# Optimization Modulo Theory: A Tutorial Using Z3 and Practical Case Studies

Mădălina Erașcu

West University of Timișoara, Romania

madalina.erascu@e-uvt.ro

September 16th, 2024

*This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI-UEFISCDI, project number PN-III-P1-1.1-TE-2021-0676.*

# Outline

# Acknowledgements


- Material resources: Erika Abraham (RWTH Aachen), https://microsoft.github.io/z3guide/, https://github.com/Z3Prover/z3.


Z3 solver online to be used during the tutorial

## Acknowledgements


- Material resources: Erika Abraham (RWTH Aachen), https://microsoft.github.io/z3guide/, https://github.com/Z3Prover/z3.
- Parts of this work is joint with: Flavia Micota, Daniela Zaharie, Eduard Laitin.

## Acknowledgements

- Material resources: Erika Abraham (RWTH Aachen), https://microsoft.github.io/z3guide/, https://github.com/Z3Prover/z3.
- Parts of this work is joint with: Flavia Micota, Daniela Zaharie, Eduard Laitin.
- Parts of this work was inspired by or polished after teaching satisfiability checking topics to Master students or supervising Bachelor/Master theses.

## Acknowledgements

- Material resources: Erika Abraham (RWTH Aachen), https://microsoft.github.io/z3guide/, https://github.com/Z3Prover/z3.
- Parts of this work is joint with: Flavia Micota, Daniela Zaharie, Eduard Laitin.
- Parts of this work was inspired by or polished after teaching satisfiability checking topics to Master students or supervising Bachelor/Master theses.


Files used in this presentation

# Contents

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{aligned} \min \quad & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\ \text{subject to} \quad & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} \end{aligned}$$

## Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{aligned} \text{min} \quad & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\ \text{subject to} \quad & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} \end{aligned}$$

**Solution approaches:**

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{array}{rl} \min & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\ \text{subject to} & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} \end{array}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{array}{rl} \min & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\ \text{subject to} & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} \end{array}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]

## Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{array}{rl} \min & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\ \text{subject to} & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} \end{array}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]
   - Satisfiability Modulo Theory (SMT)
     - Z3 [7], cvc5 [3], Yices2 [8], OpenSMT [5].
   - Advantage: provides an optimal solution

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{array}{rll} \min & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) & \forall p \geq 1 \\ \text{subject to} & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} & \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} & \end{array}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]
   - Satisfiability Modulo Theory (SMT)
     - Z3 [7], cvc5 [3], Yices2 [8], OpenSMT [5].
   - Advantage: provides an optimal solution
   - Drawback: significant computational time for large problems

## Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{array}{rll} \text{min} & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) & \forall p \geq 1 \\ \text{subject to} & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} & \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} & \end{array}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]
   - Satisfiability Modulo Theory (SMT)
     - Z3 [7], cvc5 [3], Yices2 [8], OpenSMT [5].
   - Advantage: provides an optimal solution
   - Drawback: significant computational time for large problems

2. Approximate methods

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$
\begin{array}{rl}
\min & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\
\text{subject to} & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\
 & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m}
\end{array}
$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]
   - Satisfiability Modulo Theory (SMT)
     - Z3 [7], cvc5 [3], Yices2 [8], OpenSMT [5].
   - Advantage: provides an optimal solution
   - Drawback: significant computational time for large problems
2. Approximate methods
   - heuristics and metaheuristics

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{array}{rll} \min & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) & \forall p \geq 1 \\ \text{subject to} & g_i(\mathbf{x}) = c_i & \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j & \forall j = \overline{1, m} \end{array}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]
   - Satisfiability Modulo Theory (SMT)
     - Z3 [7], cvc5 [3], Yices2 [8], OpenSMT [5].
   - Advantage: provides an optimal solution
   - Drawback: significant computational time for large problems
2. Approximate methods
   - heuristics and metaheuristics
   - Advantage: faster

# Motivation

Constrained optimization problems have applications in engineering, economics, computer science, just to name a few.

**General form:**

$$\begin{aligned} \min \quad & f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_p(\mathbf{x}) \quad \forall p \geq 1 \\ \text{subject to} \quad & g_i(\mathbf{x}) = c_i \quad \forall i = \overline{1, n} \\ & h_j(\mathbf{x}) \geq d_j \quad \forall j = \overline{1, m} \end{aligned}$$

**Solution approaches:**

1. Exact methods
   - Constrained Programming (CP)
     - Modelling languages can be used, e.g. MiniZinc [15]
     - Google OR-Tools [16], Gecode [17], Chuffed [6]
   - Mathematical Programming (MP)
     - CPLEX [1]
   - Satisfiability Modulo Theory (SMT)
     - Z3 [7], cvc5 [3], Yices2 [8], OpenSMT [5].
   - Advantage: provides an optimal solution
   - Drawback: significant computational time for large problems
2. Approximate methods
   - heuristics and metaheuristics
   - Advantage: faster
   - Drawback: provides a (sub)optimal solution

# Contents

# Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.

# Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.
- In the propositional case, a model is a truth assignment to the Boolean variables.

## Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.
- In the propositional case, a model is a truth assignment to the Boolean variables.
- In the first-order (FO) case, a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.

# Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.
- In the propositional case, a model is a truth assignment to the Boolean variables.
- In the first-order (FO) case, a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.
- Automated reasoning failure: proof-search procedures for full FO logic $\rightsquigarrow$ is FO logic the best compromise between expressivity and efficiency?

# Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.
- In the propositional case, a model is a truth assignment to the Boolean variables.
- In the first-order (FO) case, a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.
- Automated reasoning failure: proof-search procedures for full FO logic $\leadsto$ is FO logic the best compromise between expressivity and efficiency?
- Gain efficiency by:

# Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.
- In the propositional case, a model is a truth assignment to the Boolean variables.
- In the first-order (FO) case, a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.
- Automated reasoning failure: proof-search procedures for full FO logic $\leadsto$ is FO logic the best compromise between expressivity and efficiency?
- Gain efficiency by:
  - addressing only (expressive enough) decidable fragments of a certain logic.

## Satisfiability Modulo Theory (SMT)

- Satisfiability is the problem of determining if a formula has a model.
- In the propositional case, a model is a truth assignment to the Boolean variables.
- In the first-order (FO) case, a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.
- Automated reasoning failure: proof-search procedures for full FO logic $\rightsquigarrow$ is FO logic the best compromise between expressivity and efficiency?
- Gain efficiency by:
  - addressing only (expressive enough) decidable fragments of a certain logic.
  - incorporate domain-specific reasoning, e.g: arithmetic, equality, data structures (arrays, lists, stacks, ...) and valid combinations

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language

- SMT: propositional logic + domain-specific reasoning

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
- SMT: propositional logic $+$ domain-specific reasoning

## Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning
  - $+$ better expressivity

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning
  - $+$ better expressivity
  - $-$ certain (but acceptable) loss of efficiency

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning
  - $+$ better expressivity
  - $-$ certain (but acceptable) loss of efficiency
- SAT competition: https://satcompetition.github.io/

## Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning
  - $+$ better expressivity
  - $-$ certain (but acceptable) loss of efficiency
- SAT competition: https://satcompetition.github.io/
- SMT competition: https://smt-comp.github.io/

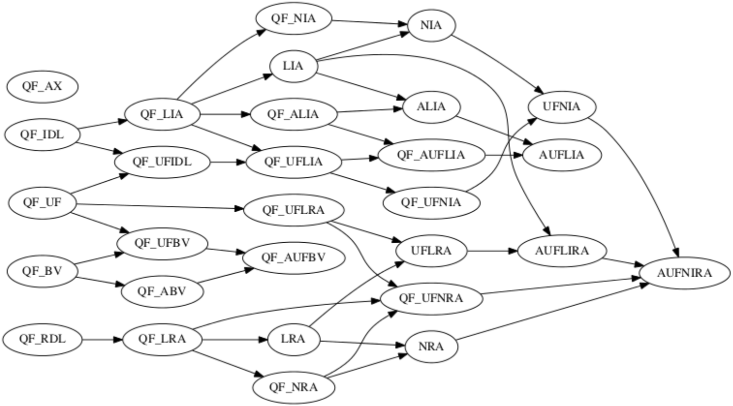
# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning
  - $+$ better expressivity
  - $-$ certain (but acceptable) loss of efficiency
- SAT competition: https://satcompetition.github.io/
- SMT competition: https://smt-comp.github.io/

