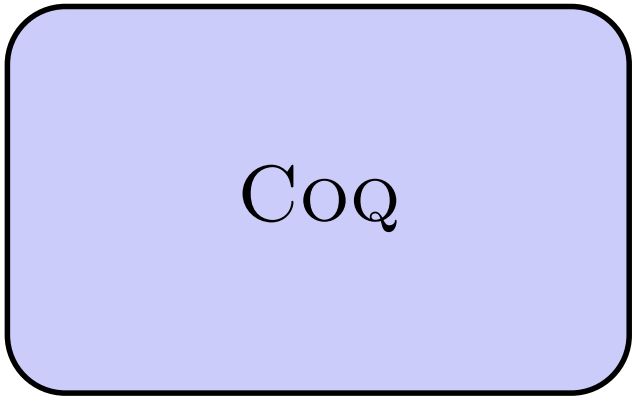
Extending Coq with Imperative Features and its Application to SAT Verification

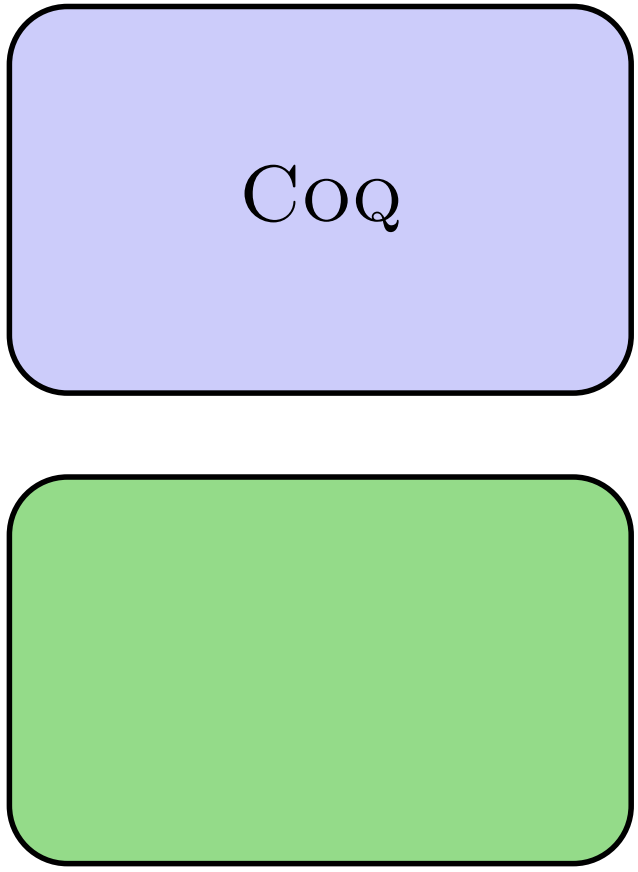
Michaël Armand (INRIA), Benjamin Grégoire (INRIA),
Arnaud Spiwack (École Polytechnique), Laurent Théry (INRIA)

