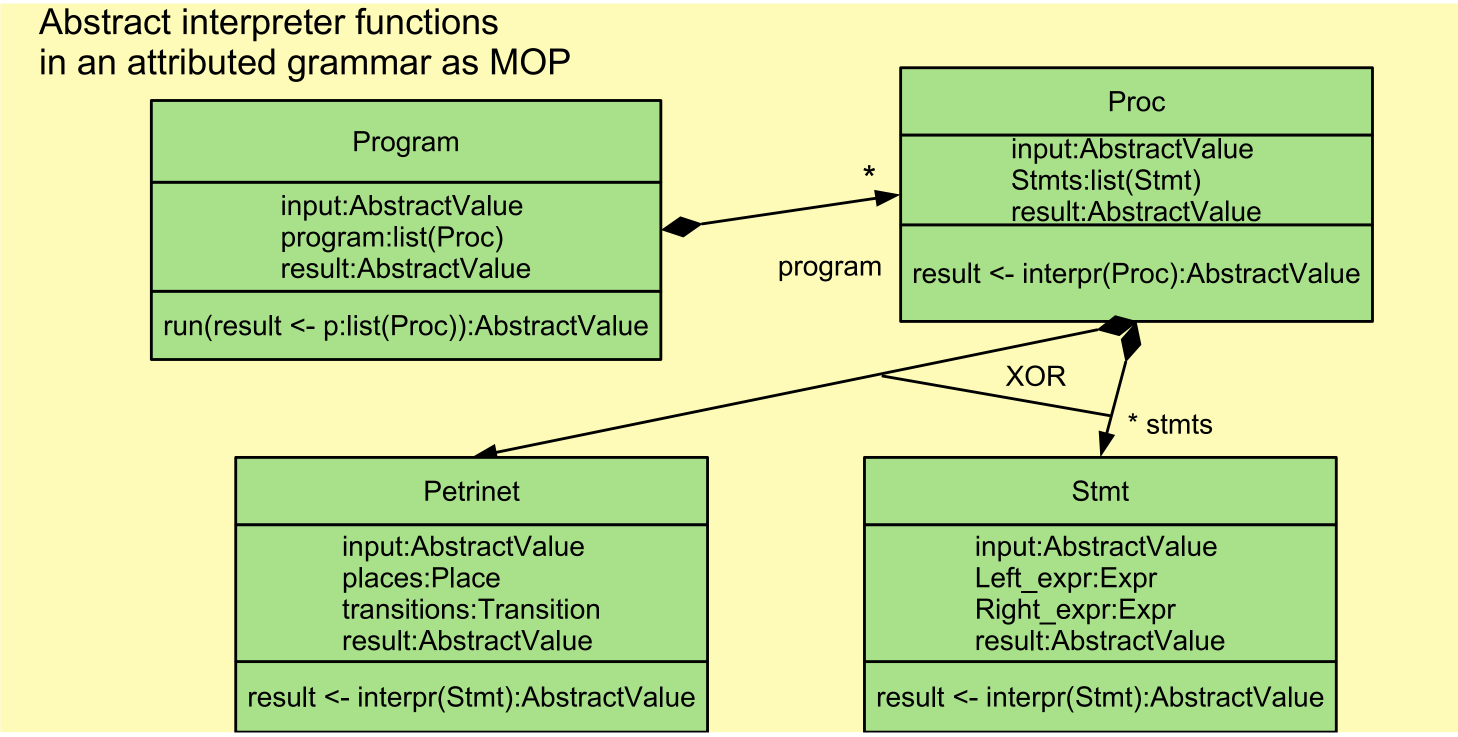
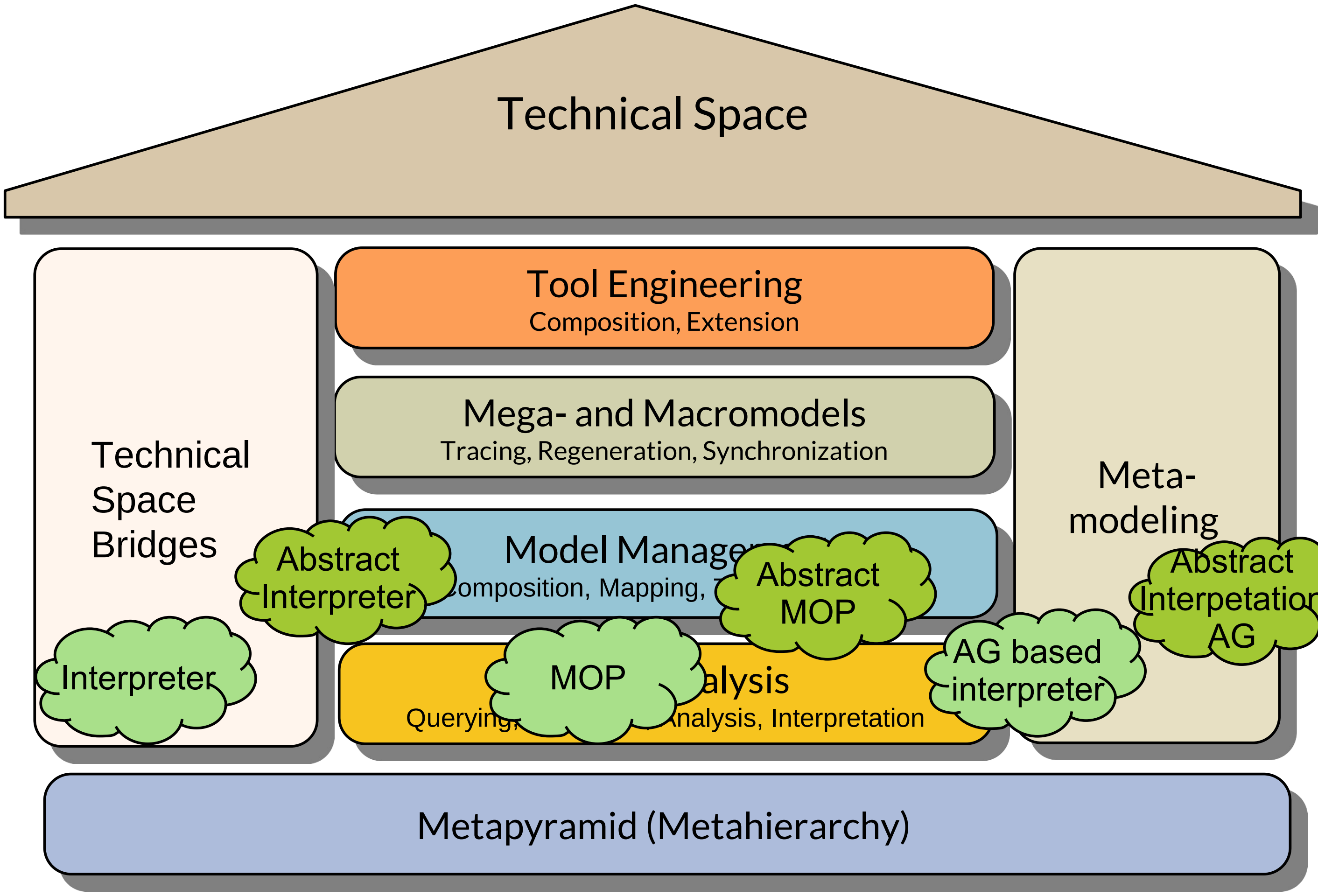
Implementation Pattern III (MOP-AG-Interpreters): Abstract Interpreters can be Specified by AG

