MIT(16.399) - Abstract Interpretation (Patrick Cousot)

Satyendra Kumar Banjare

Contents

Chapter 1

Overview of Abstract Interpretation

This is a perfect introduction to all the things which briefly comprises of the following topics :

- **Static Analysis** : It concerns the pre and post "static" conditions of a program's variables/states. All the logical conditions are checked for safety, only if it exists. By incorporating the method of Abstract Interpretation, we seek to perform basically two operations Verification & Abstraction.

- **Semantics & Undecidability** : The concrete semantics of a program formalizes (is a mathematical model of) the set of all its possible executions in all possible execution environments. This can be thought of as a set of all possible trajectories given that starting points may or may-not be same for an assumed program's mathematical model.

  Undecidability refers to non-terminatibility of a program state and we say that infinite loop has begun, conversely the solution term is not possible to achieve. The Programs structured after should use the termination of earlier as an indication to start, on not getting that, It is possible that it wont start at all.

- **Safety Proofs** : Safety Properties guarantees the correct functioning of a program. A safety proof simpy states there exists no relation between correct functioning and incorrect behaviour (Bugs and undecidability) of programs. Test are a Subset of all the possible Safety proofs for a given semantics and performance parameters.

- **Abstract Interpretation & Formal Methods** : Abstract Semantics serves as a super set of all possible executions hence it covers all posiible behaviour. Therefore proving correctness of abstract semantics should be preferred. Formal methods are just abstract interpretations of working behavious and it differs in having a pre-defined semantics (example - programming language's grammar) and has been used to perform model-checking.

  It uses some deductive methods and static analysis methods to achieve safety proofs. As a basic need these proofs / abstract interpretations must be Sound, Precise and easy to get.

  Some graphic example covers possible trajectories and forbidden zones.

- **Trace Semantics** : Collection of "Ordered" states of executions is trace and adding rules for their behaviour is simply called trace semantics. These can be of two types finite and infinite. A prefix trace semantics has pre-set maximal number of possible behaviours. This is often decided by using some type of logical condition. It is equivalent to defining subsets.

- **Collecting Semantics** : Collection of "unordered" states to executions is called collecting semantics. It just considers the presence of a given state and not how that particular state was reached.

- **Interval Abstraction** : This is simple performing abstract interpretations and tracing after fixed number of steps or fixed time.

- **Abstracting Sets & Concretization** : Consider a Concrete set 'S' the contains all the concrete objects/states (defined by the concrete semantics) and Abstract set α(S) that contains all the abstrat objects/states defined by the abstrat semantics.

  Abstraction function α maps a set of concrete objects to its abstract interpretation and a Concretization function γ that maps a set of abstract objects to its concrete ones. This easily follows that it should be monotone and that the composition $\alpha o \gamma$ is extensive and the comosition $\gamma o \alpha$ is reductive.

  Galois Connections

- **Convergence Acceleration** : This deals with fastly analyzing the behavioural semantics using some techniques listed as "upward iteration with widening" and 'upward iteration with interval widening'. Stabiliy of this method is discussed.

- **Interval Analysis** : This is a mathematical method and this basically defines and modifies sets (real numbers) as the program proceeds the execution.

- **Refinement of Abstractions** : Having defined abstractions, it is rather more important to have a refined abstraction that just covers all possible states. Graphically, It is the minimal figure [PolyHedra] covering all the state points. Beautifully explained by the usage of congruences Linear and Trapezoidal linear to achieve more refined abstractions.

  Graphicaly Explined the possiblity of having discontinuous surfaces/geometry . The abstraction should smartly covering all these.

  Partitioned iteration with Widening and interval widening are explained beautifully.

- **Combinations of Abstractions** : Various smaller abstractions can be combined to develop abstractions for the complete program. Thus it leads to using method of backward and forward analysis for approximating behaviour.

- **Backward & Forward Analysis** : As the name implies this differ in just with iterating the program in the direction of either forward, one state covered after other in an ordered manner. A mixed Backward and forward analysis involves recollecting and reusing a previous state for the analysis of current state.

Chapter 2

Software Verification Problem

- **Bugs** : Recent advancements in Computational power has also increased the possibility of bugs that are commonly known to us in form of overflows of memory. A smart and safe implementation is needed.
- **Program Verification Methods** : These are ways to guarantee the correct working of a program. It may not be the state of art but definitely better than existing methods.
  - **Testing/ Writing Tests** : It gives the presence of possible bugs relative to some cases. thus it is very possible that many bugs can be easily missed.
  - **Verification** : It gives the complete absence of bugs related to a given specification. It includes the conditions of overflows and undecidability problem. It is more useful.
- **Partial Correctness** : It involves having a PreCondition 'P', PostCondition 'Q' relative to a given program condition 'C'. Then Partial Correctness follows that P holds on the beginning of C and Q holds at the termination of C. Hoare triples notation is explained.

  Invariant is an assertion that always holds irrespective of program condition. example : the physical parameters in case of hardware specification.

  An example of Euclidian Integer Division lemma is also explained.
- **Floyd/Naur invariance proof method** : It basically involves proving two things :

  1. The assertions on the entry point of program holds.

  2. The inductively defined assertions hold. Termination of one program also follows that the invariant assertion does hold.

  - **Assignment Verification** : It involves proving 'There Exists' type of problem. The invariant should hold. An example of integer increment is explained.
  - **Conditional Verification** : It involves program 'Conditional existance' of program invariants. An example of abs(x) is explained.
  - **While Loop Verification** : It involved proving the looping in a program. Basically the while condition, the next assignment verification is carried out for the iterating variable.
- **Hoare Logic** : This involves 5 rules namely the assignment axiom, composition rule, if-else rule, while rule, consequence rule. It is used to explain the correctness in proof steps of Euclidian Integer division lemma.

- **Termination** : The most important aspect of program is that it should terminate, checking for relative completeness is also an important part of software verification. Mathematically it is expressed as having a well-founded relation.
- **Floyd termination proof method**:

  1. Exhibit a so-called ranking function from the values of the program variables to a set S and a well-founded relation r on S.

  2. Show that the ranking function takes r-related values on each program step.
- **Total Correctness = Partial Correctness Ť ermination**.
- **Manna/Pnueli logic**
- **Transformer Calculus** : Dijkstra introduced predicate transformers, namely weakest liberal PreCondition [WLP] and Weakest PreCondition [WP].

  These transformer rules included :

  - Skip (leaving the state unchanged) it's evaluation is same for WLP and WP.
  - Abort (to leave state that never ends) its evaluation is true for WLP and false for WP.
  - ';' composition in sequential order. it behaves similar way for both.

  Other aspects of transformer calculus :

  - Non-deterministic choice of commands may give different results depending on how it is binded WLP or WP.
  - Guards are used preconditions. They can be implemented in two ways like '?G' and '!G' meaning basically the if-else condition for 'G' and not condition for 'G' respectively.
- **Iteration** : Repetitive construct by Dijkstra involving guards .
- **Automatic Verification Methods** : Involves using theorem provers to auto-generate the next logical states and easy out verification by wrong-case elimination.

Chapter 3

Abstract Invariance and Termination Proofs

- **Fundamental Limitation** : The Hilbert's Axiomatic program to reform the basis of mathematics and it being proved incorrect by Godel's Incompleteness theorem. The Axiomatic system is not possible entirely !!
- **Decidable/Semidecidable/Undecidable** : Decidable if algorithm exists to solve the program in finite time. Undecidable if there exists no algorithm to solve the problem in finite time. Semidecidable if solvable in finite time for some cases and completely un-solvable for other cases.
- **Termination Problem** : Given input data and some preconditions, whether that sequential program will terminate or not. Non-Termination problem to check whether the program will not terminate.

  Using a simple example of Interpretor program, it has been shown that termination can be semi-decidable as well as Undecidable using 'reductio ad absurdum'.

  Non-Termination is not semidecidable. Simply anding semidecidable(termination) and semidecidable(Non-Termination) it results to decidable(termination).
- **Problem Reduction** : it follows that if program 'P' is decidable, it means its termination proof is also 'decidable'.

  Constant propogation problem (retaining constant value of a variable in a program) is undecidable !!
- **Approximations** : As a way to tackle (thought it is absurd to do completely) , We can have approximations of things that might happen. example - false alarms in ASTREE.
- **Computational Complexity** : Time and space required to solve a problem. It has classes such as P , NP, Co-NP
  - **NP-Hard** is observed if every NP problem can be redued to it.
  - **Boolean SAT problem** They are NP-complete Problems.
    NP-complete problems are generally solved by a fixed parameter approach.
    Probabilistic nature of solution algorithms has overall good average range of solving time.