Motivations

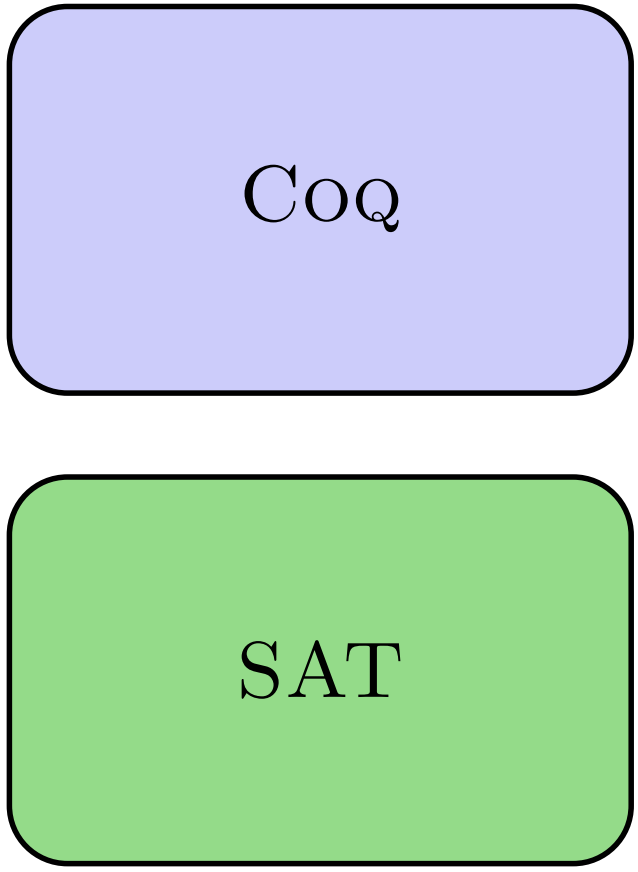
Motivations



Motivations



Motivations



Outline

Outline

Overview of SAT

Outline

Overview of SAT

Computing inside Coq

Outline

Overview of SAT

Computing inside Coq

Adding Machine Integers

Outline

Overview of SAT

Computing inside Coq

Adding Machine Integers

Adding Arrays

Outline

Overview of SAT

Computing inside Coq

Adding Machine Integers

Adding Arrays

Some Benchmarks

SAT

SAT

```
p cnf 50 80
 16   23   42 0
-16   23   42 0
 26   41  -42 0
-26   41  -42 0
 32  -41  -42 0
  6   15  -41 0
 -6   15  -32 0
  1  -32   46 0
 -1  -32   46 0
-15  -41  -46 0
-15  -21  -46 0
-23   33   38 0
-23  -33   38 0
  8   22   33 0
  8   22  -33 0
-22   37  -38 0
 13   36  -37 0
 13  -22  -36 0
-13  -22  -37 0
 11  -23   47 0
 -8   11  -47 0
 -8  -11   39 0
-11   27  -39 0
 -8  -11  -39 0
 -7   26   29 0
 -7  -26   29 0
-13   20   36 0
-13   17   20 0
  5  -17   20 0
  5  -19  -45 0
 -5  -10  -45 0
  6   25   47 0
 -6  -10   25 0
 -2  -27   37 0
-27  -36   40 0
 18   39  -40 0
 -2  -19   31 0
  5   18  -30 0
-31  -43  -50 0
 10  -30   43 0
 10  -41   43 0
 19   21   29 0
 37   42   45 0
-20   27   40 0
-21  -36   48 0
 31  -36  -48 0
  3   -9  -18 0
 16  -40  -47 0
  1  -18   21 0
  2   28   32 0
 -1  -24  -50 0
-12   35   49 0
 -6  -36   45 0
  7   12  -43 0
  7   30  -43 0
 -5    9  -17 0
  3   14   50 0
-12   17  -49 0
 24   34   49 0
 14  -20   24 0
 -9   35  -49 0
 -4  -47   50 0
  4   44  -44 0
 28  -28  -38 0
  2    4  -48 0
-20   35  -44 0
 30  -31  -43 0
-14  -29   35 0
-20   35  -35 0
 19  -22  -24 0
 25  -28   48 0
-14  -34   44 0
  9   20   44 0
 -3    9  -29 0
 17   34  -34 0
 12   48   48 0
-12  -25  -43 0
-25  -31   48 0
 14  -16   49 0
 -3   -4  -35
```

SAT

```
p cnf 50 80
 16   23   42 0
-16   23   42 0
 26   41  -42 0
-26   41  -42 0
 32  -41  -42 0
  6   15  -41 0
 -6   15  -32 0
  1  -32   46 0
 -1  -32   46 0
-15  -41  -46 0
-15  -21  -46 0
-23   33   38 0
-23  -33   38 0
  8   22   33 0
  8   22  -33 0
-22   37  -38 0
 13   36  -37 0
 13  -22  -36 0
-13  -22  -37 0
 11  -23   47 0
 -8   11  -47 0
 -8  -11   39 0
-11   27  -39 0
 -8  -11  -39 0
 -7   26   29 0
 -7  -26   29 0
-13   20   36 0
-13   17   20 0
  5  -17   20 0
  5  -19  -45 0
 -5  -10  -45 0
  6   25   47 0
 -6  -10   25 0
 -2  -27   37 0
-27  -36   40 0
 18   39  -40 0
 -2  -19   31 0
  5   18  -30 0
-31  -43  -50 0
 10  -30   43 0
 10  -41   43 0
 19   21   29 0
 37   42   45 0
-20   27   40 0
-21  -36   48 0
 31  -36  -48 0
  3   -9  -18 0
 16  -40  -47 0
  1  -18   21 0
  2   28   32 0
 -1  -24  -50 0
-12   35   49 0
 -6  -36   45 0
  7   12  -43 0
  7   30  -43 0
 -5    9  -17 0
  3   14   50 0
-12   17  -49 0
 24   34   49 0
 14  -20   24 0
 -9   35  -49 0
 -4  -47   50 0
  4   44  -44 0
 28  -28  -38 0
  2    4  -48 0
-20   35  -44 0
 30  -31  -43 0
-14  -29   35 0
-20   35  -35 0
 19  -22  -24 0
 25  -28   48 0
-14  -34   44 0
  9   20   44 0
 -3    9  -29 0
 17   34  -34 0
 12   48   48 0
-12  -25  -43 0
-25  -31   48 0
 14  -16   49 0
 -3   -4  -35
```