# Satisfiability (SMT) and Optimization Modulo Theory (OMT) (cont'd)

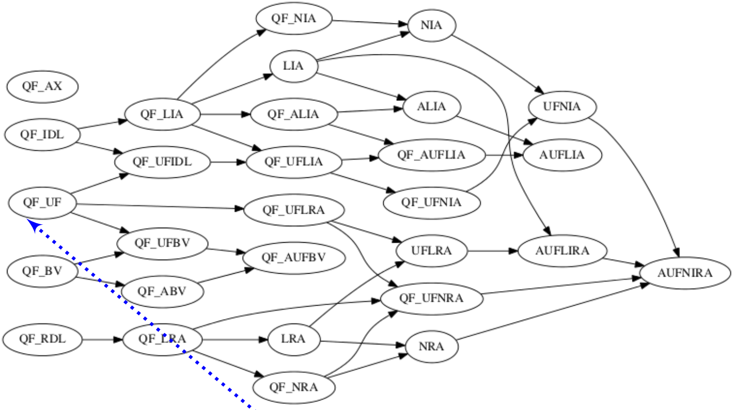
- SAT: uses propositional logic as the formalization language
  - $+$: high degree of efficiency
  - $-$: expressive but complex encodings
- SMT: propositional logic $+$ domain-specific reasoning
  - $+$ better expressivity
  - $-$ certain (but acceptable) loss of efficiency
- SAT competition: https://satcompetition.github.io/
- SMT competition: https://smt-comp.github.io/
- Some SMT solvers offer optimization features $\rightsquigarrow$ optimization modulo theory (OMT): Z3 [4], OptiMathSAT [18]; Symba [13], HAZEL [14], MAXHS-MSAT [10], PULI [11], CEGIO [2], BCLT [12].

# SMT Theories




Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

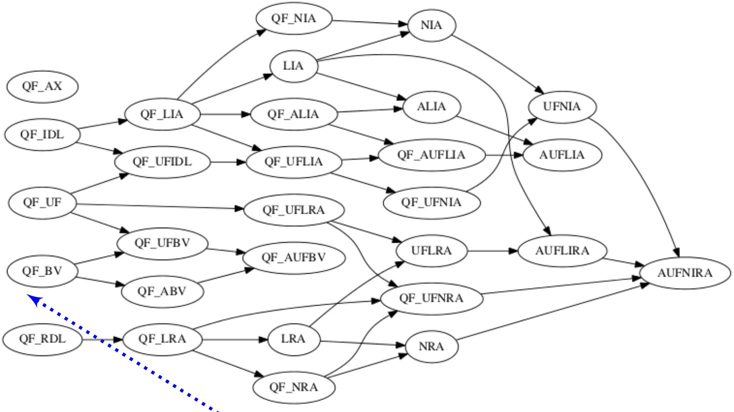
# SMT Theories




Quantifier-free equality logic with uninterpreted functions
$(a = c \wedge b = d) \rightarrow f(a, b) = f(c, d)$

Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

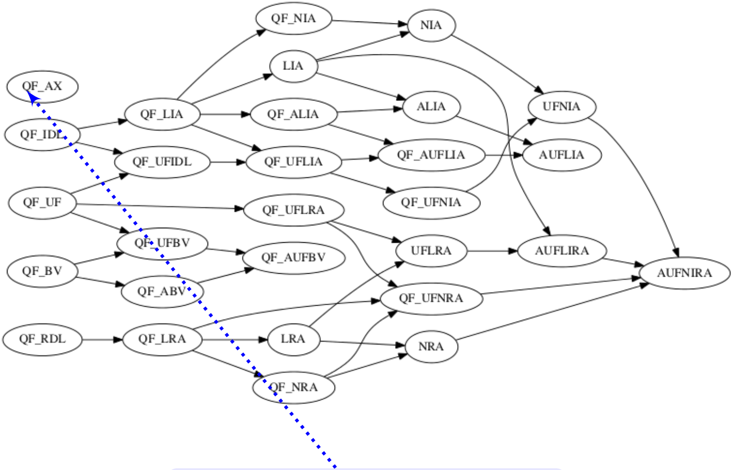
# SMT Theories




Quantifier-free bit-vector arithmetic
$a + b \geq 0 \wedge (a|b) \leq (a\&b)$

Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

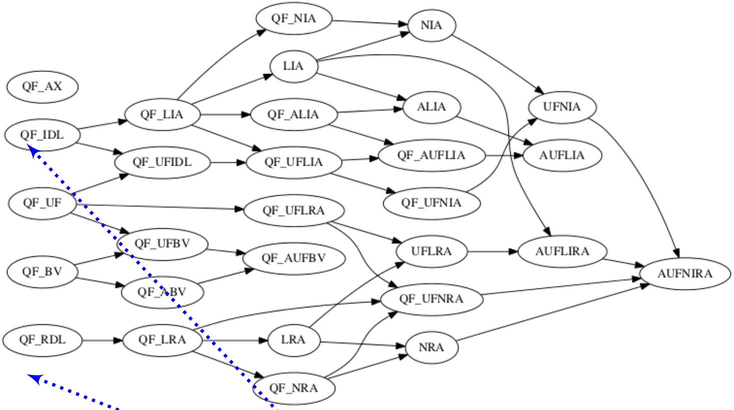
# SMT Theories




Quantifier-free array theory
$i = j \rightarrow read(write(a, i, v), j) = v$

Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

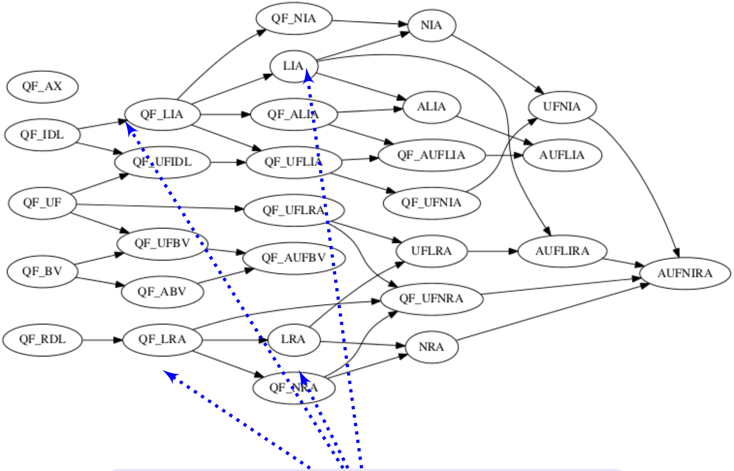
# SMT Theories




Quantifier-free integer/rational difference logic
$x - y \geq 0 \lor x - z < 0$

Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

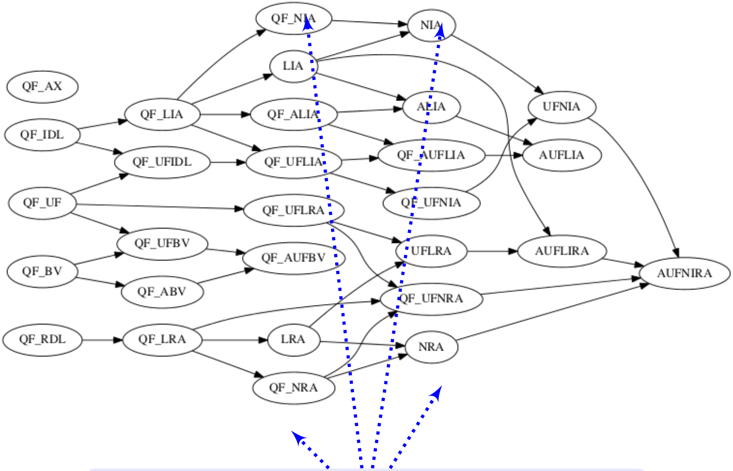
# SMT Theories




(Quantifier-free) real/integer linear arithmetic
$4x + 7y = 8 \wedge (y = 0 \vee x > y)$

Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

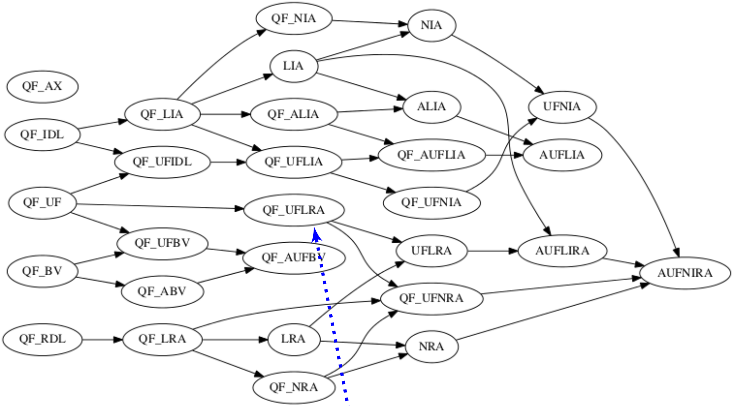
# SMT Theories




(Quantifier-free) real/integer non-linear arithmetic
$x^2 + 2xy + y^2 > 0 \vee (x \geq 1 \wedge xz + yz^2 = 0)$

Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

# SMT Theories




Combined theories
$$2f(x) + 5y > 0 \land \neg(f(x) = y \lor x + 2y = 0)$$

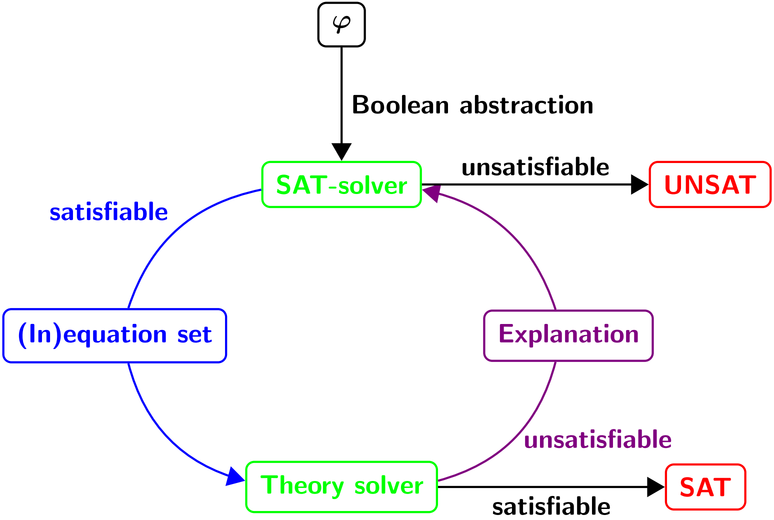
Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

## How an extension to SMT solving looks like?

There are basically two different approaches:

- Eager SMT solving transforms logical formulas over some theories into satisfiability-equivalent propositional logic formulas and applies SAT solving. ("Eager" means theory first)
- Lazy SMT solving uses a SAT solver to find solutions for the Boolean skeleton of the formula, and a theory solver to check satisfiability in the underlying theory. ("Lazy" means theory later)

# Lazy SMT solving



## Running Example

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

# Running Example

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Formalization.** Let $x_{ij}$ denote that VM $i$ is placed on the server $j$ and $y_j$ denote that server $j$ is in use.

# Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Choosing the suitable underlying theory is determined by the principles of the formalization: $x_{ij}, y_j \in \{0, 1\}$

- linear constraints with integer variables with 0,1 restriction
- linear constraints with real variables with 0,1 restriction
- linear constraints boolean variables

# Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Implicit constraints

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Implicit constraints
  - Variables are integers:

$$x_{ij}, y_j \in \mathbb{Z}, \quad \forall i, j = \overline{1,3}$$

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Implicit constraints
  - Variables are integers:

$$x_{ij}, y_j \in \mathbb{Z}, \quad \forall i, j = \overline{1,3}$$

  - Variables have only 0,1 value:

$$x_{ij} = 0 \vee x_{ij} = 1, \quad \forall i, j = \overline{1,3}$$

# Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Implicit constraints
  - Variables are integers:

$$x_{ij}, y_j \in \mathbb{Z}, \quad \forall i, j = \overline{1,3}$$

  - Variables have only 0,1 value:

$$x_{ij} = 0 \vee x_{ij} = 1, \quad \forall i, j = \overline{1,3}$$

  - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i = \overline{1,3}$$

# Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Implicit constraints
  - Variables are integers:

$$x_{ij}, y_j \in \mathbb{Z}, \quad \forall i, j = \overline{1,3}$$

  - Variables have only 0,1 value:

$$x_{ij} = 0 \vee x_{ij} = 1, \quad \forall i, j = \overline{1,3}$$

  - A VM is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i = \overline{1,3}$$

  - A used server has at least a VM on it:

$$(y_j \geq x_{1j}) \wedge (y_j \geq x_{2j}) \wedge (y_j \geq x_{3j}), \quad j = \overline{1,3}$$

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Explicit constraints

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Explicit constraints
  - Capacity constraints:
    $100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$
    $100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$
    $100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Explicit constraints
  - Capacity constraints:
    $100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$
    $100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$
    $100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$
- Optimization functions

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution.** Writing the constraints keeping in mind the underlying theory.
Assume the case: linear constraints with integer variables with 0,1 restriction

- Explicit constraints
  - Capacity constraints:
    $100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$
    $100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$
    $100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$
- Optimization functions
  - $10y_1 + 5y_2 + 20y_3$
  - $y_1 + y_2 + y_3$

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution approaches.**

1. Formalization in SMT-LIB2 format: useful for toy examples, some SMT tools are available online to try their capabilities.

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution approaches.**

1. Formalization in SMT-LIB2 format: useful for toy examples, some SMT tools are available online to try their capabilities.
2. Formalization by programming Z3, in particular Python API.

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution approaches.**

1. Formalization in SMT-LIB2 format: useful for toy examples, some SMT tools are available online to try their capabilities.
   - variant-int.smt2
2. Formalization by programming Z3, in particular Python API.

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution approaches.**

1. Formalization in SMT-LIB2 format: useful for toy examples, some SMT tools are available online to try their capabilities.
   - variant-int.smt2
   - variant-bool.smt2
2. Formalization by programming Z3, in particular Python API.

## Running Example (cont'd)

Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

- Minimize the number of servers used.
- Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

**Solution approaches.**

1. Formalization in SMT-LIB2 format: useful for toy examples, some SMT tools are available online to try their capabilities.
   - variant-int.smt2
   - variant-bool.smt2
2. Formalization by programming Z3, in particular Python API.

Is the order of the optimization functions important?

## Types of optimization (in Z3)

**Single-criteria optimization**:
$OMT(\mathcal{LIRA} \cup \mathcal{T}), OMT(\mathcal{BV} \cup \mathcal{T}), OMT(\mathcal{PB} \cup \mathcal{T})$ and MAXSMT solving [19].

# Types of optimization (in Z3)

**Single-criteria optimization**:
$OMT(\mathcal{LIRA} \cup \mathcal{T}), OMT(\mathcal{BV} \cup \mathcal{T}), OMT(\mathcal{PB} \cup \mathcal{T})$ and MAXSMT solving [19].
**Multi-criteria optimization**
To the best of our knowledge, in the SMT solver Z3, there are three ways to combine objective functions.

# Types of optimization (in Z3)

**Single-criteria optimization**:
$OMT(\mathcal{LIRA} \cup \mathcal{T})$, $OMT(\mathcal{BV} \cup \mathcal{T})$, $OMT(\mathcal{PB} \cup \mathcal{T})$ and MAXSMT solving [19].
**Multi-criteria optimization**
To the best of our knowledge, in the SMT solver Z3, there are three ways to combine objective functions.

