Microsoft

Supercharging Virtual Plant Configurations using Z3

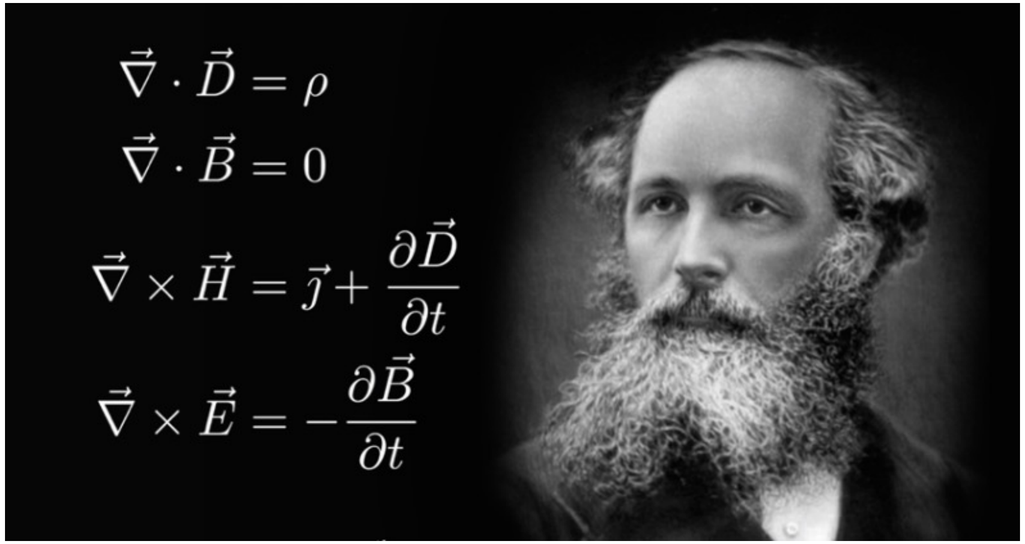
Nikolaj Bjørner, Max Levatich,
Nuno Lopes, Andrey Rybalchenko,
Chandra Vuppalapati

Microsoft

1 | Introduction and overview of Z3

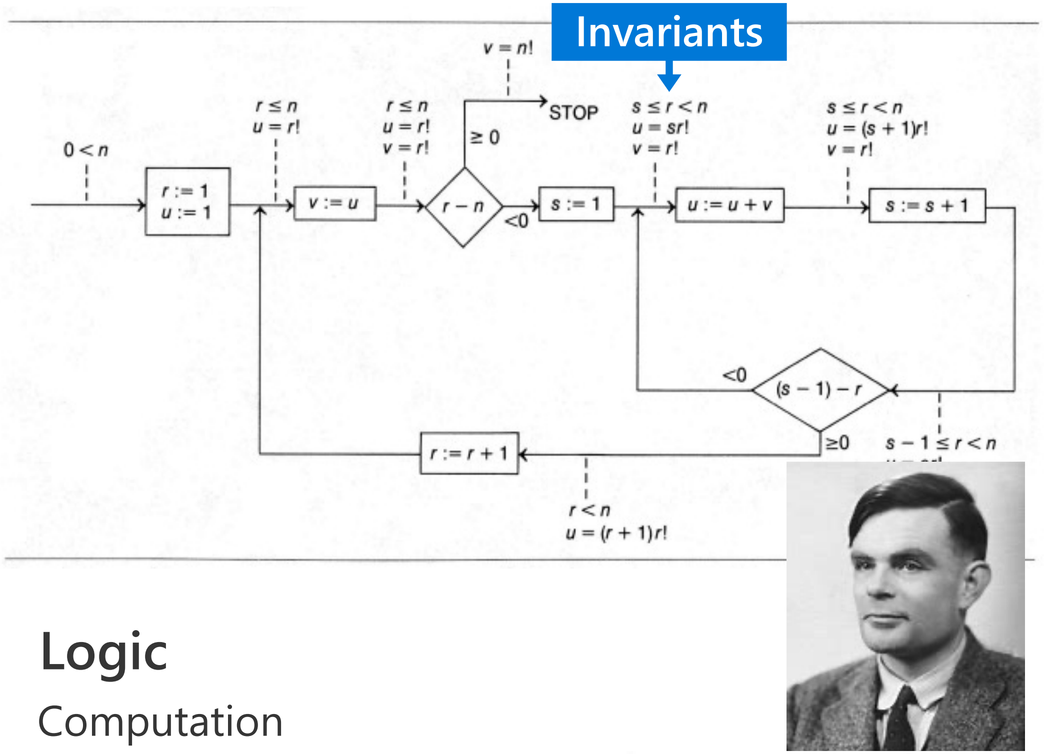
2 | Z3 for virtual plant design automation

Logic: The Calculus of Computation



Differential, Integral Calculus

Dynamics, Conduction,..
Matlab, Mathemetica, Simulink



Friday, 24th June [1949]
Checking a large routine by Dr A. Turing.

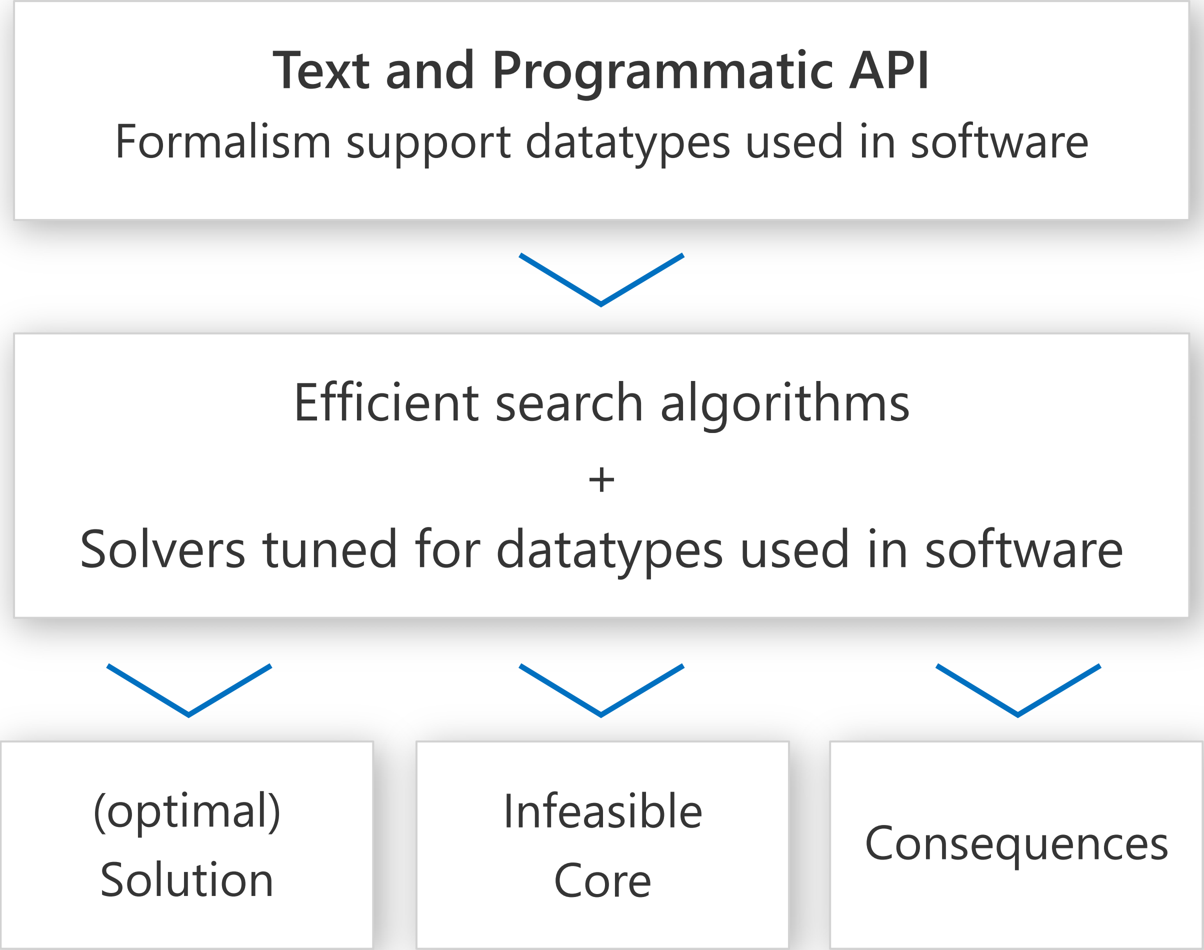
Logic

Computation

Z3

Claim: Practically all modern program analysis tools involve solving logical formulas

Z3 Efficient Solver for Symbolic Logic



Text and Programmatic API

Formalism support datatypes used in software

Efficient search algorithms

\+

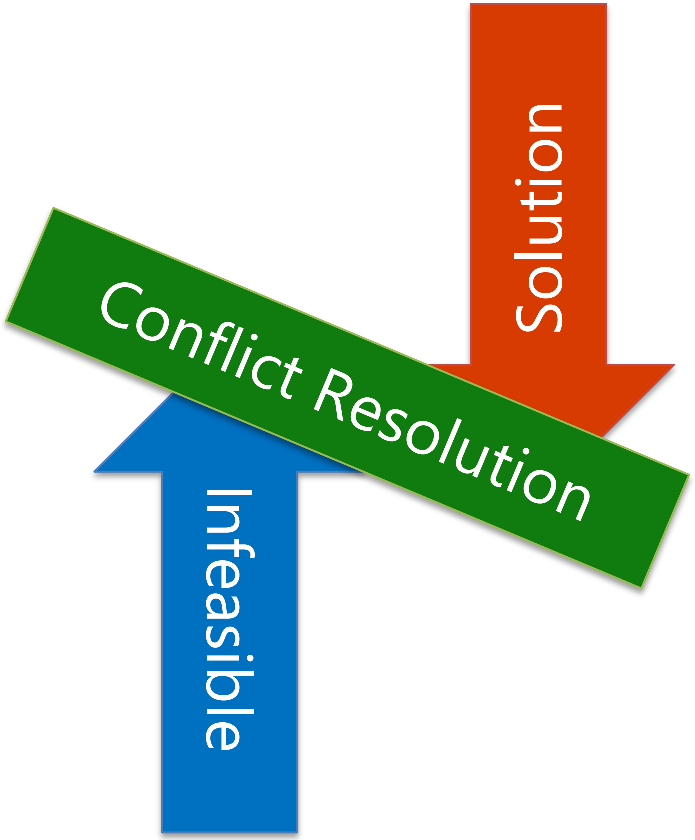
Solvers tuned for datatypes used in software