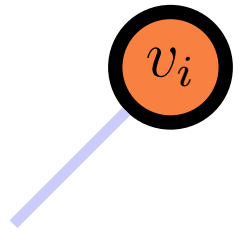

```
p cnf 50 80
 16   23   42 0
-16   23   42 0
 26   41  -42 0
-26   41  -42 0
```

DPLL

DPLL

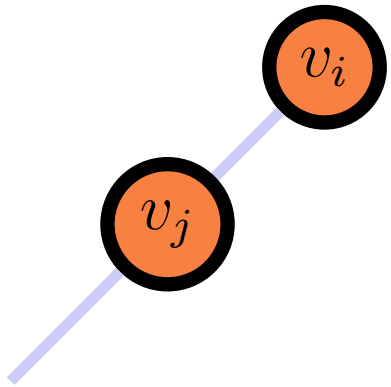
VS

DPLL



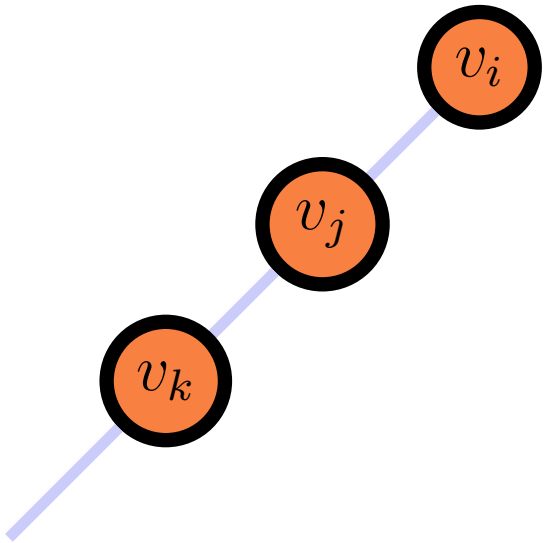
VS + UP

DPLL



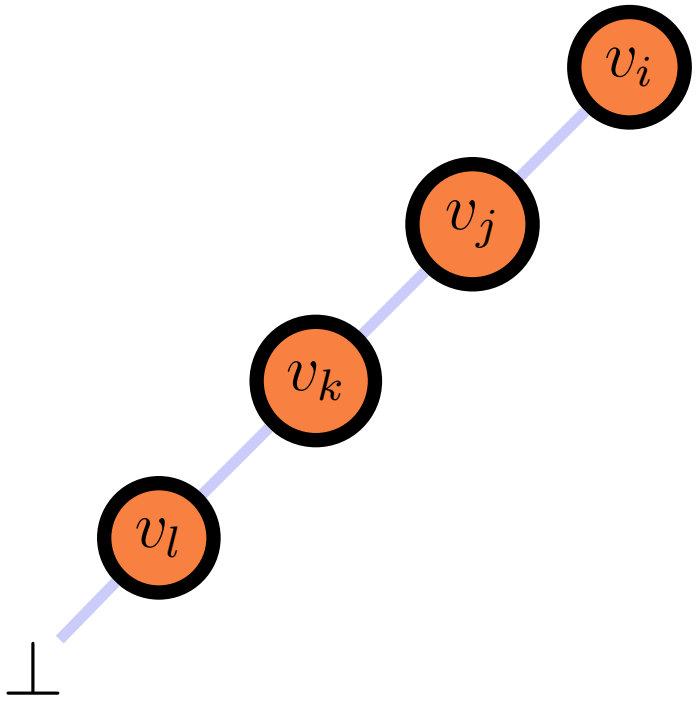
VS + UP

DPLL



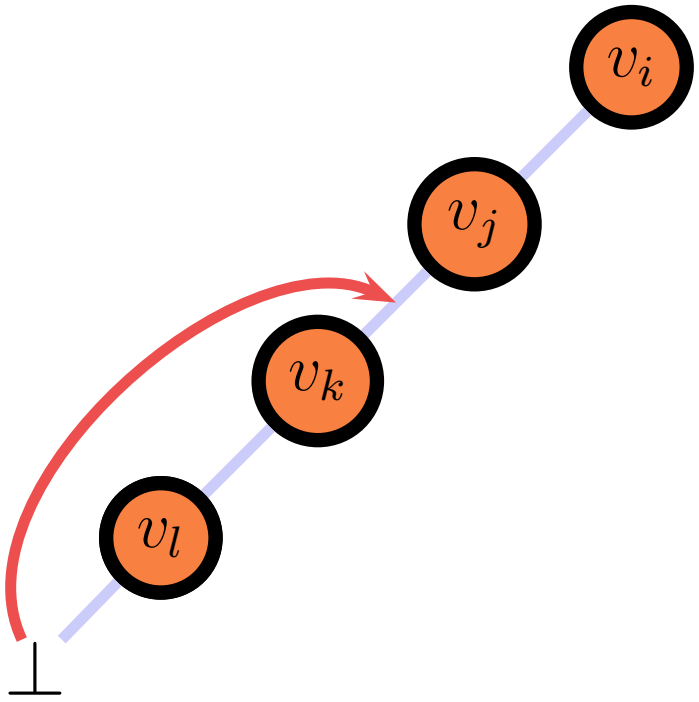
VS + UP

DPLL



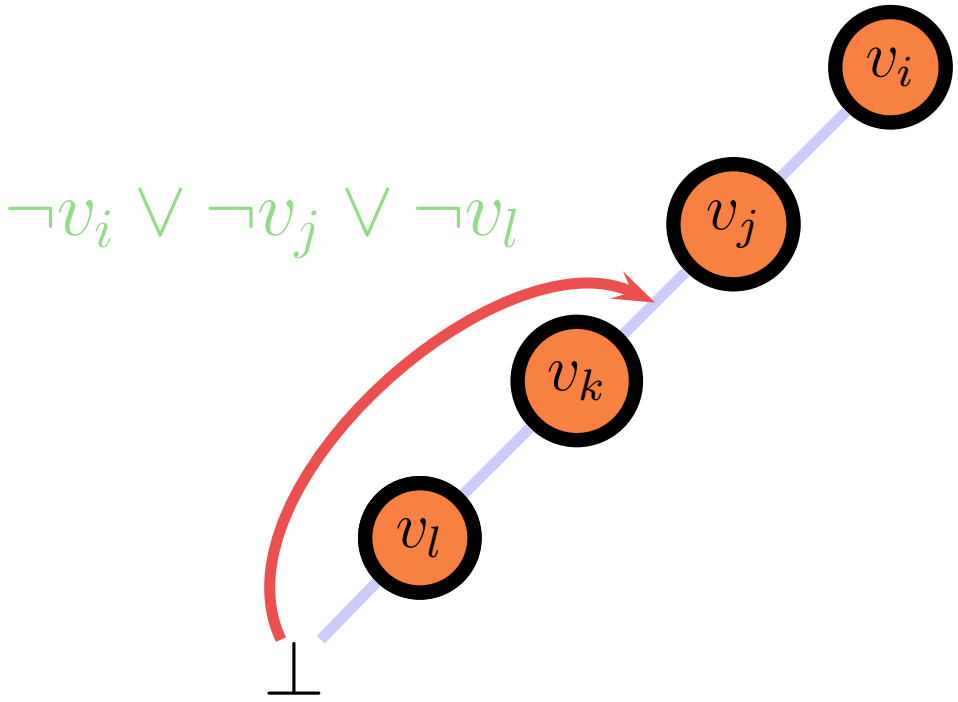
VS + UP

DPLL



VS + UP + BJ

DPLL



VS + UP + BJ + L

Replaying traces

Replaying traces

Resolution: $\dfrac{x \vee C \qquad \neg x \vee C'}{C \vee C'}$

Replaying traces

Resolution: $\dfrac{x \vee C \qquad \neg x \vee C'}{C \vee C'}$

For each conflict clause: a resolution chain

Replaying traces

Resolution: $\dfrac{x \vee C \qquad \neg x \vee C'}{C \vee C'}$

For each conflict clause: a resolution chain

A trace: a sequence of resolution chains

Traces

```
 3  2
 1  0
14 13
21 23
20 19 79
12 11
18 16 15 81 78 80
 8  7
 5  6  9 83  4 76
77 84
80 85
81 85
82 85
78 86
17 88 89
15 87 89
18 91 89 90
```