Approximations with optimal error range is done to solve the problem.

Example of SAT solver algo - DLL algorithm.

- **Termination Analysis Method** : It proves the termination conditions of a program. Example of a while loop is explained. This methos uses 4 major parameters in consideration : Termination precondition , Loop invariant, operational semantics and Ranking Function.

  Forward reachability properties and partial correctness is referred to guide the program beginning, Backward reachability for the program termination.

  - **Auxiliary termination counter method** : It involves using an auxilary counter to enforce the bound (termination) of program. The Loop invariant is found by using a forward relational static analysis. The operational semantics is then known by performing a forward static analysis.

    Floyd's Ranking Method to prove Termination then uses a ranking function to determine rank of loop/program invariant. This Ranking function should be always decreasing and Non-Negative.

    Example of arithmetic mean function's termination using ranking function is beautifully explained.

  - One method involves Expressing the loop invariant and relational semantics as numerical positivity constraints. Then the floyd's method is applied to get the termination conditions.

    The Loop Invariant for a given step of loop can be expressed as an anding of variables of op-semnatics at each previous step !!. see equation in the slides.

  - **Lagrangian Relaxation** : Involves reforming the problem by estimation the conjuctiona and implications using lagrangian relaxation.

    It is simple mathematically to understand λ_i lagrangian coefficients for the given problem. Finding these coefficients is carried using different methods. Basically we are approximating a major problem/ major curve by a smaller/easier representation.

    Yakubovichs procedure is often used to prove the completeness.

  - **Parametric Abstraction of Ranking Function** : Assuming a Ranking Function form for some unkown parameters we are trying to solve for those parameters.

  - **Approximating Universal Quantifications using Linear Matrix Inequalities (LMI)** LMI of ranking function is formed. Further problem reduction shows that this LMI problem reduces to provinf semi-definitiveness of matrix which can be done using mathematical programming and/ord using LMI solvers.

    Extending this, we should try to solve the Convex constraints (see slides pictures) also by the semidefinite programming.

    This involves using linear programming, ellipsoidlal methods and interior methods.

    Futher more we can transform non-convex constraints to convex constrains then aplly the previous steps to solve things out.

  - **Handling Non-determinancy** includes case analysis, interleaving for concurrency and encoding with explicit scheduler.

  - **Floyd's method for invariance** involves determining loop precondition and loop invariant, further also proving that the loop variant is inductive.

This transforms the problem to a Bilinear Matrix Inverse type problem.
BMI solvers by Semi-Definite programming is thus used.

Chapter 4

Set Theory

Basics of Set theory is explained in this including its representations, terminology and definitions of Relations like transitive, reflexive, symmmetric-antisymmetric and relation equivalence .

- **Partitions** : Sub parts of set the complete it on conjuction.
- **Posets** : Partially Ordered Sets (Posets) have an ordering condition associated with each of its elements.
- **Hasse Diagram** : Diagram to represent the Posets in its trnsitive reduction form. Drawn in a bottom up faishon.
- **Encoding N with sets** is representing natural numbers using set notation. Expressed as : n+1 = { 0,1,2...n}.

  Various Other things including function, pre-image concept and dual image are explained too.
- **Wosets** : posets with no infinitely strictly decreasing sequence (for natural numbers).
- **Equipotence** : It is an equivalence relation between 2 finite sets.
- **Cardinals** : That property of A which is inherent in any set B equivalent to A. Here two sets are called equivalent (or equipotent or of the same cardinality) if it is possible to construct a bijection (one-to-one correspondence) between them.

  Mathematically it is possible to carry out mathematical operations on the cardinal sets like Adding, multiplying etc.
- **Ordinals** : This means the the following well-ordered set (Woset) has previous elements as its members.

  Ordinality means the Woset has 'is and element of' as the required ordering condition.

  Ordinal number is can be represented in set notation in a form expresses above.
- **Well Founded Sets**
- **Burali-Forti Paradox** : The class 'O' of ordinals is not a set.

Chapter 5

Operational Semantics

Basic Lexer-Parser Theory has been explained with example.

- **Parsing and Translation** : Involves using a lexer to tokenize the given code. Parser to combine the tokens using the provided grammar rules/abstract syntax in a bottom-up or left-right approach.
- **Grammar semantics** : These convert the tokens into semantic values using what is characterized as semantic action.
- **Grammar rules & Priority decidance** : involves assigning priority to operators.
- **Concrete syntax *** : Basically writing all possible variations differently.
- **Abstract Syntax *** : Writing similar type of grammar declarations at same place. may be as an Inducive definition.
- **Symbol Table** : list of all the program variables
- **Program Labelling** : labelling the different part of a program.
- **Operational Semantics of SIL** : Involves Small-step and Big-step operational semantics where we keep a record of every step variable's values and only final and initial values respectively.

Chapter 6

First Order Logic

- **Formal Logics** : It has a formal language, a model-theoretic semantics and a Deduction system. It needs to be sound and complete.
- **Propositional Logics** : Formula representation of meta theories often involving the boolean algebric terms.
- **Model-Theoretic classical propositional logic** : it adds the semantics to model behaviour defined by formulas. Can be understood as imporving the contextual interpretation of same code.
- **Models** : A modelling semantics associates with the formulaes and variables. A model for a formula exits if its modelling semantics exits.

  Formulaes are further classified into a satisfiable, unsatisfiable and valid.
- **Hilbert Deductive System** : This logic system uses some basic axioms along with an inference rule. The objects on which these axioms are applied are formulae / logical statements.

  The Hypotheses in analyzed based on deduction from base cases.
- **Proof** : Provability and proof of a statement ϕ from τ exists if $\tau \in P(F)$.

  simple example cases have been explained. if you are familiar with proof assistants, this is rather the basic exmaple you would have solved.
- **Soundness and Consistency of Deductive System** : proof system is sound if provable formaulaes do hold. It is consistent if there are no contradictory proofs.
- **Normalization** : Negative form basically defines the negation of any formula. Normalization means expression the combination of formula in a better "conjuctive" (or-ed) form. It is just a method of representation but is very useful. example : CNF and negative CNF.
- **Syntax specification of first order logic system** : the important features of first order logic system includes
  - Lexems, includes symbols, constants and variable
  - Terms includes the objects whose value if determined and it can be constant, variable or a function.

- Atomic Formulae are used to represent the basic/initial facts/featueres of the proof system
- Free Variables
- First order syntax
- bound variables
- theories

Substitution refers to replacement of variables by 'Terms' of logic system.

- **Semantics of first order logic system** : Includes Assigning Environment preconditions to the variables, Interpretation using a domain of Interpretation, Assignment (or Substition speaking syntactically).
- **Deduction System for first order logic** : using the basic two inference rules (modus peanus and generalization), we define the working of the deuction system.

  Two basic examples are explained.
- **Extending first order logic system** : involves incorporating the theorems as terms in a logic system. this thus extends the logic systems's generalization rules.
- **Properties of Hilbert Deduction System** : It is sound, complete proof of which can be explained using leibnitz inequality rules. It is however Un-decidable.

Chapter 7

Program Specifications

- **Program Semantics and Specifications** : semantics refers to the properties of program it does have, specifications refers to the properties the a program should have. program semantics include different type of Operational semantics, natural and reachability semantics.
- **Small Step Semantics** : It is a transition system of program states defined with transition rules and final states.
- **Trace Semantics** : records the sequence of program states.
- **Partial Trace Semantics** : records the sequence of program states given a precondition or a prefix trace.
- **Maximal Trace Semantics** : it is performing trace semantics for the state 'S' as well as it's negation 'not(S)'.
- **Big-Step Operational Semantics** : It is the semantics of transition system. It represents deciding rules for the Small-Step semantics.
- **Natural Denotational Semantics** : decides the semantics of transition system for some blocked states. the 'natural' term possibly means the real case scenario where there are very possible blocked cases/states.
- **Natural Operational Semantics** : deciding Op-semanctics with care taken the non-termination is never possible.
- **Forward Reachability Semantics** : semantics of forward transition of program states.
- **Backward Reachability Semantics** : semantics of Backward transition of program states.