(optimal)
Solution

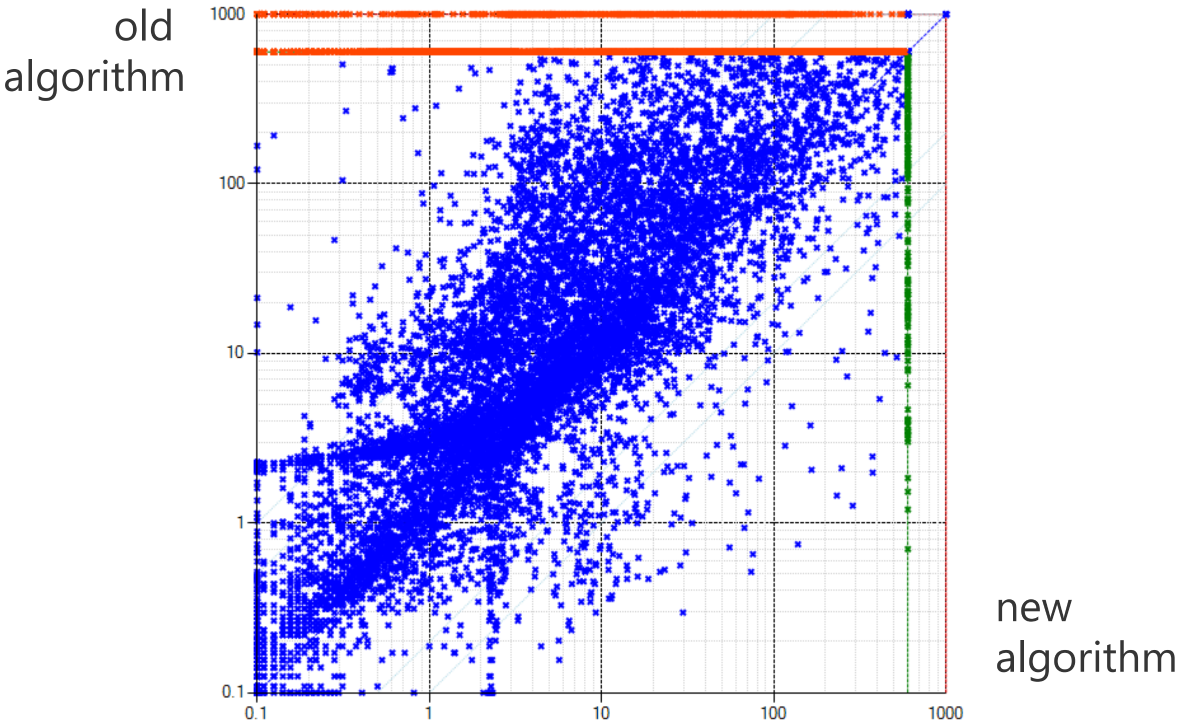
Infeasible
Core

Consequences

Symbolic Solving: Foundations and Engineering

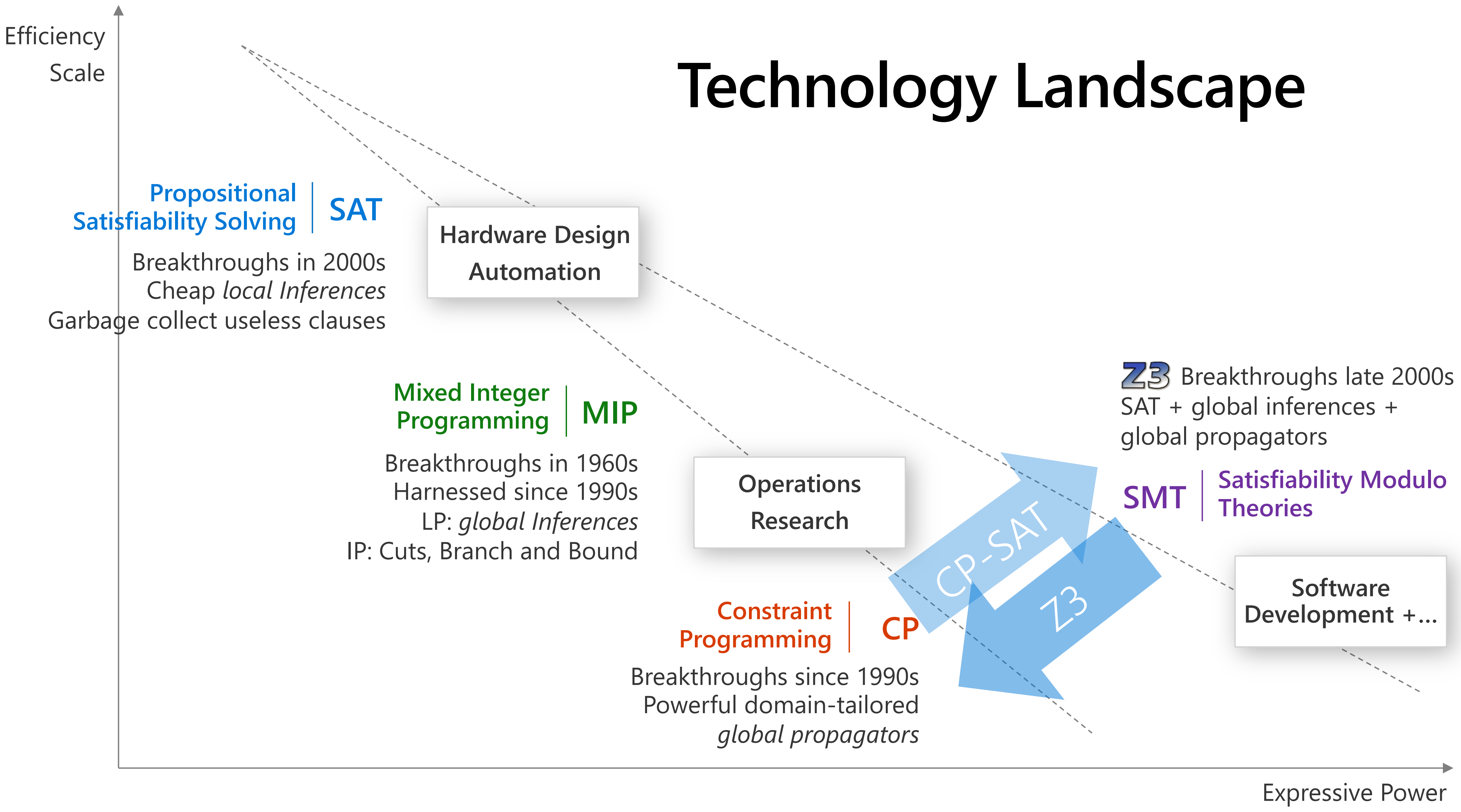


Core solvers in Z3 founded on duality between solution witness / infeasibility proof



Advances guided by application domains and evaluated by extensive benchmarking

Technology Landscape



Propositional Satisfiability Solving | SAT

Breakthroughs in 2000s
Cheap *local Inferences*
Garbage collect useless clauses

Hardware Design Automation

Mixed Integer Programming | MIP

Breakthroughs in 1960s
Harnessed since 1990s
LP: *global Inferences*
IP: Cuts, Branch and Bound

Operations Research

Constraint Programming | CP

Breakthroughs since 1990s
Powerful domain-tailored
global propagators

Z3 Breakthroughs late 2000s
SAT + global inferences +
global propagators

SMT | Satisfiability Modulo Theories

Software Development +…

SMT Solvers – Main Services

Support domains that are natural in Software and Hardware analysis

- Make it easy to translate program assertions into SMT

Holy grail of SMT: modularity + efficiency

- Combine disjoint Theory Solvers by reconciling equalities between shared variables

Quantifier-Free First Order Theories

- Int, Real, Bit-vectors, IEEE floating point numbers, arrays, algebraic data-types

Quantified First-Order, Higher-Order Logics

- As aid to proof assistants

Base theory: Uninterpreted Functions

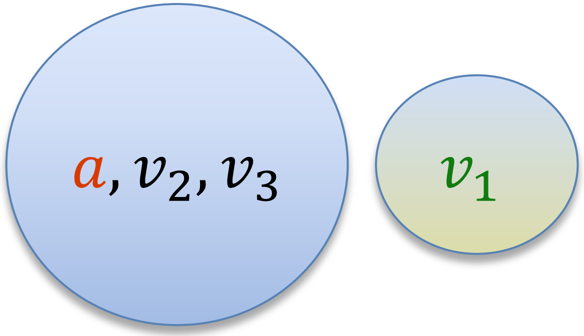
$a = f(f(a)), \qquad a = f(f(f(a))), \qquad a \neq f(a)$

- Produce Proofs
- Incremental Updates
- Propagate Literals

$a = v_2, a = v_3, a \neq v_1,$

$v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2)$

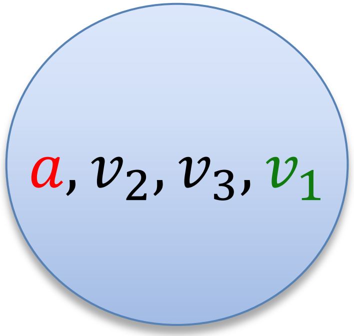
Step 1: Equivalence classes from equalities



Union Find

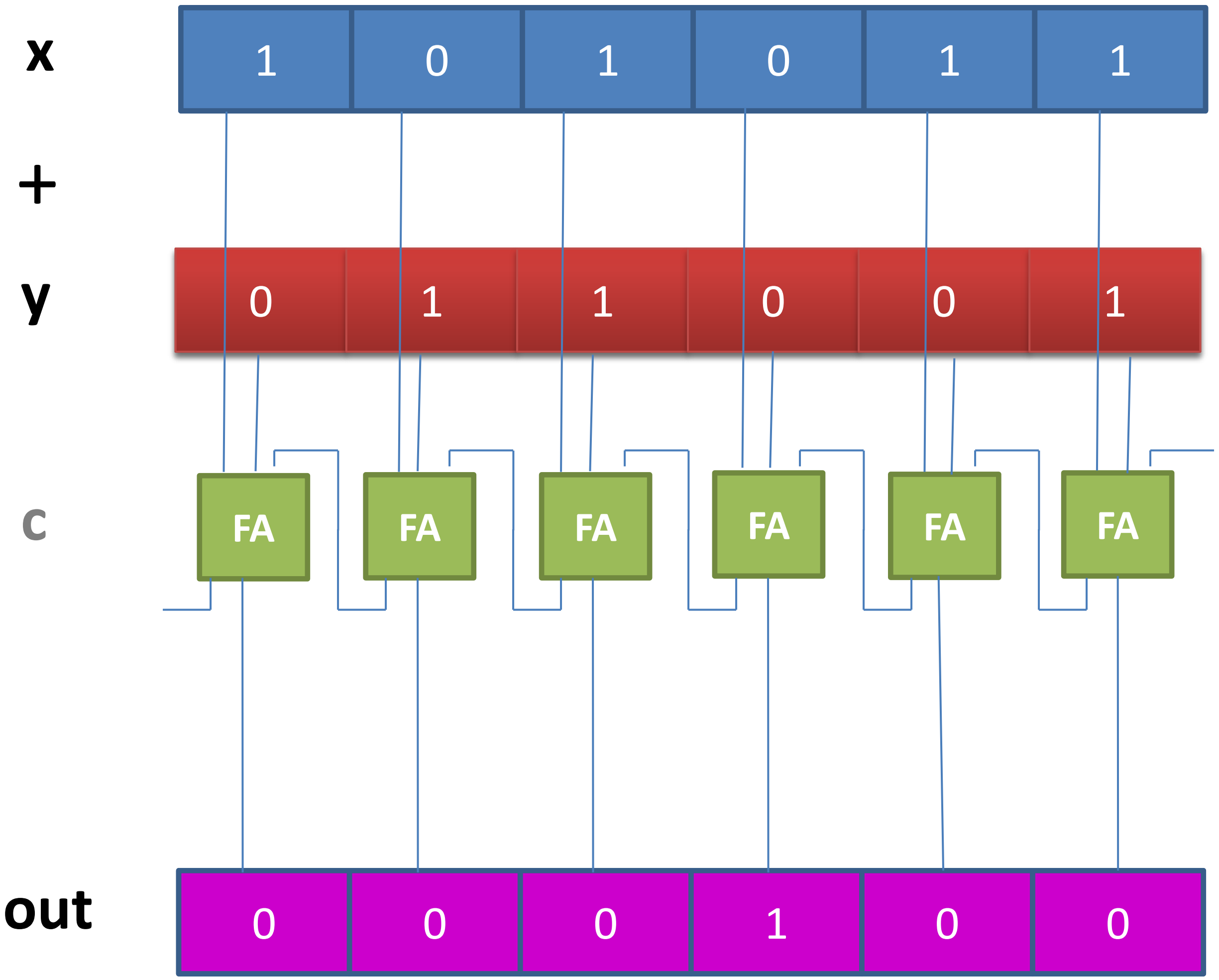
Step 2: Apply Congruence Rule:

$a \simeq v_2$ implies $f(a) \simeq f(v_2)$: $\quad v_1 \simeq v_3$



E-graph

Compiled into SAT: Bit-vectors



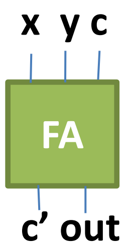
Bit-vector addition is expressible using bit-wise operations and bit-vector equalities.

$$\mathbf{out} = \mathrm{xor}(\mathbf{x}, \mathbf{y}, \mathbf{c})$$
$$\mathbf{c'} = (\mathbf{x} \wedge \mathbf{y}) \vee (\mathbf{x} \wedge \mathbf{c}) \vee (\mathbf{y} \wedge \mathbf{c})$$
$$\mathbf{c[0]} = \mathbf{0}$$
$$\mathbf{c'[N\text{-}2{:}0]} = \mathbf{c[N\text{-}1{:}1]}$$

Benefits:
- Efficient finite domain reasoning

Limitations:
- Not suitable for heavy use of linear arithmetic
- Bit-vector multiplication is super expensive

Note:



$$\mathbf{out} \leftrightarrow \mathrm{xor}(\mathbf{x}, \mathbf{y}, \mathbf{c})$$
$$\mathbf{c'} \leftrightarrow (\mathbf{x} \wedge \mathbf{y}) \vee (\mathbf{x} \wedge \mathbf{c}) \vee (\mathbf{y} \wedge \mathbf{c})$$

Combining Theories in the age of CDCL(T)

Foundations

1979 Nelson, Oppen: Framework

1996 Tinelli & Harindi: N.O Fix

2000 Barrett et al: N.O + Rewriting

2002 Zarba & Manna: “Nice” Theories

2004 Ghilardi et al: N.O. Generalized

Efficiency using rewriting

1984 Shostak: Theory solvers

1996 Cyrluk et al: Shostak Fix #1

1998 B: Shostak with Constraints

2001 Rueß & Shankar: Shostak Fix #2

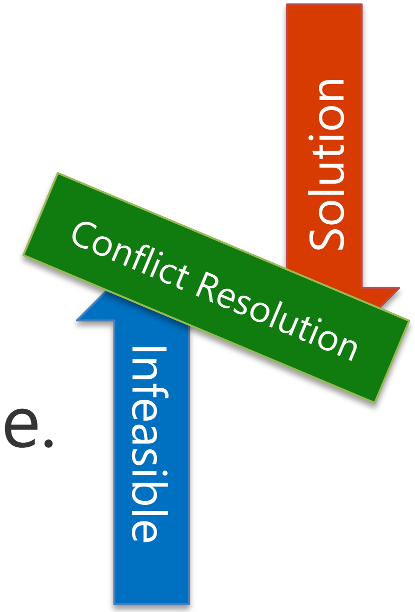
2004 Ranise et al: N.O + Superposition

1998 de Silva, Sakallah; 2001 Moskewicz et al: DPLL → CDCL made guessing cheap

2006 Bruttomesso et al: Delayed Theory Combination

2007 de Moura & B: Model-based Theory Combination

Model-based theory combination



Pre-existing methods

- Propagate all implied equalities – complicated costly.
- Delayed theory combination – $O(n^2)$ equalities, “2^{n^2}” time.

