IA169 System Verification and Assurance

CEGAR and Abstract Interpretation

Jiří Barnat

Program Analysis

Goals of Program Analysis

- Deduce program properties from the program source code and ...
- ... employ them for program optimisation.
- ... employ them for **program verification**.

Undecidability

- Validity of all interesting program properties written in general programming language is algorithmically undecidable.
- Henry Gordon Rice (1953) – Rice's Theorems.
- Alan Turing (1936) – Halting Problem.

Undecidable – So what now?

Abstraction

- To hide details in order to simplify the analysis.
- Aims at correct, even if incomplete solution.

Using Abstraction

- To build (typically finite state) model of system under verification. (Followed, e.g., by model checking approach to verification.)
- System execution within the context of abstraction – **Abstract Interpretation**.

Recall Other Approaches

- Fixed input set of values (Testing).
- Limited exploration of the full state space (Bounded MC).
- Practical undecidability (Symbolic Execution).

Data and Predicate Abstraction

Data Abstraction

Motivation

- State space explosion due to large data domains.
- Reduction employing principle of domain testing, i.e. replacing original data domain with a data domain with less number of members.

Terminology

- Abstraction: mapping concrete states to abstract ones.
- Concretisation: mapping abstract states to set of concrete states.

Example of Data Abstraction

- `Int -> { Even, Odd }`
- Concrete state: $\langle$ `PC:12, A:15, B:0` $\rangle$
- Abstract state: $\langle$ `PC:12, A:Odd, B:Even` $\rangle$

Abstract Transition System

Transitions in Concrete and Abstract Semantics

- Statement in program code, line 12: `A := A+A`
- In concrete semantics:
  $\langle$`PC:12, A:15, B:0`$\rangle \longrightarrow \langle$`PC:13, A:30, B:0`$\rangle$
- in abstract semantics:
  $\langle$`PC:12, A:Odd, B:Even`$\rangle \longrightarrow \langle$`PC:13, A:Even, B:Even`$\rangle$

Non-Determinism in Abstract Transition System

- Abstract state: $\langle$`PC:13, A:Even, B:Even`$\rangle$
- Statement in program code, line 13: `A := A div 2`
- $\langle$`PC:13, A:Even, B:Even`$\rangle \longrightarrow$

  $\langle$`PC:14, A:Even, B:Even`$\rangle$

  $\langle$`PC:14, A:Odd, B:Even`$\rangle$

Relation of Abstract and Concrete Transition Systems

Over-Approximation

- Every run of concrete system is present in concretisation of some abstract run (run of abstracted transition system).
- There may exist runs that are present in concretisation of some abstract run, but are not allowed in concrete transition system.

Under-Approximation

- Every run present in concretisation of any abstract run is an existing run of concrete transition system.
- There may exist runs of concrete transition systems that are not present in concretisation of any abstract run.

Verification of Approximated Transition Systems

Notation

- ATS – Abstract Transition System
- CTS – Concrete Transition System

Verification of Over-Approximated Systems

- Absence of error in ATS proves absence of error in CTS.
- Error in ATS may, but need not indicate error in CTS.
- Error in ATS that is not an error in CTS, is referred to as false positive (spurious error, false alarm).

Verification of Under-Approximated Systems

- Error in ATS proves presence of error in CTS.
- Absence of error in ATS does not prove absence of error in CTS.
- Error in CTS that is not present in ATS is referred to as false negative.

Example – Concrete Semantics

Task

- Is error state reachable in the following program?
- % denotes modulo operation, `A` in an integral variable

Source-code	Value of `A` in concrete semantics after execution of program statement	
1 `read(A);`	`[int]`	
2 `A = A % 2;`	`[0]`	`[1]`
3 `A = A + 1;`	`[1]`	`[2]`
4 `if (A==0)`	`<false>`	`<false>`
5 **`error;`**		
6 `else`		
7 `return;`	`<ret>`	`<ret>`

Example – Data Abstraction

Task

- Is error state reachable in the following program?
- `A` is abstracted into parity domain `{even,odd}`.

Source-code		Value of `A` in abstract semantics after execution of program statement	
1	`read(A);`	`[even]`	`[odd]`
2	`A = A % 2;`	`[even]`	`[odd]`
3	`A = A + 1;`	`[odd]`	`[even]`
4	`if (A==0)`	`<false>`	`<true/false>`
5	**`error;`**		**`<error>`**
6	`else`		
7	`return;`	`<ret>`	`<ret>`

Predicate Abstraction

Predicate Abstraction

- Predicates – Boolean expressions about variable values.
- Example of definition of abstract transition system:
  $\langle$Program Counter, Validity of selected predicates$\rangle$

Amount of Abstraction

- Amount of predicates influences the precision of abstraction.
- Less predicates $\rightsquigarrow$ big ambiguity, smaller state space.
- More predicates $\rightsquigarrow$ increased precision, bigger state space.