  From Abstract Interpretatin point of view, the different semantics are simply abstractions of each other.
- **Specification of program Semantics** : decideds the properties that a program semantics should have. Covers the Idea of Correctness of the program Semantics.
- **Trace Properties** : trace semantics should be a subset of correctness specification properties.

  includes the idea of relational properties, partial correctness and total correctness.

- **State Properties** : It is also a weaker form of correctness of program specification properties. includes the idea of forward reachability, specification and runtime errors.

  example cases of proving state properties includes ideas like : there should exist no run-time error.

- **Program Logics** : Formal Descriptions of program logic include representing programs specification in terms of sets, relation between sets and developing the trace semantics.

- **Set of States Predicate logic** : exlpined earlier, this deals with expressing the state transition in terms of disjoint mathematical sets.

  Developing Sets, we can understand the program specification in more better way.

  This can be further extended to also account for errors (on safe side we should definitely account for these error !!).

  It also involves extending the assignment properties too.

- **State Relation Predicate Logic** : It involves extending the different state properties'. this means extending syntax rules, Assignment rules, predicate rules and trace rules.

  This extends the set-wise relation of semantical attributes of the program.

- **Trace Predicate Logic** : For better assesment of trace semantics, we have used some of the predicates such as having a finite time interval, using atomic formulae.

- **Extending logic semantics** : This is performed by extending the syntax of the predicates, extending the assignment semantics of 'Control Variables'.

- **Linear Time Temporal Logic** : temporal logic specify out the execution trees, A linear time temporal logic essentially analyzes the execution trees by deciding traces one at a time.

- **Linear Time Temporal Logic syntax** : is similar to 'terms', includes definition for universal quantification, generalizations and negations.

- **LTL expression system** : Includes LTL auxiliary operators - Eventually , Henceforth, and waiting .(all these words to be understood with respect to the time dependence nature of the program.)

  includes expressing temporal tautologies. (differnt expression forms)

- **Synchronous Languages** can be used to specify sets of finite traces.

- **comparative example of specification** : a simple filter program is explained where by making it clear the temporal logic makes it difficult to understand each states.

  it is thus difficult to auto-generate useful code.

Chapter 8

Collecting Semantics

Collecting Semantics defines a whole class of static analyzers, all those one can abstract/apporximate .

- **Collecting Semantics of Arithmetic Expressions** : is a complete join morphism. Forward Collecting Semantics defines all the possible values a given arithmentic expression can evaluate to given the environment conditions.
- **Forward/bottom-up Collecting Semantics of Boolean Expressions** : defines the set of all the environment where a given boolean expression can evaluate to true.

  small-step transition system is follwed in forward collecting semantics.
- **Collecting Semantics of commands** : structural Big-step collecting semantics of 'commands' link the entry-exit points of program states not only in a natural order but also represent conection/links from any current state to other subsequently reached state.
- **Structural Op-semantics** : We define and work with entry-exit state points throughout the program analysis.

  Op-semantics of if-else or conditional command has been explained.
- **Structural big-step Semantics of iteration** : Transitive closure relation on iteration program is proved.
- **Structural big-step Semantics of Sequence** : Reflexive Transitive closure relation on sequence of programs is proved.
- **Classification of Program Trace Properties** :
- **Safety** :
  - Ensures Bad things can not happen.
  - Prefix closure (PCI) of traces if pre-conditions, set relations created for traces of a safety program.
  - limit closure defines the set of prefix closres that can have infinitely many prefixes.
  - safety property 'S' means traces set bounded by prefix closures.
    example - invariance.

  - Safety properties can be analyzed only by looking at partial program behaviour.

- **Liveliness** :

  - A property is a Liveliness property if the property's behaviour can be representedas a subset of its PCI.
  - Proving the liveliness properties requires considering the infiniteness of program's behaviours. Example - termination.

- **Dual Limit** : double negation of propery PCI of a property P.

  results in a non-monotonicity, extensivity.

- **Decomposition of total correctness** : Any Property 'P' can be broken down into partial Safety Property and Liveliness Propoerty and can be represented as a intersection of both..

Chapter 9

Lattice Theory 1

- **Covering relations** for posets basically mean if one set contains elements of other. it covers the other set.
- **Inverse of Partial Order is a partial order.** :
- **Hasse Diagram** : line diagram of posets' points. it is shown that it cna be extended.
- **Antichain** : is a poset if two random elements can be equal.
- **chain** : is a poset if two random elements are either less than one another, a¡b ob b¡a both possible.
- **Chain Condition** : Ascending and Descending chain conditions, ACC & DCC. Ascending if gradually the number of equal elments decrese on a less than or equal poset. descending if numbers increase.
- **Tree, Max - Min** : A poset tree means that it's down set is a woset.

  Max-min is same as top-bottom of a poset.
- **A poset is non-infinite if it satisfies bith ACC and DCC conditions.**
- **Duality** : Dual of a poset is its inverse. Its hasse diagram is simple vertical reflection of original poset.

  duality principle says if a statement is true for a poset, it is also true for its dual.
- **upset & downset** : upset if the superset poset has all members greater than current poset's members. downset if lower.
- **Directed Sets** : It means x less than or equal z & y less than or equal z for the elements of a subset of a poset with property 'less than or equal' .
- **Upper and Lower bounds** : max min for a subset of a poset.
- **properties of lub/glb** : lub/glb of a poset is unique.

  lub exists if and only if there is a bottom in poset and glb if there is a supremum in the poset.

- **Subset ordering** : functional values of elements of subset of a poset also follow the characteristic property of that poset.
- **Lattices** : are posets. join semi lattice if any two elements have a LUB. meet semi lattice if any two elements have a GLB.

  A lattice is both a join and a meet lattice.
- **Semi-Lattice properties** : they follow associativity, commutativity and idempotency. They also follow Duality principle.
- **SubLattices** : of a lattice is a lattice following the characteristic ordering property. all it's elements are subset of original lattice.
- **CPO & Complete Lattices** : complete partial order (CPO) is a poset that its increasing chain of P has lub in P.

  complete lattice is poset if its every subset has a lub in P.

  complete lattice is never empty and has both LUB and GLB.
- **Bottom and top of latttice** : max-min of a lattice . easy to refer from the hasse diagram.
- **Finite Lattices** : are non-infinite lattices and are complete lattices.
- **Boolean algebra for Lattice** : They follows distributive , commutative , associativity and transitivity properties.
- **Distributive Lattice** : if elements follow distribuitve law with GLB and ULB as additon and multiplication operations.
- **Semi-infinite distributive lattice** : if they follow either infinite join distributivity or infinite meet distributivity.

  infinite if it satisfies both.
- **Complements** : of an element of a poset exists if their ULB is poset's bottom and their GLB is poset's top. They are not unique
- **bounded posets and lattice** : have both a maximum and a minimum.
- **Complemented lattice** : if every lement has a complement in itself.
- **Boolean Lattices** : A Boolean latticeis a complemented distributive lattice.

Chapter 10

Lattice Theory 2

- **Moore Family** : is a subset of a poset such that glb of P is an element of M.

  it is a complete lattice
- **Moore Closure** : exists if the glb of subset of a moore family is an element of M. example - convex subset of a poset.
- **Linear sum of Posets** : is a poset, possible if glb of one is less than lub of other.
- **Combination of Posets** : bottom lifting adds a bottom to P. top lifting adds a top to P.
- **Flat ordering** is just simple ordering the equal elements of a poset, that lie on a some horizontal line in the hasse diagram.
- **Smashed & Disjoint sum of posets** : smashed means joining two posets with one same element that is glb of one poset and is also a lub of another. Is exactly one common point

  disjoint if there is no only one such common / same element. rather there can be two joining together the hasse diagrams to form donut like shape.
- **cartesian product of posets** : every possible cartesian set that can be made from poset's elements.
- **smashed cartesian product** : has only one top/bottom element.
- **cardinal power** of set x maps it to a poset with pointwise ordering.

Chapter 11

Ordered Maps Galois Connections 1

- **Maps Between Posets** : simply extending the idea of relations including various morohisms for poset.
- **Mono/Anti-tone Maps** : monotone if one to one and order preserving. anti-tone if order reversal. also referred to as dual.
- **characterization of monotone maps** : is done by checking if the lub of one poset lies in the subset of other poset.
- **Order Embedding** : refers to a poset map if the functional value of that poset's elements can also form a poset.

  It is injective.

  Order isomorphism is an order embedding which is bijective.
- **Preserving Maps** : can be of type join preserving or meet preserving. (seee lattice theory part for clear idea).

  f(lub of elements from two posets) = f(element of A) lub f(element of B).

