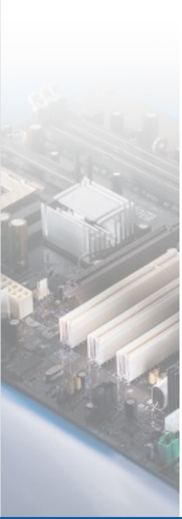


metaSMT: Focus On Your Application Not On Solver Integration

Finn Haedicke, Stefan Frehse, Görschwin Fey,
Daniel Große, Rolf Drechsler

Group of Computer Architecture, University of Bremen, Germany

DIFTS 2011

`http://www.informatik.uni-bremen.de/agra/eng/metasmt.php`

Outline

- Motivation
  - Initial Example
  - Design Goals
- Architecture
  - Syntax
  - Contexts
  - APIs
- Evaluation
- Summary
  - Features
  - Conclusions

Motivation

- Decision procedures are an important aspect of formal methods.

Motivation

- Decision procedures are an important aspect of formal methods.
- Many SAT and SMT solvers are available and increasingly powerful

Motivation

- Decision procedures are an important aspect of formal methods.
- Many SAT and SMT solvers are available and increasingly powerful
- Programming formal algorithms can be hard

Motivation

- Decision procedures are an important aspect of formal methods.
- Many SAT and SMT solvers are available and increasingly powerful
- Programming formal algorithms can be hard
- . . . even without integrating solvers.

Motivation

- Decision procedures are an important aspect of formal methods.
- Many SAT and SMT solvers are available and increasingly powerful
- Programming formal algorithms can be hard
- ... even without integrating solvers.

$\Rightarrow$ Framework to easily integrate advanved reasoning engines

  - metaSMT framework for Solver Integration
  - Domain Specific Language for SMT expression in C++
  - No algorithm changes when switching solvers

Example: Integer Factorization / Prime Test

Example

- Find a valid factorization of an integer $r = 1234567$
- Solve $r = a \times b \wedge a \neq 1 \wedge b \neq 1$ or prove its unsatisfiability
- All variables are bit-vector integers: $r, a, b \in \{0, \ldots, 2^n - 1\}$
- Easy to formulate as SMT-Lib instance

Example: Integer Factorization / Prime Test (2)

SMT-Lib 2.0

```
; declare variables
(declare-fun a () (_ BitVec 32))
(declare-fun b () (_ BitVec 32))
; assert a*b == r (1234567)
(assertion (=
    (bvmul
      ( (_ zero_extend 32) a)
      ( (_ zero_extend 32) b))
    (_ bv1234567 64 )
))
; a and be must not be 1
(assertion
  (not (= a (_ bv1 32))))
(assertion
  (not (= b (_ bv1 32))))

(check-sat)
(get-value (a))
(get-value (b))
```

Example: Integer Factorization / Prime Test (2)

SMT-Lib 2.0

```
; declare variables
(declare-fun a () (_ BitVec 32))
(declare-fun b () (_ BitVec 32))
; assert a*b == r (1234567)
(assertion (=
    (bvmul
      ( (_ zero_extend 32) a)
      ( (_ zero_extend 32) b))
    (_ bv1234567 64 )
))
; a and be must not be 1
(assertion
  (not (= a (_ bv1 32))))
(assertion
  (not (= b (_ bv1 32))))

(check-sat)
(get-value (a))
(get-value (b))
```

metaSMT (C++)

```

bitvector a=new_bitvector(bw);
bitvector b=new_bitvector(bw);

assertion( ctx, equal(
      bvmul(
        zero_extend(bw, a),
        zero_extend(bw ,b)),
      bvuint(1234567, 2*bw)
));

assertion( ctx,
    nequal(a, bvuint(1,bw)) );
assertion( ctx,
    nequal(b, bvuint(1,bw)) );

if( solve( ctx) )
  read_value ( ctx, a ),
  read_value ( ctx, b );
```

Example: Integer Factorization / Prime Test (2)

SMT-Lib 2.0

```
; declare variables
(declare−fun a () (_ BitVec 32))
(declare−fun b () (_ BitVec 32))
; assert a∗b == r (1234567)
(assertion (=
    (bvmul
      ( (_ zero_extend 32) a)
      ( (_ zero_extend 32) b))
    (_ bv1234567 64 )
))
; a and be must not be 1
(assertion
  (not (= a (_ bv1 32))))
(assertion
  (not (= b (_ bv1 32))))

(check−sat)
(get−value (a))
(get−value (b))
```

