Mathematical Programming: Turing-completeness and applications to code analysis

Leo Liberti

Joint work with F. Marinelli

IBM TJ Watson Research Center, Yorktown Heights, USA

LIX, École Polytechnique, France

Summary of Talk

- Benchmarking code
  - *finding difficult inputs (of given size) for given codes*
- Static analysis by abstract interpretation
  - *finding overapproximations of the sets of values taken by program variables during execution*

    (without actually executing the code)

$\Rightarrow$ **"code relaxations"**

Benchmarking code

The hardest input

Example: algorithm Mod(n, k): decides if $n \pmod{k} = 0$ for given n, k with $n > k$

```
1: input n, k ∈ ℕ
2: n ← n − k;
3: if n = 0 then
4:   return YES
5: else if n < 0 then
6:   return NO
7: else
8:   goto 2
9: end if
```

Find difficult instances for effective benchmarking

The hardest input

Example: algorithm Mod$_t$$(n, k)$: decides if $n \pmod k = 0$ for given n, k with $n > k$

1: input $n, k \in \mathbb{N}$; $t = 0$ (step counter)
2: $n \leftarrow n - k$; $t \leftarrow t + 2$
3: **if** $n = 0$ **then**
4: $t \leftarrow t + 2$; **return** YES
5: **else if** $n < 0$ **then**
6: $t \leftarrow t + 2$; **return** NO
7: **else**
8: $t \leftarrow t + 1$; goto 2
9: **end if**

Maximize t over varying input

Optimizing over executions

- C: a code
- $\mathcal{V}(C)$: set of values taken by the program variables during execution
- Formalize the following optimization problem:

$$\begin{array}{rrcl}
\max\limits_{n,k,t} & & t & \\
& t & \leq & T \\
& n & = & (n_0, \ldots, n_T) \\
& k & = & (k_0, \ldots, k_T) \\
& n, k & \in & \mathcal{V}(\mathsf{Mod}_t(n_0, k_0)) \\
& |n_0| + |k_0| & \leq & \textit{some given size}
\end{array}$$

where n_0, k_0 is the given input to the Mod_t code

Mathematical programming

- Translate $\mathcal{V}(\text{Mod}_t(n_0, k_0))$ to a set of constraints on n, k, t
  $=$ imperative $\rightarrow$ declarative language
- Use Mathematical Programming (MP)

Fundamental question:

Can we translate *any* code to MP?

Turing completeness

- Notation:
  - P: a MP formulation
  - $G(P)$: set of global optima of P
- Let C be a code for a Universal Turing Machine (UTM)

$$\boxed{\exists P \in \text{MP} \quad x \in G(P) \Leftrightarrow x \in \mathcal{V}(C) \text{ ?}}$$

- In other words, is MP a *Turing complete* language?

Turing completeness

- Notation:
  - P: a MP formulation
  - $G(P)$: set of global optima of P
- Let C be a code for a Universal Turing Machine (UTM)

$$\boxed{\exists P \in \text{MP} \quad x \in G(P) \Leftrightarrow x \in \mathcal{V}(C)\ ?}$$

- In other words, is MP a *Turing complete* language?

YES

- Universal Diophantine Equations (UDE):
  Negative answer to Hilbert's 10th problem
- Cook's theorem: *SAT is* ***NP****-complete*

Universal Diophantine Eqns

- $X \subseteq \mathbb{N}$ *recursively enumerable*
  $\triangleq \exists$ algorithm for listing all members of X
  $\Rightarrow \exists$ algorithm which terminates iff $x \in X$

Universal Diophantine Eqns

- $X \subseteq \mathbb{N}$ *recursively enumerable*
  $\triangleq \exists$ algorithm for listing all members of X
  $\Rightarrow \exists$ algorithm which terminates iff $x \in X$
- *there are countably many r.e. sets*
  $W_n = \{x \in \mathbb{N} \mid \mathsf{TM}_n(x) \downarrow\}$