  They are not monotone.

  all the maps may not preserve lubs or glbs.
- **Complete Join Preserve Maps** : if a preserving maps preserves all the GLBs and LUBs- then it is called complete join preserving maps.

  not all finite Join/meet preserving maps are complete.
- **Continuous Maps** : if f(lub of chain C of poset) is a subset lub of f(chain).

  they are monotone.
- **Lattice Morphisms** : extends the relations for lattices.
- **Notations for monotone lub/glb preserving co-continuous maps** : simple notations for clearing out the type of mapping done.
- **complete lattice of point wise ordered monotone maps on lattice points form a complete lattice themselves**.

- **Encoding Maps between Posets using boolean algebra** : refers to ordering and grouping the set elements in some ways. example - CNF conjuctive normal form used in SAT solvers.

- **Shanon Trees & Boolean Decision Diagrams** : shanon trees of boolean expression can be represented by BDD.

  this can be done by merging and reducing to a directed acyclic graph, DAG and forther elemeinating useless nodes.

  they are isomorphic to each other.

- **Ordered BDD** : if left subtree is not equal to right. this would create separate and different hierachy for decision dependence of elements and thus 'orderize' them.

- **Typed Shanon Tree** : uses a mathematical sign (+ / -) to represent the difffence instead to sing 0 or 1. only one is used.

- **Typed Decision Graph** : is obtained from typed shanon trees using the previous merging and eliminating rules.

  thus boolean expression can be represented by TDG.

  TDG of a boolean expression is unique.

- **TDG Operations** : boolean operation uses shannon decomposition to avoid recursive calls to same node (physical address).

  this is similar to checking for equality for a TDG.

- **Encoding of Complete Join Morphisms with join irReducibles** : element of a poset is join irreducible if its not the bottom or it is either of two elements whose lub is equal to it.

- **Descending Chain irReducibles** : Encoding of lattice satisfying DCC can be done by the image of join irreducibles.

- **Atoms and join irReducibles** : atoms of a poset refers to those satifying the duality principle with bottom of poset.

  image of boolean lattice's join irreducibles is a subset of atoms of that poset.

- **Closure Operators** : that maps p into p.

- **Upper Closure Operators** : closure operators that maps to higher elements.

- **Topological Closure Operators** : is a closure opertor and is strict, extensive, follows join morohism and is idempotent.

- **Morgado Theorem for upper closure operators** : says an operator is upper closure operator if and only is operator(x) less than or equal to operator(y) forall x less than or equal to operator(y). x, y are elements of poset.

- **Fixpoints of closure Operators** : it returns the same value.

- **Galois Connections** : Two maps A and B mappig from two posets P to Q and Q to P respectively are said to form galoin connections if A(element of P) (characteristic property of Q) (Q's Element) and similar for B.

  example - bijective functions.

- **Galois Connections - Functional Abstraction** : representing the mapping as functions and let them form galois connections.

- **Galois Connections induced by upper closure operators** : extends Morgado's condition that makes a galois connection.

- **unique adjoints** : one adjoint of galois connection determines other.

- **Properties of Galois Connections** : is monotone.

  A o B is lower closure on P. B o A is upper closure on Q.

- **Duality Principle for Galois Connections** : dual of a galois connection is simply exchange of the adjoints. (A,B) - (B,A).

- **Composition of Galois Connections** : composition if possible if resulting is also a galois connection.

- **Galois Surjections** : is one to one, (A o B) = 1. A is onto.

- **Galois Injections** : is one to one, (A o B) = 1. B is onto.

- **Conjugate Galois Connections in Boolean Algebra** : conjugate mapping results in forming a galois connection with reversed characteristic property of the earlier posets.

- **Reduction of Galois Connection** : done using Pre , Post and Duals.

- **Sum of Galois Connections** : is summing individually the mapps for the two same posets.

- **Power of Galois Connections** : is galois connection formation between two sets of galois connection having a monotine mapping done between formers and laters by same map then resulting composition is also a galois connection.

Chapter 12

Ordered Maps Galois Connections 2

- **Poset Images** : image of complete lattice by join preserving maps is a complete lattice.

  image of complete lattice by closure operators is also a complete lattice.

- **Closure Operator induced by Moore Family** : closure operator on a lattice induces a moore family if glb is preserved in the lattice as well as the moore family.

- **closure operator on monotonic functions** : least upper closure operator is said to exist for monotonous maps, if f(y) is less than or greater than y & y is less than or greater than x. for a map pair (x,y).

- **Complete lattice of Galois Connections** : complete set of upper closure operators on complete lattice is also a complete lattice.

  in terms of galois connections the effort was put to explain the importance of galois connection preservance while the transformation through closure operators take place.

- **bifinitary traces** : set of finite , finite non-empty sequences.

- **prefix closure** : prefixes, pre conditions of bifinitary traces.

- **limits of chain of traces** : unique limit exists for an increasing finite chain of traces.

- **There exists a galois connection between sets of bifinitary traces and their prefix closures.**

- **limits of bifinitary traces** : lim acts as a topological closire operator on the posets.

- **closure of prefix and limits** : composition of prefix closure operator and the lim is a topological closure operator.

- **complete lattice of Safety Properties** : safety property is a complete lattice but not necessarily a complete join morphism.

  It is possible to disprove a safety property by looking at a finite part of program behavior .

Chapter 13

FixPoint Theory 1

- **fixpoints** : operator on set that returns the same value as argument.

  There also exists pre, post fix operators.

  least fixpoint exist for if the values they return there exists a element satisfying characteristic poset condition and greatest fixpoints exist if there exists elements satisfying dual of characteristic conditions .

- **iterates** :

  Forms recursive relations. Covers all the elements of set.

  All the iterates end in same cycle called basin of attraction.

  for infinite set, iterate may endup in cycle, infinite iteration or a fixpoint.

- **numerical fixpoint** : cos(x) = x

- **equiv relation** : self understood example for a particular type of relation that should be an equivalence relation.

- **grammar semantics** : are example of fixpoint definitions.

  The semantic derivation of context free grammar rules results in a fixpoint definition.

  The concatenation of two languages also form a fixpoint definition. understand this as the concatenation of 'C' and 'assembly language'.

- **lattice of closure operators** : as discussed earlier a closure operator operation on complete lattice results in a complete lattice. (similar to a fixpoint))

- **FIXPOINT THEOREMS** :

  - **Knaster-Tarski fixpoint theorem for monotone operators on a complete lattice** : A monotonic map on a lattice has a least fixpoint and dually a greatest fixpoint.
  - In lambda calculus notation, Least fixpoint definition of reflexive transitive closure emphasises on union of disjoint sets each formed from elements and relation on elements of the poset. the strict transitive closure emphasis on relation of union of element by element of the complete set.
  - **Banachs lemma** : there exists two maps from A to B and B to A that creates partitions such that they are disjoint and their union results in A or/and B.

- **The Cantor-Schrder-Bertein theorem** : given injective maps exist from A to B and B to A, there exists a Bijective Map.
- **David Park upper fixpoint induction principle** : says that for a lattice, least fixpoint operator gives values that follow the characteristic condition for a given point P outside the Invariant considered.
- **David Park lower fixpoint induction principle** : says that for a lattice, point P follows characteristic condition relative to greatest fixpoint operator's values. in this case point P lies inside the defined invariant

  this is dual to upper fixpoint induction.
- **Least fixpoint of a monotone operator greater than or equal to a given prefixpoint exists then a least fixpoint for that point also exists.**

  for the conjugate, the postfix-point can be reasoned to have a least fixpoint.
- **Park conjugate (dual) fixpoint theorem incomplete boolean lattices** : greatest fixpoint and least FixPoint of a montone operators on a lattice are basically conjugate of one another.
- **Park unique fixpoint conditionin a complete boolean lattice** :says that GLB of gfp and lfp is bottom and LUB of gfp and lfp is top.
- **Fixpoint of the composition of monotone functions** : for two maps f,g from A to B and B to A, g((lfp f) o g) = lfp (g o f).

  for comparable operators on complete lattice, is operator f satisfies the characterization condition to operator g for a Lattice L, then lfp (f) also satisfies the characterization condition to lfp(g).
- **The BekiLeszczylowski fixpoint theorem** : for partial monotone operators on a lattice L, FixPoint of one is subset of other & least fipoint is same for both.

