Offline SMT for Arrays

Robert Brummayer and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

Alpine Verification Meeting 2008

Semmering, Austria

May 19th, 2008

Satisfiability Modulo Theories (SMT)

R. Brummayer, FMV, JKU Linz

- Generalization of the Boolean Satisfiability Problem (SAT)
- Satisfiability with respect to background theories
- Software and Hardware verification
- SMT Solvers
  - Z3, CVC3, STP, Barcelogic, Boolector, MathSAT, Spear, ...
- Theories
  - Fragments of first-order logic (typically decidable)
  - For example fixed-size Bit-vectors , extensional Arrays

Software Verification Example “next-power-of-two”

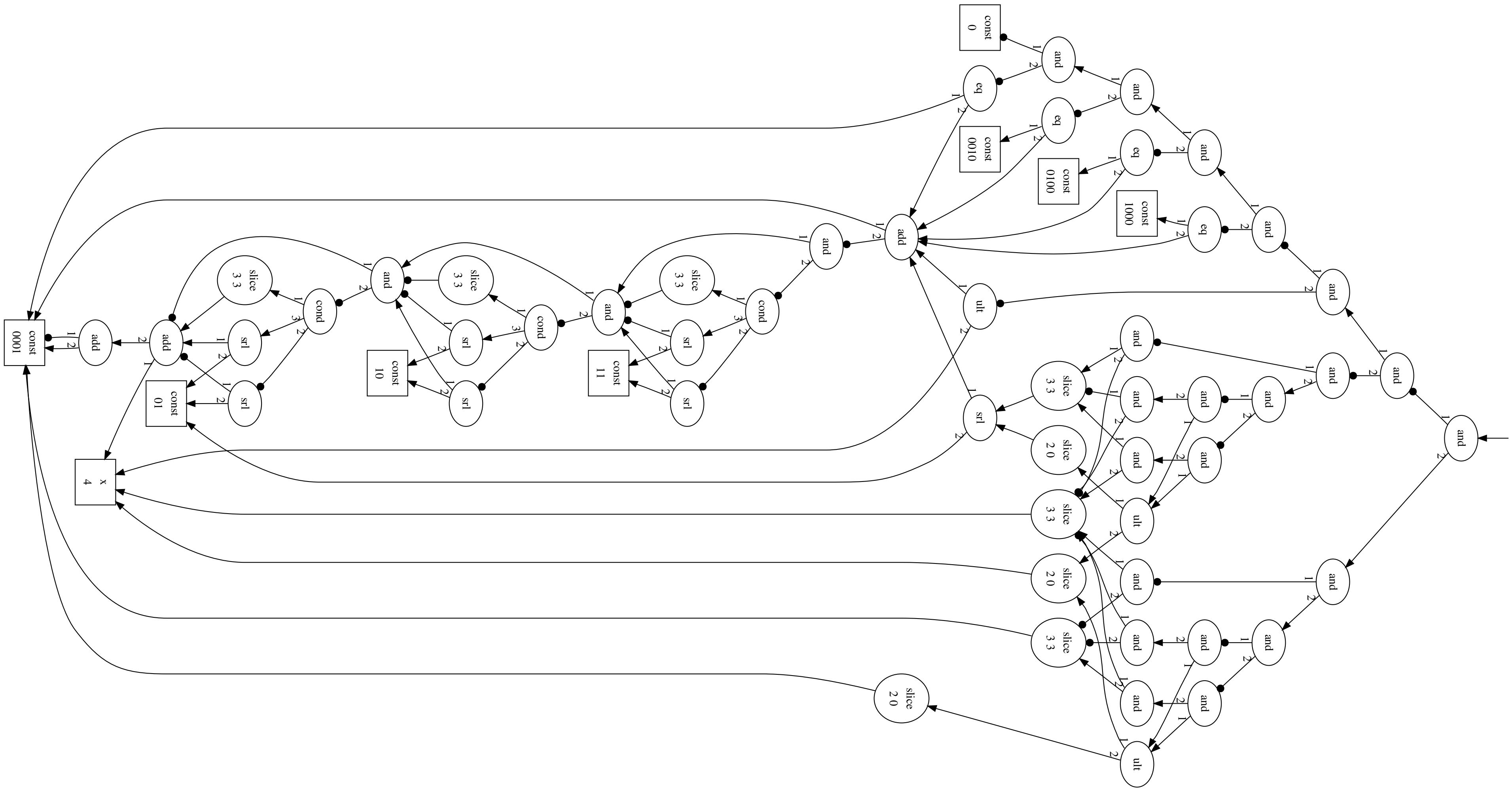
R. Brummayer, FMV, JKU Linz

```
int next_power_of_two (int x)
{
  int i;
  x--;
  for (i = 1; i < sizeof (int) * 8; i = i * 2)
    x = x | (x >> i)
  return x + 1;
}
```