Traces

```
 3  2
 1  0
14 13
21 23
20 19 79
12 11
18 16 15 81 78 80
 8  7
 5  6  9 83  4 76
77 84
80 85
81 85
82 85
78 86
17 88 89
15 87 89
18 91 89 90
```

Computing inside Coq

Computing inside Coq

Programming Language

$$a := x \mid \lambda x.a \mid a_1\ a_2 \mid C_{I,i}(\vec{a}) \mid \mathsf{case}\ a\ \mathsf{of}\ (\vec{x}_i \Rightarrow a_i)_{1 \leq i \leq |I|}$$

Computing inside Coq

Programming Language

$$a := x \mid \lambda x.a \mid a_1\ a_2 \mid C_{I,i}(\vec{a}) \mid \mathsf{case}\ a\ \mathsf{of}\ (\vec{x}_i \Rightarrow a_i)_{1 \leq i \leq |I|}$$

Reduction

$$(\lambda x.a_1)a_2 \ \Rightarrow\ a_1\{x \leftarrow a_2\}$$

$$\mathsf{case}\ C_{I,j}(\vec{a})\ \mathsf{of}\ (\vec{x}_i \Rightarrow a_i)_{1 \leq i \leq |I|} \ \Rightarrow\ a_j\{\vec{x}_j \leftarrow \vec{a}\}$$

Computing inside Coq

Programming Language

$$a := x \mid \lambda x.a \mid a_1\ a_2 \mid C_{I,i}(\vec{a}) \mid \mathsf{case}\ a\ \mathsf{of}\ (\vec{x}_i \Rightarrow a_i)_{1 \leq i \leq |I|}$$

Strong Reduction

$$\begin{aligned}
(\lambda x.a_1)a_2 &\Rightarrow a_1\{x \leftarrow a_2\} \\
\mathsf{case}\ C_{I,j}(\vec{a})\ \mathsf{of}\ (\vec{x}_i \Rightarrow a_i)_{1 \leq i \leq |I|} &\Rightarrow a_j\{\vec{x}_j \leftarrow \vec{a}\} \\
\Gamma(a) &\Rightarrow \Gamma(a') \quad \text{if } a \Rightarrow a'
\end{aligned}$$

Computing inside Coq

Symbolic variable $\tilde{x}$ + accumulator $[k]$

Computing inside Coq

Symbolic variable $\tilde{x}$ + accumulator $[k]$

$$
\begin{aligned}
(\lambda x.b)\ v &\longrightarrow b\{x \leftarrow v\} \\
[k]\ v &\longrightarrow [k\ v] \\
\text{case } C_{I,j}(\vec{v}) \text{ of } (\vec{x}_i \Rightarrow b_i)_{1\le i\le |I|} &\longrightarrow b_j\{\vec{x}_j \leftarrow \vec{v}\} \\
\text{case } [k] \text{ of } (\vec{x}_i \Rightarrow b_i)_{1\le i\le |I|} &\longrightarrow [\text{case } k \text{ of } (\vec{x}_i \Rightarrow b_i)_{1\le i\le |I|}] \\
\Gamma_v(b) &\longrightarrow \Gamma_v(b') \quad \text{if } b \longrightarrow b'
\end{aligned}
$$

where $\Gamma_v(\bullet) ::= \bullet\ v \mid b\ \bullet \mid C_{I,i}(\vec{b}\ \bullet\ \vec{v}) \mid \text{case } \bullet \text{ of } (\vec{x}_i \Rightarrow b_i)_{1\le i\le |I|}$

Computing inside Coq

$$
\begin{array}{lcl}
\mathcal{N}(b) & = & \mathcal{R}(\mathcal{V}(b)) \\
\mathcal{R}(\lambda x.b) & = & \lambda y.\mathcal{N}((\lambda x.b)\ [\tilde{y}]) \quad (y \text{ fresh}) \\
\mathcal{R}(C_{I,i}(\vec{v})) & = & C_{I,i}(\mathcal{R}(\vec{v})) \\
\mathcal{R}([k]) & = & \mathcal{R}'(k) \\
\mathcal{R}'(k\ v) & = & \mathcal{R}'(k)\ \mathcal{R}(v) \\
\mathcal{R}'(\tilde{x}) & = & x \\
\mathcal{R}'(\mathsf{case}\ k\ \mathsf{of}\ (\vec{x}_i \Rightarrow b_i)_{1\leq i\leq |I|}) & = & \mathsf{case}\ \mathcal{R}'(k)\ \mathsf{of}\ (\vec{x}_i \Rightarrow \mathcal{N}(b\ C_i([\ \vec{\tilde{y}_i}\ ])))_{1\leq i\leq |I|}
\end{array}
$$

where $b = \lambda x.\mathsf{case}\ x\ \mathsf{of}\ (\vec{x}_i \Rightarrow b_i)_{1\leq i\leq |I|}$

and $\vec{y}_i$ are sequences of fresh variables with $|\vec{y}_i| = |\vec{x}_i|$

