Lemmas on Demand for the Extensional Theory of Arrays

Robert Brummayer and Armin Biere

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

SMT 2008

Princeton, New Jersey, USA

July 8th, 2008

Overview

R. Brummayer, FMV, JKU Linz

- Introduction
- Arrays
- Verification example
- Lemmas on demand
- Consistency checking
- Lemma encoding
- Handling equalities on arrays
- Conclusion

- Bit-Vectors
  - Bit-precise modelling of computation
  - Modular arithmetic
- Arrays
  - Modelling of memories
  - Main memory in software
  - Memory components in hardware, e.g. registers

Combination of both: bit-precise modelling and reasoning

- Scenarios
  - Pipelined vs. non-pipelined hardware
  - Equivalence-check memory semantics of software
- Let both algorithm start on the same array
- Symbolically execute both algorithms
- Finally, check whether the resulting arrays are equal or not

- 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

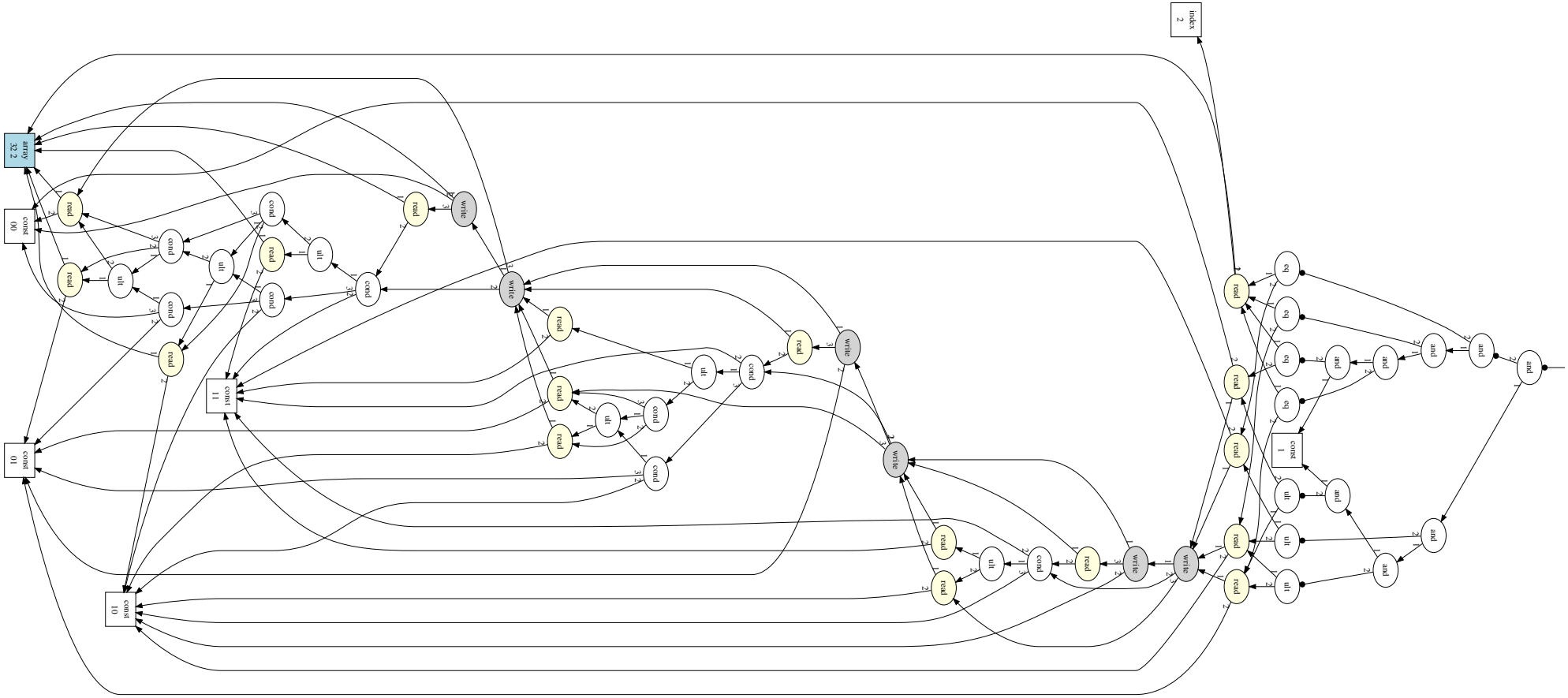
- We need additional axiom of extensionality (A4) resp. (A4′)