- **Abstraction soundness** : says that for an operator (f) so that lfp(f) satisfies the characterization condition to a point P in lattice, there exists an operator (g) such that original operator (f) satisfies the characterization condition to that (g) and lfp(g) satisfies the characterization condition to P.
- **Fixpoint clipping** : for a point P in L, lfp(f) satisfies the characterization condition to P implies that f(lfp GUB (f,P)) satisfies the characterization condition to GUB (lfp(f) , P).
- **Fixpoint induction with clipping** : clipping performed with an assumed Invariant.
- **Application to the proof ofabsence of runtime errors** : using fixpoint definitions od erronous states and initial states etc to prove, disprove the existence of least fixpoints.

Chapter 14

FixPoint Theory 2

- **fixpoint iteration** :

  Transfinite iteration on a poset involves the rules of initial starting value, successive ordinal and limit ordinals.

  finite iteration theorem for monotone operator on a lattice says that there exists an upward and increasing chain for the operator bound by a top point.

- **Kleene fixpoint iteration theoremfor continuous function** : for a upper-continuous operator on a cpo, the iterate defined till a max limit then, value of iterate at the max limit is lfp of 'f'.

  such iterative fixpoint definitions can be extedned to strict transitive closure operators to reason for equivalence of left and right hand side of relation.

- **Scott fixpoint induction principle** : says for a point in the prefix-point of an upper continuous operator and is an element of P, the lfp(a) is element of P.

- **Lower fixpoint induction principle** : given upper continuous operator on a cpo, for an element 'a' of the prefix-point of that operator and subset P of Lattice, the P satisfies the characterization condition to lfp(f) at 'a'.

- **Knaster-Tarski on cpos** : given upper continuous operator on a cpo, for an element 'a' of the prefix-point of that operator and subset P of Lattice, lfp(f) at 'a' = GLB of all x such that f(x) satisfies the characterization condition to x and a is element of x.

- **least fixpoint theorem for monotonefunctions on cpos** : given upper continuous operator on a cpo, for an element 'a' of the prefix-point of that operator and subset P of Lattice, the transfinite iterates of f form an increasing chain which is ultimately stationary andwhich limit is lfp(f) at 'a' .

- **Limit of the iterates** : for a monotonc map K, the mapping a prefixpoint from K to limit of iterates is a closure operator.

- **Ward theorem** : image of complete lattice by closure operator is a complete lattice.

- **Constructive version of Tarskis fixpoint theoremfor monotone operators on complete lattices** : set of fixpoints of a monotonic map on a complete lattice is a complete lattice.

- **Least fixpoint theorem on cpos** : for a monotonous operator on cpo with element 'a' in the prefixpoint of P, lfp(f (at a in P)) does exist.

- **Variant transfinite fixpoint iteration** : for a monotonous operator on cpo with element 'a' in the prefixpoint of P, transfinite iteration limits at a stationary rank given by lfp(f (at a in P)).

- **Fixpoints of a function and its powers** : for monotonous operators, the lfp of compositions of the operator is equivalent to that of single one.

  We can thus speed up the fixpoint teration by squaring or increasing the power of lfp on operator.

- **Least common fixpoints of continuous operators on a cpo** : for a family of monotone and extensive operators on cpo, lfp for a point in the prefixpoint set of operators exist.

  For two commuting operators f and g. following earlier analogy, lfp (f) = lfp (g) at the common point.

- **Fixpoints of extensive functions on cpos** : the operator's value on the a given cpo's elements is non-empty.

  this does not hold is the operators are non-continuous.

- **Hoare's fixpoint theorem** : says the fixpoint of composition of two operators is the intersection of fixpoints of indivdual operatos acting on the lattice/poset.

- **Asynchrounous Iterations** : deals with system of simultaneous fixpoint equations. solving this equations using the iterative techniques like jacobi, gausss siedel etc.

- **convergence theorems** : for a given prefix operator for the given pointwise ordering in a complete lattice , the chaotic iterations generates an increasing chain which is ultimately stationary and is fixed at the value given by fp(F) at D. D is a point of perfix (F).

- **async iteration for a system of fixpoint equations** : gives the idea of parallelization. the process must be finally complete and they also may be ccarried by more than one computer at a given time.

- **the μ calculus** : for the given lattice, this is a method of representation of semantics. it consists of using μ-expressions over operators and the lattice elements. there is a monotonous ordering achieved.

  the μ-expression can be extended to composition of various other expression too.

  It has properly defined semantics.

- **Kozen's μ calculus** : Uses three fundamental elements to define semantics namely propositional formulae, variables and some actions.

  For a given transition system, (S,R,I) where S is set of states, R is relations defined and I is the interpretation of propositional formulae.

  Kozen's μ calculus eventualy states that the formula is a set of states for which the given formula is satisfied. the set of states and the formula are one and same.

- **Lattice theoretic fixpoint-based and rule-based formal definitions** :

  Rule-based formal system is based on axiom and inference rules.

  - **equivalence of rule based and fixpoint based formal definitions** : proofs in rule based formal system uses the finite set of possible points satisfying the rules anf universal conditions. the set formed by these points needs to be finite and hence there should be a limit which can be referrd to as fixpoint.

    The proof-theoretic and fixpoint-theoretic definitions on the set S defined by a formal system are equivalent.

    example: rule based constraint definition and closure conditions.
  - **set of regular expressions generatedby the grammar of regular expressions** : uses the closure operator definition to justify the equivalence.

    the constraints and rule based definitions are proved to be equivalent.
  - **Formal definition of the reflexive transitive closure of a binary relation** : includes fixpoint definition, equational definition, constraint based definition and closure condition based formal definition and rule based definition.

    all the them are equivalent, the different method of representation has been explained the th lecture.
  - **Definition, semantics and equivalence of lattice theoretic formal definitions** : explains that there exists a limit often referred to as the 'least solution' in the lecture that can also be understood a a fixpoint. this existence is defined in all the types of formal definitions and thus they can be thought of being equivalent.

- **Derivation (generalizing proofs** : of some element 'x' if a lattice generates a transfinite sequence such that elements of that sequence satisfy the ordering condition and there exists a bottom and limit given by that elment 'x' .

  beautifully explained the example of derivation of regular expression, there are sets formed with increasing number of elements.

  FixPoint based and rule based derivation semantics can also be proved equivalent using reasonings similar to previous.

- **Join-Irreducible Rules** : says the lfp defined by set of rule instances and corresponding operator are equal.

  if given a poset (proof) satisfying the rule based definition ,its subset also satisfies the rule based definition.

  example : forming a set of even numbers using a rule based and operator definition.

- **Inductive definition of the finite traceoperational semantics of a transition system** : uses a finite traces transformer for the given lattice's elements. this transformer is proved to satisfy Union-morphism and intersection-morphism.

  this also states the quivalence of lfp and gfp at a extremeties (lfp at min and gfp at max).

  Co-inductive definitions prove the transformer to be complete intersection-morphic and the set having a lfp.

- **continuity of trace transformer** : the previously defined trace transformer is contnuous, though it can be non deterministic for the fixpoint semantics.

- **Bi-inductive definition of the finite and infinitetrace operational semantics of (a variant of) SIL** : it is just a revision to the formal rule based semantics and explains the various parts of a SIL that can be achieved through rule based definitions.
- **Well-formedness of the Inductive Definition of the Complete Traces Operational Semantics** : the strict definitive component used for defining the inductive definitions is well formed.
- **Beyond Action Induction** : For loops we cannot use syntactic structural induction becauseof the recursive definition of the traces.

  For finite traces we can reason by induction on the length of traces, thus it is not possible to reason for the infinite traces.

Chapter 15

Abstraction 1

- **Informal introduction to abstract interpretation** : using the graphical language, the abstract interpretation is introduced. using objects to represent an abstract idea.
- **Concretization** : mathematically the lfp satisfies the constrants, our abstract objects also satisfy the constraints. thus abstract objects (interpretations) also seem to behave likewise.

  iteration and upper approximation form the set of satisfying points .
- **abstraction examples** the abstraction function maps elements to their abstract definitions.

  the Concretization examples include the galois connections
- **Abstract ordering** : for the elements of set satisfying a given ordering, the abstraction function defined over them also follows the same ordering.
- **Abstract Rotations** : mathematically defined method to prove graphical rotations on the abstract elements.
- **Abstract transformer** : mathematically it defines the lfp called as abstract fixpoint for given abstract definition. graphically it means the abstraction defined on the petals (abstract objects).
- **Property Abstractions** :Properties are a set of objects (or set of states) satisfying some given conditions.

  it can be understood as variables, statements, heaps etc.