- next_power_of_two (5) = 8, next_power_of_two (8) = 8, ...
- From the book “Hacker’s Delight” [Warren02]
- Do you trust this algorithm?

Verification instance for bit-width = 4

R. Brummayer, FMV, JKU Linz



- Theory of Arrays [McCarthy62]

$$
\begin{array}{lllll}
\text{(A1)} & a = b \wedge i = j & \Rightarrow & read(a,i) = read(b,j) \\
\text{(A2)} & i = j & \Rightarrow & read(write(a,i,e),j) = e \\
\text{(A3)} & i \neq j & \Rightarrow & read(write(a,i,e),j) = read(a,j)
\end{array}
$$

- With (A1) to (A3) we cannot handle array inequalities

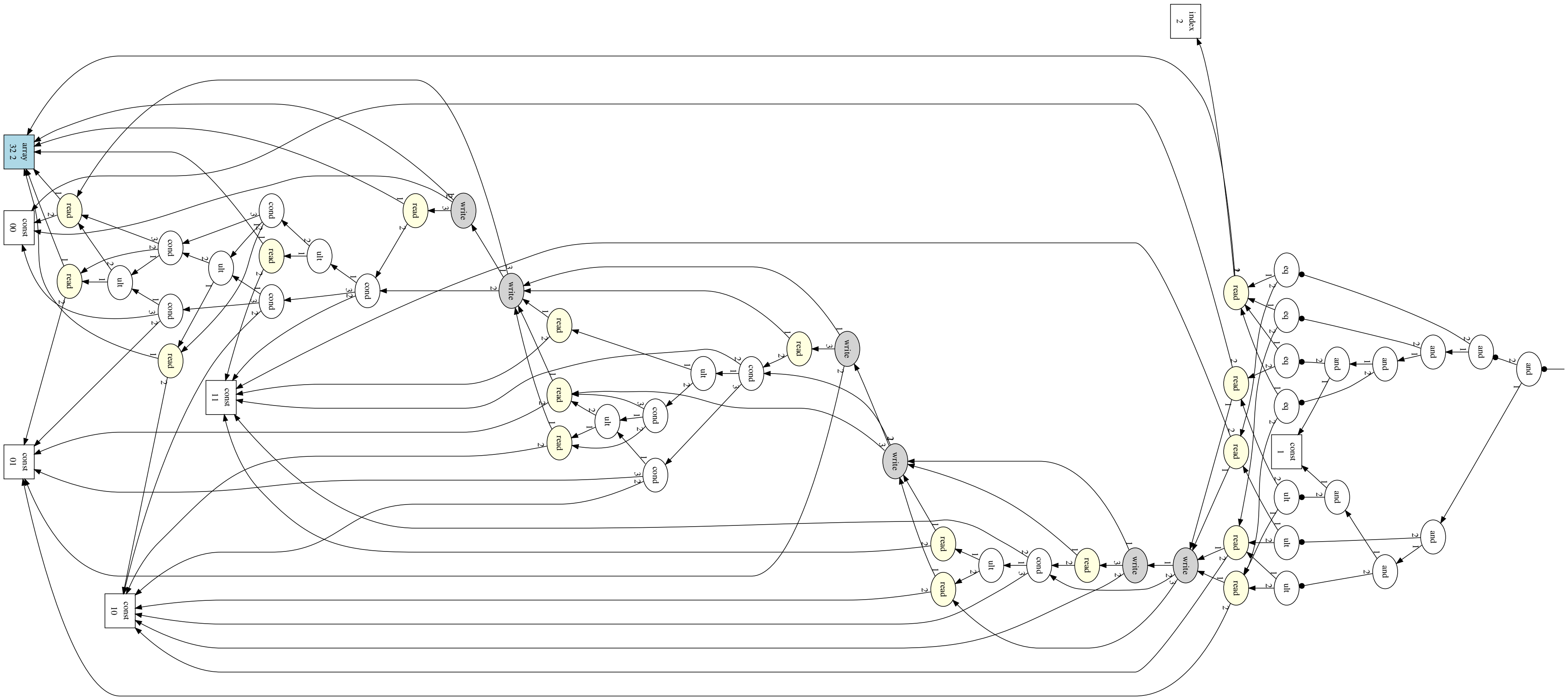
- We need additional axiom of extensionality (A4) resp. (A4′)