Universal Diophantine Eqns

- $X \subseteq \mathbb{N}$ *recursively enumerable*
  $\triangleq \exists$ algorithm for listing all members of X
  $\Rightarrow \exists$ algorithm which terminates iff $x \in X$
- *there are countably many r.e. sets*
  $W_n = \{x \in \mathbb{N} \mid \mathsf{TM}_n(x) \downarrow\}$
- *Imperative* $\Leftrightarrow$ *declarative: integer roots of polys in* $\mathbb{Z}^{<\omega}[a]$

$W_n = \{x \in \mathbb{N} \mid x \text{ is composite}\} \Leftrightarrow \exists a_1, a_2 \in \mathbb{N}\ (a_1 + 2)(a_2 + 2) = x$

Universal Diophantine Eqns

- $X \subseteq \mathbb{N}$ *recursively enumerable*
  $\triangleq \exists$ algorithm for listing all members of X
  $\Rightarrow \exists$ algorithm which terminates iff $x \in X$
- *there are countably many r.e. sets*
  $W_n = \{x \in \mathbb{N} \mid \mathsf{TM}_n(x) \downarrow\}$
- *Imperative* $\Leftrightarrow$ *declarative: integer roots of polys in* $\mathbb{Z}^{<\omega}[a]$
  $W_n = \{x \in \mathbb{N} \mid x \text{ is composite}\} \Leftrightarrow \exists a_1, a_2 \in \mathbb{N}\ (a_1 + 2)(a_2 + 2) = x$
- Thm. [Davis, Matiyasevich, Putnam, Robinson]:
  $\exists$ **one** polynomial encoding **every** r.e. set

Universal Diophantine Eqns

- $X \subseteq \mathbb{N}$ *recursively enumerable*
  $\triangleq \exists$ algorithm for listing all members of X
  $\Rightarrow \exists$ algorithm which terminates iff $x \in X$
- *there are countably many r.e. sets*
  $W_n = \{x \in \mathbb{N} \mid \mathsf{TM}_n(x) \downarrow\}$
- *Imperative* $\Leftrightarrow$ *declarative: integer roots of polys in* $\mathbb{Z}^{<\omega}[a]$
  $W_n = \{x \in \mathbb{N} \mid x \text{ is composite}\} \Leftrightarrow \exists a_1, a_2 \in \mathbb{N}\ (a_1+2)(a_2+2) = x$
- Thm. [Davis, Matiyasevich, Putnam, Robinson]:
  $\exists$ **one** polynomial encoding **every** r.e. set
- UDE: $\exists\ p(n, x, y_1, \ldots, y_t) \in \mathbb{Z}[n, x, y]$ s.t.

$$\forall n \in \mathbb{N}, x \in W_n \quad \Leftrightarrow \quad \exists y \in \mathbb{N}^t \quad p(n, x, y) = 0$$

Universal Diophantine Eqns

- $X \subseteq \mathbb{N}$ *recursively enumerable*
  $\triangleq \exists$ algorithm for listing all members of X
  $\Rightarrow \exists$ algorithm which terminates iff $x \in X$
- *there are countably many r.e. sets*
  $W_n = \{x \in \mathbb{N} \mid \mathsf{TM}_n(x) \downarrow\}$
- *Imperative* $\Leftrightarrow$ *declarative: integer roots of polys in* $\mathbb{Z}^{<\omega}[a]$
  $W_n = \{x \in \mathbb{N} \mid x \text{ is composite}\} \Leftrightarrow \exists a_1, a_2 \in \mathbb{N}\ (a_1 + 2)(a_2 + 2) = x$
- Thm. [Davis, Matiyasevich, Putnam, Robinson]:
  $\exists$ **one** polynomial encoding **every** r.e. set
- UDE: $\exists\ p(n, x, y_1, \ldots, y_t) \in \mathbb{Z}[n, x, y]$ s.t.
  $$\forall n \in \mathbb{N}, x \in W_n \quad \Leftrightarrow \quad \exists y \in \mathbb{N}^t \quad p(n, x, y) = 0$$
- $\Rightarrow \exists$ (polynomial integer) MP encoding every TM

Cook's theorem

- Cook's theorem: a reduction
  [nondeterministic polytime bounded UTM] $\to$ SAT
- We need UTM $\to$ MP, we have SAT $\to$ MP
- UTM $\to$ SAT: generalized Cook's reduction
- Remove boundedness: get an infinite SAT

Practical computation?