Task

- For the given program code and set of predicates, draw the abstract transition system formed using predicate abstraction.
- Check if there is path in your ATS that is not a spurious run and leads to an error state.

```
1   read(A);
2   A = A % 2;
3   A = A + 1;
4   if (A==0)
5      error;
6   else
7      return;
```

a) $P1 \equiv A = 0$

b) $P1 \equiv A = 0$,
$P2 \equiv A \geq 0$

Remarks on Predicate Abstraction

Analysis of Abstract Runs Leading to Error

- Decision about validity of the run (Is it a false alarm?)
- Deduction of new predicates to make abstraction more precise.

Size of Abstract Transition System

- The size of the state space grows exponentially with the number of predicates.

Possible Solution

- Predicates are bound to particular program locations.

CEGAR Approach

Counter-Example Guided Abstraction Refinement

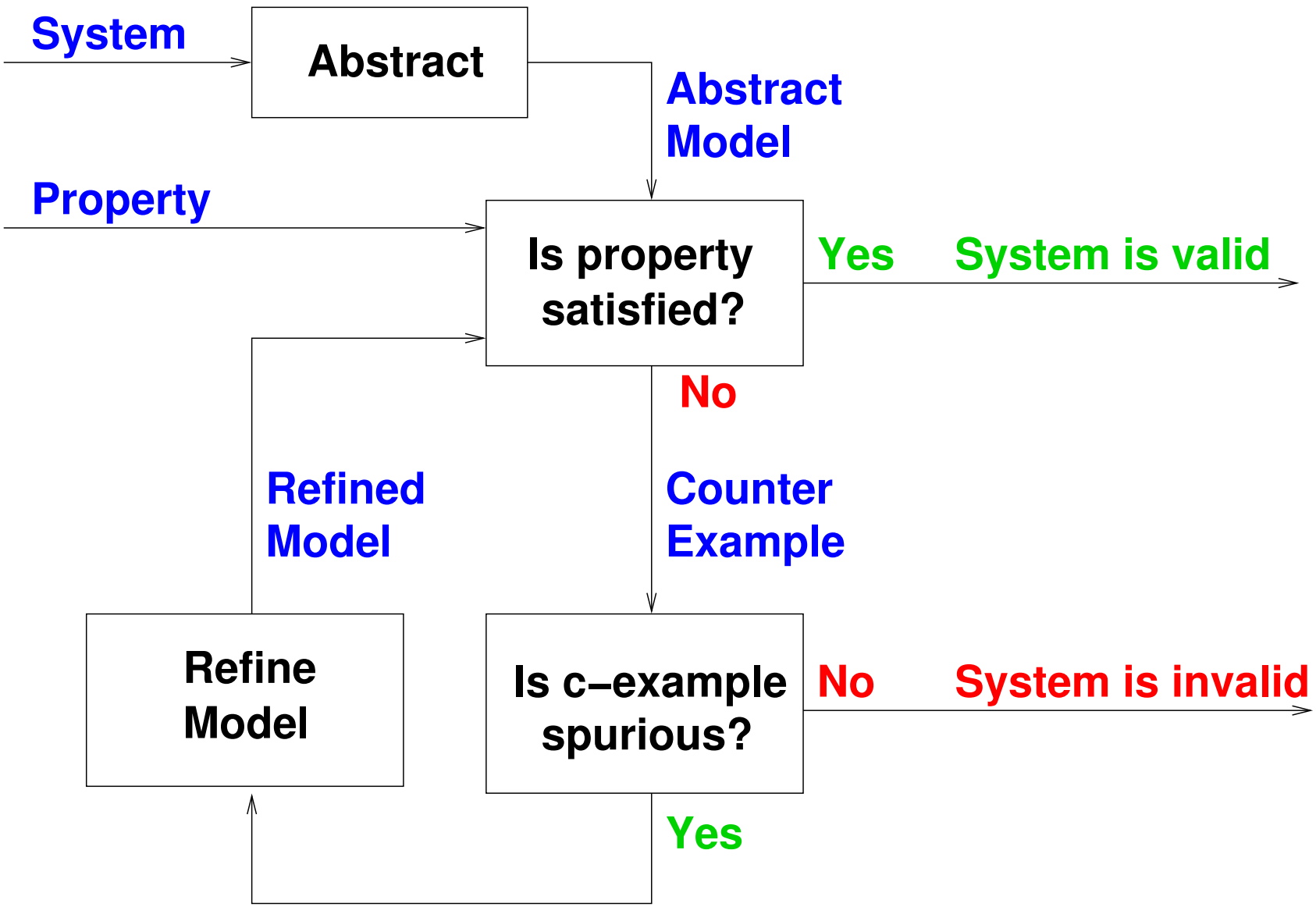
Principle of CEGAR Approach

- Given an initial set of predicates, system is abstracted with predicate abstraction.
- Abstract transition system (over-approximation) is verified with a model checking procedure.
- In the case a reported counterexample is find spurious, it is analysed in order to deduce new predicates and refine predicate abstraction.
- Procedure repeats until real counterexample is find or system is successfully verified.