  Property abstraction deals with defining those sets mathematically which be represented as lattices.
- **Abstraction, informal introduction** : Abstraction replace someting concrete with a schematic description that account for some, and in general not all properties, either known or inferred i.e. an abstract model or concept.

  it cannot define all the properties as concrete model.
- **Direction of abstraction** : can be abstraction from above or below in reference to the set of elements. Other schema may be using probabilistic reasoning to define the direction of abstraction.

  the from-above/below abstraction can be thought of abstracting the points of the x-y graph from up tp below or below to up.

- **Minimal abstraction** : in the absence of an upper abstractions, finite approximations are used.

  the minimal abstraction is defined for the smallest defined property for a given set/lattice.

  minimal upper-approximations are carried out if abstractions are not definable there.

  A classical example of absence of minimal abstract upper-approximations is that of a disk with no mini-mal convex polyhedral approximation.

  there may be many minimal abstraction possible for the same property however the most cost-efficient abstraction should be used.

  example : rule of signs

- **Best abstraction** : is an abstraction of property 'A' for the set 'P', is such the glb of P lies inside A.

- **The abstract domain is a Moore Family.**

- **closure operator based abstraction** : defines mapping from the lattice set P to abstract property set A.

  for Best abstractions, the Moore family definitions and the closure operator based definitions are equivalent.

- **Generalizing to complete lattices** : reasonings/mappings on abstraction property defined on a lattice can be generalized into complete lattices.

  this helps in forming a compositional approach to define all the complete abstraction property.

- **Specification of an abstract domain by a Galois surjection** : Correspondance between the concrete and abstract properties can be explained using compositional rules of a surjective operator and its inverse. this compostion con further be explained using galois connections using an abstraction function and concretization operator.

- **Abstract domains are complete lattices in the presence of best abstractions** : as stated previously by the use of closure operators and moore family analogy.

- **Standard examples of abstractions formalized by Galois connections** :

  - **Subset restriction abstraction**
  - **Elementwise/homomorphic abstraction** :
  - **Subset inclusion abstraction** :
  - **Functional Abstraction** :
  - **Relational Abstraction** :
  - **Relational lattice Abstraction** :
  - **minimal abstraction** :
  - **interval abstraction** :
  - **abstraction of function at a point** :
  - **Abstraction of a function at a set of point** :

– **Abstraction of a set of functions by a function** :
– **Abstraction of lattice functionals** :
– **Cartesian abstraction of a setof pairs** :
– **Pointwise abstraction composition** :
– **Componentwise abstraction composition** :
– **Higher-order abstraction composition** :

Chapter 16

Abstraction 2

- **Exact fixpoint Abstraction** : property transformer abstraction deals with finding out the abstraction operator value without having to find the lfp value for the whole lattice.

  this value (or fixpoint, since we know they are equivalent) though can not be easily achieved, the method of higher order abstraction is used in accomplishing our goal.

  thus there are possible exact as well as approximate fixpoint abstraction.

- **Kleene exact fixpoint abstraction with Galois connections** : this is also called as fixpoint fusion, fixpoint lifting , fixpoint morphisms etc.

  Given two lattices and an operaor defined mapping between them, the overall abstraction through galois connections can be broken to smaller galois connection.

  this involves having a strictly increasing sets having a limit defined by the max of lattice's elements.

  it also inolves that the operator must be monotonous and the lfp exists for the lattice.

- **Kleene exact fixpoint abstraction with continuous abstraction** : using more abstraction operators, defined and composed over the set of operators eventually results in the lfp .

- **Tarski exact fixpoint abstraction** : it also defines the composition of operator to result in the same fixpoint value.

- **transition systems** : has a set of states and a transition relation. can be reflexive transitive closure.

- **post images, post image galois connections** : post image galois connections involves establishing galois connection between the post images anf the abstraction domain.

- **reachable states in fixpoint iterations** : the states are reachable in the fixpoint iterations if the postimage of transtion relation is satisfied by the invariant.

Chapter 17

Reachability and Post Condition Semantics

- a small revision to **Forward collecting semantics of arithmetic expressions** , **Forward collecting semantics of boolean expressions** , **Big-step operational semantics ofcommands** is explained.
- **Postcondition collecting semantics of commands** : The postcondition semantics of a command (of a given programP) specifies the strongest postcondition satisfied by environments resulting from the execution of thecommandCstarting in any of the environments satisfying thepreconditionR, if and when this execution terminates.

  it forms a complete join morphism over the set of states and hence is monotonic and transitive.

  It can be understood as an abstraction of big-step operational semantics. (elaborately proved in the lecture slides.)
- **Transition system of a program** : is defined as a set of states and a set of relations (commands) defined between those states and environment.
- **Forward reachability collecting semantics of commands** : gives set of reachable states during any execution of a command starting at its starting point in any of the enviornments satisfying the precondition.
- **property and semantics of forward reachabilty of commands** : it is complete join isomorphism, it means it is continuous, monotone and strict.

  post condition semantics is a abstraction of forward reachability collecting semantics. we can thus design either the termination conditions or the forward reachability of commands.

  the complexity of either case should be looked in first before construction.
- **Implementation of the forward reachability collecting semantics**.
- **trace of the fixpoint computation with two nested loops**.
- **Totally ordered types in OCaml**.
- **Implementation: sets in OCaml**.

Chapter 18

Approximation

- **motivation & intuition** : the improved spedd of convergence of iterative fixpoint transition with widening.
- **kleene, galois based continuous transformer, fixpoint Approximation** : as in the case of abstraction, this is too an abstraction but using approximation. uses composition of multiple operators to get the result.
- **kleene, galois based monotone transformer, fixpoint Approximation** : as in the case of abstraction, this is too an abstraction but using approximation.
- **soudness of abstraction** : to prove the soundness, equivalence of lfp (operator F) and lattice P we overestimate the lfp(F) and under estimate P's element values.

  completeness of abstraction is resulted by applying abstraction function to the lfp of operators's value of pre-fixpoint element and proving its equivalent to operators's value of abstraction function applied over the pre-fixpoint element.
- **kleene, galois abstract functions based, fixpoint Approximation** : for a monotonic, upper-continuous operator function, this means applying approximation function to lfp of (abstract function value at a prefix-point).
- **kleene, galois concretization based, fixpoint Approximation** : this is kind of dual to previous method (simple analogy intended), the concretization function is applied to the approximation of lfp values iteratively.

  this is monotinic and upper continuous. it is a well defined increasing chain and has limiting ordinal.
- **sufficient conditions for iterative fixpoint convergence** : a programming language's program properties can be defined in fixpoint form together with structural induction on syntactic semantics.

  the different encodings of those properties, terms and functions can be explained with the help of abstract interpreters and syntactic analyzer.

  A static analyzer is guided by abstraction and induction on syntactic structure of the program.

  mathematically this static analyzer is expresses as a lattice togrther with an operator.

  so for termination conditions, we need to have this lattice to be finite and lattice to follow ascending chain conditions.

- **iteration acceleration by extrapolation** : the lattice of intervals for example is a lattice not satisfying the ACC thus finding a solution for this case is quite difficult.

  the example case has been explained of a simple while loop.

  the computation is very slow and involves a lot of steps.

  therefore we need to perform convergence accelerations using widening and narrowing.

- **Widening** : extrapolates unstable bounds to infinity. not monotonous transform.

- **upward iteration with widening** : is iteration sequence with the next subsequent term is a result of widening operation applied between previous term and operator.

  It is sound because of increasing and monotonic nature of operations performed with an existing limit.

- **Narrowing** : limits/imporves the infinite bounds. reduces the acheiveable values set.

- **decreasing iteration sequence with narrowing** : is iteration sequence with the next subsequent term is a result of narrowing operation applied between previous term and operator.

  its correctness is explained as similar to widening.

  narrowing for an operator starting at a postfixpoint is decreasing in nature. the limit exists and is an overapproximation of leastfix point of the operator at the postfix-point.

- **static analysis with widening/narrowing** : iteration convergence accelaration.

  automatic interval analysis with widening and narrowing.

  fixpoint approximation using widening and narrowing. the approximation function may be very precise but the widening may not be very precise. this approximation depeends on states, it is done once for all propor to applying widening.

  the static program analyzer iterates over the program states and performs narrowing/widening at the states.

  galois based static program analyzer uses galois continuous ab the basic method to perform abstraction and give reasoning.