$$\text{(A4)} \quad a = b \quad \Leftarrow \quad \forall i\,(read(a,i) = read(b,i))$$

$$\text{(A4}'\text{)} \quad a \neq b \quad \Rightarrow \quad \exists \lambda\,(read(a,\lambda) \neq read(b,\lambda))$$

Verification of Selection Sort for bit-width = 32 and size = 4

R. Brummayer, FMV, JKU Linz



Lemmas on demand (LOD) [DeMoura02]

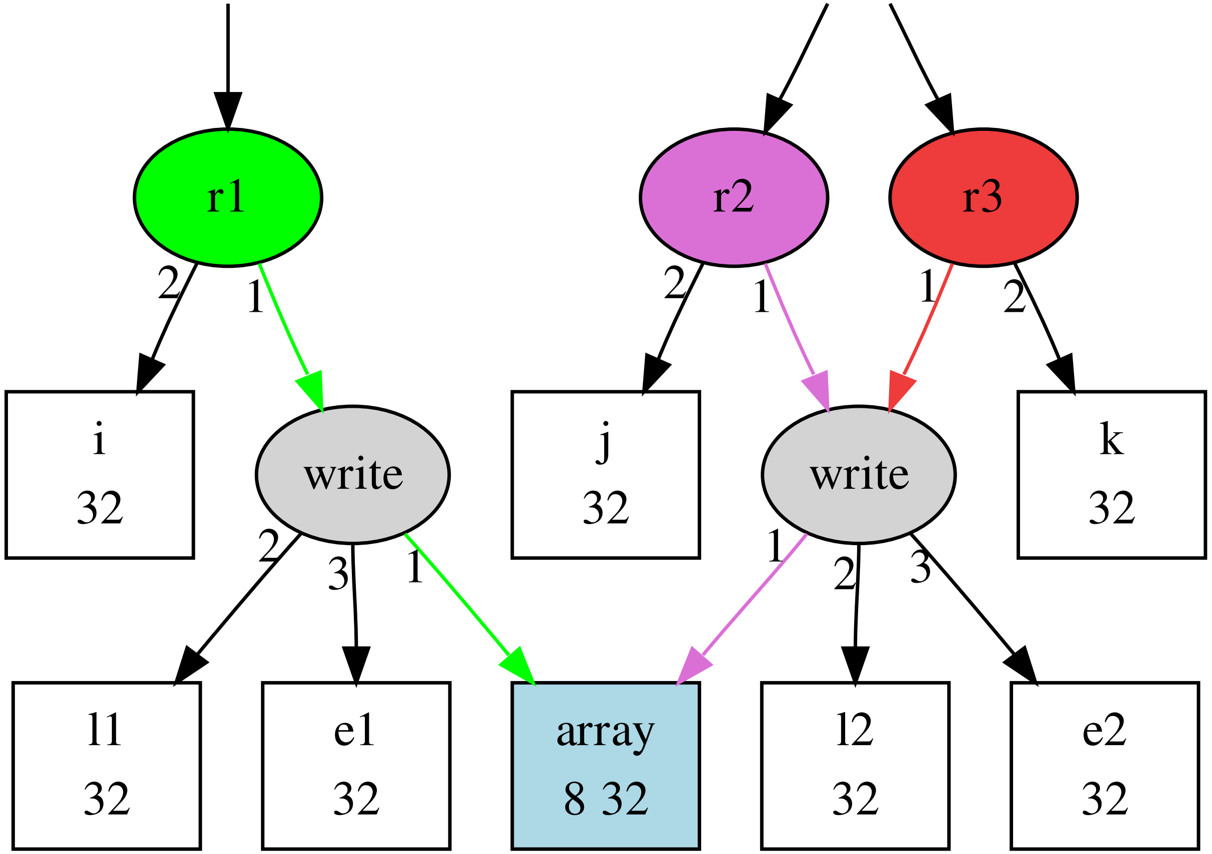
R. Brummayer, FMV, JKU Linz

- Do not translate the whole formula to SAT
- Let SAT solver "guess" solution
  - If SAT solver cannot find a solution, terminate with *unsatisfiable*
- Explicitly check constraints that were not translated to SAT
- If check succeeds then terminate with *satisfiable*
- If check fails
  - Add lemma to refine formula
  - Let SAT solver "guess" a new solution

- Translate bit-vector but not array part of the formula
- Let SAT solver "guess" solution
  - If SAT solver cannot find a solution, terminate with *unsatisfiable*
- Explicitly check Array Axioms A1 to A3
- If check succeeds then terminate with *satisfiable*
- If check fails
  - Add lemma to refine formula
  - Let SAT solver "guess" a new solution

- Propagation-based algorithm
  - Direct application of (A1) to (A3)