- The *interpretation functions (transfer functions)* of an abstract interpretation may be arranged **in the metaclasses of an attributed grammar M2**
  - Then, the syntax trees (hierarchic) are described by a grammar
- Then, we call the abstract interpreter a **abstract-interpretation attribute grammar**
  - storing the results in attributes of the tree.



Q10: The House of a Technical Space

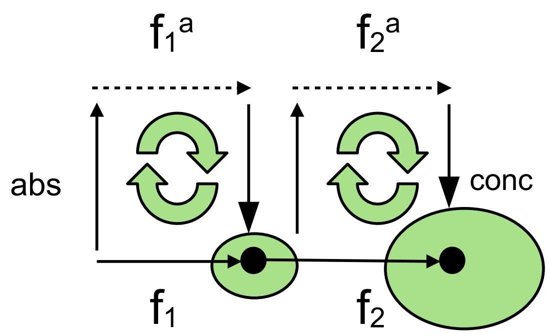


22.3. The Laws of Abstract Interpretation for Deep Analysis of Programs

The Iron Law of Abstract Interpretation: Faithfullness

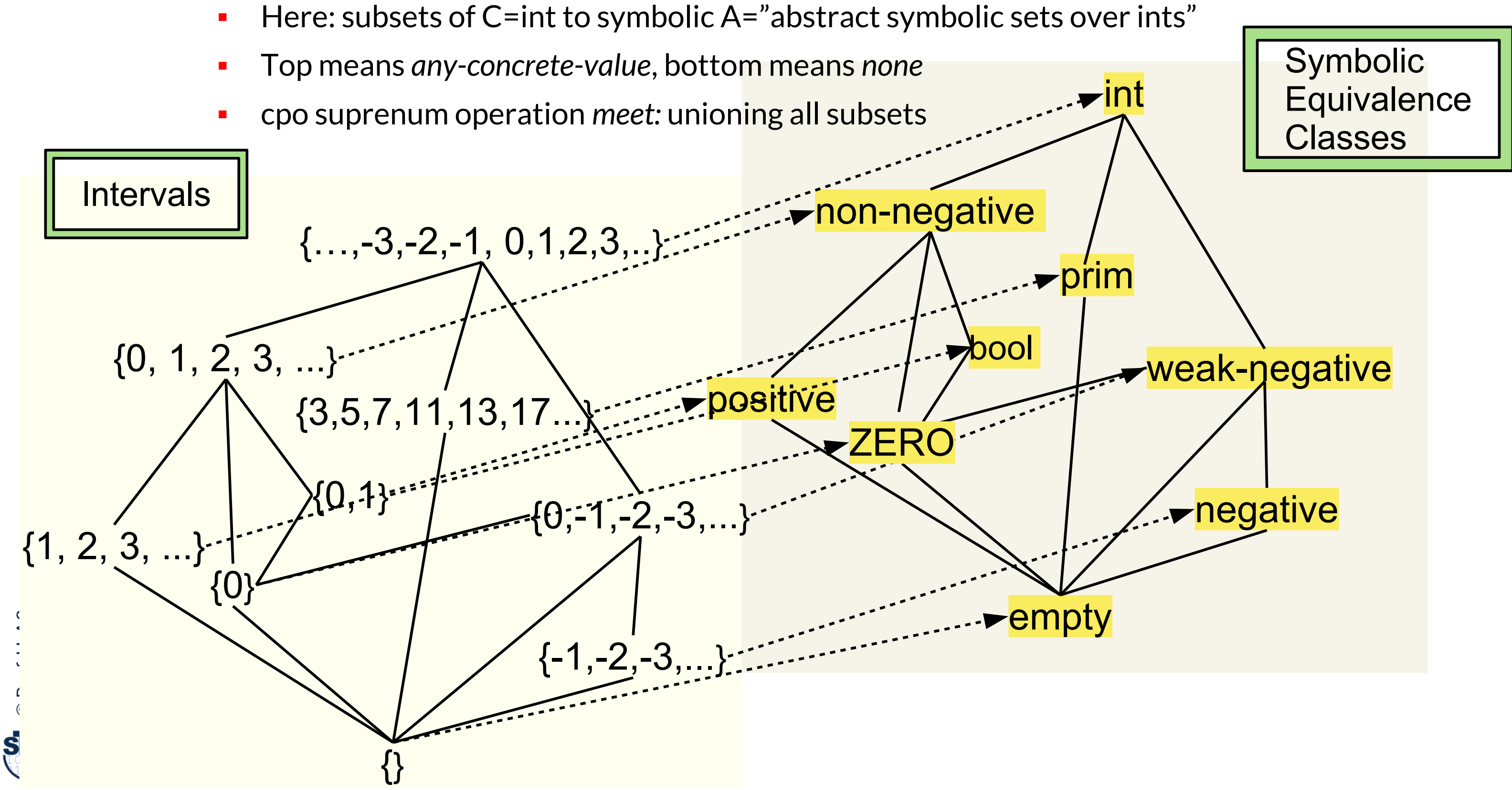
The abstract interpretation must be *correct (conservative)*, i.e., faithfully abstracting the run-time behavior of the program („reality proof"):
$f \subset conc \circ f^a \circ abs$

- The shadow must be faithful; the ***corridor of possible values must contain the real value***
- abs (abstraction function), conc (concretization function), and f^a (abstract interpretation function) must form a commuting diagram
  - The abstract interpretation should deliver all correct values, but may be more
  - They must be "interchangeable", formally: a Gaulois connection
- The interpretation must be a subset of the abstract interpretation:
  - $f \subset conc \circ f^a \circ abs$
  - The concrete semantics must be a subset of the concretization of the abstract semantics (conservative approximation)
  - $conc \circ f^a \circ abs \supset f$
  - The abstract semantic value must be a superset of the concrete semantic value after application of the transfer function