Computing inside Coq

Byte Code (ZAM)

$$
\begin{array}{lcl}
\mathcal{N}(b) & = & \mathcal{R}(\mathcal{V}(b)) \\
\mathcal{R}(\lambda x.b) & = & \lambda y.\mathcal{N}((\lambda x.b)\ [\tilde{y}]) \quad (y \text{ fresh}) \\
\mathcal{R}(C_{I,i}(\vec{v})) & = & C_{I,i}(\mathcal{R}(\vec{v})) \\
\mathcal{R}([k]) & = & \mathcal{R}'(k) \\
\mathcal{R}'(k\ v) & = & \mathcal{R}'(k)\ \mathcal{R}(v) \\
\mathcal{R}'(\tilde{x}) & = & x \\
\mathcal{R}'(\mathsf{case}\ k\ \mathsf{of}\ (\vec{x}_i \Rightarrow b_i)_{1\le i\le |I|}) & = & \mathsf{case}\ \mathcal{R}'(k)\ \mathsf{of}\ (\vec{x}_i \Rightarrow \mathcal{N}(b\ C_i([\ \vec{\tilde{y}_i}\ ])))_{1\le i\le |I|}
\end{array}
$$

where $b = \lambda x.\mathsf{case}\ x\ \mathsf{of}\ (\vec{x}_i \Rightarrow b_i)_{1\le i\le |I|}$

and $\vec{y}_i$ are sequences of fresh variables with $|\vec{y}_i| = |\vec{x}_i|$

Adding Machine Integers

Adding Machine Integers

Minimal change

Adding Machine Integers

Minimal change

$$\mathbf{I}(b_1,\, b_2\, \ldots, b_{31})$$

Adding Machine Integers

Minimal change :

Two representations for numbers in bytecode

$$\mathbf{I}(b_1,\, b_2\, \ldots, b_{31}) \;\Rightarrow\; m$$

Adding Machine Integers

Minimal change :

Two representations for numbers in bytecode

$$\mathbf{I}(b_1, b_2 \ldots, b_{31}) \Rightarrow m$$

Two sets of functions:

Adding Machine Integers

Minimal change :

Two representations for numbers in bytecode

$$\mathbf{I}(b_1, b_2 \ldots, b_{31}) \Rightarrow m$$

Two sets of functions:

$$m_1 + m_2 \Rightarrow m_1 +_n m_2$$

Adding Machine Integers

Minimal change :

Two representations for numbers in bytecode

$$\mathbf{I}(b_1, b_2 \ldots, b_{31}) \Rightarrow m$$

Two sets of functions:

$$m_1 + m_2 \Rightarrow m_1 +_n m_2$$

$$v_1 + v_2 \Rightarrow v_1 +_C v_2$$

Adding Machine Integers

Minimal change :

Two representations for numbers in bytecode

$$\mathbf{I}(b_1,\, b_2\, \ldots, b_{31}) \;\Rightarrow\; m$$

Two sets of functions:

$$m_1 + m_2 \;\Rightarrow\; m_1 +_n m_2$$

$$v_1 + v_2 \;\Rightarrow\; v_1 +_C v_2$$

Readback

$$\mathcal{R}(m) \;=\; \mathbf{I}(b_1,\, b_2\, \ldots, b_{31})$$

Adding Arrays

Adding Arrays

Purely Functional Interface

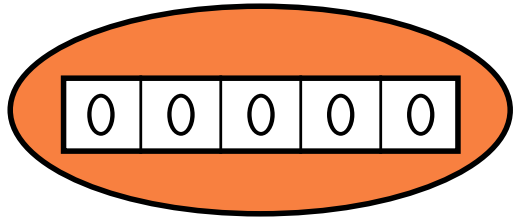
```
make: forall A, int31 → A → array A
length: forall A, array A → int31
get: forall A, array A → int31 → A
set: forall A, array A → int31 → A → array A
```

