Solving systems of fixpoint equations

an algorithmic perspective

Master student: Denis Mazzucato
Supervisor: Prof. Paolo Baldan
Co-supervisor: Postdoc Tommaso Padoan
September 23, 2020

Università degli Studi di Padova

Cost of software failure

- 2016, 1.1 trillion USD in financial losses
- 2017, 1.7 trillion USD in financial losses

Examples:

- 1996: Ariane 5, $\sim$ 370 million USD caused by an arithmetic overflow
- 2005: Toyota electronic throttle control system failure, at least 89 death

In recent history:

- December 2018: $\sim$ 30 million O2 users in UK lost access to mobile data services.
- February 2020: $\sim$ 100 flights disrupted in London's Heathrow.

Motivating program verification

How to avoid such malfunctions?

- Choose a **safe programming language**.
- Carefully **design** and develop software (with appropriate time and funds).
- Adapt **software verification** techniques.

Software verification

- **Dynamic analysis**, usually associated to **testing**.
- **Formal verification**, rigorous methods to formally ensure that the software respects some requirements.

Dynamic analysis

Pros:

- A testing system is easier and quicker to be built.
- Operates on the **running code**.

Cons:

- Operates on the running code.
- Cannot **exhaustively test** all the input/output combinations.
- Cannot certify system properties.

Formal methods

Cons:

- Requires a whole specialized team.
- Requires a **formal model** of the system.

Pro: ensures that the system respects some desirable properties.

Formal methods

Common approaches:

- Abstract interpretation
- Model checking
- Behavioral equivalences

Reduce to **systems of fixpoint equations**, over suitable lattices of information.

Abstract interpretation

- Analyse sw properties like **constant propagation analysis**, **bounds analysis**.
- Interprets the code over a proper abstract domain.
- Leads to **systems of least fixpoint equations** (roughly speaking, one equation per statement).

Model checking

- [Kozen1983] **Formal model** of the system and a **logic** to express properties.
- Checks whether the property is satisfied by the system's model.
- Produces least and greatest fixpoint equations.

Systems of fixpoint equations

Università degli Studi di Padova

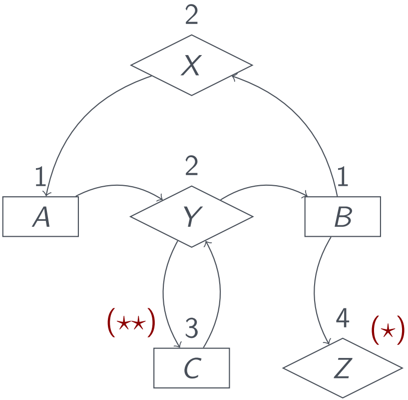
System of fixpoint equations

Given a complete lattice L, a **system of fixpoint equations** E over L is a list of m equations of the form

$$x_i =_{\eta_i} f_i(x_1, \ldots, x_m)$$

where f_i are monotone functions and $\eta_i \in \{\mu, \nu\}$.

The **solution** of systems of fixpoint equations can be characterized as a **parity game**, [Hasuo2016; Baldan2018].



The game could finish either:

(⋆) **finite play**, the winner is the player whose opponent is unable to move, or

(⋆⋆) **infinite play**, the winner is determined by the priorities appearing in the play.

Solution of a system of equations characterized via parity games.

Fixpoint game

L be a complete lattice, B_L a basis for L. Given a system of fixpoint equations E over L, the corresponding **fixpoint game** is a parity game defined as follows:

Position	Player	Moves
(b, i)	$\exists$	$\boldsymbol{X}$ such that $b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})$
$(X_1, \ldots, X_m)$	$\forall$	(b', j) such that $b' \in X_j$

Intuition: $\exists$ wins at (b, i) if $b \sqsubseteq$ solution of i-th equations.

- For model-checking: a state satisfies a property.
- For behavioral equivalence: two states are equivalent.

Fixpoint games

- Alternation between player $\exists$ and $\forall$.
- $\exists$ plays sets of elements in which is supposed to win.
  i.e., $\boldsymbol{X}$ such that $b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})$
- Afterwards, $\forall$ replies with one of them, asking $\exists$ to prove her/his guess.
  i.e., (b', j) such that $b' \in X_j$
- And the game continues until a winner is retrieved.

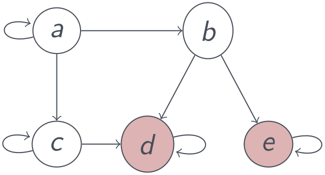
Model checking μ-calculus

μ-calculus:

- Expressive logic for system properties.
- Systems of equations over the sets of states.

Example: liveness properties

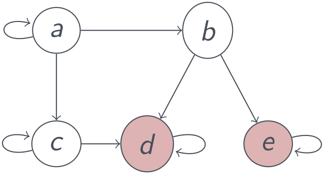
- Property P **eventually** holds.
- $X =_\mu P \vee \Diamond X$



E.g., the system will eventually enter the critical section.

Safety properties/invariants

- Property P always holds as **invariant**.
- $X =_\nu P \wedge \Box X$



E.g., absence of deadlocks (it is invariant that the system can progress).