- UDE:
  - tradeoff between #vars and degree
  - large coefficients
- SAT:
  - SAT solver: no objective
    (we need it to optimize t)
  - MP solver: high degree polynomials
    (boolean "and" $\Leftrightarrow$ product of binary variables)

MP is Turing complete
A new proof

Approach

- A simple universal register machine
- Register $=$ program variable
- *Imperative* $\rightarrow$ *declarative*: use MP constraints
- r_{jt} $=$ value of register j at iteration t
- Objective function maximizes number of steps

Minsky's Register Machine

- Infinitely many registers R_j
- Each R_j holds an arbitrary natural number
- Two types b of instructions: given j,
  1. $b = 0$: increase R_j, go to instruction $c = k$
  2. $b = 1$: if $R_j > 0$ decrease R_j, go to $c = k$; else go to $c = \ell$
  3. **Thm.**: the MRM is a Universal Turing Machine
- Each instruction is a quadruplet (j, b, k, ℓ)

MRM Example

Problem: Given $n \geq k \in \mathbb{N}$, is $n \pmod{k} = 0$?

Algorithm: Mod$_t(n, k)$

1: $n \leftarrow n - k$
2: **if** $n = 0$ **then return** YES
3: **else if** $n < 0$ **then return** NO
4: **else** go to 1

$R_1 = n$, $R_2 = k$, $R_3 = k'$ (k backup), $R_4 = a$ (output: 1 iff $k|n$)

input $(n, k, 0, 0)$

Line	(j, b, k, ℓ)				**Meaning**	**Comment**
0	-	-	-	-	stop	
1	2	1	2	4	if $k > 0$ decrease k and goto *2*, else *4*	start here
2	3	0	3	0	increase k' and goto *3*	invariant: $k + k'$
3	1	1	1	0	if $n > 0$ decrease n and goto *1*, else *0*	$n = 0$ before $k \Rightarrow k \nmid n$
4	1	1	5	8	if $n > 0$ decrease n and goto *5*, else *8*	$n, k = 0$: $\Rightarrow k|n$
5	1	0	6	0	increase n and goto *6*	
6	3	1	7	1	if $k' > 0$ decrease k' and goto *7*, else *1*	restore k using k'
7	2	0	6	0	increase k and goto *6*	
8	4	0	0	0	increase a and goto *0*	set $a = 1$

MP (essentially)

$$\begin{aligned} b = 0 \quad &\Rightarrow \quad (R_j = R_j + 1) \wedge (c = k) \\ (b = 1) \wedge R_j = 0 \quad &\Rightarrow \quad c = \ell \\ (b = 1) \wedge R_j > 0 \quad &\Rightarrow \quad (R_j = R_j - 1) \wedge (c = k) \end{aligned}$$

Encode by means of decision vars and constraints

- infinite number of variables and constraints
- polynomials of degree ≤ 3 (some trilinear terms)
- if bounded, can be reformulated to finite MILP
- integer linear feasibility problem
- can be solved with CPLEX in practice
- *Yet another* **Thm**: MP is Turing complete

Some details

- Decision variables
  - $r_{jt} \in \mathbb{N}_+$: content of register j at time t
  - $\rho_{jt} \in \{0, 1\}$: 1 iff $R_j = 0$ at time t
  - $x_{it} \in \{0, 1\}$: 1 iff $c = i$ at time t
- Examples of constraints:
  - if $c = i$, $b = 0$, set $R_j = R_j + 1$:
    $\forall t, i \quad x_{i,t-1}(1 - b)r_{jt} = x_{i,t-1}(1 - b)(r_{j,t-1} + 1)$
  - if $c = i$, $b = 1$ and $R_j > 0$, set $c = k$:
    $\forall t, i \quad x_{i,t-1}b\rho_{j,t-1}x_{kt} = x_{i,t-1}b\rho_{j,t-1}$
  - if $c = 0$, stop
    $\forall t \quad x_{0t}x_{0,t-1} = x_{0,t-1}$