Boolector API

```
BtorExp∗ a,b;
a = boolector_var(btor, bw, "a");
b = boolector_var(btor, bw, "b");

boolector_assert(btor,boolector_eq( btor,
 boolector_mul(btor,
    boolector_uext(btor, a, bw),
    boolector_uext(btor, b, bw)),
 boolector_unsigned_int(btor, 1234567, 2∗bw)
));

boolector_assert( btor, boolector_ne(btor, a,
  boolector_unsigned_int(btor, 1, bw)) );
boolector_assert( btor, boolector_ne(btor, b,
  boolector_unsigned_int(btor, 1, bw)) );

if( boolector_sat( btor ) == BOOLECTOR_SAT )
  boolector_bv_assignment(_btor, a),
  boolector_bv_assignment(_btor, b);
```

Example: Integer Factorization / Prime Test (2)

SMT-Lib 2.0

```
; declare variables
(declare-fun a () (_ BitVec 32))
(declare-fun b () (_ BitVec 32))
; assert a*b == r (1234567)
(assertion (=
    (bvmul
      ( (_ zero
      ( (_ zero
    (_ bv123456
))
; a and be must not be 1
(assertion
  (not (= a (_ bv1 32))))
(assertion
  (not (= b (_ bv1 32))))

(check-sat)
(get-value (a))
(get-value (b))
```

Boolector API

```
BtorExp* a,b;
a = boolector_var(btor, bw, "a");
b = boolector_var(btor, bw, "b");

boolector_assert(btor,boolector_eq( btor,
 boolector_mul(btor,
                                       , a, bw),
                                       , b, bw)),
                                       (btor, 1234567, 2*bw)


boolector_assert( btor, boolector_ne(btor, a,
  boolector_unsigned_int(btor, 1, bw)) );
boolector_assert( btor, boolector_ne(btor, b,
  boolector_unsigned_int(btor, 1, bw)) );

if( boolector_sat( btor ) == BOOLECTOR_SAT )
  boolector_bv_assignment(_btor, a),
  boolector_bv_assignment(_btor, b);
```

This example has memory leaks.
Boolector requires explicit release of expressions.

Example: Integer Factorization / Prime Test (2)

SMT-Lib 2.0

```
; declare variables
(declare−fun a () ( BitVec 32))

(check−sat)
(get−value (a))
(get−value (b))
```

Solver State

Every (partial) expression needs solver state

```
boolector_eq(btor, ...)
sword.addOperator(...)
Z3_mk_eq(z3, ...)
```

Boolector API

```
BtorExp∗ a,b;
a = boolector_var(btor, bw, "a");
b = boolector_var(btor, bw, "b");

boolector_assert(btor,boolector_eq( btor,
 boolector_mul(btor,
    boolector_uext(btor, a, bw),
    boolector_uext(btor, b, bw)),
 boolector_unsigned_int(btor, 1234567, 2∗bw)
));

boolector_assert( btor, boolector_ne(btor, a,
  boolector_unsigned_int(btor, 1, bw)) );
boolector_assert( btor, boolector_ne(btor, b,
  boolector_unsigned_int(btor, 1, bw)) );

if( boolector_sat( btor ) == BOOLECTOR_SAT )
  boolector_bv_assignment(_btor, a),
  boolector_bv_assignment(_btor, b);
```

Example: Integer Factorization / Prime Test (2)

SMT-Lib 2.0

```
; declare variables
(declare-fun a () (  BitVec 32))
```

Solver State

Every (partial) expression needs solver state

```
boolector_eq(btor, ...)
sword.addOperator(...)
Z3_mk_eq(z3, ...)
```

```

(check-sat)
(get-value (a))
(get-value (b))
```

metaSMT

```

bitvector a=new_bitvector(bw);
bitvector b=new_bitvector(bw);

assertion( ctx, equal(
      bvmul(
        zero_extend(bw, a),
        zero_extend(bw ,b)),
      bvuint(1234567, 2*bw)
));

assertion( ctx,
    nequal(a, bvuint(1,bw)) );
assertion( ctx,
    nequal(b, bvuint(1,bw)) );

if( solve( ctx) )
  read_value ( ctx, a ),
  read_value ( ctx, b );
```

Problems so far

- Solver specific API or SMT-file handling.
- Series of API calls instead of clear SMT expressions.
- Different APIs or SMT compliance issues for different solvers.