Persistent Arrays

```
let a := make 5 0 in
```

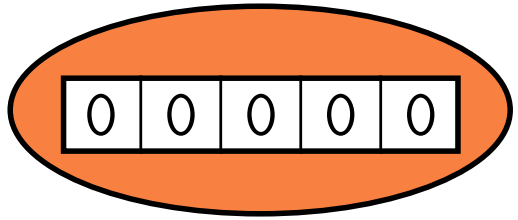
Persistent Arrays

```
let a := make 5 0 in
```



Persistent Arrays

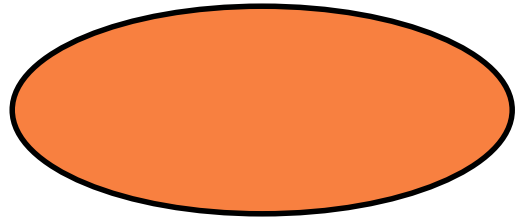
```
let a := make 5 0 in
```



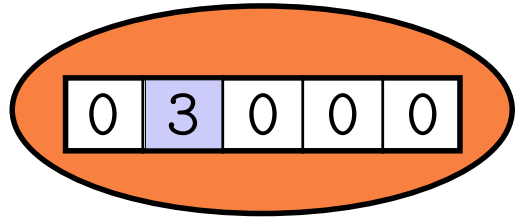

```
let b := set a 1 3 in
```

Persistent Arrays

```
let a := make 5 0 in
```



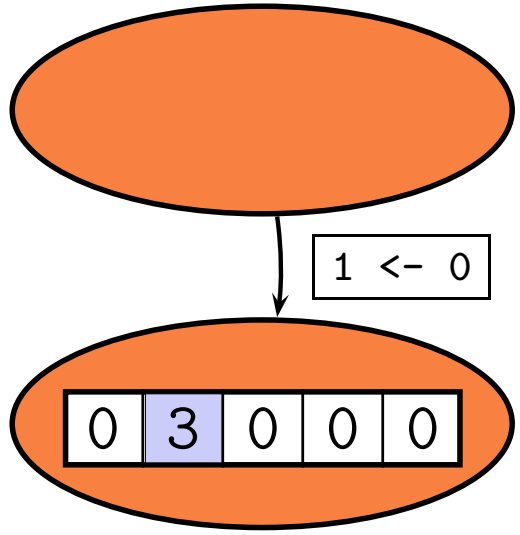
```
let b := set a 1 3 in
```



Persistent Arrays

```
let a := make 5 0 in
```

```
let b := set a 1 3 in
```

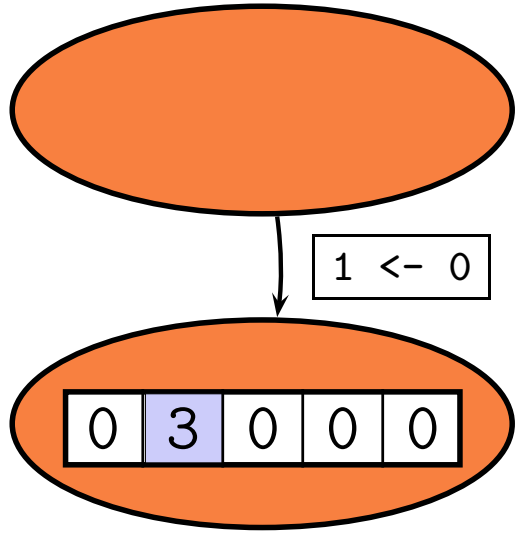


Persistent Arrays

```
let a := make 5 0 in

let b := set a 1 3 in

let c := set b 1 2 in
```

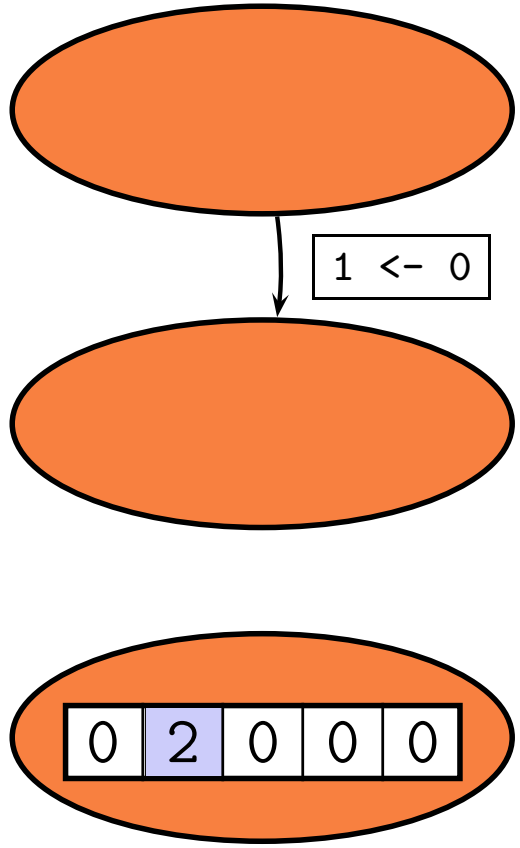


Persistent Arrays

```
let a := make 5 0 in
```

```
let b := set a 1 3 in
```

```
let c := set b 1 2 in
```