Model-based theory combination

- Each theory constructs a candidate model.
- Propagate all equalities implied by candidate model, hedging that other theories will agree.
- If not, use backtracking to fix the model.

Propagating Equalities

Equality inferences require addition/subtraction operations

Asserted inequalities

$x + u \le z \qquad z - 1 \le y \qquad y \le x \qquad 1 \le u \le 1$

$$\frac{x + 1 = y \qquad y - 1 = z}{x = z}$$

How the solver sees the constraints

$x + u + s_1 = z \qquad z - 1 + s_2 = y \qquad y + s_3 = x \qquad 1 \le u \le 1 \qquad 0 \le s_1 \qquad 0 \le s_2 \qquad 0 \le s_3$

After pivoting

$x = z - u - s_1 \qquad y = z - 1 + s_2 \qquad s_3 = -s_2 - u + 1 - s_1 \qquad 1 \le u \le 1 \qquad 0 \le s_1 \qquad 0 \le s_2 \qquad 0 \le s_3$

After propagating bounds on s_1, s_2, s_3

$x = z - u - s_1 \qquad y = z - 1 + s_2 \qquad s_3 = -s_2 - u + 1 - s_1 \quad 1 \le u \le 1 \qquad 0 \le s_1 \le 0 \qquad 0 \le s_2 \le 0 \qquad 0 \le s_3 \le 0$

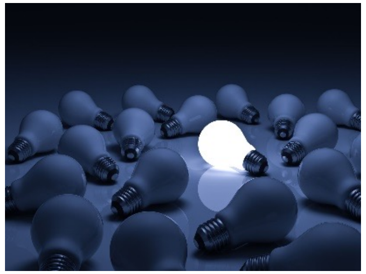
Subtract first two equalities to infer

$x = y$

Subtle complexity: Every row can have many fixed variables.
Adding values of constant bounds requires significant runtime.

Propagating Equalities - Efficiently

$$\frac{x = z - u \; - s_1 \qquad y = z \; - 1 + s_2 \qquad 1 \le u \le 1 \qquad 0 \le s_1 \le 0 \qquad 0 \le s_2 \le 0 \qquad 0 \le s_3 \le 0}{x = y}$$



Instead of adding up rows to *prove* implied equality

Use fact: all variables have values assigned by Simplex solver

Then two variables are equal if
- They are connected through offset equalities
- They have the same value

offset equality

$$x = y + a_1 z_1 + a_2 z_2 + \cdots$$

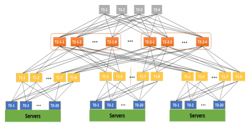
$$\overbrace{b_1 \le z_1 \le b_1, \; b_2 \le z_2 \le b_2,}$$

Example: If solver assigns $x = 3$ then $z = 4, y = 3$

x, y are connected (over $z - 1$).
x, y have the same value (3)

[Lev Nachmanson, B]

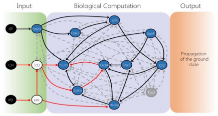
Z3 for Software +…



Azure Network
Verification

Verifying C
Compiler

Quantum
Compilation

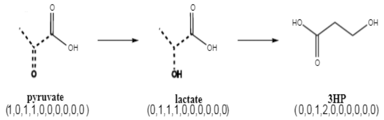


Biological
Computations

Verified Crypto
Libraries & Protocols

Dynamics
AX

SVACE
Static Analysis Engines

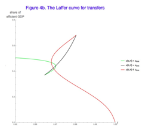


Artificial
Life

Security Risk
Detection

Smart Contract
Verification

ALIVE2
Translation Validation
for LLVM & Visual C++

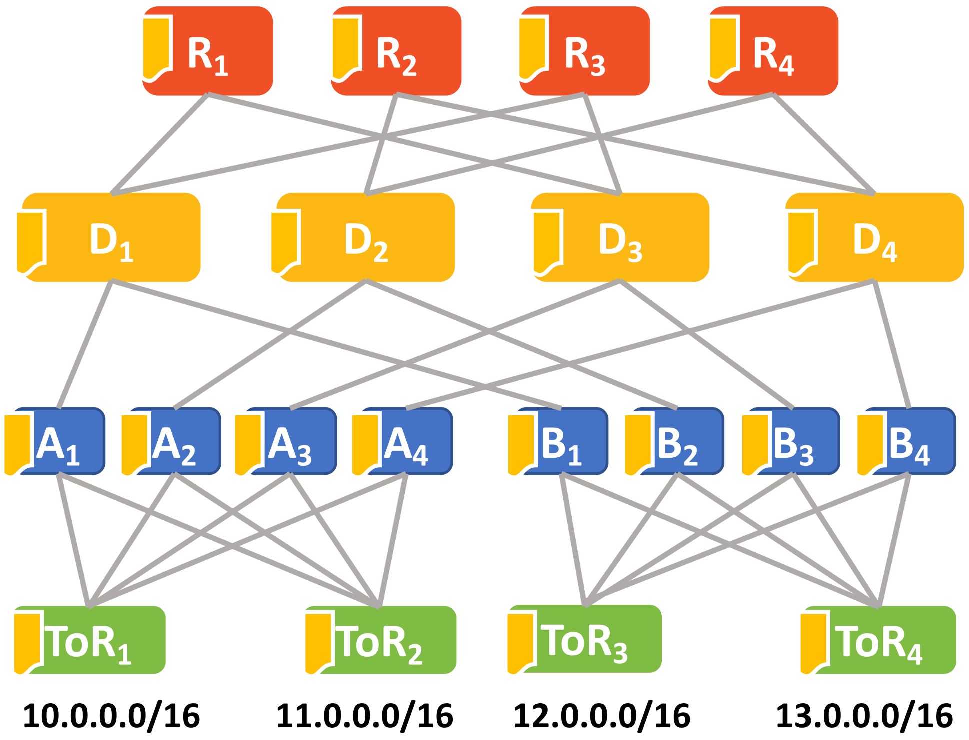


Axiomatic
Economics

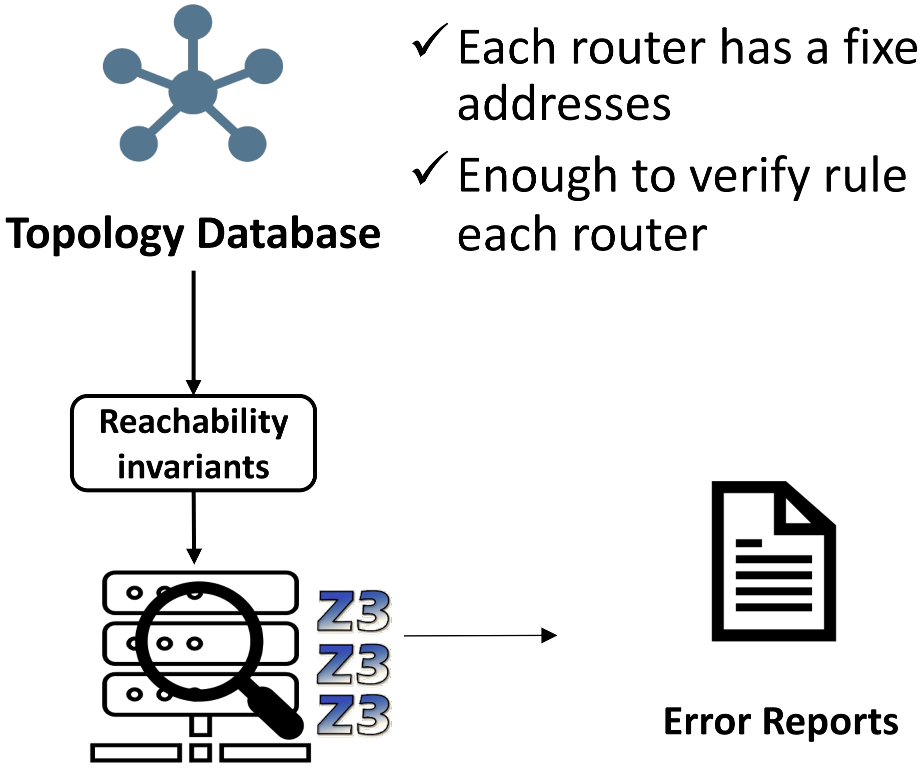
NFL
Scheduling

Live Monitoring of Forwarding Behavior

Global reachability as **local contracts**



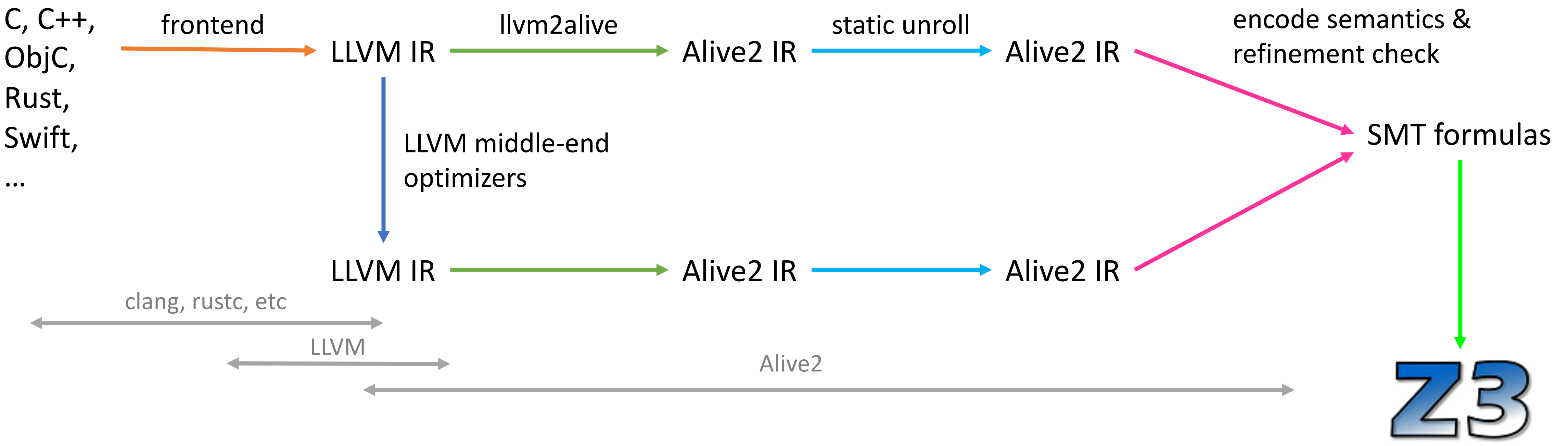
- ✓ Each router has a fixed rule for a set of addresses
- ✓ Enough to verify rule is enforced on each router



5 Billion Z3 queries per day

[Jayaraman et al, Sigcomm 2019]

Alive2: Integration with LLVM



C, C++,
ObjC,
Rust,
Swift,
…

frontend

LLVM IR

llvm2alive

Alive2 IR

static unroll

Alive2 IR

encode semantics &
refinement check

SMT formulas

LLVM middle-end
optimizers

LLVM IR

Alive2 IR

Alive2 IR

UNSAT
or **partial** model

clang w/ alive2 plugin:

```
$ alivecc file.c
```

opt plugin:

```
$ opt -tv -instcombine -tv file.ll
```

[Nuno Lopes, Juneyoung Lee, Chung-Kil Hur, Zhengyang Liu, John Regehr, PLDI 2021]

Tools and internals developed in a feedback loop

SecGuru

Dafny

FORMULA

Modeling Foundations.