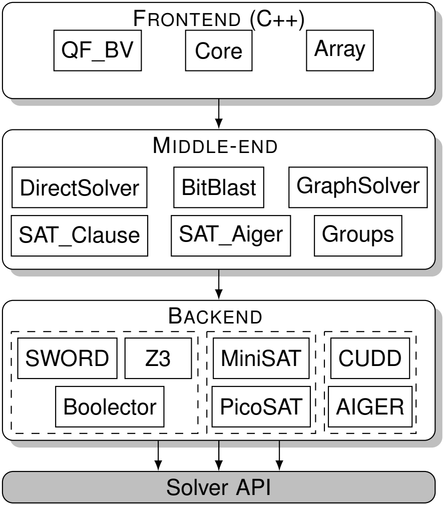
Design Goals

metaSMT . . .

- . . . provides an unified interface to different SMT solvers.
- . . . uses C/C++ interface of the solvers where available.
- . . . makes common/repetitive tasks easy.
- . . . is extensible with new logics, solvers and APIs.
- . . . is customizable for a specific purpose.

Architecture

- Three layer architecture
- Frontend: input languages
- Middle-End: Transformation, representation, APIs and optimization.
- Backend: Solvers, formal engines
- Context ⇒ a metaSMT configuration



Syntax (Commands)

solve	check the satisfiability of an instance.
assertion	add an expression to the instance.
assumption	add an expression to the instance (only for the next call to solve).
read_value	get the assignment of a variable
evaluate	get a run-time respresentation from the backend.

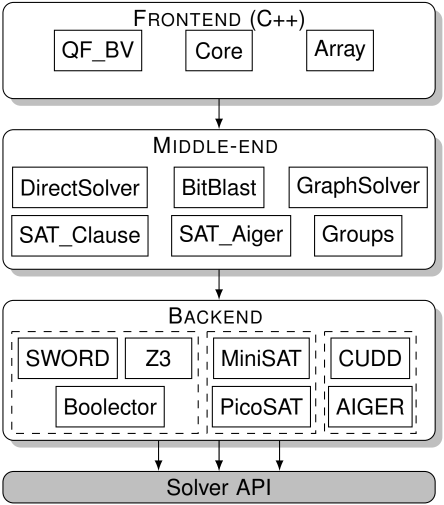
Syntax (Core Logic)

prediate	the type of a boolean variable.
True, False	Boolean constants.
new_variable	create a new boolean variable.
And, Or, Not, Implies	Boolean operations.
Nand, Nor, Xor, Xnor	Mor Boolean operations
Equal, Nequal	Compare to expressions (of same type)
Ite	If Then Else, Then and Else can be any type

Syntax (Bit-Vector)

bitvector	the type of a boolean variable.
new_bitvector(n)	create a new bitvector variable with n bits.
bvuint, bvsint, bit0, . . .	bitvector constants
bvand, bvor, bvnot, . . .	bitwise operations.
bvadd, bvmul, bvsub,. . .	arithmetic operations
bvult, bvsle, . . .	comparison operation
extract, concat, *_extend, . . .	bitvector Structure operations

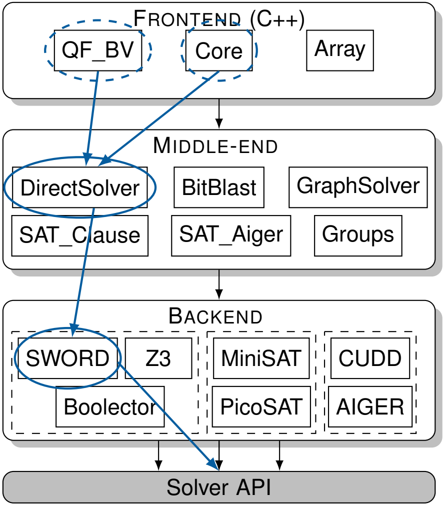
Contexts


FRONTEND (C++)
QF_BV
Core
Array
MIDDLE-END
DirectSolver
BitBlast
GraphSolver
SAT_Clause
SAT_Aiger
Groups
BACKEND
SWORD
Z3
Boolector
MiniSAT
PicoSAT
CUDD
AIGER
Solver API

Contexts

`DirectSolver<SWORD>`

- *Direct Evaluation*
- Pass all expressions directly to the backend.
- No optimizations, no intermediate representation.



