialect

**Problem Simplification**
PoTATo streamlines the points-to analysis by reducing it to a conversion task into its specialized dialect. The dialect’s simplicity facilitates the straightforward conversion of any MLIR program representation into it for subsequent analysis.

**Efficiency**
Encoding the points-to problem in a dialect enables the application of compiler optimizations to reduce the problem size. Furthermore, by keeping solely on essential information the complexity of the analyzer is also reduced.

**Flexibility**
PoTATo is built with adaptability in mind, enabling users to conduct a range of points-to analyses. By choosing the points-to lattice representation and adjusting the conversion process, users can customize the analysis.

Integration
PoTATo is a tool for points-to analysis, using a novel approach, we try to reduce the problem size to provide faster analysis without losing information. We have designed a simple MLIR dialect that models memory in the IR. To simplify the problem, we apply compiler-like optimizations to the IR. In cooperation with LLVM, we then extract the analysis result back to the source IR.

Leveraging the power of VAST Tower of IRs, we can precisely lift the IR to the original representation. Using this information in hand, we can pry the planning queries about the program in the original IR and extract the information from the computed analysis. The proxy, after getting the mapping of the IR, returns the analysis result using the MLIR dataflow analysis framework and answers the user queries.

PoTAToIR

**Field-insensitive Dialect**
Memory allocation abstracts all location creations for points-to analysis, including both local stack or heap allocations.

Memory dereference represents all operations that access memory content, such as the load operation in LLVM.

Memory assignment denotes writing to memory content, copying its points-to information into the content. It has similar semantics to the LLVM store operation.

Copy abstracts all operations that do not alter points-to information. It transfers it from the source to the destination. Such operations are pointer casts and pointer arithmetic in the case of field-insensitive analysis.

Address operation is used to abstract operations that create references, suit it to model more high-level manipulations with addresses (such as field access). Caddr

Copy is a pointer to an address that creates a copy of any non-pointer value. The value is an instance of the analysis framework, enabling copying of non-pointer values. Cconst is a pointer to a constant value.

Compile

The analysis result can be obtained by following the chain of meta-locations. For example, if we have %o in LLVM, it corresponds to %o in the PT dialect. From there, we can trace to the fused location in simplified IR and retrieve the corresponding points-to set from its original representation, enabling querying of the analysis result via the MLIR data flow interface. Each IR location is associated with a lattice value representing the analysis result.

Dataflow Reduction

Default PoTATo IR corresponds directly to the standard interpretation-based points-to analysis. Each source IR operation has a corresponding PoTATo IR operation, which transforms points to sets. However, it includes numerous irrelevant operations for points-to analysis, which have no impact on the analysis result. In particular, copies do not modify points-to information in this example.

Result: Utilizing location metadata or VAST Tower, the domain-specific IR remains linked to its original representation, enabling querying of the analysis result via the MLIR data flow interface. Each IR location is associated with a lattice value representing the analysis result.

Analysis Procedure

**Transform**
The transformation step, provided by the user of the analysis tool, abstracts away point-to-relevant information of the program in the PoTATo dialect. Here, the user can opt for including field-sensitive operations and types in the resulting IR, tagging this aspect of the analysis. The tool does not mandate the user to lower the control flow operations and function calls, as long as the source dialect implements relevant MLIR interfaces. Users only need to provide abstraction for pointer manipulation, or they can use the conversion from the LLVM.

**Simplification**
Leveraging the dialect’s simplicity, we can apply canonicalization to reduce the size of the problem, thus accelerating the points-to analysis. A key step is constant folding, which eliminates irrelevant operations in the analyzed IR.

**Analysis**
The points-to analysis procedure is implemented using the MLIR dataflow framework, enabling the flow-sensitive analyses. When integrating the analysis into the framework runner, users can select their preferred lattice representation, or provide their own implementation, which determines the analysis algorithm applied.

**Result**
Utilizing location metadata or VAST Tower, the domain-specific IR remains linked to its original representation, enabling querying of the analysis result via the MLIR data flow interface. Each IR location is associated with a lattice value representing the analysis result.