  - The concrete value of f must be a subset of the abstracted value after application of the transfer function



Ex. Concrete and Abstract Values (Equivalence Classes) over Integers

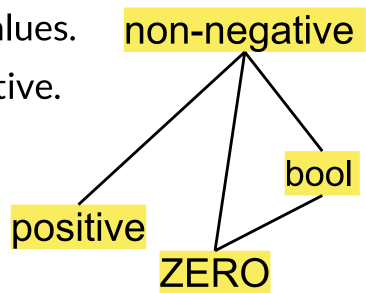
- A program variable v has a value from a concrete domain C (here Integers)
- At a point in the program, v can be typed by a subset of C (an equivalence class)
- This concrete domain C is mapped to symbolic abstract domain A
  - Here: subsets of C=int to symbolic A="abstract symbolic sets over ints"
  - Top means *any-concrete-value*, bottom means *none*
  - cpo suprenum operation *meet*: unioning all subsets



Law of Join of Control Flow in an Abstract Interpreter

When the abstract interpreter does not know what the type of a variable will be from 2 or n incoming control-flow paths at a join,
it takes the suprenum („union“) of the equivalence classes of the abstract domain

- In a *join point* of the control flow (at the end of an If, Switch, While, Loop, Call), an abstract interpreter will not know from which incoming path it should select the value
  - **If**: two paths
  - **Switch:** finitely many paths
  - **While, Loop:** infinitely many paths
  - **Call:** from a return of the called procedure
- In order to proceed, the interpreter chooses the *suprenum* of the equivalence-class values of all paths (the *meet* of all values of all incoming paths), i.e. it will choose the union or the most simple abstraction of all equivalence-class values.
- Ex.: in a Switch the values of the branches are ZERO, bool, positive.
  - The interpreter will choose “non-negative”, to cover all.