1. lexicographic combinations (by default) variant-int.smt2

---

**Algorithm 3** Sequential algorithm for general objectives

---

**for** $t = 1$ to $n$ **do**
  Solve the single-objective problem:

$$\begin{aligned} \max \quad & f_t(x) \\ \text{subject to} \quad & x \in X, \\ & f_k(x) \geq z_k \text{ for all } k \in 1, \ldots, t-1. \end{aligned}$$

  **if** the problem is infeasible or unbounded **then**
    print "no solution"
  **else**
    Add as additional constraints the values of the decision variables $x$ and $f_k(x) = z_t$
  **end if**
**end for**

---

# Types of optimization (in Z3) (cont'd)

2. Boxes are used to specify independent optima subject to given constraints: variant-int-box.smt2

# Types of optimization (in Z3) (cont'd)

2. Boxes are used to specify independent optima subject to given constraints: variant-int-box.smt2
3. Pareto optimization involves more than one objective function to be optimized simultaneously. variant-int-pareto.smt2

# Programming Z3 (Python API)

- variant-int.py

# Programming Z3 (Python API)

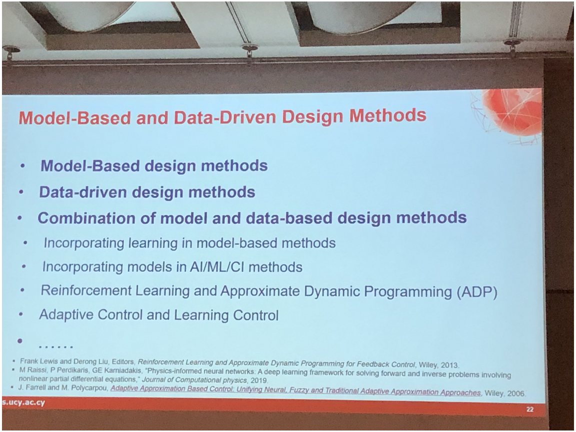
- variant-int.py
- variant-bool.py

# Feedback Part 1

Please fill out the form!

# Contents

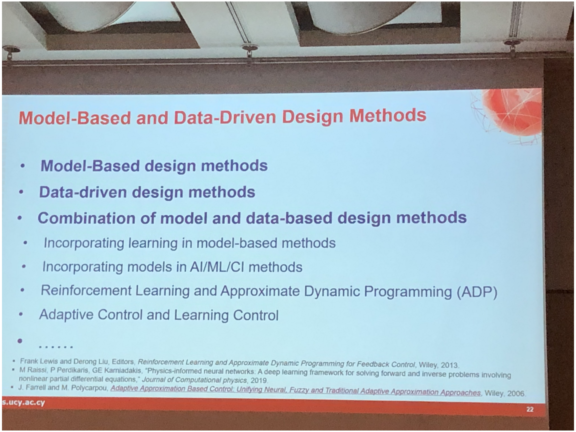
# Motivation




- The data-driven approach focuses on using data to drive the development and improvement of AI systems.
- The model-based approach focuses on developing a mathematical model of the system or process being studied.

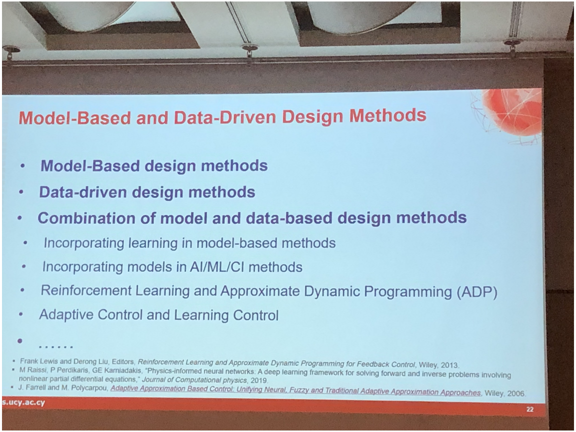
# Motivation

From Marios M. Polycarpou talk



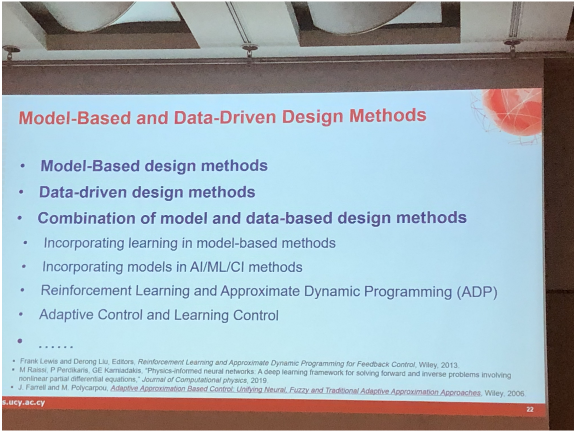

- The data-driven approach focuses on using data to drive the development and improvement of AI systems.
- The model-based approach focuses on developing a mathematical model of the system or process being studied.

# Motivation




- The data-driven approach focuses on using data to drive the development and improvement of AI systems. $\rightsquigarrow$ graph neural networks
- The model-based approach focuses on developing a mathematical model of the system or process being studied.

# Motivation




- The data-driven approach focuses on using data to drive the development and improvement of AI systems. $\rightsquigarrow$ graph neural networks
- The model-based approach focuses on developing a mathematical model of the system or process being studied. $\rightsquigarrow$ satisfiability/optimization modulo theory

## Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

# Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

**Benefits** of optimal deployment:

1. the synthesis of deployment plans that are optimal by design

# Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

**Benefits** of optimal deployment:

1. the synthesis of deployment plans that are optimal by design
2. the integration of such deployment plans into the application modeling process enables formal reasoning on a model of the deployed application.

## Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

**Benefits** of optimal deployment:

1. the synthesis of deployment plans that are optimal by design
2. the integration of such deployment plans into the application modeling process enables formal reasoning on a model of the deployed application.

**Automated deployment** of component-based applications in the Cloud consists of:

# Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

**Benefits** of optimal deployment:

1. the synthesis of deployment plans that are optimal by design
2. the integration of such deployment plans into the application modeling process enables formal reasoning on a model of the deployed application.

**Automated deployment** of component-based applications in the Cloud consists of:

1. selection of the computing resources,

# Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

**Benefits** of optimal deployment:

1. the synthesis of deployment plans that are optimal by design
2. the integration of such deployment plans into the application modeling process enables formal reasoning on a model of the deployed application.

**Automated deployment** of component-based applications in the Cloud consists of:

1. selection of the computing resources,
2. distribution/assignment of the application components over the available computing resources,

## Motivation (cont'd)

Advent of Cloud computing $\rightsquigarrow$ loosely-coupled architecture $\rightsquigarrow$ DevOps paradigm $\rightsquigarrow$ application modeling $\rightsquigarrow$ optimal deployment

**Benefits** of optimal deployment:

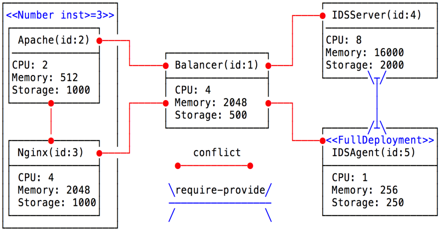
1. the synthesis of deployment plans that are optimal by design
2. the integration of such deployment plans into the application modeling process enables formal reasoning on a model of the deployed application.

**Automated deployment** of component-based applications in the Cloud consists of:

1. selection of the computing resources,
2. distribution/assignment of the application components over the available computing resources,
3. its dynamic modification to cope with peaks of user requests.

# Motivating Example

**Problem:** finding the best offer for a Secure Web Container

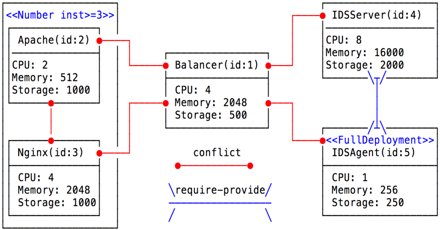



## Components

- two Web Containers (e.g. Apache Tomcat or Nginx)
- a Balancer
- an IDSServer (Intrusion Detection System)
- an IDS Agent

# Motivating Example

**Problem:** finding the best offer for a Secure Web Container




## Components

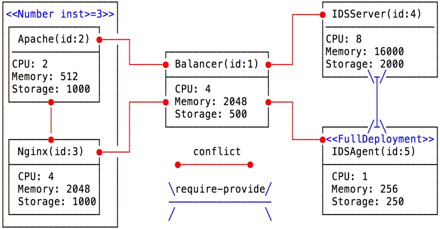
- two Web Containers (e.g. Apache Tomcat or Nginx)
- a Balancer
- an IDSServer (Intrusion Detection System)
- an IDS Agent

## Constraints

- Conflicts: Apache and Nginx cannot be deployed on the same VM.
- Conflicts: Balancer needs exclusive use of machines.
- Equal bound: exactly one Balancer has to be instantiated.
- Lower bound: at least 3 instances of Apache and/or Nginx are required.
- Require-provides: one IDSServer for 10 IDS Agents.
- Full deployment: one instance of the IDS Agent on all VMs except for those containing the IDSServer and the Balancer.
- Hardware constraints: components hardware requirements.

