CVC4 and Query Dispatcher

Query Dispatcher

- Input: SMT-LIB file
- Chooses up to 4 SMT-solvers
  - Based on results of last SMT-COMP
  - Filters solvers that can solve the input problem
  - Decision is guided by the **logic** of the input problem
- Runs selected solvers in parallel
- Once the first solver finishes, the rest are terminated
- Demo

Syntax Guided Interpolation in CVC4

- Interpolation:
  - Given formulas A and B such that A=>B
  - Find a formula C such that:
    - A=>C and C=>B
    - C only has symbols that are shared between A and B
- Syntax Guided Synthesis
  - Given a grammar and a property
  - Synthesize a formula in the grammar that satisfies the property
- Syntax Guided Interpolation:
  - Grammar: Symbols that are shared between A and B
  - Property: A=>C and C=>B

Interpolation for Model Checking

- Interpolants are used to over-approximate the reachable states
- Several interpolants are needed until a fixed-point is reached

- Interpolants for Bit-vectors:
  - Word-level: x+y=z, x&y>z, …
  - Bit-level: x[0]&y[0]=z[0], …
- Word-level interpolants have more potential:
  - Explainability: Easier to understand
  - Scalability: Amenable to more bit-vector solvers techniques

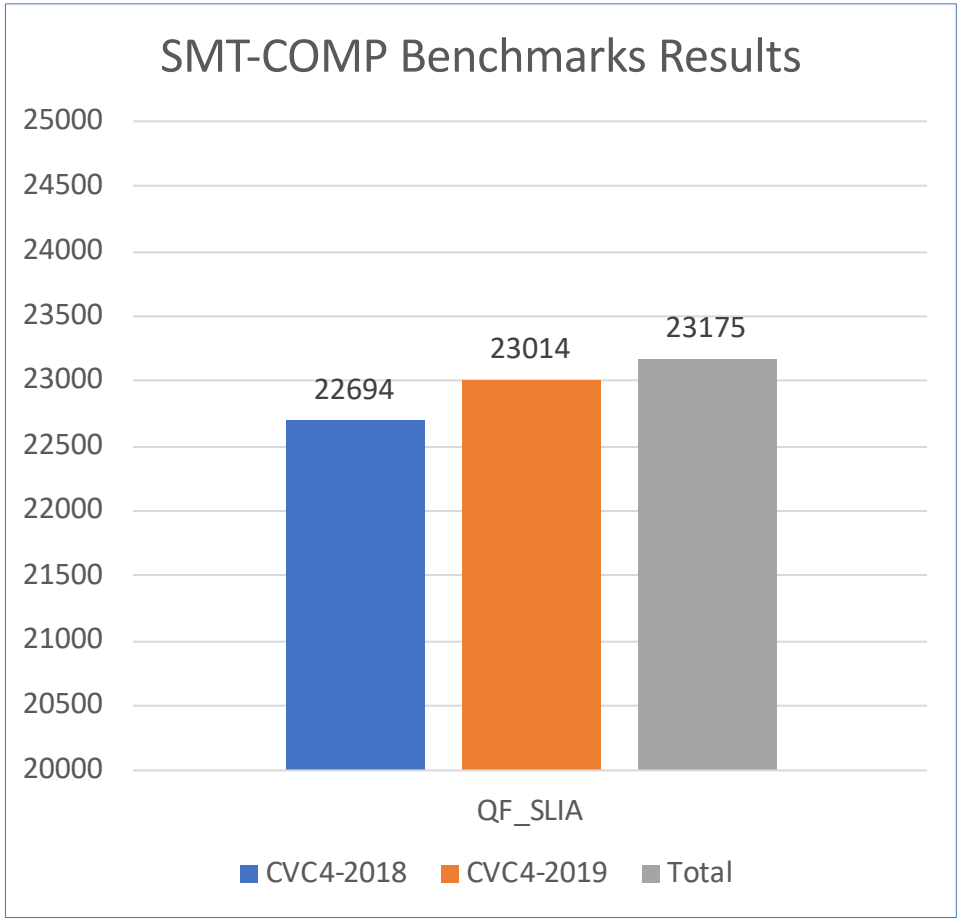
Work In Progress: CVC4-Interpolants in CoSA