Ubiquituous Abstract Interpretation for Deep Analysis of Programs and Models

- Any program in any programming or specification language can be interpreted abstractly, if
  - A syntax tree (link tree, or a graph model) is given
  - An abstract semantics is given, mapping the tree nodes to interpretation functions over abstract values
- The abstract interpreter is an implementation of the metaclasses of the M2 metamodel
- Examples:
  - Imperative Programs: A.I. of embedded C, C++, Java, C#, Scala programs
  - Rule-based Programs: A.I. of Prolog rule sets, A.I. of ECA-rule bases
  - Models: A.I. of state machines, A.I. of Petri Nets
- Functional analysis
  - Value analysis ("data-flow analysis") for numeric values and pointers
    - Range check analysis, null check analysis
    - Heap analysis, alias analysis
- Quality analyses:
  - Worst case execution time analysis (WCETA)
  - Worst case energy analysis (WCENA)
  - Security analysis

22.4 Iteration Strategy of Abstract Interpreters (Intra- and Interprocedural Visit Order)

Example: Interpretation of a Procedure with a Worklist Algorithm

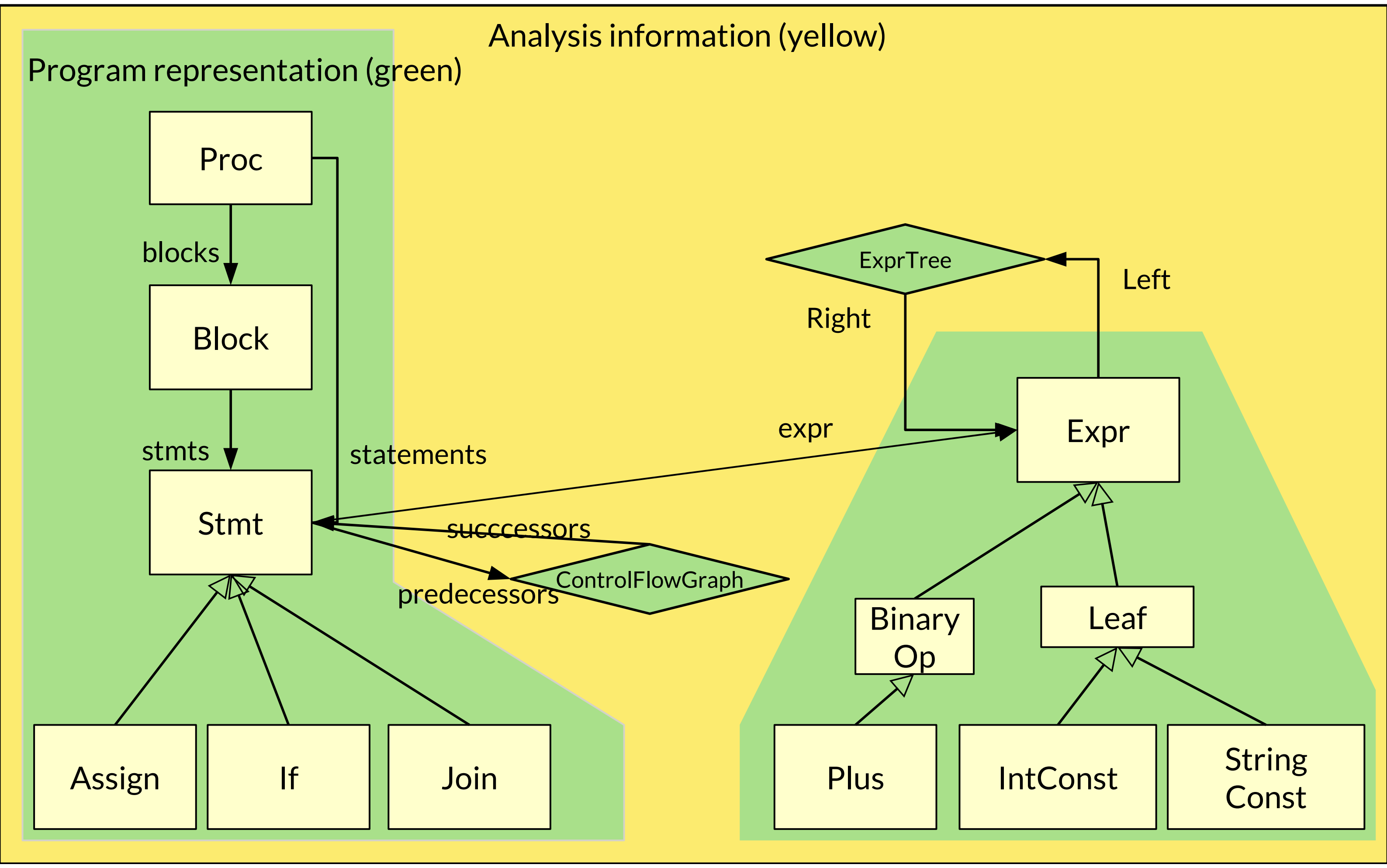
- Iteration can be done *forward* over a worklist of statements that contains "nodes of the syntax tree not finished"
- The abstract interpretation functions `fᵃ(p)` are applied as long as there are changes in the attributes
- For a AG this means: application of attribution functions is free-choice

```
worklist := nodes of syntax tree;
WHILE (worklist != NULL) DO
SELECT n:node FROM worklist;
// forward propagation from predecessors to n
            FORALL p in n.ControlFlowGraph.predecessors
               X := meet( fᵃ(p.abstract_value()) );
            // test fixpoint condition
            IF (X != n.abstract_value()) THEN
               // reattribution
               n.abstract_value() = X;
               worklist += n.ControlFlowGraph.successors;
           END
END
```