# Motivating Example

**Problem:** finding the best offer for a Secure Web Container




## Components

- two Web Containers (e.g. Apache Tomcat or Nginx)
- a Balancer
- an IDSServer (Intrusion Detection System)
- an IDS Agent

## Constraints

- Conflicts: Apache and Nginx cannot be deployed on the same VM.
- Conflicts: Balancer needs exclusive use of machines.
- Equal bound: exactly one Balancer has to be instantiated.
- Lower bound: at least 3 instances of Apache and/or Nginx are required.
- Require-provides: one IDSServer for 10 IDS Agents.
- Full deployment: one instance of the IDS Agent on all VMs except for those containing the IDSServer and the Balancer.
- Hardware constraints: components hardware requirements.

Goal: find a set of virtual machines (VMs) which satisfies the components requirements and lead to the minimum cost.

# Motivating Example (cont'd)

Spot Instance Prices

Spot Instances | Defined Duration for Linux | Defined Duration for Windows

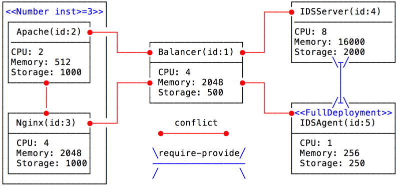
Region: EU (Ireland)

| | Linux/UNIX Usage | Windows Usage |
|---|---|---|
| **General Purpose - Current Generation** | | |
| t2.micro | $0.0038 per Hour | $0.0084 per Hour |
| t2.small | $0.0075 per Hour | $0.0165 per Hour |
| t2.medium | $0.015 per Hour | $0.033 per Hour |
| t2.large | $0.0302 per Hour | $0.0582 per Hour |
| t2.xlarge | $0.0605 per Hour | $0.1015 per Hour |
| t2.2xlarge | $0.121 per Hour | $0.183 per Hour |
| m3.medium | $0.0073 per Hour | $0.0633 per Hour |
| m3.large | $0.0306 per Hour | $0.1226 per Hour |
| m3.xlarge | $0.0612 per Hour | $0.2452 per Hour |

| Model | vCPU | CPU Credits / hour | Mem (GiB) | Storage |
|---|---|---|---|---|
| t2.nano | 1 | 3 | 0.5 | EBS-Only |
| t2.micro | 1 | 6 | 1 | EBS-Only |
| t2.small | 1 | 12 | 2 | EBS-Only |
| t2.medium | 2 | 24 | 4 | EBS-Only |
| t2.large | 2 | 36 | 8 | EBS-Only |
| t2.xlarge | 4 | 54 | 16 | EBS-Only |
| t2.2xlarge | 8 | 81 | 32 | EBS-Only |

Remark: [snapshot from https://aws.amazon.com/ec2/] tens of thousands of price offers corresponding to different configurations and zones

# Motivating Example (cont'd)




## Example solution

- $VM_1$ (CPU:8, RAM: 15 GB, Storage: 2000 GB, Price: 0.0526 \$/hour): Nginx + IDS Agent
- $VM_2$ (CPU:4, RAM: 7.5 GB, Storage: 2000 GB, Price: 0.0283 \$/hour): Balancer
- $VM_3$ (CPU:4, RAM: 30 GB, Storage: 2000 GB, Price: 0.0644 \$/hour): IDSServer
- $VM_4$ (CPU:4, RAM: 7.5 GB, Storage: 2000 GB, Price: 0.0283 \$/hour): Apache + IDS Agent
- $VM_5$ (CPU:4, RAM: 7.5 GB, Storage: 2000 GB, Price: 0.0283 \$/hour): Apache + IDS Agent

# Contents

# Problem Specification

**Automated deployment** of component-based applications in the Cloud consists of:

1. selection of the computing resources,
2. distribution/assignment of the application components over the available computing resources,
3. dynamic modification to cope with peaks of user requests.

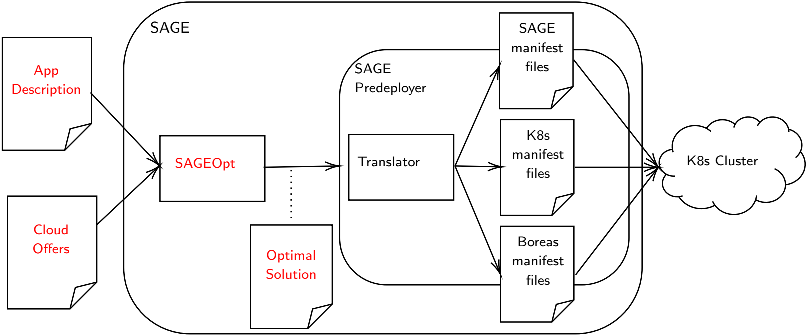
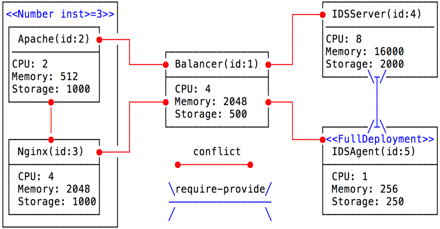



Figure: SAGE General Architecture

# Contents

# Recall Motivating Example

**Problem:** finding the best offer for a Secure Web Container




## Components

- two Web Containers (e.g. Apache Tomcat or Nginx)
- a Balancer
- an IDSServer (Intrusion Detection System)
- an IDS Agent

## Constraints

- Conflicts: Apache and Nginx cannot be deployed on the same VM.
- Conflicts: Balancer needs exclusive use of machines.
- Equal bound: exactly one Balancer has to be instantiated.
- Lower bound: at least 3 instances of Apache and/or Nginx are required.
- Require-provides: one IDSServer for 10 IDS Agents.
- Full deployment: one instance of the IDS Agent on all VMs except for those containing the IDSServer and the Balancer.
- Hardware constraints: components hardware requirements.

Goal: find a set of virtual machines (VMs) which satisfies the components requirements and lead to the minimum cost.

# Model-driven approach: Formulation of the Satisfiability/Optimization Modulo Theory Problem

## General constraints

| | | |
|---|---|---|
| *Basic allocation* | $\sum_{k=1}^{M} a_{ik} \geq 1$ | $\forall i = \overline{1, N}$ |
| *Occupancy* | $\sum_{i=1}^{N} a_{ik} \geq 1 \Rightarrow v_k = 1$ | $\forall k = \overline{1, M}$ |
| *Capacity* | $\sum_{i=1}^{N} a_{ik} \cdot R_i^h \leq F_{t_k}^h$ | $\forall k = \overline{1, M}, \forall h = \overline{1, H}$ |
| *Link* | $v_k{=}1 \wedge t_k{=}o \Rightarrow \bigwedge_{h=1}^{H} \left(r_k^h{=}F_{t_k}^h\right) \wedge p_k{=}P_{t_k}$ | $\forall o = \overline{1, O},\ \ O \in \mathbb{N}^*$ |
| | $\sum_{i=1}^{N} a_{ik} = 0 \Rightarrow t_k = 0$ | $\forall k = \overline{1, M}$ |

where:

- $R_i^h \in \mathbb{N}^*$ is the hardware requirement of type $h$ of the component $i$;
- $F_{t_k}^h \in \mathbb{N}^*$ is the hardware characteristic $h$ of the VM of type $t_k$.