- Property to satisfy with $P = \{E, D\}$:

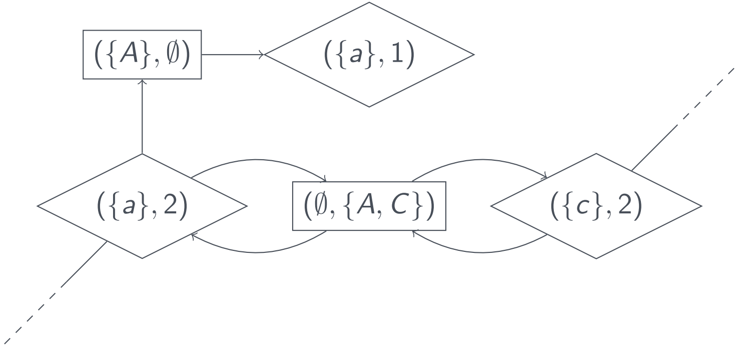
$$\varphi = \nu Y.((\mu X.(P \vee \Diamond X)) \wedge \Box Y)$$

e.g., the system can always eventually enter the critical section.

- The equivalent **system of fixpoint equations** is shown below:

$$\begin{cases} x_1 & =_\nu \{E, D\} \cap \blacksquare x_1 \\ x_2 & =_\mu x_1 \cup \blacklozenge x_2 \end{cases}$$

Graphical representation of some unfolding steps of the fixpoint game:



In practice

- Over **finite lattices**, the game-characterization allows one to determine the solution **constructively**, step-by-step.

- Using **progress measures**, a mechanism of propagation of the information for every position of the gameboard.
- For each position, the progress measure characterizes a winning strategy for $\exists$, if any.

Progress measure equations

The *progress measure equations* for E over the lattice $[\lambda_L]^m_\star$, is represented by Ψ_E the corresponding endofunction on $L \to \underline{m} \to [\lambda_L]^m_\star$ which is defined for $R : B_L \to \underline{m} \to [\lambda]^m_\star$, by

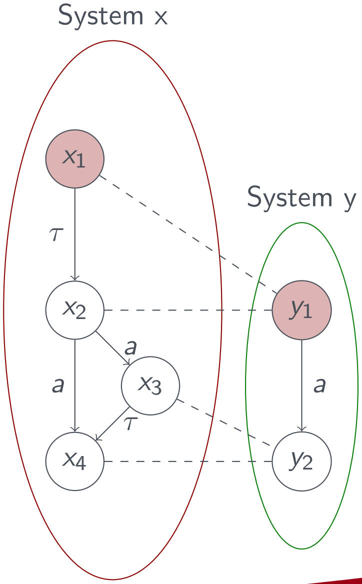
$$\Psi_E(R)(b)(i) =$$

$$min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}_i^{\eta_i} \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \boldsymbol{E}(b, i)\}$$

Progress measures

- Cons: requires to **explore all the moves** for each position.
- A naive application of the theory is highly **impractical**, too many moves.
- Global vs Local approach:
  1. Knowing the winner at each position can be of no interest.
  2. Moves can be very many.

Global vs Local approach



Example: systems equivalence

- We need to establish the equivalence of (x_1, y_1).
- The global approach computes all the possible pairs.

Local approach

- Determine the winner in a restricted set of positions.
- Usually the set of initial positions.

Pro: **on-demand exploration**, only what you need.

Reducing the moves:

- Number of moves can be impractically high.
- Notion of **selections** to limit all the possible moves player $\exists$ has to play each turn.
- Basic idea: dependencies between moves
  - if I know that a player wins over a defined move,
  - then more other moves will be winning for the given player.

Selections formally

- **Hoare preorder** on moves $(2^{B_L}, \sqsubseteq_H)$, where

$$X \sqsubseteq_H Y \text{ if } \forall x \in X, \exists y \in Y \text{ s.t. } x \sqsubseteq y$$

- **Upward-closure** with respects to Hoare as

$$\uparrow_H X = \{x \in (2^{B_L})^m \mid \exists y \in X \text{ s.t. } y \sqsubseteq_H x\}$$

selection

Function from positions of player $\exists$ to sets of moves. Such that, for all (b, i) it holds $\uparrow_H \sigma(b, i) = \boldsymbol{E}(b, i)$.

- Restricting the game to selections gives an equivalent game.

Symbolic $\exists$-moves

Logical representation of $\exists$-moves to efficiently explore selections.

logic for upward closed sets with respect to $\sqsubseteq_H$

Let L be a lattice and let B_L be a basis for L. Let m be the number of equations.
The logic $\mathcal{L}_m^H(B_L)$ has formulae defined as follows, where $b \in B_L$ and $j \in \underline{m}$:

$$\varphi ::= [b, j] \mid \bigwedge_{k \in K} \varphi_k \mid \bigvee_{k \in K} \varphi_k$$

Local approach and selections lead to a considerable saving in terms of exploration space.

Algorithm outline

- **Local approach** based on a **depth-first** exploration.
- **Assumptions/decisions** are made on the winner of some positions explored.
- **Assumptions** are added when the current position is possibly a winning position.
- **Decisions** are added when an evidence of a winning strategy for the current position is found.
- Already found positions are remembered to unfold loops.
- Only the sufficient set of moves is explored.
- Decisions and assumptions are **withdrawn** when there is a witness against them.

Algorithm outline

Functions Explore and Backtrack are mutually called throughout the execution.

- Function Explore goes downward until finds:
  - positions with no move left,
  - decisions/assumptions for the current position.

  Afterwards, we backtrack.
- Function Backtrack goes upward until finds:
  - the root,
  - positions controlled by the current winner's opponent with moves left.

Conclusion

- Game-based characterization of systems of fixpoint equations.
- Restriction of possible moves using selections.
- Progress measure to prove the equivalence of the game restricted to selections.
- Efficient logical representation of selections.
- Local algorithm to solve the game.

Future work

- Working tool, prototypal to solve verification tasks
- Infinite (height) lattices via abstraction
- Up-to functions