- Annotate every array expression with its set of reads ρ
- For every read $read(b, i) \in \rho(write(a, j, e))$
  - (A2): If current assignment $\sigma(i) = \sigma(j)$, check if $\sigma(read(b, i)) = \sigma(e)$
  - (A3): If current assignment $\sigma(i) \neq \sigma(j)$, add read to $\rho(a)$
- Check read congruence (A1) on all array expressions
- Propagation only downwards and can be implemented with DFS or BFS

Example 1



$$\sigma(i) = \sigma(j),\ \sigma(r1) \neq \sigma(r2),\ \sigma(i) \neq \sigma(l1),\ \sigma(j) \neq \sigma(l2),\ \sigma(k) = \sigma(l2),\ \sigma(r3) \neq \sigma(e2)$$

Lemma construction

R. Brummayer, FMV, JKU Linz

- We have found two inconsistent reads r_1 and r_2

- We collect all assignments that has been responsible for propagation

- We add a lemma of the following kind:

  - $i \neq l_1 \wedge j \neq l_2 \wedge \ldots \wedge i = j \quad \Rightarrow \quad r_1 = r_2$

- Lemma for inconsistency of r_1 and r_2 in example 1

  - $i \neq l_1 \wedge j \neq l_2 \wedge i = j \quad \Rightarrow \quad r_1 = r_2$

- Lemma for inconsistency of r_3 and right *write* in example 1

  - $k = l_2 \quad \Rightarrow \quad r_3 = e_2$

Adding equalities on arrays (1/2)