# Problem Formalization (cont'd)

## Application-specific constraints

| | | |
|---|---|---|
| *Conflicts* | $a_{ik} + a_{jk} \leq 1$ | $\forall k = \overline{1, M}, \forall (i,j)\ \mathcal{R}_{ij} = 1$ |
| *Co-location* | $a_{ik} = a_{jk}$ | $\forall k = \overline{1, M}, \forall (i,j)\ \mathcal{D}_{ij} = 1$ |
| *Exclusive* | *deployment* | |
| | $\mathcal{H}\left(\sum_{k=1}^{M} a_{i_1 k}\right) + \ldots + \mathcal{H}\left(\sum_{k=1}^{M} a_{i_q k}\right) = 1$ | for fixed $q \in \{1, \ldots, N\}$ |
| | | $\mathcal{H}(u) = \begin{cases} 1 & u > 0 \\ 0 & u = 0 \end{cases}$ |
| *Require-* | *Provide* | |
| | $n_{ij} \sum_{k=1}^{M} a_{ik} \leq m_{ij} \sum_{k=1}^{M} a_{jk}$ | $\forall (i,j) \mathcal{Q}_{ij}(n_{ij}, m_{ij}) = 1$ |
| | $0 \leq n \sum_{k=1}^{M} a_{jk} - \sum_{k=1}^{M} a_{ik} < n$ | $n, n_{ij}, m_{ij} \in \mathbb{N}^*$ |

where:

- $\mathcal{R}_{ij} = 1$ if components $i$ and $j$ are in conflict (can not be placed in the same VM);
- $\mathcal{D}_{ij} = 1$ if components $i$ and $j$ must be co-located (must be placed in the same VM);
- $\mathcal{Q}_{ij}(n, m)$=1 if $C_i$ requires at least $n$ instances of $C_j$ and $C_j$ can serve at most $m$ instances of $C_i$

# Problem Formalization (cont'd)

## Application-specific constraints

*Full deployment* $\sum_{k=1}^{M}\left(a_{ik} + \mathcal{H}\left(\sum_{j,\mathcal{R}_{ij}=1} a_{jk}\right)\right) = \sum_{k=1}^{M} v_k$

*Deployment with bounded number of instances*

$\sum_{i\in\overline{C}}\sum_{k=1}^{M} a_{ik}\langle\text{op}\rangle n \qquad |\overline{C}| \leq N,\ \langle\text{op}\rangle\in\{=,\leq,\geq\}, n\in\mathbb{N}$

Find:

- assignment matrix $a$ with binary entries $a_{ik} \in \{0,1\}$ for $i = \overline{1,N}$, $k = \overline{1,M}$, which are interpreted as follows:

$$a_{ik} = \begin{cases} 1 & \text{if } C_i \text{ is assigned to } V_k \\ 0 & \text{if } C_i \text{ is not assigned to } V_k. \end{cases}$$

- the type selection vector $t$ with integer entries $t_k$ for $k = \overline{1,M}$, representing the type (from a predefined set) of each VM leased.

Such that: the leasing price is minimal $\sum_{k=1}^{M} v_k \cdot p_k$

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
- VM nodes

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
- VM nodes

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
- VM nodes

# Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*
  - No edges between two VM nodes!

## Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*
  - No edges between two VM nodes!

*Feature scaling:*

# Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*
  - No edges between two VM nodes!

*Feature scaling:*

- to $[0, 1]$ is needed in order to prevent one feature from dominating while preserving min-max relationships, crucial for prediction algorithms
- is needed for the hardware specifications/requirements and the VM price.

# Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*
  - No edges between two VM nodes!

*Feature scaling:*

- to $[0, 1]$ is needed in order to prevent one feature from dominating while preserving min-max relationships, crucial for prediction algorithms
- is needed for the hardware specifications/requirements and the VM price.

*Edge classification*

# Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*
  - No edges between two VM nodes!

*Feature scaling:*

- to $[0, 1]$ is needed in order to prevent one feature from dominating while preserving min-max relationships, crucial for prediction algorithms
- is needed for the hardware specifications/requirements and the VM price.

*Edge classification*

- Initially, all edges between a node of type component and one of type VM are of type *unlinked*.

# Graph Neural Network Formulation

The first step in solving the edge classification problem is to model it as graph data.

*Heterogeneous graph*:

- component nodes
  - The features are *(ID, CPU, Mem, Sto, FullDepl, UpperB, LowerB, EqualB)*.
  - Edges between component nodes are determined by the application-specific constraints except FullDepl, UpperB, LowerB, EqualB.
  - Edges are specified using one-hot encoding on the possible application-specific constraints: *(Conflict, Co-location, RequireProvide, ExclusiveDeployment, UpperB, LowerB, EqualB)*.
- VM nodes
  - The features are *(CPU, Mem, Sto, Price)*
  - No edges between two VM nodes!

*Feature scaling:*

- to $[0, 1]$ is needed in order to prevent one feature from dominating while preserving min-max relationships, crucial for prediction algorithms
- is needed for the hardware specifications/requirements and the VM price.

*Edge classification*

- Initially, all edges between a node of type component and one of type VM are of type *unlinked*.

**The task is to implement a GNN model in order to predict the type (linked/unlinked) for all the edges.**

# Contents

## Solution

1. Generate the dataset which is used to train a GNN model of the application to be deployed. This dataset, representing optimal deployment plans, is obtained by multiple runs of the exact solver previously developed by us.

# Solution

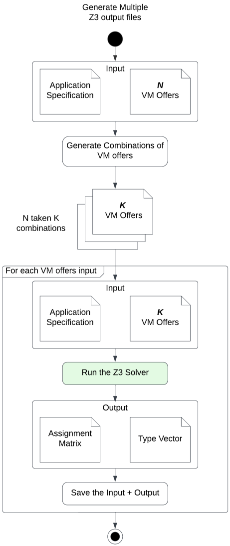
1. Generate the dataset which is used to train a GNN model of the application to be deployed. This dataset, representing optimal deployment plans, is obtained by multiple runs of the exact solver previously developed by us.
2. Train a GNN model which *predicts* the assignments of components to VMs as well as the VM Offers.

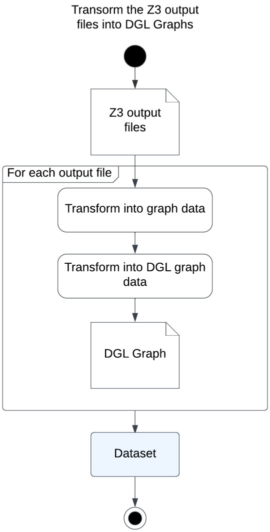
# Solution

1. Generate the dataset which is used to train a GNN model of the application to be deployed. This dataset, representing optimal deployment plans, is obtained by multiple runs of the exact solver previously developed by us.
2. Train a GNN model which *predicts* the assignments of components to VMs as well as the VM Offers.
3. Transform the predictions into *soft constraints* to guide the search exploration of the Base solver towards an optimal solution.

# Dataset generation

Large dataset to train the model $\binom{20}{15} \approx 15000$ different VM Offers inputs.



# Training a GNN model for edge classification

Supervised GNN learning approach:

## Training a GNN model for edge classification

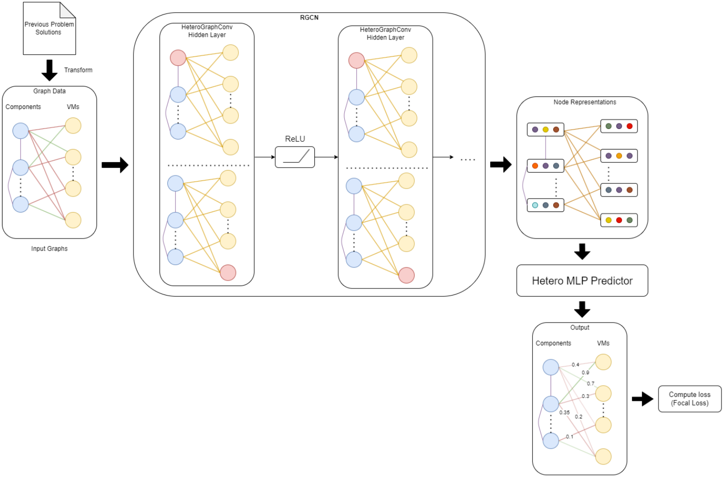
Supervised GNN learning approach:

1. Data Preparation: graph representation and nodes and edges feature extraction.
2. Graph Construction: application graph's structure (nodes, edges, and their relationships)

# Training a GNN model for edge classification

Supervised GNN learning approach:

3. Choosing Model Architecture which allows heterogeneity modeling and edge classification.



# Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.

# Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.
5. Adam optimizer with the default parameters provided by DGL library.

# Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.
5. Adam optimizer with the default parameters provided by DGL library.
6. Model Training, Validation and Testing

## Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.
5. Adam optimizer with the default parameters provided by DGL library.
6. Model Training, Validation and Testing
   - train, test, validation: $60\%, 20\%, 20\%$

## Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.

5. Adam optimizer with the default parameters provided by DGL library.

6. Model Training, Validation and Testing
   - train, test, validation: $60\%, 20\%, 20\%$
   - batch size $= 50$

# Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.

5. Adam optimizer with the default parameters provided by DGL library.

6. Model Training, Validation and Testing
   - train, test, validation: $60\%, 20\%, 20\%$
   - batch size $= 50$
   - training dataset size: min $= 50$, max $= 10000$

# Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.
5. Adam optimizer with the default parameters provided by DGL library.
6. Model Training, Validation and Testing
   - train, test, validation: $60\%, 20\%, 20\%$
   - batch size $= 50$
   - training dataset size: min $= 50$, max $= 10000$
   - #epochs: min $= 30$, max $= 600$

# Training a GNN model for edge classification

Supervised GNN learning approach:

4. The loss function is focal loss suitable for imbalanced sets.

5. Adam optimizer with the default parameters provided by DGL library.

6. Model Training, Validation and Testing
   - train, test, validation: $60\%, 20\%, 20\%$
   - batch size $= 50$
   - training dataset size: min $= 50$, max $= 10000$
   - #epochs: min $= 30$, max $= 600$

| # | Sample Size | #Ep | Acc | Time | Pred. $\mathbb{T}$ Links | Pred. $\mathbb{F}$ Links | GT True Links |
|---|---|---|---|---|---|---|---|
| 3 | 50 | 200 | 0.95 | 21.36 | 7 | 10 | 8 |
| 7 | 100 | 100 | 0.95 | 21.92 | 7 | 13 | |
| 11 | 100 | 400 | 0.95 | 87.92 | 8 | 13 | |

# Integrated GNN and Exact Techniques: Experimental Results

- the scalability of the GNN approach for increasing number of VM offers, possibly previously unseen (see table), and

# Integrated GNN and Exact Techniques: Experimental Results

- the scalability of the GNN approach for increasing number of VM offers, possibly previously unseen (see table), and
- the generalization of the GNN approach for applications characterized by similar constraints between components but with different hardware requirements (see the paper [9]).

# Experimental Analysis (cont'd)

## Explanation of the FV symmetry breaker

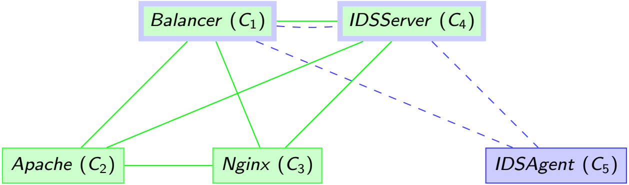



Figure: Secure Web Container conflict graph. The components with green background belong to the clique $\overline{G}$.

| | $VM_1$ | $VM_2$ | $VM_3$ | $VM_4$ | $VM_5$ | $VM_6$ |
|---|---|---|---|---|---|---|
| $C_1$ | $\boxed{1}$ | $\boxed{0}$ | $\boxed{0}$ | $\boxed{0}$ | 0 | 0 |
| $C_2$ | $\boxed{0}$ | $\boxed{0}$ | $\boxed{1}$ | $\boxed{0}$ | 1 | 0 |
| $C_3$ | $\boxed{0}$ | $\boxed{0}$ | $\boxed{0}$ | $\boxed{1}$ | 0 | 0 |
| $C_4$ | $\boxed{0}$ | $\boxed{1}$ | $\boxed{0}$ | $\boxed{0}$ | 0 | 0 |
| $C_5$ | 0 | 0 | 1 | 1 | 1 | 0 |

Table: Effect of FV symmetry breaking strategy

# Integrated GNN and Exact Techniques: Experimental Results

- the scalability of the GNN approach for increasing number of VM offers, possibly previously unseen (see table). Base = Z3

| #o | Solver | Model#3 | Model#7 | Model#11 | Opt. Price |
|---|---|---|---|---|---|
| 20 | Base | | 0.24 | | 3.759 |
| | Base+FVPR | | 0.11 | | |
| | Base+GNN | 0.12 | 0.12 | 0.07 | |
| | Base+FVPR+GNN | 0.12 | 0.09 | 0.10 | |
| 40 | Base | | 0.54 | | 2.676 |
| | Base+FVPR | | 0.27 | | |
| | Base+GNN | 0.28 | 0.33 | 0.28 | |
| | Base+FVPR+GNN | 0.29 | 0.28 | 0.29 | |
| 250 | Base | | 2.82 | | 1.622 |
| | Base+FVPR | | 0.98 | | |
| | Base+GNN | 0.76 | 0.77 | 1 | |
| | Base+FVPR+GNN | 1.40 | 1.39 | 1.15 | |
| 500 | Base | | 8.71 | | 1.582 |
| | Base+FVPR | | 2.42 | | |
| | Base+GNN | 4.56 | 2.92 | 1.5 | |
| | Base+FVPR+GNN | 3.09 | 3 | 3.01 | |
| 27 | Base | | 0.26 | | 2.400 |
| | Base+FVPR | | 0.09 | | |
| | Base+GNN | 0.10 | 0.14 | 0.14 | |
| | Base+FVPR+GNN | 0.10 | 0.10 | 0.07 | |

# Experimental Analysis (cont'd)

## Using the GNN Prediction as Soft Constraints

Formalization of the assignment predictions as a binary 3D tensor *pred* with $pred_{iko} \in \{0,1\}$ for $i = \overline{1,N}$, $k = \overline{1,M}$ and $o = \overline{1,O}$:

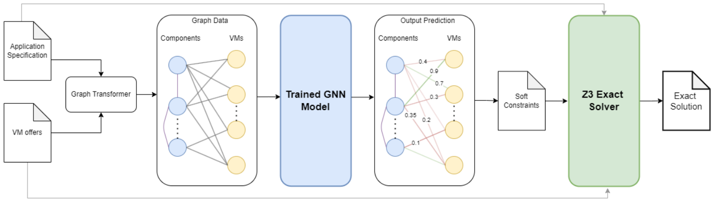
$$pred_{iko} = \begin{cases} 1 & \text{if } C_i \text{ is assigned to } V_j \text{ of type } O_o \\ 0 & \text{if } C_i \text{ is not assigned to } V_j \text{ of type } O_o \end{cases}$$

From tensors to soft constraints

$$\exists o \in \overline{1,O} \text{ s.t. } pred_{iko} = 1 \implies a_{ik} = 1 \wedge \bigwedge_{h=1}^{H} \left( r_k^h = F_o^h \right) \wedge p_k = P_o$$

$$\nexists o \in \overline{1,O} \text{ s.t. } pred_{iko} = 1 \implies a_{ik} = 0$$

Diagram describing the integration of the GNN model with the Base solver



## Experimental Analysis (cont'd)

**Using the GNN Prediction as Soft Constraints** The *pred* tensor for the *first component* of the Secure Web Container application, generated from running the GNN model prediction on the case study application with 10 VM offers, looks like:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where $M = 6$ rows and $O = 10$ columns.