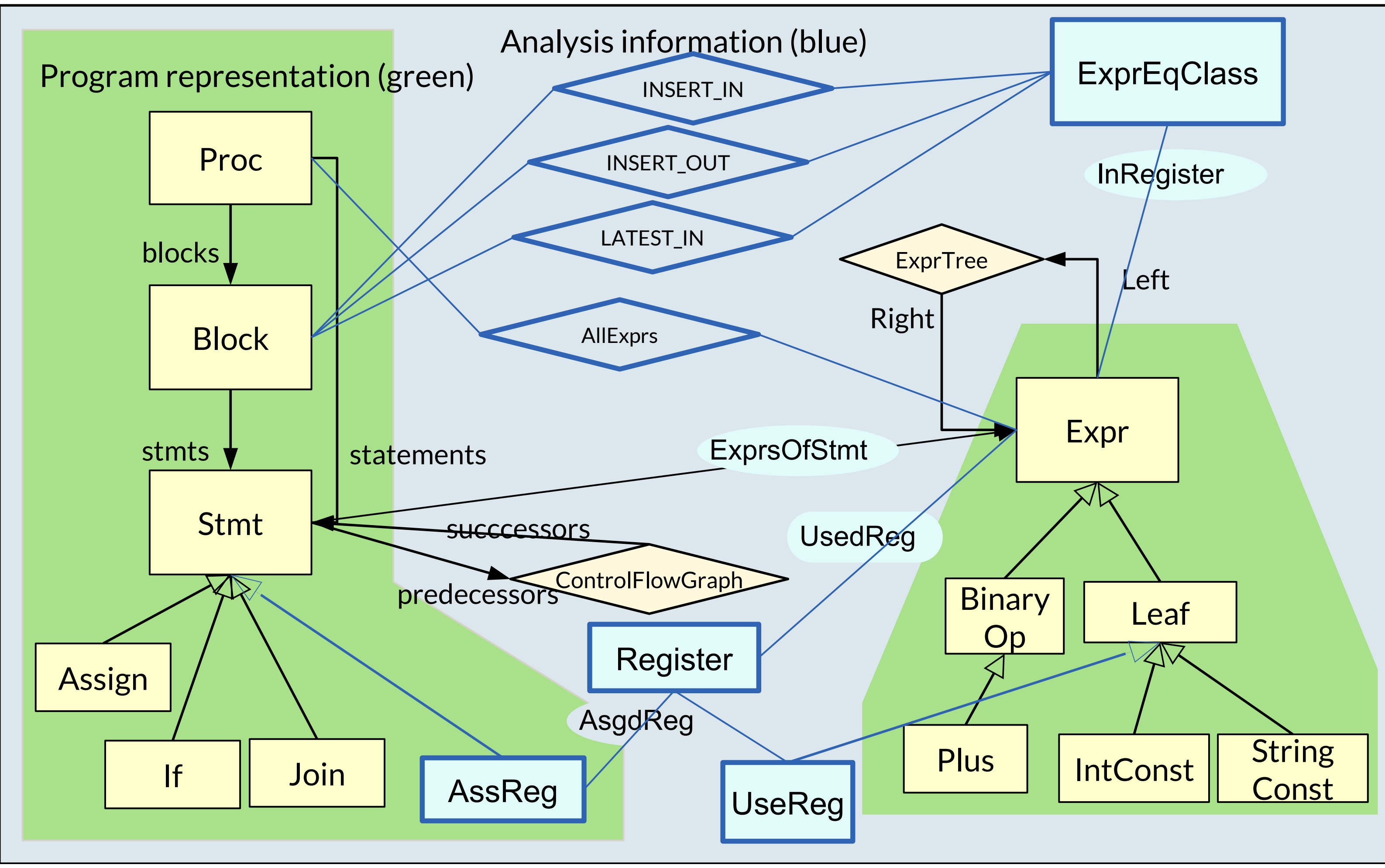
Building Abstract Interpreters on M2

- In the TAM style, the interpreter works basically with Design Pattern “`Interpreter`”, as from the Gamma book
- What has to be modeled:
  - A model of the program (program representation), with Class, Proc, Stmt, Expr, etc
    - Most often, this is a syntax tree (with links)
  - A model of the analysis information
