Enhancing VASCO Framework

Rashmi Kokare
Guide: Prof. Uday Khedkar
Department of Computer Science, IIT Bombay
IIT Bombay

What is VASCO

- **Enhance and generalize the VASCO framework within LLVM** to improve its scalability.
- **VASCO** is a context-sensitive data flow analysis framework where the context is defined by the data flow values reaching specific program points.
- Incorporate **context-insensitive information** for comparison purposes, evaluating the benefits of context sensitivity, and further generalizing the framework to handle multiple analyses.
- Includes **testing** the implementation with modifications.

VASCO 1.0 vs VASCO 2.0

- VASCO 1.0:
  - Context-sensitive analysis with unique data flow contexts.
  - Uses a context-transition graph for call-return matching.
  - Supports only one analysis at a time.
- VASCO 2.0:
  - Supports multiple simultaneous analyses.
  - Employs the DFV class for centralized analysis type management.
  - Enhances worklist and context handling with a product lattice.

Example

Available Expression Analysis

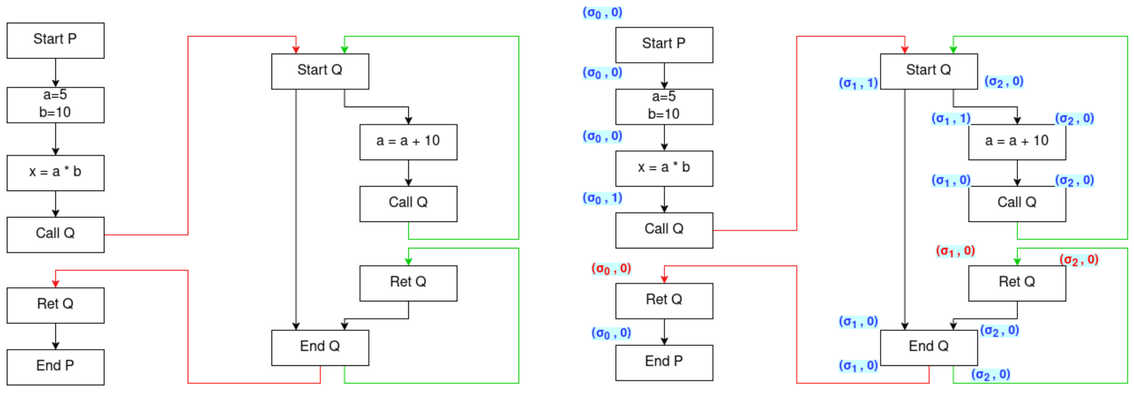
An expression *e* is available at a program point *p* if, along every path from the entry to *p*, *e* is computed and no operand of *e* is modified thereafter.

Source Code

```
void Q{
    if(cond){
        a = a+10;
    }
}
```

Example

- Understanding VASCO with Available Expression Analysis



Need For DFV Class

The Problem:

The previous implementation passed analysis types as **void***, causing:

- Manual type casting,
- Null pointer errors,
- Incorrect type interpretation.

These issues even led to **segmentation faults** during constant propagation tests.

The DFV Solution:

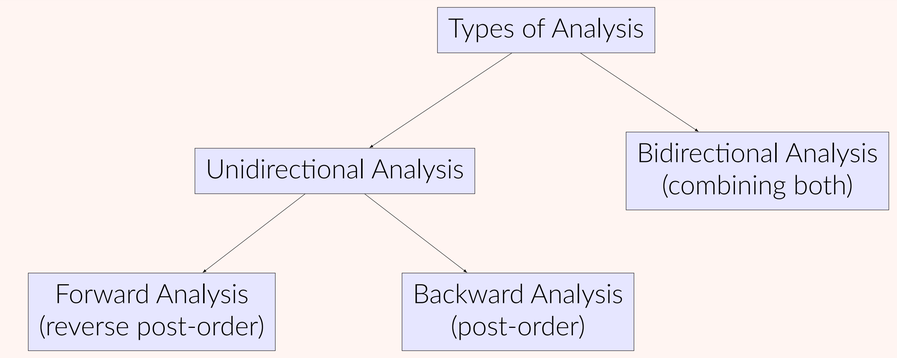
The **DFV class** is a templated utility that sets the **analysis type** once for the entire VASCO framework. All data flow values then adhere to this specified type, thereby:

- Improving type safety,
- Reducing runtime errors,
- Ensuring consistent analysis behavior.