F

Vcc

Yogi

SLAM

SAGE

Pex

TERMINATOR

SLAYER

HAVOC

BOOGIE

Tools

SLS, floats

νZ: Opt+MaxSMT

μZ: Datalog

Generalized PDR

Existential Reals

Model Constructing SAT

CutSAT: Linear Integer Formulas

Quantified Bit-Vectors

Z3 Internals

Linear Quantifier Elimination

Model Based Quantifier Instantiation

Generalized, Efficient Array Decision Procedures

Engineering DPLL(T) + Saturation

Effectively Propositional Logic

Model-based Theory Combination

Relevancy Propagation

Efficient E-matching for SMT solvers

Z3 for Virtual Plant Design Automation

An ongoing collaboration

Solving Virtual Plants (in a nutshell)

Solve for:

- Assign every task to a station and an operator

Subject to:

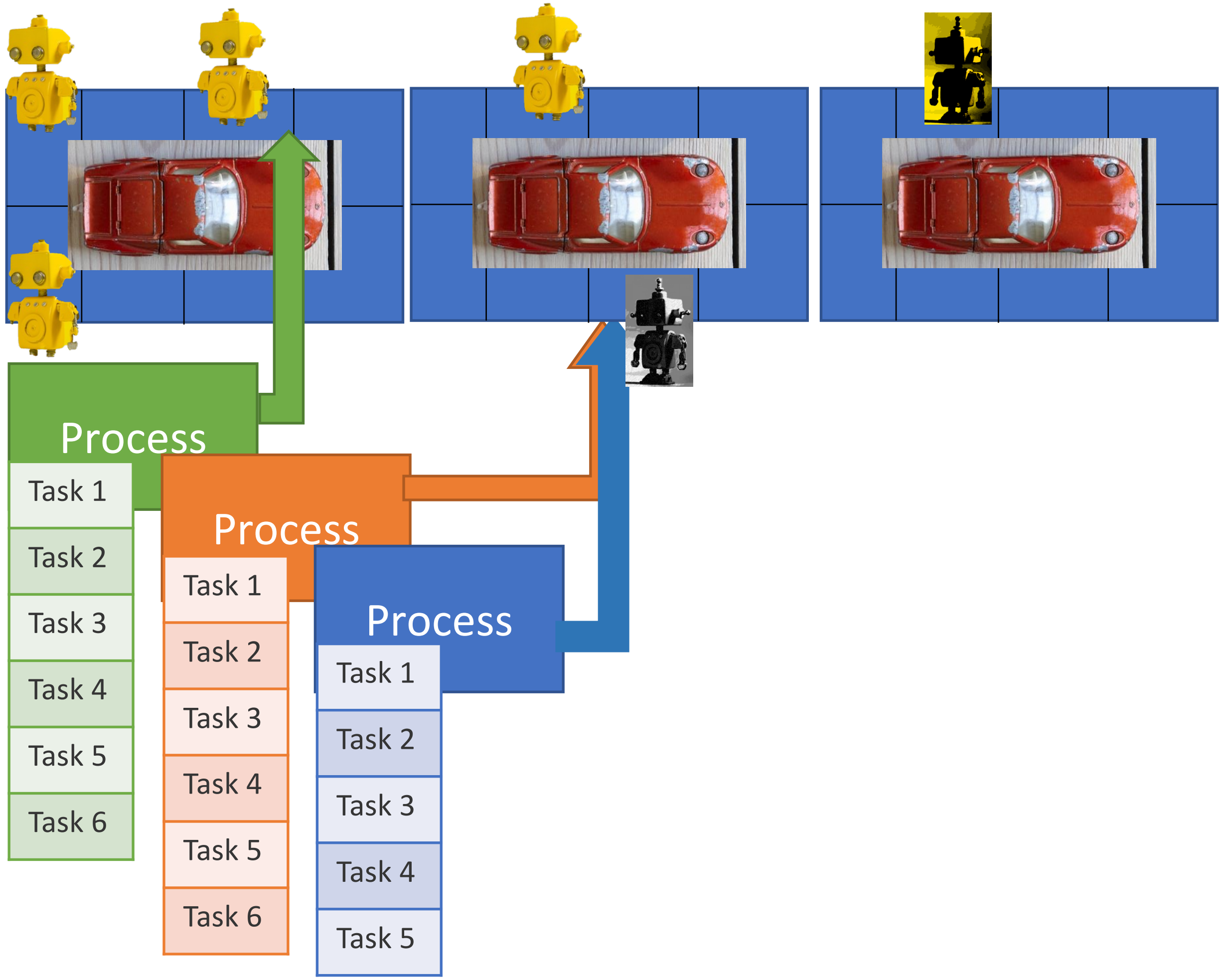
- Bounded completion time
- Partial order of stations and processes
- What operations stations can perform

Objectives:

- Minimize resource consumption
- Minimize operator congestion

Enable:

- Automate manual puzzle
- Optimize over design space
- Scale and be nimble: new factories, new models
- Track and manage inventory

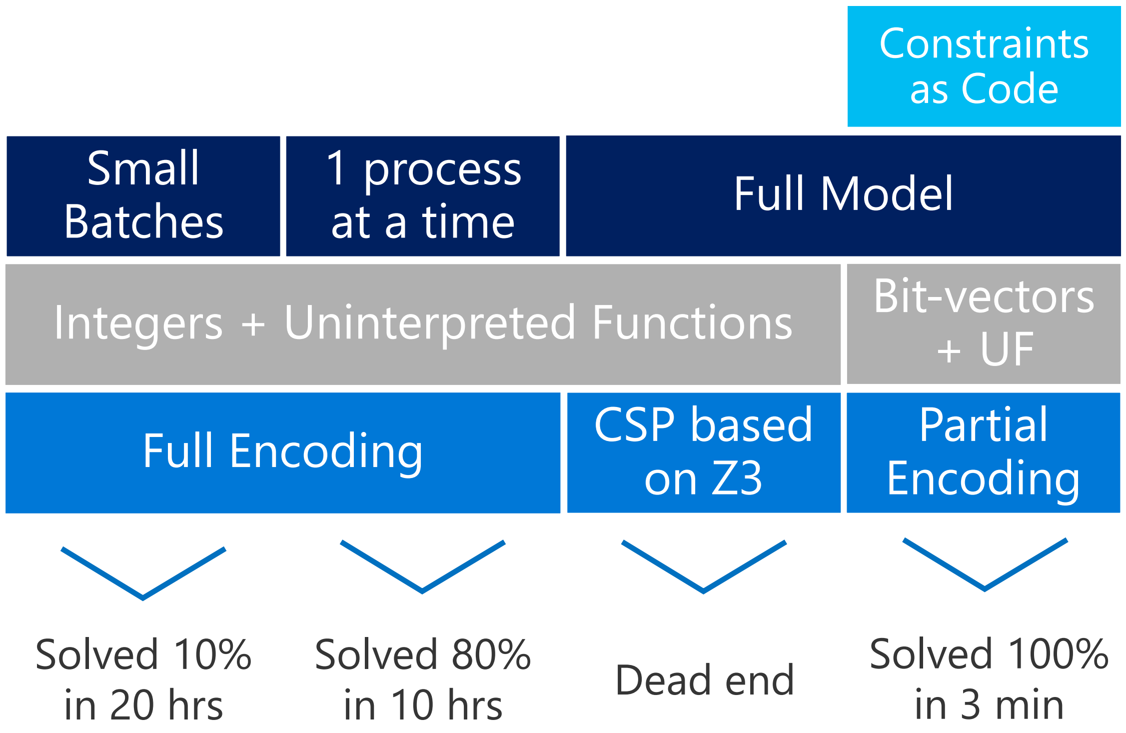


Experiences Summary

Domain Engineering

Model Formalization	Data Validation
Develop mathematically precise model	Visualization
Relational Object Model of SQL data	Semantic validation of SQL data

Solver Engineering

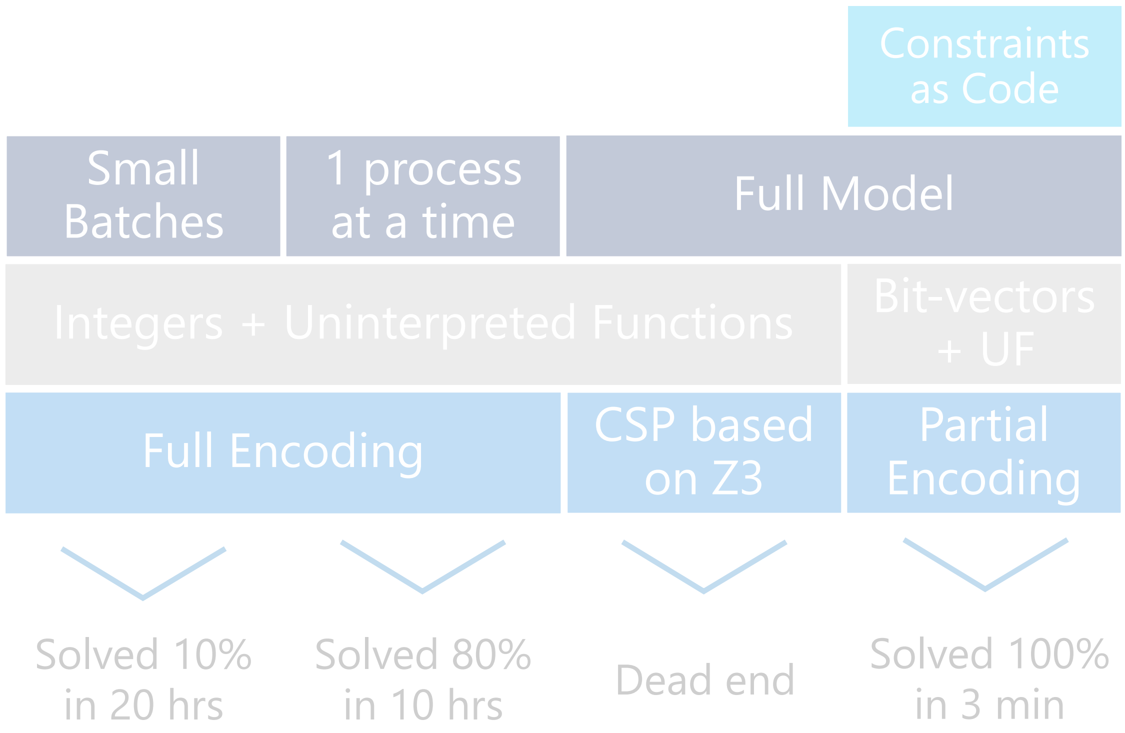


Experiences Summary

Domain Engineering

Model Formalization	Data Validation
Develop mathematically precise model	Visualization
Relational Object Model of SQL data	Semantic validation of SQL data

Solver Engineering



Constraints as Code

Small Batches

1 process at a time

Full Model

Integers + Uninterpreted Functions

Bit-vectors + UF

Full Encoding

CSP based on Z3

Partial Encoding

Solved 10% in 20 hrs

Solved 80% in 10 hrs

Dead end

Solved 100% in 3 min

Domain Engineering – Mathematical Modeling

Solve for:

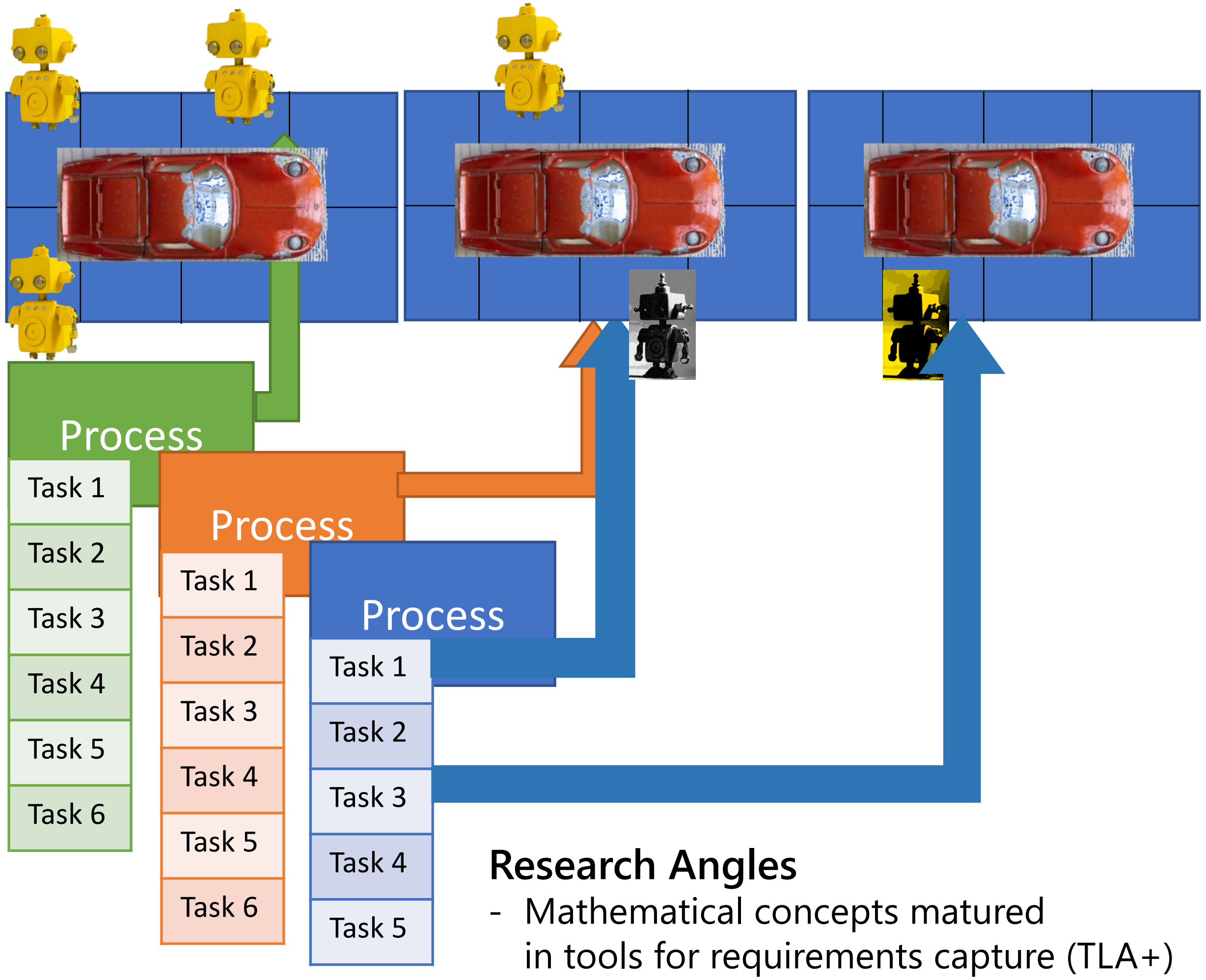
$assign_p : Station$ each process p

Auxiliary Functions:

$$maxHeight : Station \to Nat$$
$$operator : Station \times Zone \to Operator$$

Assignment Constraints:

$$operator(assign_p, z) \in \{op_1, op_2\}$$
$$maxHeight(assign_p) \geq height_p$$



Process

Task 1
Task 2
Task 3
Task 4
Task 5
Task 6

Process

Task 1
Task 2
Task 3
Task 4
Task 5
Task 6

Process

Task 1
Task 2
Task 3
Task 4
Task 5

Mapping to Z3 at *same* the level of model

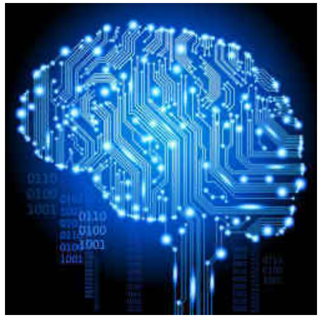
- Uninterpreted functions
- Nested formulas (no tuning for big Ms)
- Finite domains using bit-vectors

Research Angles

- Mathematical concepts matured in tools for requirements capture (TLA+)
- A sweet spot for Formal Methods skillsets
- Uncovered many subtle implicit assumptions

Domain Engineering – Semantic Validation

Grand Goal



Deep Solving

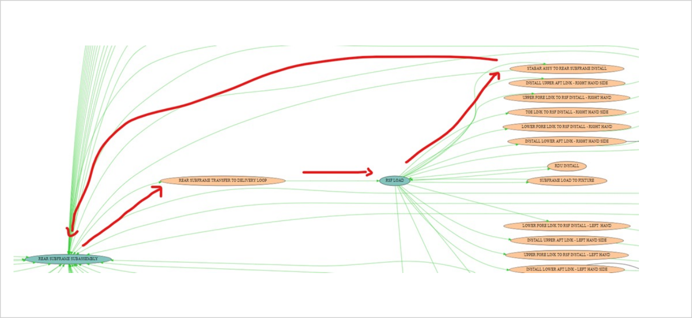
Humble Path

Deep Validation

Validation Approaches

1. Scripts that check invariants of database entries

2. Provenance information using infeasible cores from Z3



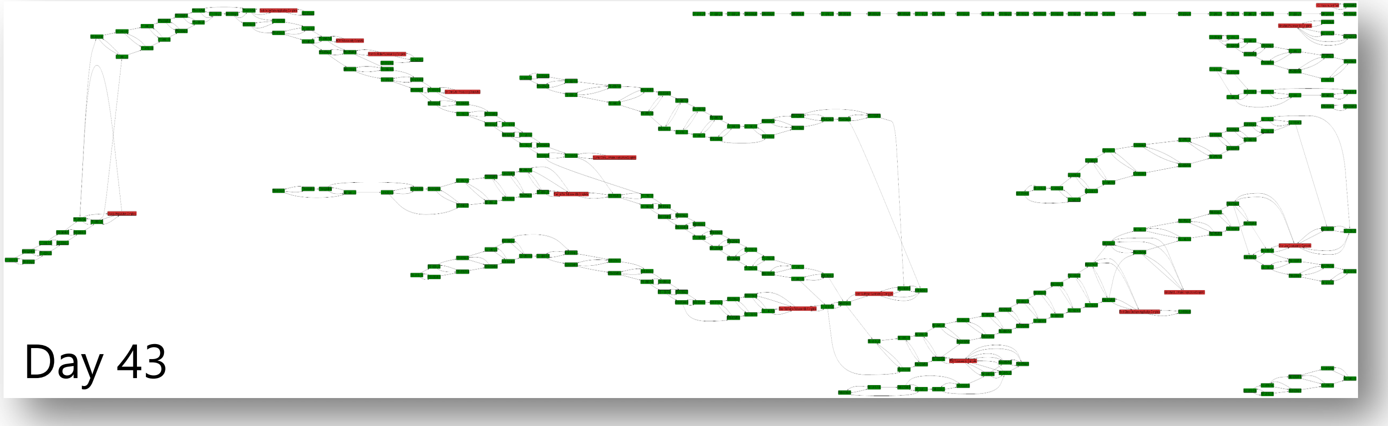
Z3 Features

- Native core minimization in SAT solver
- Core and correction set enumeration
- Software bug localization and repair

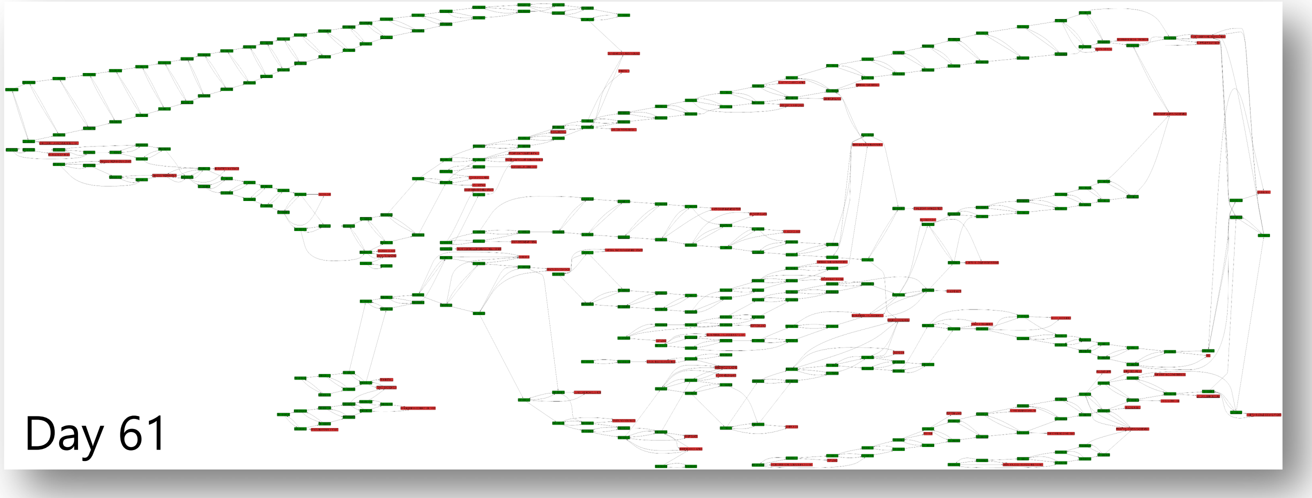
Practical impact

- Invariant checker and provenance tools in hands of collaborators
- Used to fix a significant set of data entry bugs

Domain Engineering – Visualization



Aid to understand model stored in database and spot bugs by simple inspection



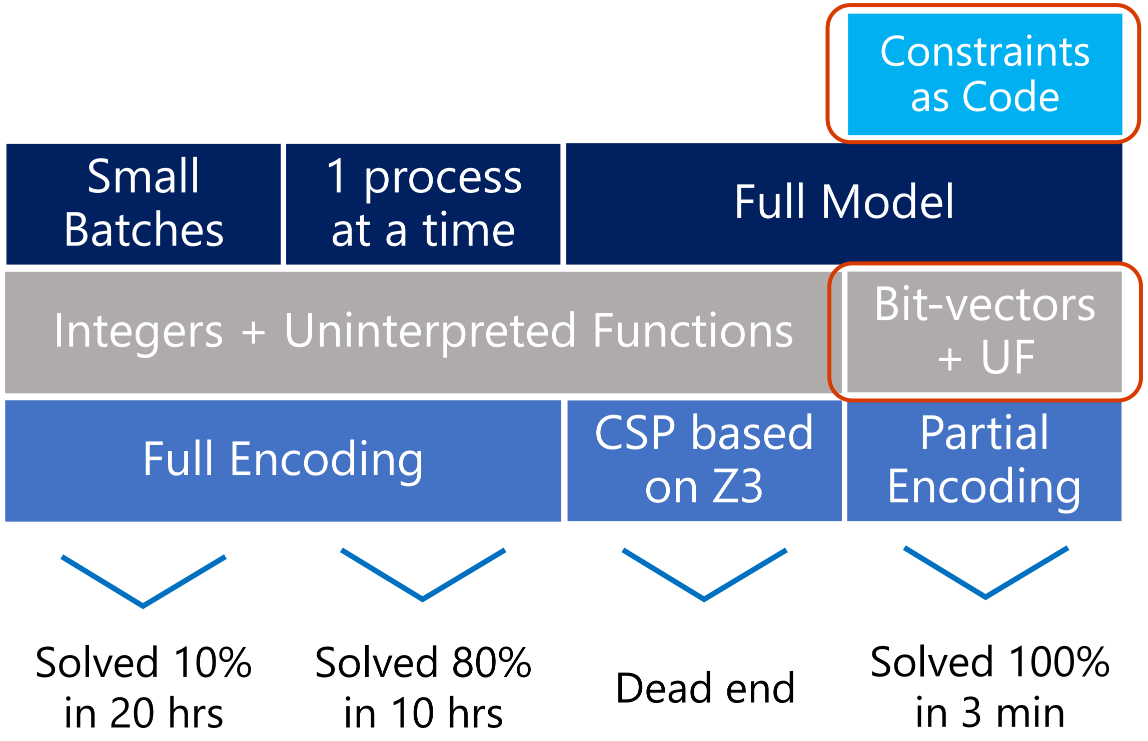
MSAGL – Automated Graph Layout engine

Experiences Summary

Domain Engineering

Model Formalization	Data Validation
Develop mathematically precise model	Visualization
Relational Object Model of SQL data	Semantic validation of SQL data

Solver Engineering



Constraints as Code

Small Batches

1 process at a time

Full Model

Integers + Uninterpreted Functions

Bit-vectors + UF

Full Encoding

CSP based on Z3

Partial Encoding

Solved 10% in 20 hrs

Solved 80% in 10 hrs

Dead end

Solved 100% in 3 min

A Fly in the Ointment

and a Wasp in the Rose

Domain Overload

Number of Processes = O(1K)

Number of Stations = O(1K)

Number of Tasks = O(10K)