CVC4	mathsat
(! (self.c__AT1 u< self.a__AT1)) & (! (self.c__AT1 u< self.b__AT1))	((! (self.c__AT1 u< self.b__AT1)) & (! (self.c__AT1 u< self.a__AT1))
((config_reg.conf_reg.value.out__AT1 = config_reg.conf_reg.value.in__AT1) & (config_reg.conf_reg.value.out__AT1 u< 1_2))	((! (config_reg.conf_reg.value.clk__AT1 = 0_1)) & ((0_1 = config_reg.conf_reg.value.out__AT1[0:0]) & (0_1 = config_reg.conf_reg.value.out__AT1[1:1])))
((config_reg.conf_reg.value.in__AT1 u< 1_2) & (config_reg.conf_reg.value.out__AT1 = 0_2))	((H__state_id1__H__AT1 = 1_1) & (((config_reg.conf_reg.value.in__AT1 = config_reg.conf_reg.value.out__AT1) & ((0_1 = config_reg.conf_reg.value.in__AT1[0:0]) & (0_1 = config_reg.conf_reg.value.in__AT1[1:1]))) | ((config_reg.conf_reg.value.in__AT1 = config_reg.conf_reg.value.out__AT1) & (((config_reg.conf_reg.value.in__AT1 = config_reg.conf_reg.value.out__AT1) & ((0_1 = config_reg.conf_reg.value.in__AT1[0:0]) & (0_1 = config_reg.conf_reg.value.in__AT1[1:1]))) | (config_reg.conf_reg.value.out__AT1 = 0_2)))))
	((H__state_id1__H__AT1 = 1_1) & ((config_reg.conf_reg.value.in__AT1 = config_reg.conf_reg.value.out__AT1) & ((0_1 = config_reg.conf_reg.value.in__AT1[0:0]) & (0_1 = config_reg.conf_reg.value.in__AT1[1:1]))))

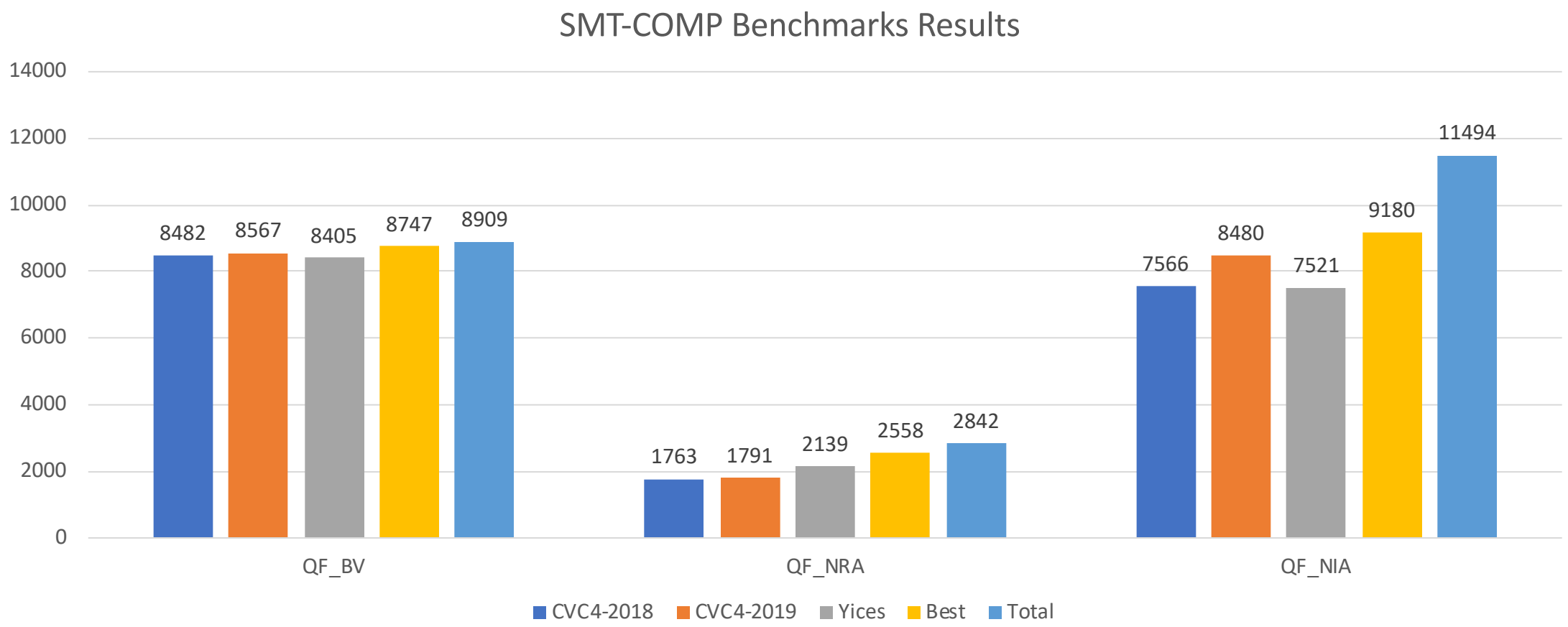
- CVC4's interpolants are **word level**
- One less iteration was needed!
- However… each interpolant computation is slower.

CVC4 Scalability: Strings

- String solving:
  - concat, substing, replace, etc.
  - len(s)
  - Regular expressions
- CVC4's String solver is under constant improvement
- Used by AWS for verifying access control policies
- Improved since 2018 (+320 solved)
- Experiments were re-run on the same cluster



CVC4 Scalability: Bit-vectors + Non-linear Arithmetic



- Improvement from 2018 to 2019 in all 3 divisions
- Some problems are better for CVC4, some are better for Yices
- Portfolio is **always** strictly better than either solver