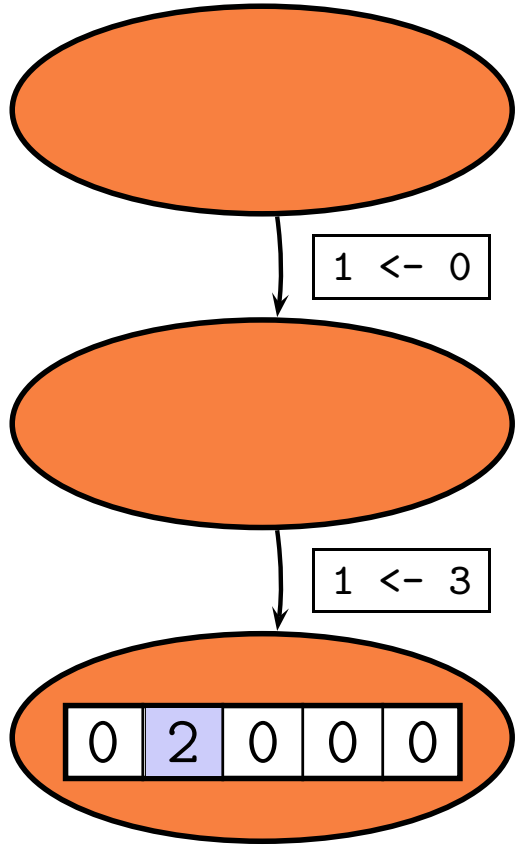


Persistent Arrays

```
let a := make 5 0 in

let b := set a 1 3 in

let c := set b 1 2 in
```



SAT

SAT

Take ZVERIFY code and Coquify it.

SAT

Take ZVERIFY code and Coquify it.

Code: 230 lines Proof: 1190 lines

Benchmark

Problem	Vars	Clauses	zChaff	zVerify
dubois50	150	400	0.00	0.01
barrel5	1407	5383	0.50	0.07
barrel6	2306	8931	1.74	0.14
barrel7	3523	13765	5.20	0.26
6pipe	15800	394739	42.21	2.86
longmult14	7176	22390	408.55	7.34
hole11	132	738	14.82	0.90
hole12	156	949	144.49	4.85
hole13	182	1197	5048.23	-

Benchmarks

Problem	zVerify	Coq	Isabelle
dubois50	0.01	0.04	0.04
barrel5	0.07	0.47	1.10
barrel6	0.14	1.15	1.74
barrel7	0.26	1.45	5.20
6pipe	2.86	24.73	-
longmult14	7.34	73.63	-
hole11	0.90	9.51	9.36
hole12	4.85	58.28	61.10
hole13	-	88.15	-

Benchmarks

Problem	zVerify	Coq	Cert	Typing	Check
dubois50	0.01	0.04	0.00	0.02	0.02
barrel5	0.07	0.47	0.00	0.32	0.15
barrel6	0.14	1.15	0.08	0.62	0.45
barrel7	0.26	1.45	0.17	0.80	0.48
6pipe	2.86	24.73	0.98	13.92	9.83
longmult14	7.34	73.63	7.72	27.07	38.84
hole11	0.90	0.41	2.96	6.14	1.39
hole12	4.85	58.28	2.44	18.47	37.38
hole13	-	1068.30	88.15	387.44	592.72

Benchmarks

Problem	Check	Array	Parray
dubois50	0.02	0.00	0.00
barrel5	0.15	0.00	0.00
barrel6	0.45	0.06	0.14
barrel7	0.48	0.07	0.16
6pipe	9.83	2.05	4.74
longmult14	38.84	9.10	16.92
hole11	6.14	1.39	2.89
hole12	37.38	13.12	16.88
hole13	592.72	183.47	275.14

Conclusions

Conclusions

Light way of introducing unpure features

Conclusions

Light way of introducing unpure features

Relatively efficient computation

Conclusions

Light way of introducing unpure features

Relatively efficient computation

Compact representation for large objects