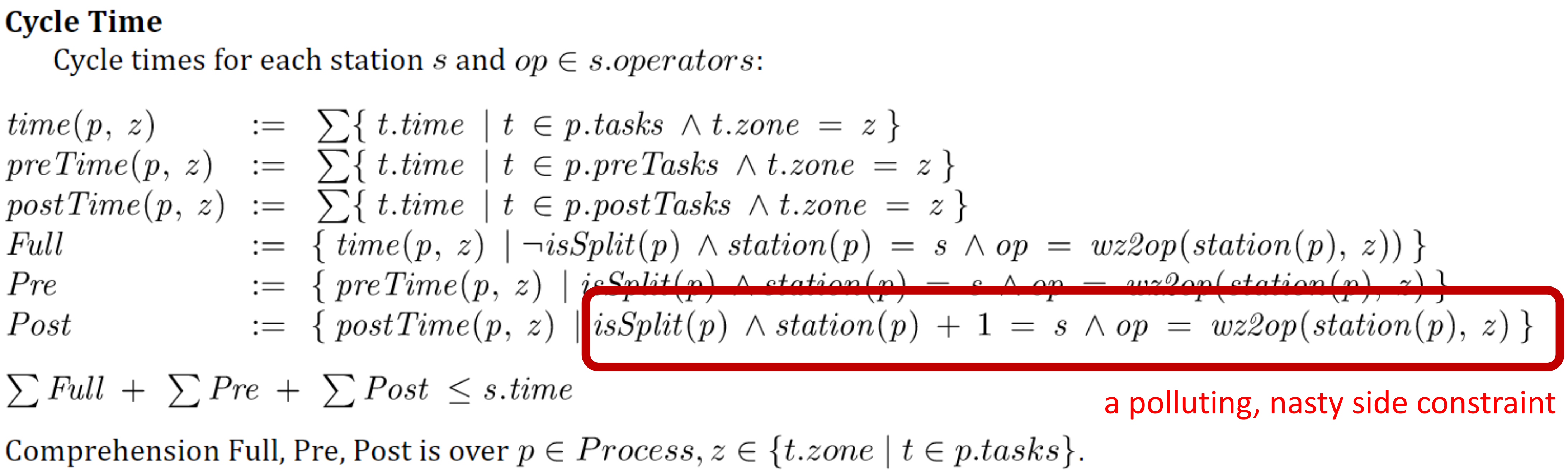
Up to O(10) different operators per station

Direct MIP-style encoding: $t_{i,s,op}$ - Task i is at station s using operator op

$$10K\times1K\times10 = 100M \text{ variables}$$

Our approach: Use uninterpreted functions for “symbolic indices”

Constraint Overload



Cycle Time

Cycle times for each station s and $op \in s.operators$:

$$
\begin{aligned}
time(p,\ z) &:= \sum\{\ t.time \mid t \in p.tasks\ \wedge t.zone\ =\ z\ \} \\
preTime(p,\ z) &:= \sum\{\ t.time \mid t \in p.preTasks\ \wedge t.zone\ =\ z\ \} \\
postTime(p,\ z) &:= \sum\{\ t.time \mid t \in p.postTasks\ \wedge t.zone\ =\ z\ \} \\
Full &:= \{\ time(p,\ z) \mid \neg isSplit(p)\ \wedge station(p)\ =\ s\ \wedge op\ =\ wz2op(station(p),\ z))\ \} \\
Pre &:= \{\ preTime(p,\ z) \mid isSplit(p)\ \wedge station(p)\ =\ s\ \wedge op\ =\ wz2op(station(p),\ z)\ \} \\
Post &:= \{\ postTime(p,\ z) \mid \boxed{isSplit(p)\ \wedge station(p)\ +\ 1\ =\ s\ \wedge op\ =\ wz2op(station(p),\ z)\ \}}
\end{aligned}
$$

$$\sum Full\ +\ \sum Pre\ +\ \sum Post\ \leq s.time$$

a polluting, nasty side constraint

Comprehension Full, Pre, Post is over $p \in Process, z \in \{t.zone \mid t \in p.tasks\}$.

Constrained multi-knapsack:

A set of items, each is added to one knapsack, subject to side-constraints

Our approach: program constraint as an ad-hoc theory

Solver Engineering – Mathematical Modeling

Solve for:

$assign_p : Station$ each process p

Auxiliary Functions:

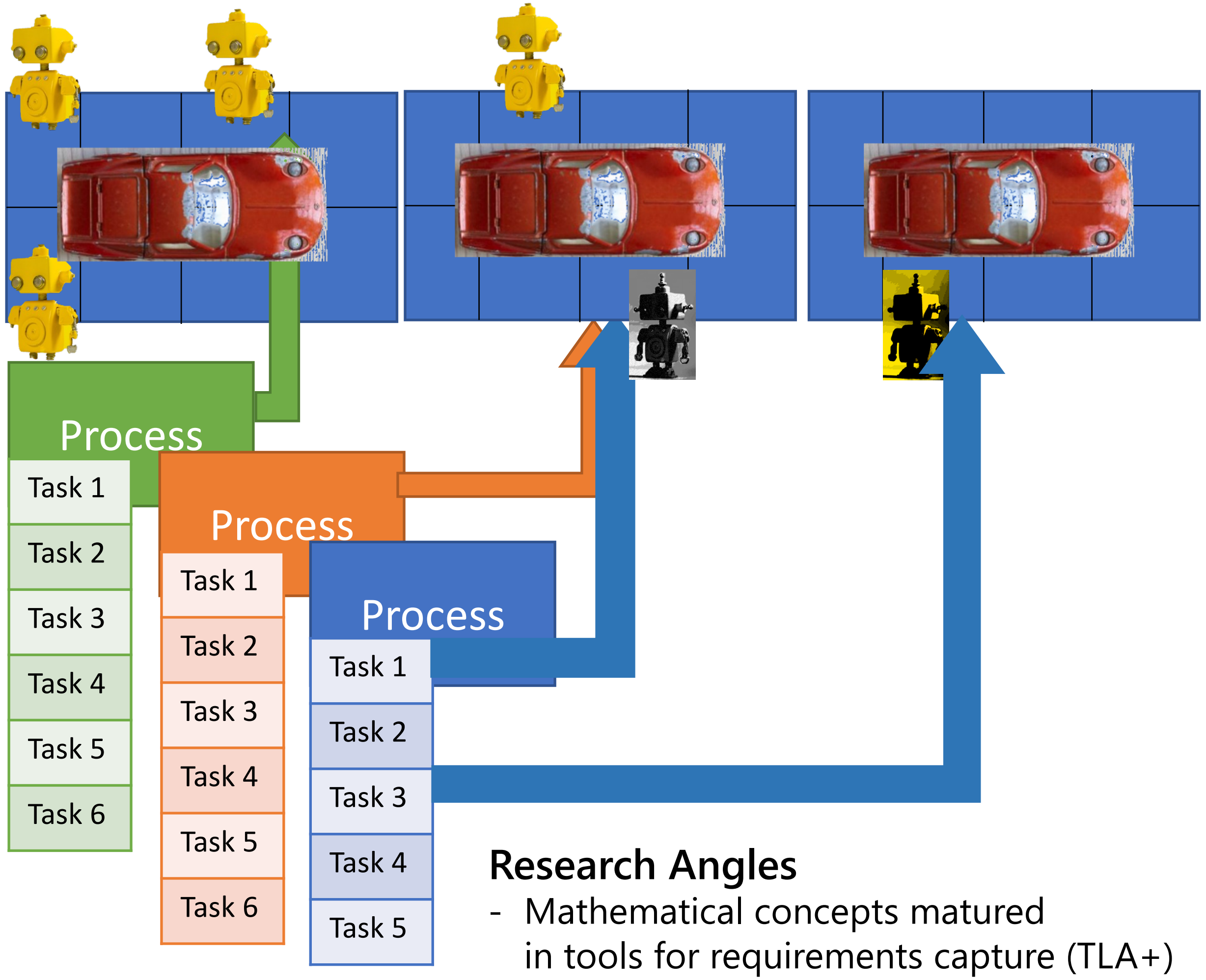
$$maxHeight : Station \rightarrow Nat$$
$$operator : Station \times Zone \rightarrow Operator$$

Assignment Constraints:

$$operator(assign_p , z) \in \{op_1, op_2\}$$
$$maxHeight(assign_p) \geq height_p$$

Mapping to Z3 at *same* the level of model

- Uninterpreted functions
- Nested formulas (no tuning for big Ms)
- Finite domains using bit-vectors



Research Angles

- Mathematical concepts matured in tools for requirements capture (TLA+)
- A sweet spot for Formal Methods skillsets
- Uncovered many subtle implicit assumptions

Solver Engineering – Mathematical Modeling

Solve for:

$assign_p\colon Station$ each process p

Auxiliary Functions:

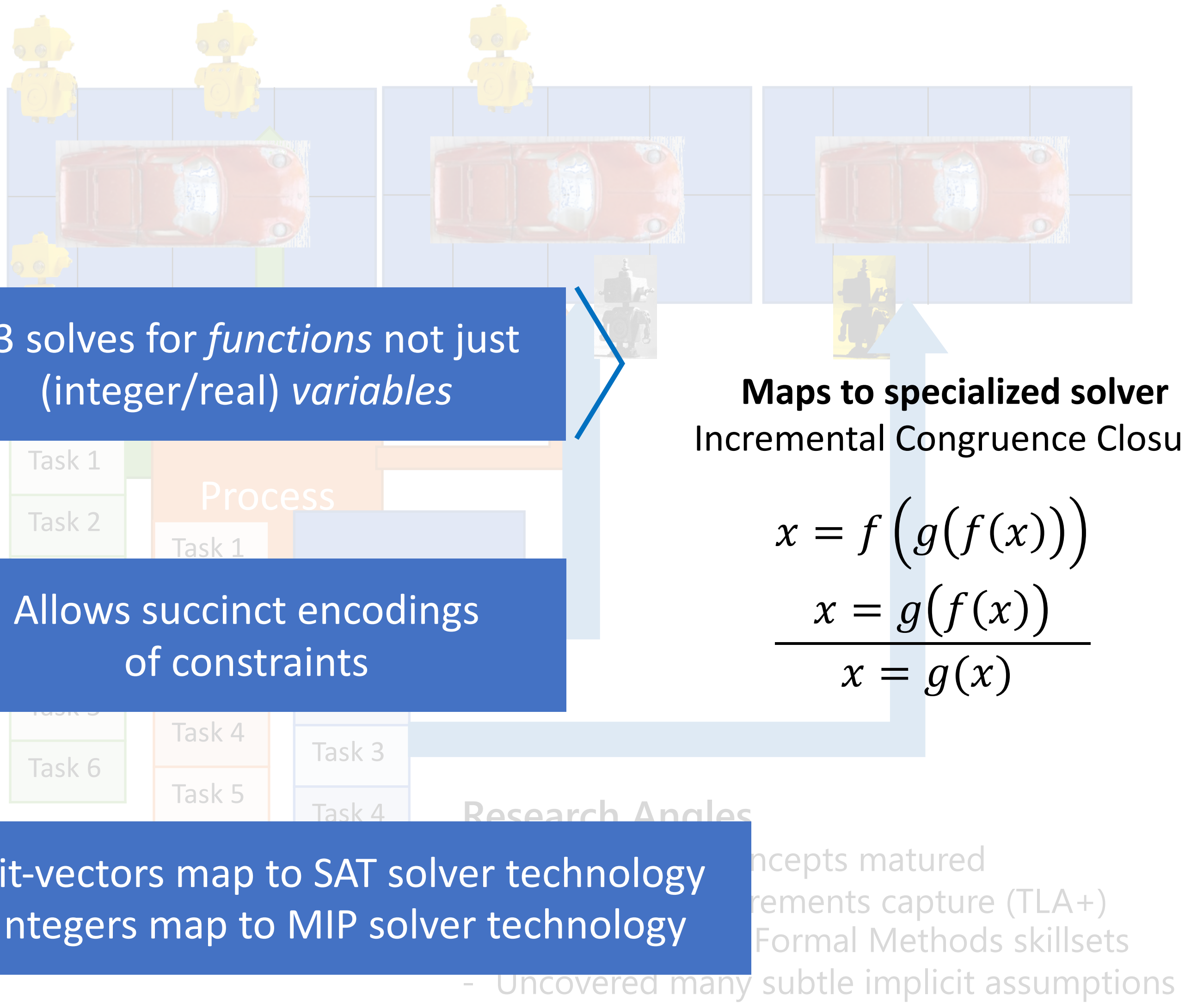
$$maxHeight\colon Station \rightarrow Nat$$
$$operator\colon Station \times Zone \rightarrow Operator$$

Assignment Constraints:

$$operator(assign_p, z) \in \{op_1, op_2\}$$
$$maxHeight(assign_p) \geq height_p$$

Mapping to Z3 at *same* the level of model

- Uninterpreted functions
- Nested formulas (no tuning for big Ms)
- Finite domains using bit-vectors



Z3 solves for *functions* not just (integer/real) *variables*

Allows succinct encodings of constraints

Bit-vectors map to SAT solver technology
Integers map to MIP solver technology

Maps to specialized solver
Incremental Congruence Closure

$$\frac{\begin{array}{c} x = f\left(g(f(x))\right) \\ x = g(f(x)) \end{array}}{x = g(x)}$$