    - ControlFlowGraph: has inserted Join nodes representing control flow joins in If#s and While's
    - AbstractValue domains: e.g., abstract integers, abstract intervals and ranges, abstract heap configurations
    - Environments binding variables to abstract values

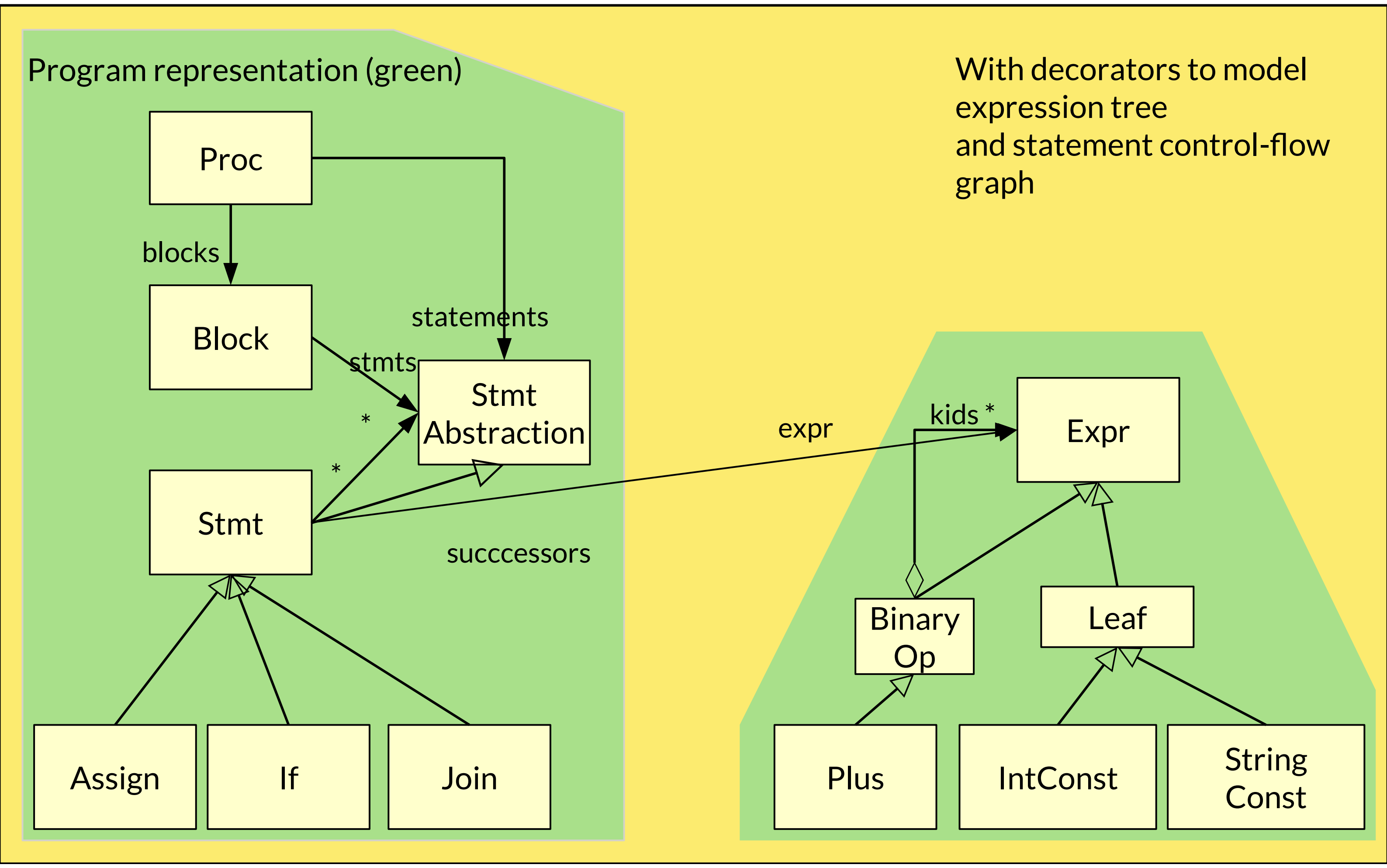
A Simple Intraprocedural Program (Code) Model (Schema) in MOF