Notes

- Deducing new predicates is very difficult.
- Often done within model checking procedure (on-the-fly).
- Berkeley Lazy Abstraction Software Verification Tool (BLAST).

Schema of CEGAR Approach



Basics of Abstract Interpretation and Static Analysis of Programs

Program Representation – Flow Graph

- "Special version" of Control-Flow Graph.
- Every edge is either guarded with a single guard or defines a single assignment.

Goal

- Compute properties of individual vertices of the flow-graph.

Goal Examples

- Deduce range of values a particular variable may take in a given program location.
- Compute a set of live variables in a given program location.
- ...

A General View on Abstract Interpretation

Property Decomposition

- The property to be verified by general program analysis procedure can be decomposed into local data values assigned to individual vertices of the (control-)flow graph.
- The result of verification is compound or deduced from the values local to the flow-graph vertices.

Value Improvement

- It is defined how an edge of the (control-)flow graph improves (locally updates) the decomposed value of the property to be verified.
- Application of local update functions gradually (monotonously) improve (approximate) the overall solution.

General Algorithm for Computing Abstract Interpretation

Initialisation

- Initially, a suitable value is assigned to every vertex.
- Every edge of the graph is marked as unprocessed.

Computation

- Pick an unprocessed edge of the graph and perform the local update relevant to the edge and associated vertices. If the update improved the solution, mark the edge as unprocessed again.
- Repeat until there are unprocessed edges, i.e. until overall fix-point is reached.

Example – Computing Live Variables

Setup

- Initial value associated with graph vertices is $\emptyset$.
- Every edge from vertex u to vertex v updates value associated with the vertex u as follows:

$$V(u) = V(u) \cup \Big(V(v) \setminus \mathit{assigned}(u, v) \cup \mathit{used}(u, v) \Big),$$

where $V(x)$ denotes value associated with vertex x, $\mathit{assigned}(u, v)$ and $\mathit{used}(u, v)$ denote variables redefined and used along the edge (u, v), respectively.

Observation

- In every moment of computation there is some (approximating) solution to the verified property.
- Reaching a fix-point indicate, no more information can be deduced by program analysis in the current setup.

Abstract Interpretation

Observation

- The procedure presented on previous slides is quite general. Choosing proper setup may result in verification (computation) of many interesting program properties.
- Often this is combined with some data abstraction for variables.
- May be performed on partially unwinded graphs.
- Generally referred to as to **abstract interpretation**.

Parameters

- What abstract domain is used.
- Direction of update function (forward, backward, both).
- What does update function do.
- How are the values merged over multiple incoming edges.
- Order of processing of unprocessed edges.
- Termination detection.

Relevant Questions of Abstract Interpretation

Is there a fix-point?

- Often the composition of domains of values associated to vertices forms a complete lattice.
- Knaster-Tarski theorem says that every monotonous function over such a domain has a fix point.

Does computation terminates?

- If there is no infinitely ascending sequence of possible solutions in the composition of local domains, then yes.
- Otherwise, need not terminate.

Abstract Interpretation and Widening

Widening

- Auxiliary transformation of intermediate results such that it preserves correctness, and at the same times prevents existence of long (possibly infinite) ascending sequence of values in the corresponding domain.

Example of widening

- Let be given boundaries of precision in which we want to know the range of possible values of some variable.
- Beyond the precision boundaries values will be extended to infinity, i.e. $+\infty$ or $-\infty$.
- Sequence

  $$[0,1] \subset [0,2] \subset [0,3] \subset [0,4] \subset [0,5] \subset [0,6] \ldots$$

  will for precision bound [0,3] turn into:

  $$[0,1] \subset [0,2] \subset [0,3] \subset [0,+\infty]$$

Abstract Interpretation and Narrowing

Narrowing

- Using widening may lead to very imprecise results.
- Widening can be used to accelerate analysis of cycles.
- After widening-based analysis of cycle, the values are made more precised with narrowing (similar but dual technique).

Example

- Interval $[0,+\infty]$ will after narrowing shrink to $[0,n]$.

Commentary

- Precise usage of widening and narrowing is beyond the scope of this lecture.

Other Notes Related to Abstract Interpretation

Other Corners of Program Analysis

- Inter-procedural analysis.
- Analysis of parallel programs.
- Generation of invariants.
- Pointer analysis and analysis of dynamic memory structures.
- ...

CPA checker

- The Configurable Software-Verification Platform
- `http://cpachecker.sosy-lab.org/`
- Very successful competitor in Software Verification Competition.

Homework

Wanna Chalenge?

- Get acquainted with CPAchecker (`https://github.com/dbeyer/cpachecker/blob/trunk/README.txt`)
- Comment counterexample report (`http://cpachecker.sosy-lab.org/counterexample-report/ErrorPath.0.html`)

Homework

- Install and try BLAST verification tool.