- **properties of widening and narrowing** : widening - starts below lfp and ends above postfixpoint.

  narrowing - starts above lfp and ends above lfp.

  dual widening - starts above greatest-fp and ends below prefixpoint.

  dual narrowing - starts below gfp and ends below .

  narrowing not always is able to recover the information lost by widening.

- **parametrized meta-example of interval invarianty by fixpoint approximation with convergence accelaration by widening/narrowing** : a given interpretor will perform same for similar type of program. the constraint may be different valued leaving us with more number of programs of similar nature, but the behaviour of interpretor will be same.

- **usefulness of widening** : using interval analysis it can be showed that there can be infinitely many abstract domains are needed.

example : Kildall's constant propogation problem using the lattice.

the lattice is finite used to obtain program behavioural properties.

there are infinitely many abstract prameters required to reasn about he working of the program which can't be referrred directly by looking at the text.

- **Relative precision of widening** : its proved that limit of states is more precise for iteration with widening than iteration without widening
- **weakening of hypotheses on widening** : over approximation and upper bound hypotheses can be made better by using a concretization function .
- **widening are not monotone, its immediate effects** : widening can not be monotone since for an equal element, the widening will result the same an hence loss of extrapolation.

  a small change in static analyzer may lead to global deterioration of the complete program.
- **design of widening/narrowing** :
  - iteration threshold. (do not perform widening/narrowing till some iterations).
  - Unrolling. semantically unroll the first iterates of loop.
  - using cutpoints
  - history based extrapolations
- **threshold widening** : involves widening with thresholds (some limiting values of elements of the lattice).
- **widening for pairs and tuples** : is a set product of widening/ narrowing operators.
- **first order functional widening** :
- **second order functional widening** :

Chapter 19

Non-relational monotonic finitary static analysis 1

- **Design of a non-relational abstract interpreter for SIL** : Non-relational abstraction deals with having a point-wise independent abstraction function for the latttice elements or the set of the states.

  this may be understod as catersian abstraction or component wise abstraction trying ot represnt the different point in the cartesian plane.

  the resulting states are not related.

- **Forward non-relational finitary reachability static analysis of SIL** : non-relational is easy to program but is less approximate. relational is cost effective but difficult to program. Global non-relational and local relational finitary analysis is caaried out.

  non-relational finitary analysis of set of environments result in having a less precise set of states. the galois connection is used to improve the approximation of co-domain. To form the approximate lattice a composition of various galois connection is used. this abstract lattice is then used eliminate abstract 'useless' enviromments.

- **Forward non-relational finitary reachability static analysis of arithmetic expressions** : picking up from previous chapter on static analysis of arithmetic expressions, this part deals with forming structural definitions of evaluation of arithmetic expression semantics.

- **Forward non-relational finitary reachability static analysis of boolean expressions** : extends the structural definitions for the static analysis of boolean expressions semantics.

- **Forward non-relational finitary reachability static analysis ofcommands** : extends the structural definitions for the static analysis of commands, the precondition and the post condition analysis of the commands.

- **initialization and simple sign analysis** : is simply defining rules for the mathematical sign allotment for the number.

- **Ocaml implementation of abstract interpreter**.

Chapter 20

Non-relational monotonic finitary static analysis 2

Abexp = abstract expression.
Bexp = boolean expression.
comm = command.

- **Generic backward/bottom-up static analysis of arithmetic expressions** : is not useful in tests where the end result is known.

  Is more helpful in the case of initialization and sign allotment.

  It follows that any information on the possible result of an arithmetic expression should bring informationon the values of the variables involved in the arith-metic expressions for its result to satisfy the knwon information.

- **Backward/bottom-up collecting semantics of arithmetic expressions is a lower closure operator** : collecting semantics of arithmetic helps in finding the environment variable for which the arithmentic expression will evaluate to true.

  A lower Closure Operator definition of collecting semantics of arithmetic expressions follows monotonicity, reductivity and idempotency.

- **Structural definition of the backward/bottom-up collecting semantics of arithmetic expressions** : revised definition and relational mapping that represent the changes in the environment variables it depends on.

- **Generic backward/bottom-up non-relational abstract semantics of arithmetic expressions** : it deals with getting the over approximations of backward collecting semantics.

- **Structural definition of the genericbackward/bottom-up non-relational abstract semantics of arithmetic expressions.**

- **Calculational design of the generic backward/bottom-up non-relational abstract semantics of arithmetic expressions** : uses compositional rules and compositions of various abstraction and concretization functions to realize the semantics.

- **Ocaml Implementation of the primitive backward/bottom-up non-relational abstract arithmetic operations for initialization and simple sign analysis**.

- **Improving the non-relational analysis of boolean expressions using the backward analysis of its arithmetic subexpressions** :

  The abstract interpretation of boolean expressions can be revised using the backward abstract interpretation of arithmetic expressions and can be parametried using comparison operators.

- **Calculational design of the revisited non-relational abstract interpretation of boolean expressions** : uses rule based derivation to find the abstraction of boolean expression.

- **Abstract arithmetic comparison operations forthe initialization and simple sign analysis** : deals with Generic abstract boolean equality.

  The calculational design ofthe abstract equality operation does not depend upon the specific choice of Lattice 'L'.

  the table with all the different possible combinations of data type and sign initialization is specified.

- **Local decreasing iterations** : the lower closure operaotor is better abstract interpretation for the monotonous reductive type mapping on a given lattice.

- **The forward/top-down nonrelational abstractsemantics of arithmetic expressions is monotone** :

- **The forward/top-down nonrelational abstractsemantics of boolean expressions is monotoneand reductive** :

- **reductive expression for boolean formulae** :

Chapter 21

Forward Non-relational Infinitary Static Analysis

- **Proving the correctness of static analyzers** : the principle for proving the soundness of static analyzers is to proceed by induction on the syntax of programs, which yields a proof for the whole programming language.

  The proof involves use of quite general form of collecting semantics, abstraction and abstract definitions concretized by formal definition (galois connections).

- **abstract formalization of finitary structural analysis by abstract interpretation** : The abstract syntax defines a collection of syntactic categories and a well-founded relation.

  It consists of defining concrete domains, concrete semantics.

  Type-ing the structural concrete semantics uses components like language, types, abstract domains typing component (base types), typing semantic expression, typing soundness of semantic equational semantics.

- **forward collecting semantics of arithmetic expressions** : defines rules from simple arithmetic expressions to mapping in semantic expressions.

- **Example of structural concrete semantics** : are commands, sequences and programs, that deal with reachable states, transition system and operational semantics.

- **Well-definedness of structural semantics** : If fixpoints exist then the structural seman-tic definition is well-defined.

- **Fixpoint existence** : transfinite sequence has a limit ordinal defined.

- **Monotonic structural semantics** : the structural semantics is said to be Monotonic if all the structural semantic mappings defined on a given poset are monotonic.

  they are well defined if lfp exist for such cpos.

- **Structural abstract semantics** : it is similar to structural collecting semantics. uses idea of abstract domain, abstract transformers and abstract semantics.

- **Local abstraction** : is idea is used to create correspondence between concrete and abstract semantics.

is said to be defined whenever similar definitions and correspondingly similar structural expressions exist for abstract domains and concrete domains. The abstract domain are subset of concrete domains.

this correspondence between concrete and abstract is well defined if there exists both concrete and abstract lfps.

- **abstract formalization of infinitary structural analysis by abstract interpretation** :

  Uses widening structral semantics that are similar to collecting semantics.

- **soundness theorem on the correspondencebetween concrete semantics and its local abstraction by a structural abstract semantics with widening** : says that the local abstraction and widening if hold for the structurally same expressions, they follow ordering w.r.t one other.

- **Hypotheses on narrowings** : there are two defined hypotheses namely - the narrowing is subset of larger set from which narrowing is made. and the sequence generated is decreasing.

  A structural definition with widenings and narrowings respectively satisfying narrowness and widening hypotheses is well-defined.

- **soundness theorem on the correspondencebetween a concrete semantics and its local abstraction by a structural abstract semanticswith widening/narrowing** : if two structurally identical expressions / definitions exist and narrowing and widening hypotheses hold then one can be represented as an concretization of other.