Q14: A Simple Intraprocedural Program (Code) Model (Schema) in MOF

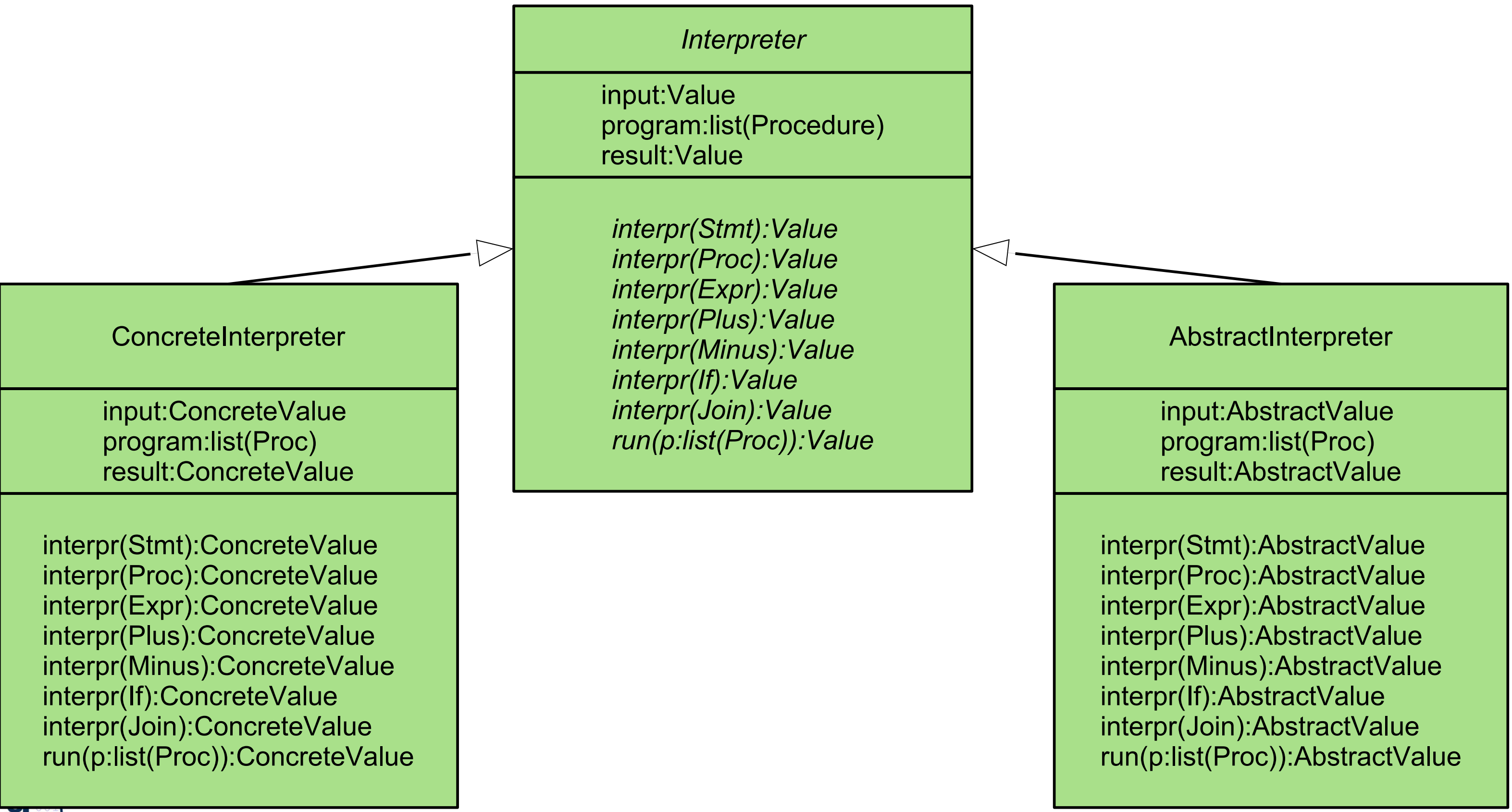


A Simple Program (Code) Model (Schema) in EMOF



An TAM-Design of an Interpreter Family of a Programming Language

- Concrete and abstract interpreters are “twins”, i.e., have the same interface but working on concrete vs abstract values



Example: TAM-Interpretation of a Procedure with a Worklist Algorithm

- Simplified assumption: one value per statement is computed by the abstract interpreter.
- The value at the return statement of the interpreted procedure is the final result of the abstract interpretation

```
CLASS AbstractInterpreter EXTENDS Interpreter {
…
  FUNCTION interpr(p:Proc):AbstractValue {
    worklist:list(Statement) := p.statements;
    WHILE (worklist != NULL) {
      SELECT current:Statement FROM worklist;
      // forward propagation from current.predecessors to current
      FORALL pred in current.ControlFlowGraph.predecessors {
        NewValue := meet( pred.abstract_value() );
      }
      // test whether fixpoint is reached
      IF (NewValue != current.abstract_value()) {
        current.abstract_value() := NewValue;
        worklist += current.ControlFlowGraph.successors;
      }
    }
    RETURN p.statements.last.abstract_value;
  }
}
```

22.2.2 Free-Choice Visit Theorem (Intraprocedural Coincidence Theorem) for Abstract Interpreters

[Kam/Ullman] Intraprocedural Coincidence Theorem:

The maximum fixpoint of an iterative evaluation of the system of abstract-interpretation functions f_n at a node n
is equal
to the value of the meet-over-all-paths to the node n (MOP(n)).