DFV Class

- **Overview:** The **DFV class** is a templated utility that centralizes the management of **analysis types** within the VASCO framework, thereby improving **type safety** and overall **consistency**.
- Key Methods:
  - `setType(const T& type)`: Configures the analysis type once, eliminating the need for repetitive type handling.
  - `getType() const`: Provides a uniform approach to retrieve the current analysis type.
- **Evaluation:** The DFV class has been successfully tested with two forward analyses: **pointer analysis** and **constant propagation**.

Types of Analysis



Note: Bidirectional analysis applies both forward (reverse post-order) and backward (post-order) traversals of the control flow graph.

Worklist Implementation in VASCO 2.0

Purpose:

Coordinates analyses across contexts and basic blocks until data flow values converge.

Call Handling:

Reuses an existing context if a call was already analyzed; otherwise, performs a new analysis to compute data flow values.

Benefits:

- Rapid identification of unprocessed blocks.
- Minimal memory overhead via a compact array.

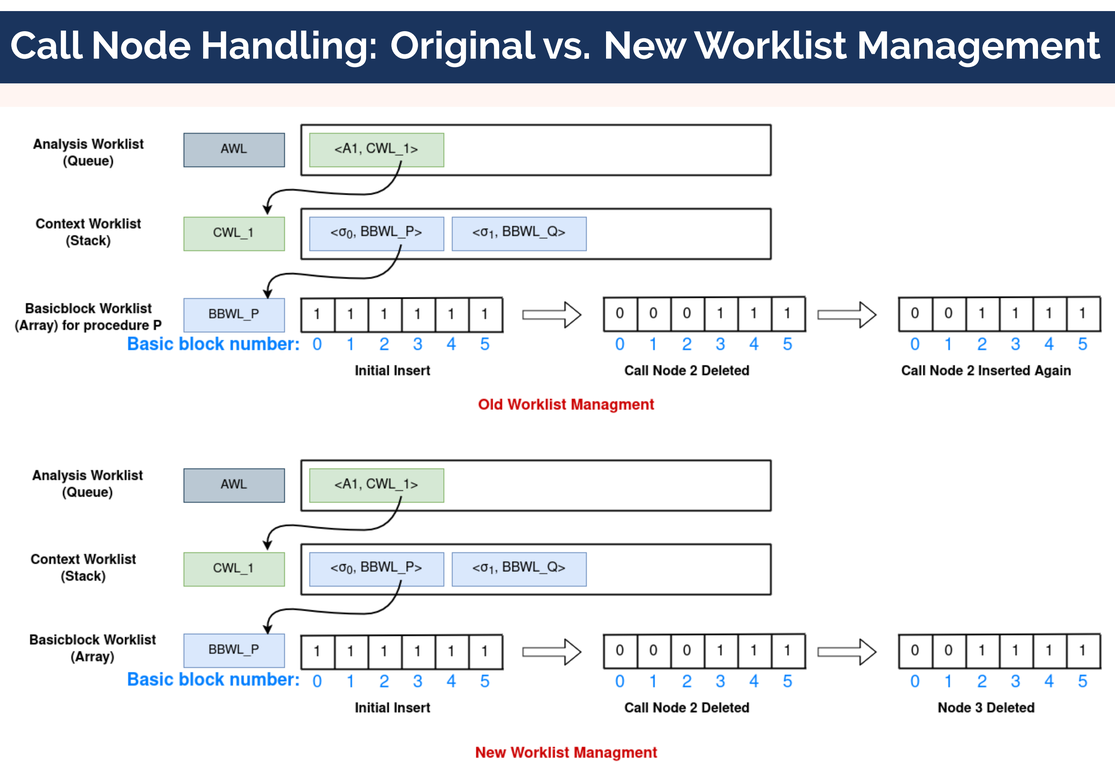
Structure:

Analyses Queue: Pairs each analysis with its context worklist for proper scheduling.

Context Stack: Maintains a stack of contexts, each linked to its basic block worklist.

Basic Block Array: Uses a compact boolean array to flag unprocessed blocks, enabling rapid and efficient traversal of the control flow graph.

Call Node Handling: Original vs. New Worklist Management



Context Insensitive With VASCO