- **Abstract formalization of structural verification by abstract interpretation** : Deals with structural safety proof and abstract safety specification. It involves design abstract semantics that are computable with a given choice of abstract domains, transfer functions and widening/narrowings.

- **Abstract structural safety verification** : Checks if a concretization exists for a given command.

  An abstract structural safety verification is sound.

- **Example of abstract structural safety specification for arithmetic expressions:proper initialization** execution of arithmetic expression in a given environment is without any initialization error if forward collecting semantics do not cause errors.

  We can only strengthen the analysis by refining theabstraction or weaken the specification.

- **choosing an abstract semantics more refined than an abstract specifications** : The objective is to check the conformance of a semantics to a specification.

  The fact that the abstract semantics should be morerefined than the abstract specification is similar theproof of theorem requiring stringer arguments in the proof.

  By choosing abstract specification we are simply satisfying the concretization conditions for supposed specs but it is difficult to represent it in as a abstract domain.

- **Error Analysis, error abstraction** : Error Abstraction : takes initialization and arithmetic errors into account. We can draw a hasse diagram for error complete lattice.

- **Parity analysis** : analyzing parity conditions.
- **Design of the abstract properties** :
  - **interval analysis** : involves error and interval abstraction. lattice modelling of intervals. proving existence of lubs to show the existence of complete lattice of intervals.

    Interval Abstraction represented with galois connection and defining composition rules to accurate the working intervals. Defining reduced products on posets. Includes defining concretizations for products.
  - **error-interval abstraction** : We define the interval and error abstraction as the reduced product of the interval and error abstractions.

    this analysis uses parity analysis too to achieve more reasonable lattice.
  - **interval abstraction as the reduced product of the minimum and maximum abstractions** : says that the lattice having a maxima, minima has a dual by galois connection thus a new pair can be repreented as a reduced product.

These are Ocaml implemenations discussed in lecture.

- **Design of Abstract Transformers** : implementation in ocaml. uses symbolic execution property to check errors.
- **Design of Abstract Transformers -backward integer constant** : implements a bool check for if a given integer lies in between a given pair 'V'.
- **Design of the abstract transformers: forward integer addition** : implements addition semantics using forward abstract semantics.
- **Design of the abstract transformers: backwardinteger addition** : implements addition semantics using backward/ bottom-up abstract semantics.
- **Design of the abstract transformers: forward integer comparison** :
- **Implementation of the abstract transformers** :
- **Design of the abstractconvergence accelerators (Widening)** :
- **Generic abstract interpreter** : The global structure of the analyzer is the same whichever is the abstract domain chosen to approximate sets of values.

  Up to the use of widening/narrowing when no con-vergence acceleration is needed (e.g. finite domains,domains satisfying the ACC with rapid convergence.

  For non-relational analyzes, the structure of the ab-stract domain approximating sets of environements only depends on the abstract doamin for sets of values.
- **Forward static analyzer** :

Chapter 22

Forward Relational Infinitary Static Analysis

- **Motivation** : We want to combine abstract analyzes that are defined independently of one another. Each analysis is defined on the collecting semantics by a closure operator. the combination produces a reduced product that corresponds to lub of closure operators.
- **lub of closure operators** : a monotone and extensive function defined on a cpo then lfp is a least upper closure on the cpo greater or equal to that function.
- **Iterative reduced product** : two complete lattices can be shown to be related in galois connection with the functions of product and sum to visualize the idea of universalization.

  Cartesian product of abstract domains (complete lattices). The cartesian product of abstractions discovers in one shot the information found separately by the component analyzes.
- **Reduction Operator and Reduced product** : reduction operator is a least closure operator that defines a galois connection between complete lattice.

  the reduced product is a poset having a componenet wise ordering defined using composition rules for complete lattices.

  The advantage of the reduced product over the cartesian product of analyses is that each analysis in the abstract composition benefits from the information brought by the other analyses.
- **The reduced product of three abstract domains or more** : We can do iterated reduction on groups. the advantage being that pairwise reductions are easy to design than the global reductions for the complete domain.
- **Generic forward abstract interpreter with reduced product** : Ocaml implementation of the ternary iterated reduction, parity and initialization and simple sign reduction, parity and intervals reduction.
- **Generic Abstract interpreter - usage and construction specifications** : Ocaml implementation.
- **linearization** : Reduction can cancel convergence enforcement by widening/narrowing. The linearization or linear abstraction of variables of SIL program is vectorization of states. there

are two ways to analyze the state namely dynamic and static. the dynamic analysis may/may-not predict the static state correctly. linearization helps in improving dynamic analysis accuracy.

- **correctness of abstract abstraction** : The syntactic transformation of an arithmetic expression A of a program P into its linear form yields an upper approximation of its forward collecting semantics.
- **Linearization of boolean expressions** : The extension of linearization to boolean expressions is trivial since it essentially concerns the arithmetic expressions within the boolean expression.
- **program linearization** : the concept of linearization can be extended to commands and programs. supporting implementation added in lecture.
- **implementation of static linear abstraction** : Ocaml implementation.
- **linear abstraction of programs** : Basically means the vector coefficients for different used variables.
- **generic linear relational abstract interpreter** : the expenses are analyzed as whole under relational analysis.
- **Generic linear relational abstract domains** : for linear relational analysis, these are array, variables, set of environments, initialization functions etc.
- **Generic linear relational analysis of boolean expressions**
- **Generic linear relational analysis of commands** : similar to non-relational analysis, but the checking is performed on linear/ vectorized attributes.
- **Fixpoint computation with widening/narrowing** : introduces traces and trace fixpoint computation.
- **Polyhedral relational static analysis** : For the vector space defined over a field, the abstract affine inequalities form a convex closed polyhedra. this may or may not be closed.
- **Affine Hull** : called as convex hull is combination of set points and their affine combinations. not defined for infinite number of points. example - a ray formed from collection of points. We use concretization function only.
- **Minimal representations** : polyhedra representations includes representation by constraints, generators. it is colled minimal if no constraints can be eleminated without changing the polyhedra.
- **Chenikova Algorithm** : helps in achieving a minimal representations of polyhedra by checking for constraints.
- **Minimization of the system of generators by Le Verge algorithm** : helps in achieving a minimal representations of polyhedra by checking for generators.

- **lattice structure of polyhedra** : test for emptiness - No verterx.

  inclusion test - subset present.

  equality - overlapping polyhedra.

  union.

- **Abstract Polyhdral transfer functions** : involves linear transformation. polyhedral widening is also a transformation that changes some of the constraints.
- **polyhedral libraries** : set of libraries that help in polyhedral analysis.
- **implementation of polyhedral analysis** : Ocaml implementation.
- **polyhedral abstract environment** : Ocaml implementation.

Chapter 23

Iterated Forward backward Relational infinitary Static Analysis

- **revisits** : retouched topics operational semantics, program transition relation, Big-Step Operational Semantics, Structural Big-step operational semantics, forward collecting semantics of boolean and arithmetic expressions.
- **Backward/precondition collecting semantics** : Backward Precondition is the weakest precondition for execution of P from entry point of command C to have the possibility to reach a final state in R on exit of the command.

  Structural semantic specifications of it include rules for skipping, conditional statement, looping.

  in slides a proof has been explained for the correctness of backward collecting semantics.
- **Backward/precondition relational abstract semantics** : an abstract relation semantics is abstraction of backward collecting semantics in a given environment conditions.

  an extra condition for the environment is added and applied with every Structural specifications of backward semantics.

  its correctness is also proved in the lecture.
- **Generic backward/precondition linear relational analyzer** : Ocaml implementation.

  this adds a mapping function to linearized commands at the sarting point.
- **Polyhedral backward/precondition analysis** : Ocaml implementation.

  inntroduces the idea of handling backward assignments.
- **The precondition/postcondition collecting semantics** : revisits the idea of transition system.

  the forward/backward semantics is a pair.

  for abstraction we can over-approximate this pair.
- **The precondition/postcondition abstract semantics and its implementation** : Given a forward/backward semantics pair and we want to over approximate it, we can use greatest fixpoint characterization.

Pre/postcondition static analysis - The chaotic iterations convergence need to be enforced via narrrowing.

- **The forward/backward reachability collecting semantics** : forward reachability is the set of descendents of initial states by specified transitions.

  backward reachability is the set of ascendents of final states by specified transitions.

- **The forward/backward reachability abstract semantics** :

- **Optimality** : the forward-backward combination is the optimal approach to get the best result.

- **The ASTRE static analyzer** : properties and some functionality discussed.