Correctness proof: by induction on t

Back to benchmarking

- input values for registers: **decision variables**
- $x_{it} = 1$ iff instruction i executed at time t
- instruction 0: **stop** (by convention)
- minimize $\sum_{t \in \mathbb{N}} x_{0t}$

Sanity check

$\mathsf{Mod}_t(n, k)$ yields $k = 1$ for all tested values of n

Issue: MRM too simple to run meaningful codes

Code relaxations

The trace of a code

- [Böhm & Jacopini '66]: language Turing-complete if it has:
  1. Tests
  2. Loops
  3. Juxtaposition of commands

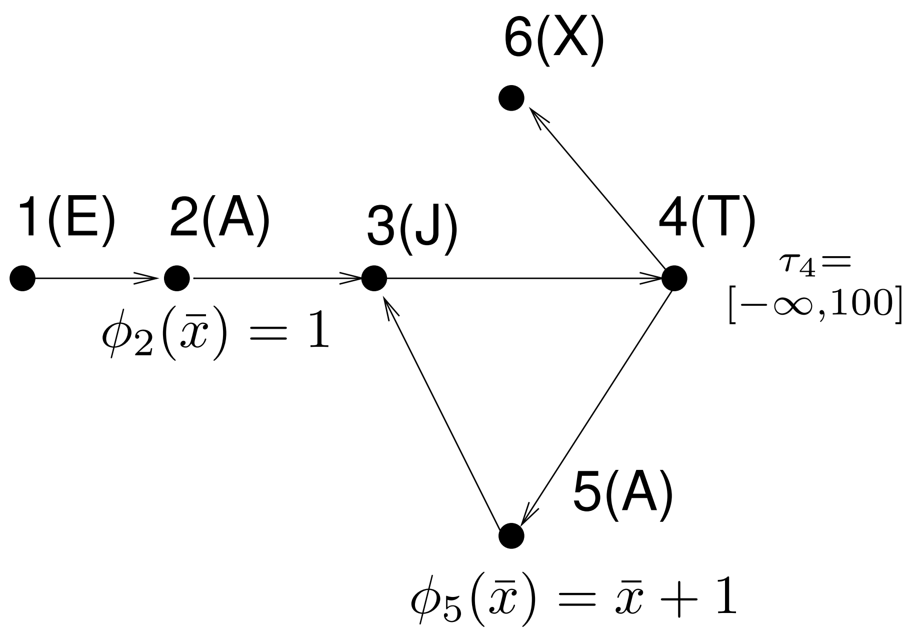
- $\mathscr{L}$: a Turing complete language
- $C(x)$: a program code in $\mathscr{L}$ involving variables $x = (x_1, \ldots, x_n)$
- $\forall i \leq n$, $\bar{X}_i$ sequence of values taken by x_i during execution of C
- $\bar{X} = (\bar{X}_1, \ldots, \bar{X}_n)$: the **trace** of $C(x)$
- The trace is not computable

Semantics

- Relaxation of Turing completeness: *remove Property 3*
- $\forall i$ look at $\hat{X}_i =$ set of values occurring in $\bar{X}_i$
- Concrete semantics $\hat{X} = (\hat{X}_1, \ldots, \hat{X}_n)$: not computable
- X: relaxation(=overapproximation) of $\hat{X}$
- E.g. $X \in$ boxes, polyhedra, etc.
- For some set classes, X is computable
- **Abstract semantics**: assignment of X to x

Flowgraph of a code

```
//(1)
int x = 1; //(2)
//(3)
while(x <= 100){ //(4)
  x = x + 1; //(5)
}
//(6)
```