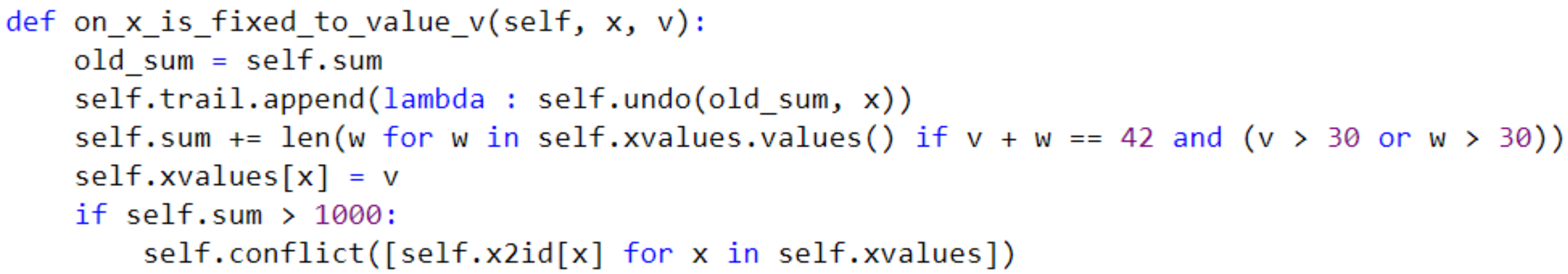
Solver Engineering – Constraints as Code

When supported formalisms lag, or encoding is impractical

As constraints:
O(100K) atomic formulas

$$\left\langle \; 1000 \geq \sum_{0\leq i<j\leq 500} x_i + x_j = 42 \wedge \left(x_i > 30 \vee x_j > 30\right) \; \right\rangle$$

As code:
callback + conflicts on demand



```
def on_x_is_fixed_to_value_v(self, x, v):
    old_sum = self.sum
    self.trail.append(lambda : self.undo(old_sum, x))
    self.sum += len(w for w in self.xvalues.values() if v + w == 42 and (v > 30 or w > 30))
    self.xvalues[x] = v
    if self.sum > 1000:
        self.conflict([self.x2id[x] for x in self.xvalues])
```

Solving Strategy

v1: Pre-solving

- Assign a small batch of 4-10 processes to stations at a time
- Stations and processes, station heights and task time are integers
- Solved 100 processes very slowly.

v2: Pre-solving take two

- Assign just one process at a time and only encode process constraints when a process gets assigned.
- The resulting solver can assign 950 out of 1050 processes in a few minutes.

v3: A custom CDCL / CSP solver

- Perform branching and propagation of cycle time constraints on top of repeated calls to Z3.
- Maintain backtracking stack and add lemmas based on the chosen branches.
- This was complex to engineer and only exercised in preliminary form.

v4: with Custom Propagator and Bit-vectors

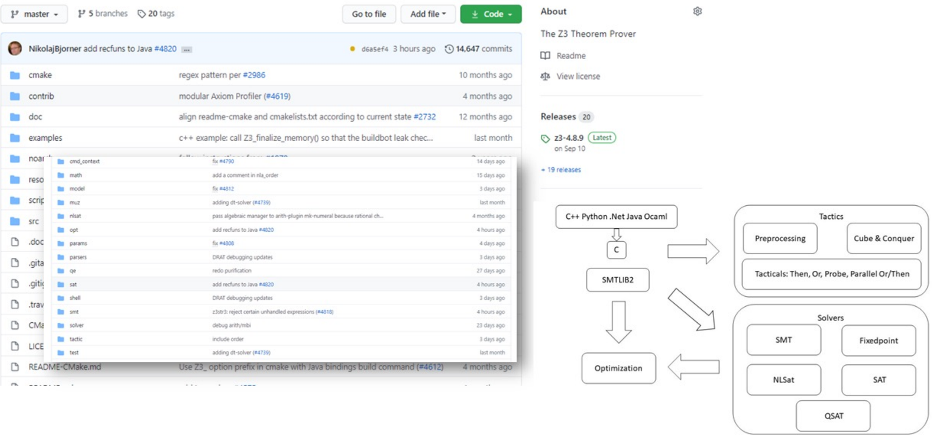
- With bit-vectors, without cycle-time: solvable in 30 seconds.
- With bit-vectors and cycle-time: solvable for 300 processes in a few minutes, but not all processes.
- With bit-vectors, programmable-propagator for cycle-time: patching + solving
- Initial: a few hours
- Current: a few minutes.

SMT for OR?

- Already happened: CP-SAT uses CDCL(T) for OR domains
- Approach here: Uninterpreted Functions, Bit-Vectors, Constraints as Code
- From experiences to tuning:
  - LNS for Modulo Theories?
  - A modernized core solver for Z3: In-processing for SMT?
  - Sound MIP is too costly for CP: Specialized LP for modular machine arithmetic

Summary

Z3 – an efficient SMT solver



master 5 branches 20 tags Go to file Add file Code

NikolajBjorner add recfuns to Java #4820 d6a5ef4 3 hours ago 14,647 commits

cmake	regex pattern per #2986	10 months ago
contrib	modular Axiom Profiler (#4619)	4 months ago
doc	align readme-cmake and cmakelists.txt according to current state #2732	12 months ago
examples	c++ example: call Z3_finalize_memory() so that the buildbot leak chec...	last month
cmd_context	fix #4790	14 days ago
math	add a comment in nla_order	15 days ago
model	fix #4812	3 days ago
muz	adding dt-solver (#4739)	last month
nlsat	pass algebraic manager to arith-plugin mk-numeral because rational ch...	4 months ago
opt	add recfuns to Java #4820	4 hours ago
params	fix #4808	4 days ago
parsers	DRAT debugging updates	3 days ago
qe	redo purification	27 days ago
sat	add recfuns to Java #4820	4 hours ago
shell	DRAT debugging updates	3 days ago
smt	z3str3: reject certain unhandled expressions (#4818)	4 hours ago
solver	debug arith/nbi	23 days ago
tactic	include order	3 days ago
test	adding dt-solver (#4739)	last month
README-CMake.md	Use Z3_ option prefix in cmake with Java bindings build command (#4612)	4 months ago

About

The Z3 Theorem Prover

Readme

View license

Releases 20

z3-4.8.9 Latest
on Sep 10

+ 19 releases

Theories

Congruence Closure

$$\begin{array}{c} x = f\left(g\left(f(x)\right)\right) \\ x = g\left(f(x)\right) \\ \hline x = g(x) \end{array}$$

Constraints as Code

```python
def on_x_is_fixed_to_value_v(self, x, v):
    old_sum = self.sum
    self.trail.append(lambda : self.undo(old_sum, x))
    self.sum += len(w for w in self.xvalues.values() if v + w == 42 and (v > 30 or w > 30))
    self.xvalues[x] = v
    if self.sum > 1000:
        self.conflict([self.x2id[x] for x in self.xvalues])
```

Bit-vectors

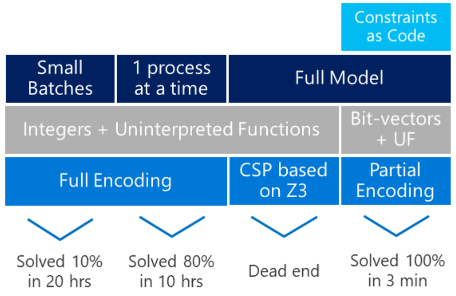
```python
def encoding(self):
line_bits      = math.ceil(math.log(len(self.model.lines), 2))
self.Line      = BitVecSort(line_bits)
self.Station   = BitVecSort(station_bits)
self.Operator  = BitVecSort(operator_bits)
self.Segment   = BitVecSort(segment_bits)
self.op_used     = Function( 'op_used',    self.Station, self.Operator, BoolSort()    ) # Is operator used
self.wz_used     = Function( 'wz_used',    self.Station, self.Zone,     BoolSort()    ) # Which workzones used
self.min_height = Function( 'min_height', self.Station,                self.Height   ) # Min height of station

...
yield Implies(p.is_split, self.min_height(p.to_station + 1) <= min_height), E.suf_height_lo(p, min_height)
yield Implies(p.is_split, self.max_height(p.to_station + 1) >= max_height), E.suf_height_hi(p, max_height)
```

Experiences

Domain Engineering

Model Formalization	Data Validation
Develop mathematically precise model	Visualization
Relational Object Model of SQL data	Semantic validation of SQL data

Solver Engineering



			Constraints as Code
Small Batches	1 process at a time	Full Model	
Integers + Uninterpreted Functions			Bit-vectors + UF
Full Encoding		CSP based on Z3	Partial Encoding
Solved 10% in 20 hrs	Solved 80% in 10 hrs	Dead end	Solved 100% in 3 min

Humble Path

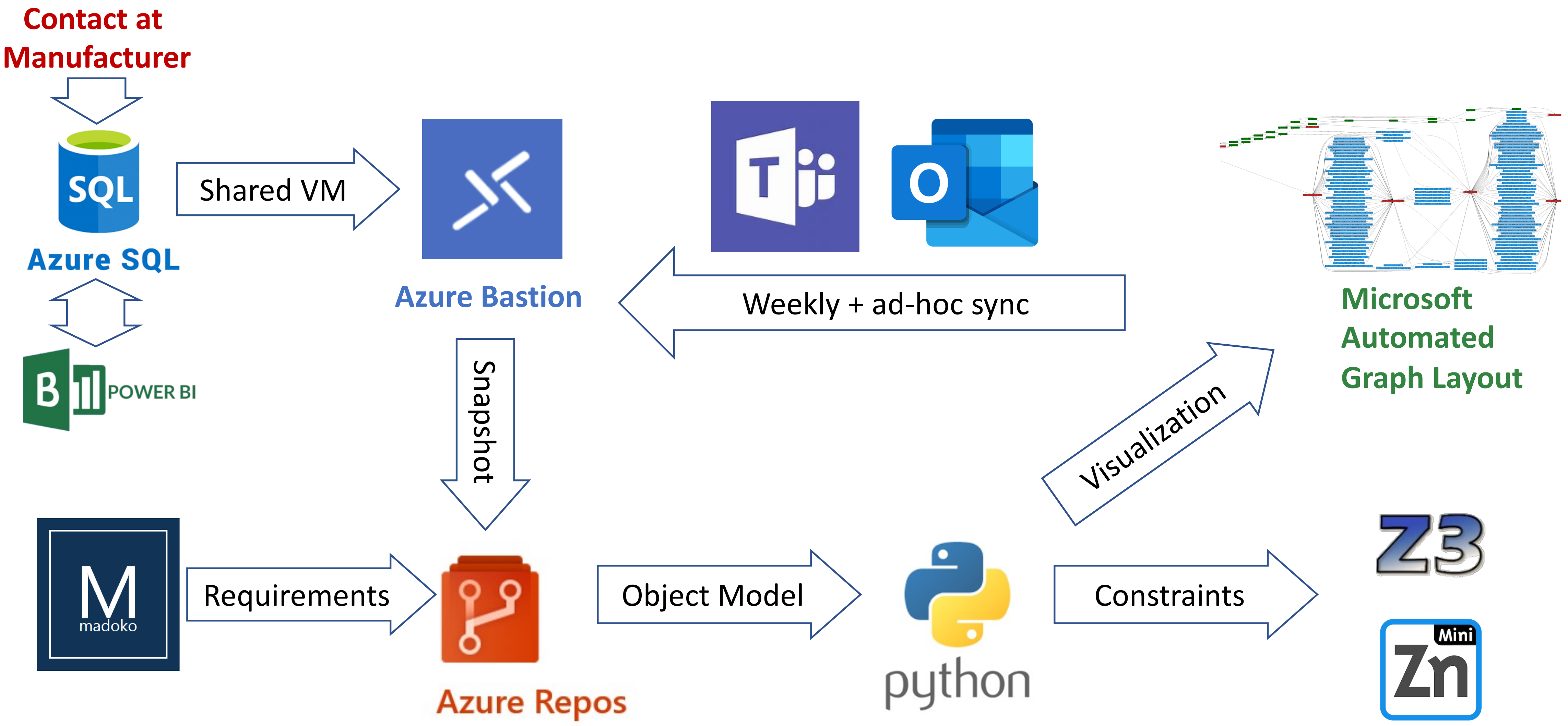
Data Validation

Fly in the Ointment

Nimble Constraints

Extra Slides

Tools used as part of collaboration



Contact at Manufacturer

Azure SQL

Shared VM

Azure Bastion

Weekly + ad-hoc sync

Microsoft Automated Graph Layout

Snapshot

Visualization

Requirements

Azure Repos

Object Model

Constraints

Optimization Objectives

Currently at early stage

Understanding what best serves our scenario

Likely main objective

Reduce number of operators, reduce number of tools used overall.

First approach is by programmable Branch & Bounding to find a Pareto Front per run.