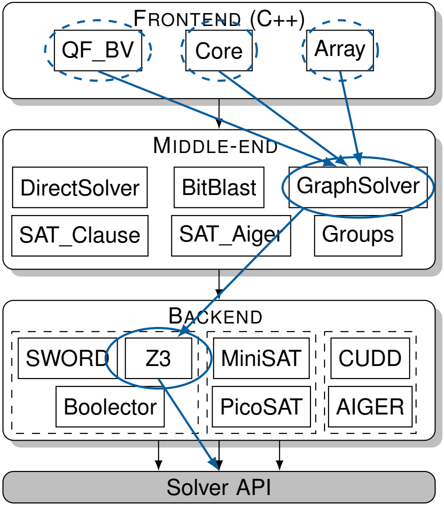
Contexts

DirectSolver<SWORD>

- *Direct Evaluation*
- Pass all expressions directly to the backend.
- No optimizations, no intermediate representation.

GraphSolver<Z3>

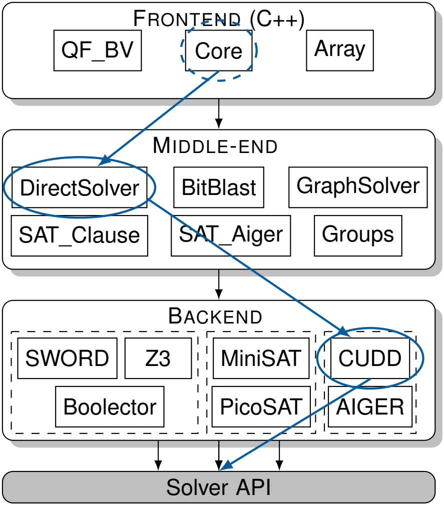
- Intermediate representation
- Collapse common subexpression before passing to the backend



Contexts

`DirectSolver< CUDD >`

- Direct Evaluation
- Only supports core logic.



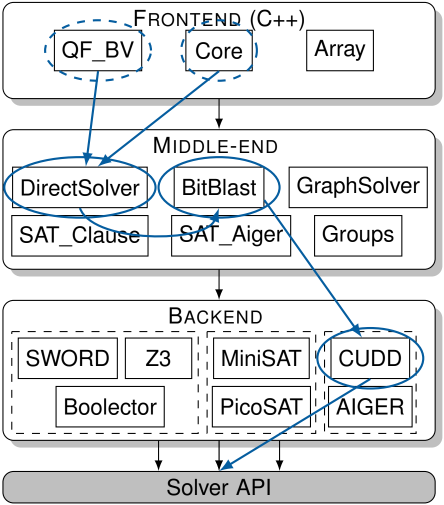
Contexts

`DirectSolver< CUDD >`

- Direct Evaluation
- Only supports core logic.

`DirectSolver<BitBlast< CUDD > >`

- Direct Evaluation
- Emulates QF_BV Logic



Explicit solver APIs

Remark

- Solvers use different APIs and features.
- E.g. Incremental SAT: assumption vs. push/pop vs. none.

Explicit solver APIs

Remark

- Solvers use different APIs and features.
- E.g. Incremental SAT: assumption vs. push/pop vs. none.

Proposed Solution

- Backends/contexts declare features they support.
- If possible Emulation for unsupported features is provided by metaSMT.
- Define a *command* interface to pass arbitrary API commands.
- Check that the Contexts support the API at compile time.

Explicit solver APIs (Example)

- Z3 and SMT-Lib 2.0 support the assertion-stack API
- API functions `push` and `pop`
- Required interface: `stack_api`
- Stack emulation provided for assumption based backends.

```
struct stack_api {};
```

Explicit solver APIs (Example)

- Z3 and SMT-Lib 2.0 support the assertion-stack API
- API functions `push` and `pop`
- Required interface: `stack_api`
- Stack emulation provided for assumption based backends.

```
struct stack_api {};

template<>
struct supports< Z3_Backend,
    stack_api>
: boost::mpl::true_ {};
```

Explicit solver APIs (Example)