The corresponding soft constraints are:

- for assignment matrix $a$:

```
(assert-soft (= a11 0))    (assert-soft (= a13 1))    (assert-soft (= a15 0))
(assert-soft (= a12 1))    (assert-soft (= a14 1))    (assert-soft (= a16 0))
```

- for type vector $t$:

```
(assert-soft (and (= PriceProv2 8.403)...))
(assert-soft (and (= PriceProv3 8.403)...))
(assert-soft (and (= PriceProv4 0.093)...))
```

  where the predictions obtained for the VM offers type were $t_2 = 7$, $t_3 = 7$, $t_4 = 5$. For example, the VM Offer 5 has the specification $(1, 3750, 1000, 0.093)$.

# Experimental Analysis (cont'd)

**Implementation of soft constraints in Z3 optimization**

Optimization in Z3:

1. If the soft constraints are added before the optimization function, then the solver tries to satisfy as many as possible and will use the intermediate results for the optimization function (multi-criterial optimization with lexicographic option by default). $\rightsquigarrow$ hence the soft constraints might destry the actual optimim

# Experimental Analysis (cont'd)

**Implementation of soft constraints in Z3 optimization**

Optimization in Z3:

1. If the soft constraints are added before the optimization function, then the solver tries to satisfy as many as possible and will use the intermediate results for the optimization function (multi-criterial optimization with lexicographic option by default). $\rightsquigarrow$ hence the soft constraints might destry the actual optimim
2. If the soft constraints are declared after the optimization function, the optimal value will be found without the soft constraints $\rightsquigarrow$ this does not exploit the benefits of soft constraints in guiding the solver to find faster the solution.

# Experimental Analysis (cont'd)

**Implementation of soft constraints in Z3 optimization**

Optimization in Z3:

1. If the soft constraints are added before the optimization function, then the solver tries to satisfy as many as possible and will use the intermediate results for the optimization function (multi-criterial optimization with lexicographic option by default). $\rightsquigarrow$ hence the soft constraints might destry the actual optimim
2. If the soft constraints are declared after the optimization function, the optimal value will be found without the soft constraints $\rightsquigarrow$ this does not exploit the benefits of soft constraints in guiding the solver to find faster the solution.

We simulated soft constraints with *pseudo-boolean and cardinality constraints.*

# Experimental Analysis (cont'd)

**Implementation of soft constraints in Z3 optimization**

Optimization in Z3:

1. If the soft constraints are added before the optimization function, then the solver tries to satisfy as many as possible and will use the intermediate results for the optimization function (multi-criterial optimization with lexicographic option by default). $\rightsquigarrow$ hence the soft constraints might destry the actual optimim
2. If the soft constraints are declared after the optimization function, the optimal value will be found without the soft constraints $\rightsquigarrow$ this does not exploit the benefits of soft constraints in guiding the solver to find faster the solution.

We simulated soft constraints with *pseudo-boolean and cardinality constraints.*

## Pseudo-Boolean constraint

A linear *pseudo-Boolean constraint* has the form: $\sum_j a_i l_j \triangleright b$ where $a_i$ and $b$ are integer constants, $l_j$ are literals and $\triangleright$ is a relational operator.

# Experimental Analysis (cont'd)

**Implementation of soft constraints in Z3 optimization**

Optimization in Z3:

1. If the soft constraints are added before the optimization function, then the solver tries to satisfy as many as possible and will use the intermediate results for the optimization function (multi-criterial optimization with lexicographic option by default). $\rightsquigarrow$ hence the soft constraints might destry the actual optimim
2. If the soft constraints are declared after the optimization function, the optimal value will be found without the soft constraints $\rightsquigarrow$ this does not exploit the benefits of soft constraints in guiding the solver to find faster the solution.

We simulated soft constraints with *pseudo-boolean and cardinality constraints.*

## Pseudo-Boolean constraint

A linear *pseudo-Boolean constraint* has the form: $\sum_j a_i l_j \triangleright b$ where $a_i$ and $b$ are integer constants, $l_j$ are literals and $\triangleright$ is a relational operator.

## Cardinality Constraint

A *cardinality constraint* is a constraint on the number of literals which are true among a given set of literals.

# Experimental Analysis (cont'd)

**Implementation of soft constraints in Z3 optimization**

Optimization in Z3:

1. If the soft constraints are added before the optimization function, then the solver tries to satisfy as many as possible and will use the intermediate results for the optimization function (multi-criterial optimization with lexicographic option by default). $\rightsquigarrow$ hence the soft constraints might destry the actual optimim
2. If the soft constraints are declared after the optimization function, the optimal value will be found without the soft constraints $\rightsquigarrow$ this does not exploit the benefits of soft constraints in guiding the solver to find faster the solution.

We simulated soft constraints with *pseudo-boolean and cardinality constraints.*

## Pseudo-Boolean constraint

A linear *pseudo-Boolean constraint* has the form: $\sum_j a_i l_j \triangleright b$ where $a_i$ and $b$ are integer constants, $l_j$ are literals and $\triangleright$ is a relational operator.

## Cardinality Constraint

A *cardinality constraint* is a constraint on the number of literals which are true among a given set of literals.

In our case: `atmost(`$k$`,` $\{x_1, x_2, \ldots, x_n\}$`)` is true if and only if at most $k$ literals among $x_1, x_2, \ldots, x_n$ are true.

# Contents

# Future Work

- Investigate better the timings of Base+GNN and Base+FV+GNN on this use case and others.

# Future Work

- Investigate better the timings of Base+GNN and Base+FV+GNN on this use case and others.
- Investigate the characteristics of the datasets.
- Improve the GNN model.
- New case studies, from different application domains.

# References I


IBM ILOG CPLEX Optimization Studio.
*Version*, 12(1987-2018):1, 1987.

R. F. Araujo, H. F. Albuquerque, I. V. De Bessa, L. C. Cordeiro, and J. E. Chaves Filho.
Counterexample guided inductive optimization based on satisfiability modulo theories.
*Science of Computer Programming*, 165:3–23, 2018.

H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, et al.
CVC5: A versatile and industrial-strength SMT solver.
In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442. Springer, 2022.

N. Bjørner, A. Phan, and L. Fleckenstein.
$\nu$Z - an optimizing SMT solver.
In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 194–199, 2015.

# References II

R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich.
The OpenSMT solver.
In *Tools and Algorithms for the Construction and Analysis of Systems: 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings 16*, pages 150–153. Springer, 2010.

G. Chu, P. Stuckey, A. Schutt, T. Ehlers, G. Gange, and K. Francis.
Chuffed, a lazy clause generation solver.

L. de Moura and N. Bjørner.
Z3: An efficient SMT solver.
In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

B. Dutertre.
Yices 2 manual.
*Computer Science Laboratory, SRI International, Tech. Rep*, 2014.

# References III

M. Erașcu.
Fast and exact synthesis of application deployment plans using graph neural networks and satisfiability modulo theory.
In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2024.

K. Fazekas, F. Bacchus, and A. Biere.
Implicit hitting set algorithms for maximum satisfiability modulo theories.
In *Automated Reasoning: 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings 9*, pages 134–151. Springer, 2018.

G. Kovásznai, C. Biró, and B. Erdélyi.
Puli – a problem-specific OMT solver.
In *Proc. 16th International Workshop on Satisfiability Modulo Theories (SMT 2018)*, volume 371, 2018.

D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio.
Minimal-model-guided approaches to solving polynomial constraints and extensions.
In *International Conference on Theory and Applications of Satisfiability Testing*, pages 333–350. Springer, 2014.

# References IV

Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik.
Symbolic optimization with SMT solvers.
In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 607–618, 2014.

A. Nadel and V. Ryvchin.
Bit-vector optimization.
In *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings 22*, pages 851–867. Springer, 2016.

N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack.
MiniZinc: Towards a standard CP modelling language.
In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

L. Perron and V. Furnon.
OR-Tools.

# References V


C. Schulte, M. Lagerkvist, and G. Tack.
Gecode.
*Software download and online material at the website: http://www.gecode.org*, pages 11–13, 2006.

R. Sebastiani and P. Trentin.
OptiMathSAT: A tool for optimization modulo theories.
In D. Kroening and C. S. Păsăreanu, editors, *Computer Aided Verification*, pages 447–454, Cham, 2015. Springer International Publishing.

P. Trentin.
*Optimization Modulo Theories with OptiMathSAT*.
Phd thesis, University of Trento, May 2019.