E=entry, X=exit, A=assignment, J=join, T=test

Semantic eqns. of a flowgraph

$$
\begin{aligned}
X_{11} &= \mathsf{Id}(\texttt{input}) \\
X_{21} &= \phi_2(X_{11}) \\
X_{31} &= X_{21} \cup X_{61} \\
X_{41} &= X_{31} \cap \tau_4 \\
X_{51} &= \phi_5(X_{41}) \\
X_{61} &= X_{31} \cap (X \setminus \tau_4)
\end{aligned}
$$

In the abstract semantics:

$$
\begin{aligned}
X_{11} &= [-\infty, \infty] \\
X_{21} &= [1, 1] \\
X_{31} &= [1, 1] \cup X_{61} \\
X_{41} &= X_{31} \cap [-\infty, 100] \\
X_{51} &= X_{41} + [1, 1] \\
X_{61} &= X_{31} \cap [101, \infty]
\end{aligned}
$$

Fixed points

- $\mathcal{C}$: an abstract domain (intervals, polyhedra, etc.)
- Semantic equations: $\forall i \leq m, j \leq n \quad X_{ij} = F_{ij}(X)$

$$\Rightarrow \quad X = F(X)$$

- Solutions in $\mathcal{C}$ are called *fixed points* (FP)
- If X is a FP, then it is invariant w.r.t. F
  - $\Rightarrow$ action of the code $C(\cdot)$ on X does not change it
  - $\Rightarrow X \supseteq \hat{X}$
- Tightest relaxation: **least FP** (LFP) w.r.t. set inclusion

Example

$$\begin{aligned} X_{11} &= [-\infty, \infty] \\ X_{21} &= [1, 1] \\ X_{31} &= [1, 1] \cup X_{61} \\ X_{41} &= X_{31} \cap [-\infty, 100] \\ X_{51} &= X_{31} \cap [101, \infty] \\ X_{61} &= X_{41} + [1, 1]. \end{aligned}$$

$$\begin{array}{lll} X_{11} = [-\infty, \infty] & X_{21} = [1, 1] & X_{31} = [1, 101] \\ X_{41} = [1, 100] & X_{51} = [101, 101] & X_{61} = [2, 101] \end{array}$$

Debugging

- Suppose x is an index for an array y
- Suppose y has > 100 allocated memory cells
- LFP $X_{41} = [1, 100] \Rightarrow$ proves that $\nexists$ memory overflow due to x
- Automated debugging :
  fundamental in critical system codes, e.g.:
  - Ariane rockets
  - A380

MP for semantic eqns

- Fix a particular abstract domain, e.g. intervals
- $X = F(X)$: system of interval equations
- Look for LFP: $\inf_{\subseteq}\{X \mid X \supseteq F(X)\}$
- Model using MP

Decision variables

- For each instruction $i \leq m$ and variable x_j with $j \leq n$:
- Consider an interval $X_{ij} = [x^L_{ij}, x^U_{ij}]$
- Let $\bar{x}_{ij} = 1$ iff $X_{ij} = \varnothing$
- Represent X_{ij} by triplet $(x^L_{ij}, x^U_{ij}, \bar{x}_{ij})$

Objective function

- Define an interval width $|X_{ij}| = \bar{x}_{ij}(x^U_{ij} - x^L_{ij}) + \log \bar{x}_{ij}$
- If interval $\neq \varnothing$, $|\cdot|$ gives the interval length
- If interval $= \varnothing$, $|\cdot|$ unbounded below ($-\infty$)
- Extend to $|X| = \sum\limits_{\substack{i \le m \\ j \le n}} |X_{ij}|$
- **Lemma**: $|\cdot|$ monotonic with interval inclusion lattice
- Objective function: $\min |X|$

The interval MP

- MP constraints to model the interval semantics of:
  1. **Assignments**
     - constant assignment, identity assignment
     - positive and negative constant products
     - positive odd and even powers
     - general sum and products
     - also attempt to model division
  2. **Loops** (by means of interval unions)
  3. **Tests** (by means of interval intersections)
- Get a MINLP *(products, exps, binary, integer, continuous vars)*
- Solution? **Forget it!** But theoretical interest