Research Angles

- Pareto Strategies
- Any-time optimization
- Local Neighborhood search
- MaxSAT based on:
    - Cores
    - Hitting sets
    - Correction sets
    - Branch and bound

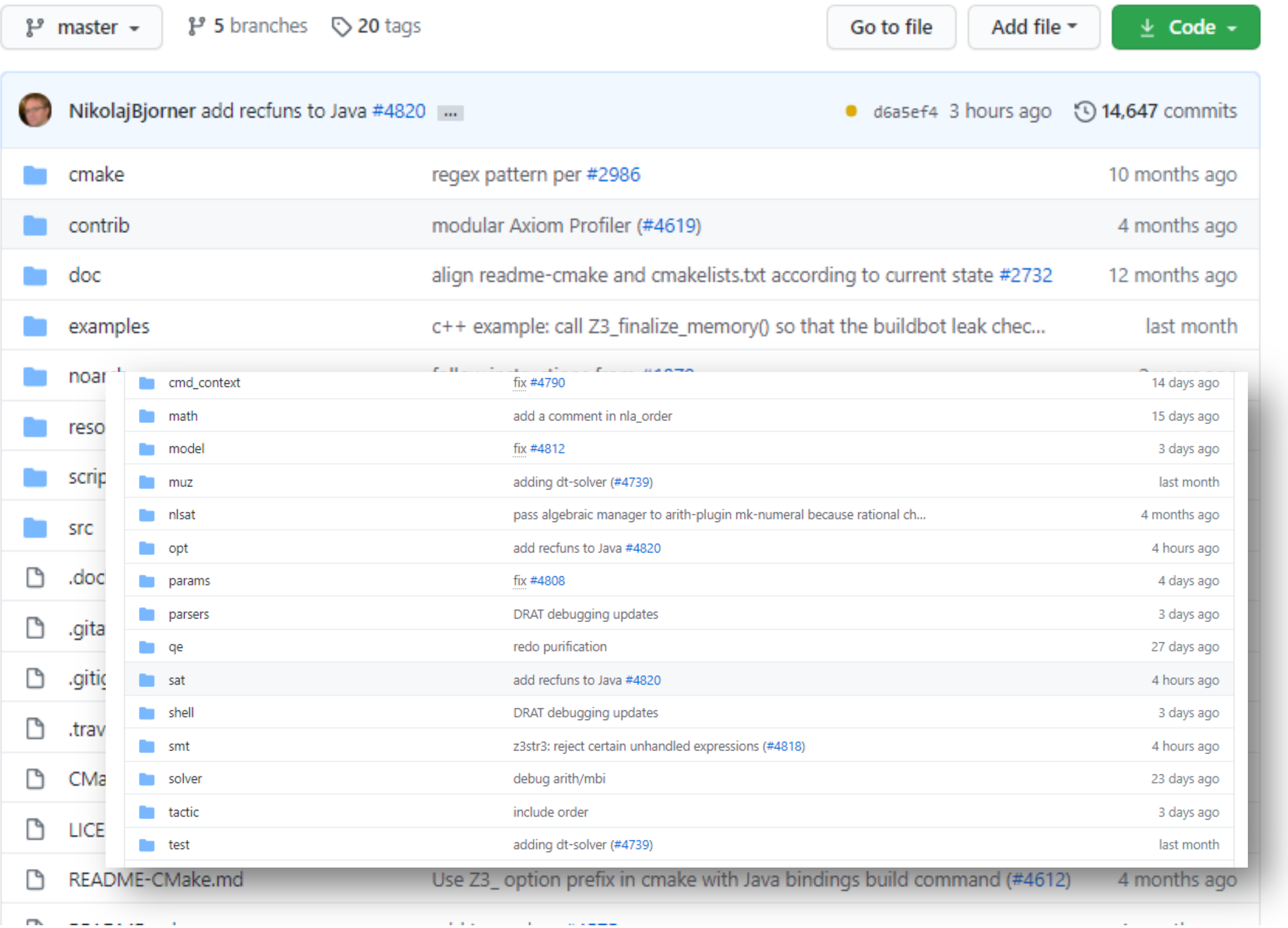
Z3 Technologies

- Core based MaxSAT
- Primal Simplex
- Multi-objective optimization: Pareto, Lex, Box

Some years ago

Used Azure cloud scaling (cube & conquer) and large neighborhood search to optimize NFL schedules

Z3 – an efficient SMT solver



master · 5 branches · 20 tags · Go to file · Add file · Code

NikolajBjorner add recfuns to Java #4820 … d6a5ef4 3 hours ago 14,647 commits

cmake	regex pattern per #2986	10 months ago
contrib	modular Axiom Profiler (#4619)	4 months ago
doc	align readme-cmake and cmakelists.txt according to current state #2732	12 months ago
examples	c++ example: call Z3_finalize_memory() so that the buildbot leak chec…	last month
noar		
reso		
scrip		
src		
.doc		
.gita		
.gitig		
.trav		
CMa		
LICE		
README-CMake.md	Use Z3_ option prefix in cmake with Java bindings build command (#4612)	4 months ago

cmd_context	fix #4790	14 days ago
math	add a comment in nla_order	15 days ago
model	fix #4812	3 days ago
muz	adding dt-solver (#4739)	last month
nlsat	pass algebraic manager to arith-plugin mk-numeral because rational ch…	4 months ago
opt	add recfuns to Java #4820	4 hours ago
params	fix #4808	4 days ago
parsers	DRAT debugging updates	3 days ago
qe	redo purification	27 days ago
sat	add recfuns to Java #4820	4 hours ago
shell	DRAT debugging updates	3 days ago
smt	z3str3: reject certain unhandled expressions (#4818)	4 hours ago
solver	debug arith/mbi	23 days ago
tactic	include order	3 days ago
test	adding dt-solver (#4739)	last month

About

The Z3 Theorem Prover

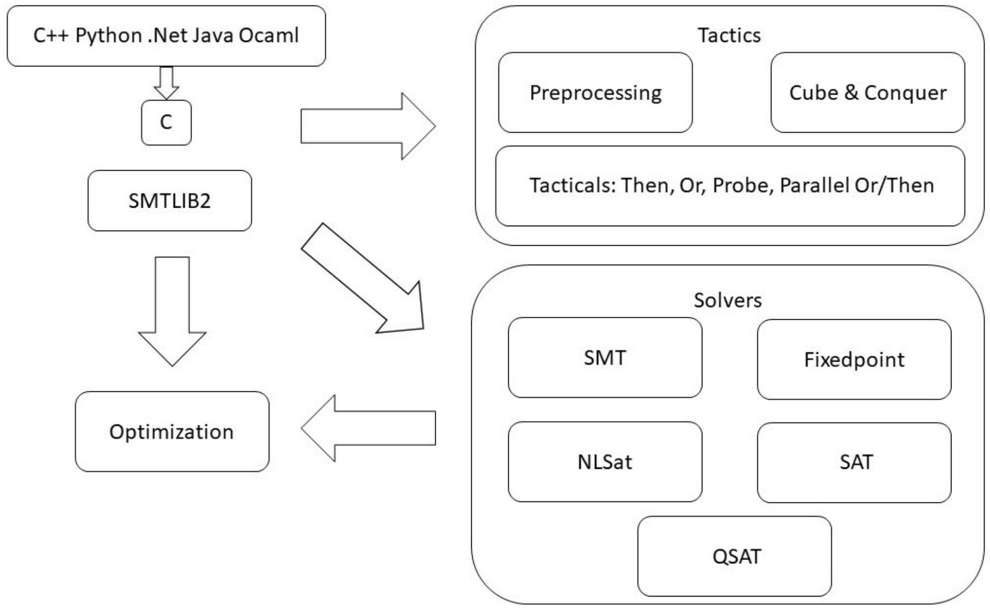
Readme

View license

Releases 20

z3-4.8.9 Latest
on Sep 10

+ 19 releases



C++ Python .Net Java Ocaml

C

SMTLIB2

Optimization

Tactics

Preprocessing

Cube & Conquer

Tacticals: Then, Or, Probe, Parallel Or/Then

Solvers

SMT

Fixedpoint

NLSat

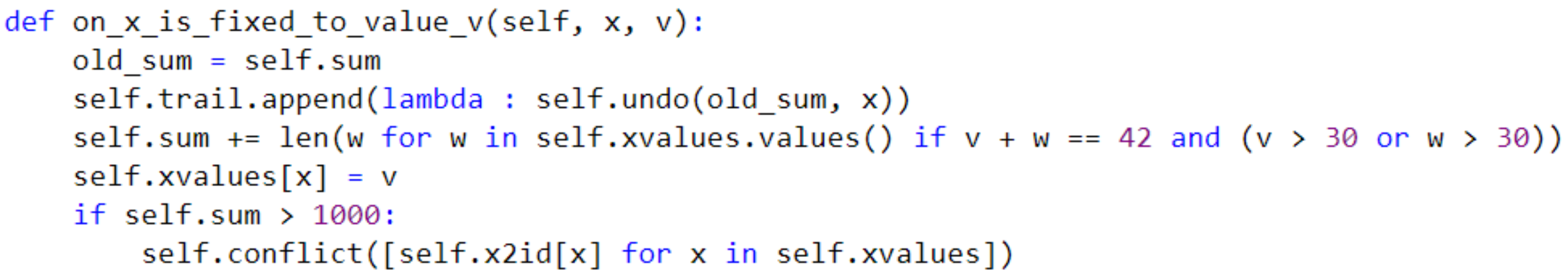
SAT

QSAT

Congruence Closure

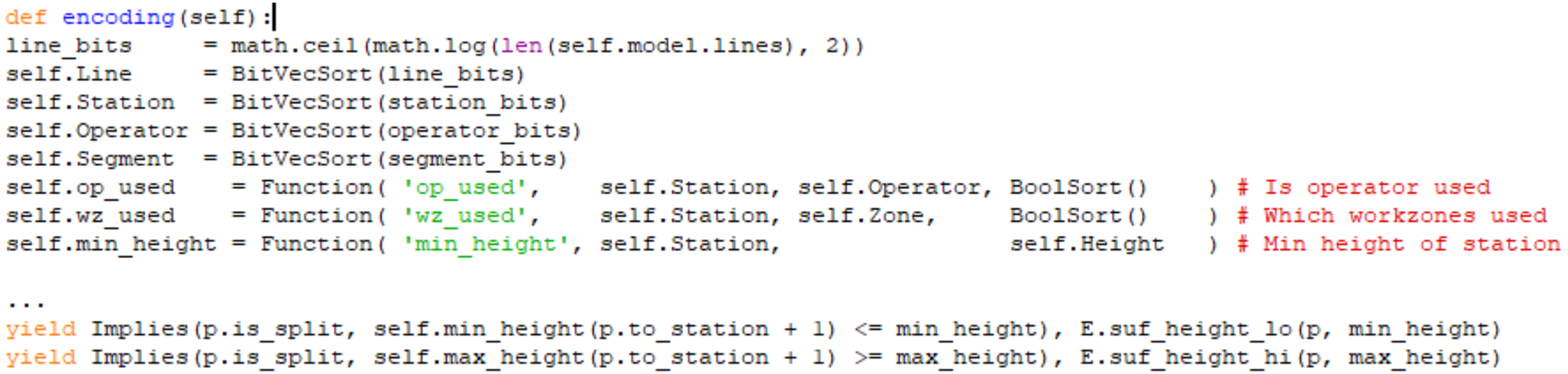
$$\frac{\begin{array}{c} x = f\left(g(f(x))\right) \\ x = g(f(x)) \end{array}}{x = g(x)}$$

Constraints as Code



```
def on_x_is_fixed_to_value_v(self, x, v):
    old_sum = self.sum
    self.trail.append(lambda : self.undo(old_sum, x))
    self.sum += len(w for w in self.xvalues.values() if v + w == 42 and (v > 30 or w > 30))
    self.xvalues[x] = v
    if self.sum > 1000:
        self.conflict([self.x2id[x] for x in self.xvalues])
```

Bit-vectors



```
def encoding(self):
line_bits      = math.ceil(math.log(len(self.model.lines), 2))
self.Line      = BitVecSort(line_bits)
self.Station   = BitVecSort(station_bits)
self.Operator  = BitVecSort(operator_bits)
self.Segment   = BitVecSort(segment_bits)
self.op_used     = Function( 'op_used',    self.Station, self.Operator, BoolSort()     ) # Is operator used
self.wz_used     = Function( 'wz_used',    self.Station, self.Zone,     BoolSort()     ) # Which workzones used
self.min_height  = Function( 'min_height', self.Station,                self.Height    ) # Min height of station

...
yield Implies(p.is_split, self.min_height(p.to_station + 1) <= min_height), E.suf_height_lo(p, min_height)
yield Implies(p.is_split, self.max_height(p.to_station + 1) >= max_height), E.suf_height_hi(p, max_height)
```