- Z3 and SMT-Lib 2.0 support the assertion-stack API
- API functions `push` and `pop`
- Required interface: `stack_api`
- Stack emulation provided for assumption based backends.

```
struct stack_api {};

template<>
struct supports< Z3_Backend,
    stack_api>
: boost::mpl::true_ {};

struct stack_push
{ typedef void result_type; };
struct stack_pop
{ typedef void result_type; };
```

Explicit solver APIs (Example)

- Z3 and SMT-Lib 2.0 support the assertion-stack API
- API functions `push` and `pop`
- Required interface: `stack_api`
- Stack emulation provided for assumption based backends.

```
struct stack_api {};

template<>
struct supports< Z3_Backend,
    stack_api>
: boost::mpl::true_ {};

struct stack_push
{ typedef void result_type; };
struct stack_pop
{ typedef void result_type; };

template <typename Context >
typename boost::enable_if<
  supports<Context, stack_api>
>::type
push( Context & ctx, unsigned N
    =1) {
  ctx.command(stack_push(), N);
}
```

Explicit solver APIs (Example)

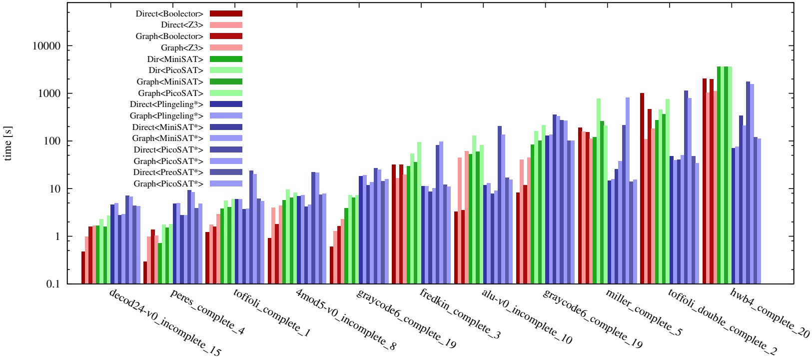
```
class Z3_Backend {
  ...
void command(stack_push const&,
                unsigned n) {
  while (howmany > 0) {
    Z3_push(z3_);
    --howmany;
  }
}

void command(stack_pop const&,
                unsigned n) {
  Z3_pop(z3_, howmany);
}

};
```

```
struct stack_api {};

template<>
struct supports< Z3_Backend,
    stack_api>
: boost::mpl::true_ {};

struct stack_push
{ typedef void result_type; };
struct stack_pop
{ typedef void result_type; };

template <typename Context >
typename boost::enable_if<
  supports<Context, stack_api>
>::type
push( Context & ctx, unsigned N
    =1) {
  ctx.command(stack_push(), N);
}
```

Evaluation

- Experiment on RevKit synthesis algorithms for reversible and quantum circuits circuits.
- Replace custom abstraction by metaSMT.
- More solvers available (previously only 2 implemented).
- Run extensive comparison with 16 metaSMT contexts.
- How scalable are these contexts?
- All results are very easily obtained by using metaSMT.
- A single switch to choose a different solver
- Contexts based on incremental SMT, incremental SAT and file-based SAT backends with DirectSolver as well as GraphSolver.

Comparison



Direct<Boolector>
Direct<Z3>
Graph<Boolector>
Graph<Z3>
Dir<MiniSAT>
Dir<PicoSAT>
Graph<MiniSAT>
Graph<PicoSAT>
Direct<Plingeling*>
Graph<Plingeling*>
Direct<MiniSAT*>
Graph<MiniSAT*>
Direct<PicoSAT*>
Graph<PicoSAT*>
Direct<PreoSAT*>
Graph<PicoSAT*>
time [s]
10000
1000
100
10
1
0.1
decod24-v0_incomplete_15
peres_complete_4
toffoli_complete_1
4mod5-v0_incomplete_8
graycode6_complete_19
fredkin_complete_3
alu-v0_incomplete_10
graycode6_complete_19
miller_complete_5
toffoli_double_complete_2
hwb4_complete_20

Features

A summary of current and future features in metaSMT. Several major additions since the paper submission.