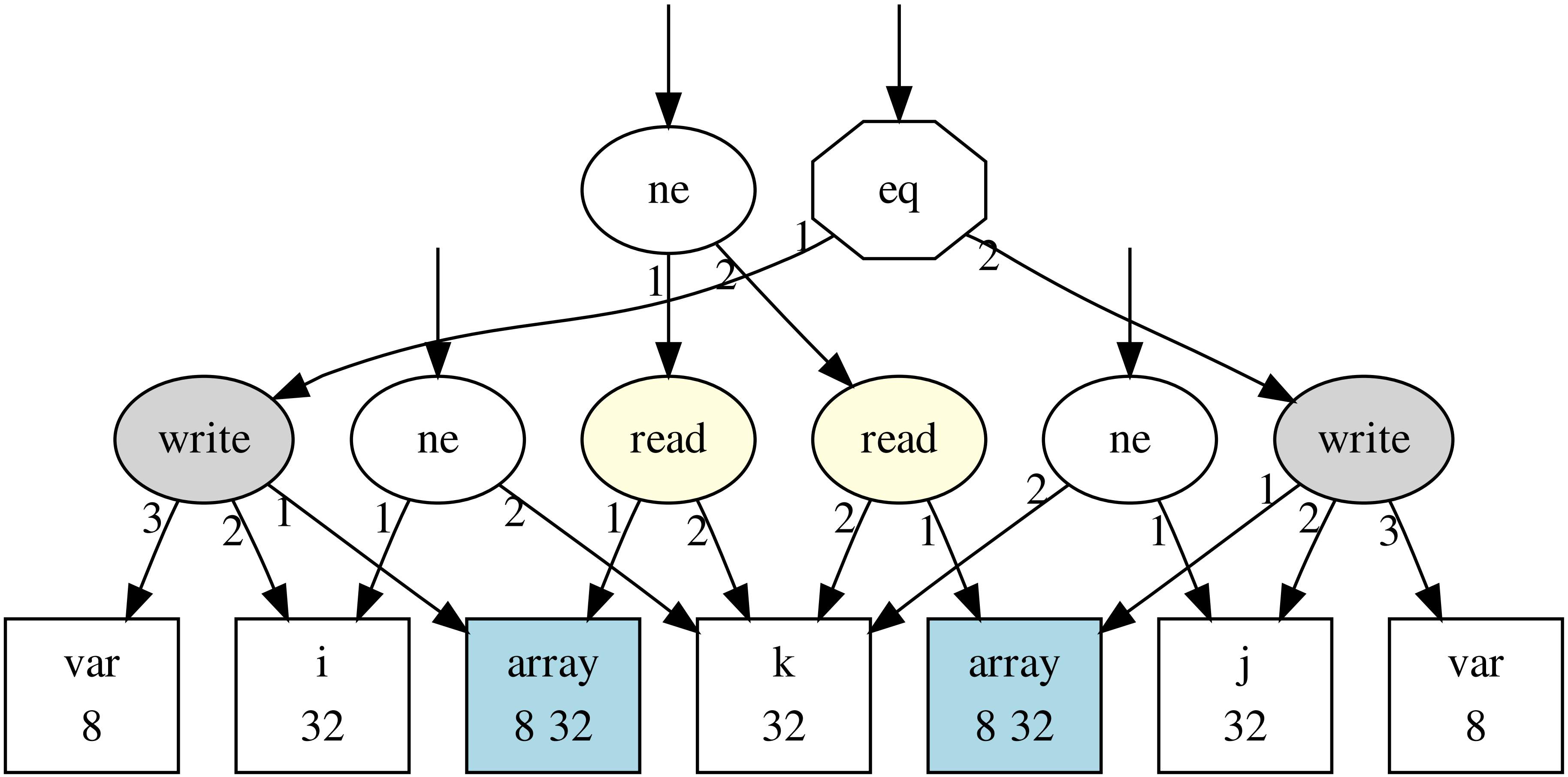
R. Brummayer, FMV, JKU Linz

- For every array equality a = b
  - Introduce fresh Tseitin variable $e_{a,b}$
  - Introduce two virtual reads $read(a,\lambda)$, $read(b,\lambda)$, for a fresh λ
  - Virtual reads are used as witness for array inequality
  - Encode $\bar{e}_{a,b} \quad \Rightarrow \quad read(a,\lambda) \neq read(b,\lambda)$
- If $\sigma(e_{a,b}) = 1$, propagate reads over array equalities
  - Ensures read congruence over equal arrays
  - Propagation can now be cyclic, e.g. $a = b \wedge b = c \wedge c = a$
  - We need fix-point algorithm

Adding equalities on arrays (2/2)

R. Brummayer, FMV, JKU Linz

- We must ensure congruence on write values for equal writes
- For example $write(a,i,e_1) = write(a,i,e_2)$ implies that $e_1 = e_2$
- We can treat every $write(a,i,e)$ as $read(a,i)$, where $read(a,i) = e$
- Propagate writes as reads
- In order to reach every array equality
  - We must also propagate upwards with respect to (A2) and (A3)
  - Only propagate upwards if value has not been overwritten



$$write(a,i,e_1) = write(b,j,e_2) \ \wedge\ i \neq k \ \wedge\ j \neq k \ \wedge\ read(a,k) \neq read(b,k)$$

Conclusion

R. Brummayer, FMV, JKU Linz

- SMT
- Lemmas on demand for Extensional Theory of Arrays
  - In our examples with Bit-vectors, but approach is more general
  - SAT solver is used offline as black box
- Algorithm based on propagation and direct application of array axioms
  - Non-extensional algorithm with DFS or BFS
  - Introducing equality on arrays requires fix-point algorithm
  - Virtual reads as witnesses for array inequalities
- Algorithm implemented in our SMT solver Boolector