Example of constraints

- Consistency: $\forall i \le m, j \le n \quad x^L_{ij} \le x^U_{ij}$
- Sum: $\forall i, k, \ell \le m$ and $j, h, f \le n$

$$\begin{aligned} \bar{x}_{ij} &= \bar{x}_{kh}\bar{x}_{\ell f} \\ \bar{x}_{ij} \to x^L_{ij} &\le x^L_{kh} + x^L_{\ell f} \\ \bar{x}_{ij} \to x^U_{ij} &\ge x^U_{kh} + x^U_{\ell f}. \end{aligned}$$

- Union: $\forall i, k, \ell \le m$ and $j, h, f \le n$

$$\begin{aligned} (1 - \bar{x}_{ij}) &= (1 - \bar{x}_{kh})(1 - \bar{x}_{\ell f}) \\ \bar{x}_{ij} \to x^L_{ij} &\le \min(x^L_{kh}, x^L_{\ell f}) \\ \bar{x}_{ij} \to x^U_{ij} &\ge \max(x^U_{kh}, x^U_{\ell f}) \end{aligned}$$

Properties of the MP

- P: interval MINLP modelling the LFP X^* of $X = F(X)$
- **Thm**: P is feasible and bounded iff X^* is non-empty and bounded
- **Thm**: P is feasible and unbounded iff X^* is empty and bounded (empty box with bounded interval components)
- **Thm**: P is infeasible iff X^* is unbounded

Solving $P \Leftrightarrow$ Determining X^*

Bounded codes

- General MINLP : for <u>*universal computation*</u>
- Given two additional assumptions:
  1. X^* is bounded ($=$ execution terminates)
  2. Solution to P approximates X^* to a given $\epsilon > 0$
- $\Rightarrow \quad \exists$ MILP approximation
  (lots of equality constraints and big Ms)
- Solves reasonably fast

Computational results

Using MILP approximation with $\top = [-M, M] = [-5000, 5000]$

Instance			MP				PI			
Instance	*Lines*	*Vars*	CPU	$\lvert\cdot\rvert$	$\lvert\top\rvert$	$\lvert\neg\top\rvert$	CPU	$\lvert\cdot\rvert$	$\lvert\top\rvert$	$\lvert\neg\top\rvert$
`short_31`	32	3	0.008	250002	**25**	**2**	0	250023	25	23
`short_32`	20	3	0.02	270077	27	77	0	270077	27	77
`short_35`	22	3	0.02	32028	3	2028	0	32028	3	2028
`short_37`	25	3	0.008	420000	**42**	**0**	0	470000	47	0
`short_38`	35	3	0.12	34501	3	4501	0	34501	3	4501
`long_1`	213	4	0.768	90000	**9**	**0**	0.004	90052	9	52
`long_2`	217	4	0.916	80000	**8**	**0**	0.008	90002	9	2
`long_3`	130	4	0.64	120426	**12**	**426**	0.06	4.36e+06	436	246
`long_4`	195	4	0.412	120000	**12**	**0**	0.008	120002	12	2
`long_5`	216	4	0.772	110000	**11**	**0**	0.004	120010	12	10
`arrays`	22	6	0.04	300139	30	139	-	-	-	-
`fun_arrays`	53	6	0.016	30000	3	0	-	-	-	-
`functions`	62	7	0.112	101190	10	1190	-	-	-	-
`subway`	62	34	9.25258	1.77e+07	1766	675	-	-	-	-

WARNING: Comparison against research code, not commercial codes

It would be nice to

- **Benchmarking**: model C instead of MRM
- Answer more questions

  E.g. "best nontrivial algorithm for given input/output"
- **Code relaxations**: integrate black-boxes
  (for the analysis of complex codes)

References:

1. [L. et al. ENDM 2010]
2. [Goubault et al., ENTCS 2010]
3. [L. and Marinelli, JOCO, OnlineFirst]