- In the Paper
- New
- Planned, work in progress

Published Features

- Groups
- SMT backends: Boolector, SWORD, Z3
- SAT backends: MiniSAT, PicoSat, CNF-files
- CUDD backend
- AIG based (SAT) representation
- Graph based representation

New Features

- Cardinality Constraints
- weighted BDD (solution distribution)
- Multi-Threaded (2, portfolio approach)
- Explicit APIs
- Stack (emulation)

Planned Features

- Multi-Threaded (arbitrary many) and Multi-Process
- Python bindings
- SMT 2 input Parser

Conclusions

- Lower barrier of entry
- We find it is easier to write SMT based algorithms, even when you get different solvers for *free*.
- metaSMT as abstraction layer let easy to evaluate different contextes in term of optmization by very low programming effort.
- Use solvers that are not SMT-Lib 2.0 compliant with a unified interface.

metaSMT

`http://www.informatik.uni-bremen.de/agra/eng/metasmt.php`

Bibliography

[aig] Aiger.
`http://fmv.jku.at/aiger/`.

[BB09] R. Brummayer and A. Biere.
Boolector: An efficient SMT solver for bit-vectors and arrays.
In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 174–177, 2009.

[Bie08] Armin Biere.
Picosat essentials.
JSAT, 4(2-4):75–97, 2008.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner.
Z3: An efficient smt solver.
In *TACAS*, pages 337–340, 2008.

[HFF+11] F. Haedicke, S. Frehse, G. Fey, D. Große, and R. Drechsler.
metaSMT: Focus on your application not on solver integration.
In *DIFTS'11: 1st International workshop on design and implementation of formal tools and systems*, 2011.

[Som09] F. Somenzi.
CUDD: CU Decision Diagram Package Release 2.4.1.
University of Colorado at Boulder, 2009.

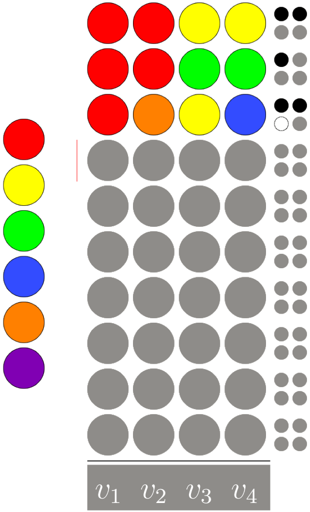
[WFG+07] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler.
Sword: A SAT like prover using word level information.
In *VLSI of System-on-Chip*, pages 88–93, 2007.

[WGHD09] R. Wille, D. Große, F. Haedicke, and R. Drechsler.
SMT-based stimuli generation in the SystemC verification library.
In *Forum on Specification and Design Languages*, 2009.

Slide Index

Example: Mastermind

- Guess a hidden combination of colors
- Hints: #correct color at correct place (black), #correct color at wrong position (white)
- Question: Is there a valid assignment given the hints (which)?
- Good strategy presented by Knuth 1977



Example: Mastermind

Constraint for a single `guess` with `num_correct` correct colors.

```
result_type sum_equal = evaluate( ctx, bvuint(0 , w));
for (unsigned i = 0; i < width; ++i) {
  sum_equal = evaluate( ctx,
      bvadd( sum_equal,
        zero_extend(w-1,
          bvcomp(  v[i], bvuint( guess[i], bw)))
  ));
}

assertion( ctx, Equal( sum_equal,
      bvuint( num_correct, width)));
```

Example: Mastermind

```
// count the colors in the guess
vector<unsigned> by_color ( num_colors, 0);
foreach( unsigned g, guess) { ++by_color[g]; }

vector<result_type> count_by_color( num_colors,
    evaluate( ctx, bvuint(0 , w)));

for (unsigned i = 0; i < width; ++i) {
  // Symbolically count the colors in v (cmp. Knuth '77)
  for (unsigned c = 0; c < num_colors; ++c) {
    count_by_color[c] = evaluate( ctx, Ite(
          And( Equal( v[i], bvuint( c, bw) ),
            bvult(count_by_color[c], bvuint( by_color[c], w)))
          , bvadd( count_by_color[c], bvuint(1, w))
          , count_by_color[c]  ));
}}
result_type sum = evaluate( ctx, bvuint(0 , w));
foreach(result_type r, count_by_color) {
  sum = evaluate(ctx, bvadd( sum, r));
}
assertion( ctx, Equal(sum, bvuint(anywhere, w)) );
```

Lessons learnt

- There is no one API to for all purposes.
- Build an API only if you use it.