- Forall n:Node: $MFP(n,f_n) = MOP(n,f_n)$
- The theorem means, that no matter how the abstract-interpretation functions are iterated over a procedure (free-choice visit), if they stop at a fixpoint, they stop at the meet-over-all-paths
  - Any iteration algorithm can be used to reach the abstract values at each node (i.e., the maximal fixpoint of the function system)
  - The paths through a procedure need not be formed (there may be infinitely many), instead, free iteration can be used until the fixpoint is found (until termination of the iteration)
- The application of an attribution function is similar to a free rewriting step

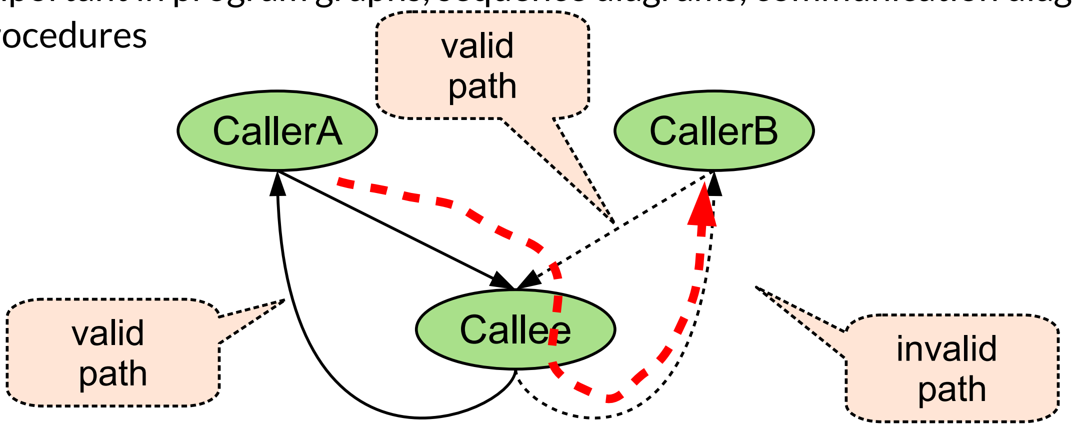
Example: Backward TAM-Interpretation with Worklist Algorithm

- Iteration can be done with many strategies
- E.g., iterating *backward* over a worklist that contains “nodes not finished”
- Other alternatives: innermost-outermost, lazy, etc.

```
CLASS AbstractInterpreter EXTENDS Interpreter {
…
  FUNCTION interpr(p:Procedure):AbstractValue {
    worklist:list(Statement) := p.statements;
    WHILE (worklist != NULL) {
      SELECT current:Statement FROM worklist;
      // backward propagation from current.successors to current
      FORALL succ in current.ControlFlowGraph.successors {
        NewValue := meet( succ.abstract_value() );
      }
      // test whether fixpoint is reached
      IF (NewValue != current.abstract_value()) {
        current.abstract_value() := NewValue;
        worklist += current.ControlFlowGraph.predecessors;
      }
    }
    RETURN p.statements.last.abstract_value();
  }
}
```

Interprocedural Control Flow Graphs and Valid Paths

- Transfer Functions f# can be defined on Nodes f#(n), or even on Edges f#(e)
- **Interprocedural edges** are call edges from caller to callee
- **Local edges** are within a procedure from "call" to "return"
- Problem: not all interprocedural paths will be taken at the run time of the program
  - Call and return are *symmetric*
  - From whereever I enter a procedure, to there I leave
- An **interprocedurally valid path** respects the symmetry of call/return
- Important in program graphs, sequence diagrams, communication diagrams, Petri-net procedures



Interprocedural Problems

- Non-valid interprocedural paths invalidate the coincidence for the interprocedural case
- Knoop found a restricted one [CC92]:
  - No global parameters of functions
  - Restricted return behavior

The End

- Explain the differences of an interpreter and an abstract interpreter!
- Why are interpreters and abstract interpreters specified on an abstract syntax tree specified by an RTG?
- Can models be interpreted?
- What are the differences of an abstract interpreter and an attributed grammar?
- Why is a reference attributed grammar (RAG) more expressive than a pure AG?
- What happens at a control-flow join during an abstract interpretation?
- Explain abstract domains and the iron law of abstract interpretation.