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

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

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

procedure lemmas-on-demand (ϕ)
  encode-to-sat (ϕ)
  loop
    $(r, \sigma) \leftarrow$ sat(ϕ)
    if ($r = unsatisfiable$) **return** *unsatisfiable*
    if (consistent (ϕ, σ)) **return** *satisfiable*
    add-lemma (ϕ, σ)

- Do not eagerly encode all constraints up front
- Explicitly check constraints that have not been encoded
  - Use SAT solver assignment σ
- Incrementally refine formula

- 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

Checking Array Axioms

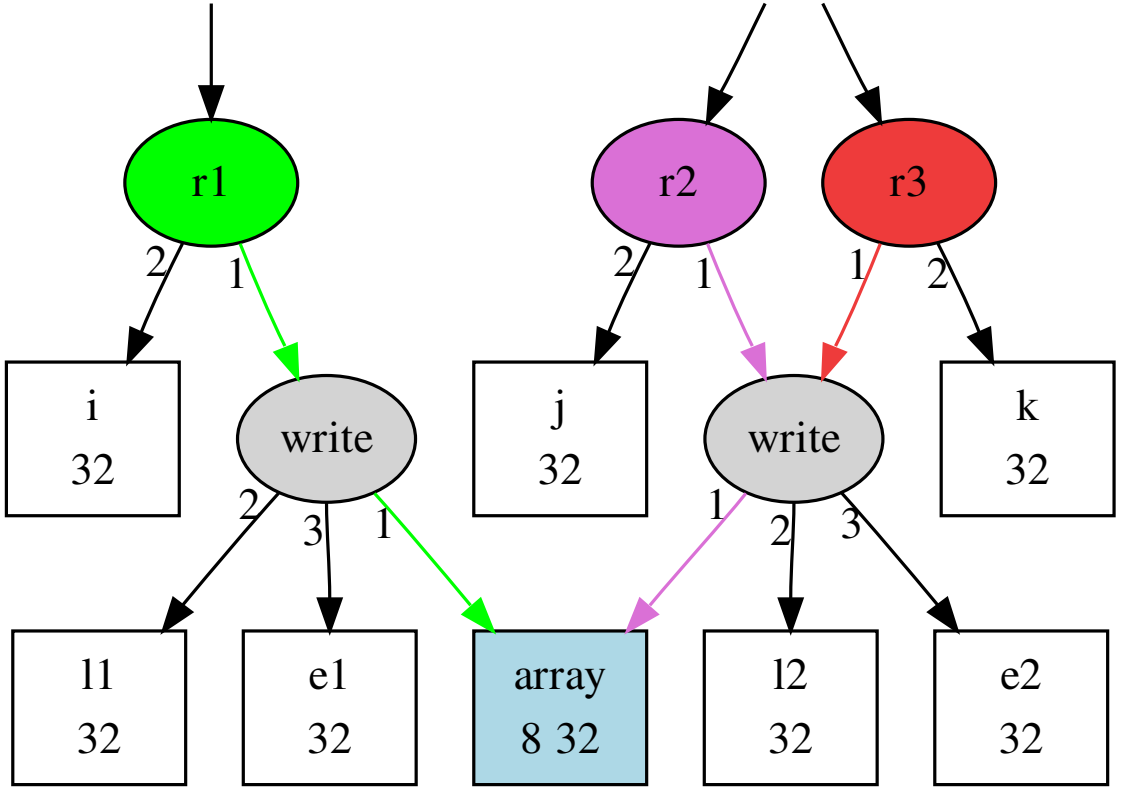
R. Brummayer, FMV, JKU Linz

- 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: Inconsistent Assignment

R. Brummayer, FMV, JKU Linz

$$\sigma(i) = \sigma(j), \quad \sigma(k) = \sigma(l2)$$



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

Lemma Construction

R. Brummayer, FMV, JKU Linz

- We collect all assignments 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
  - $k = l_2 \quad \Rightarrow \quad r_3 = e_2$
- Lemmas can be directly encoded into CNF (no new expressions)

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

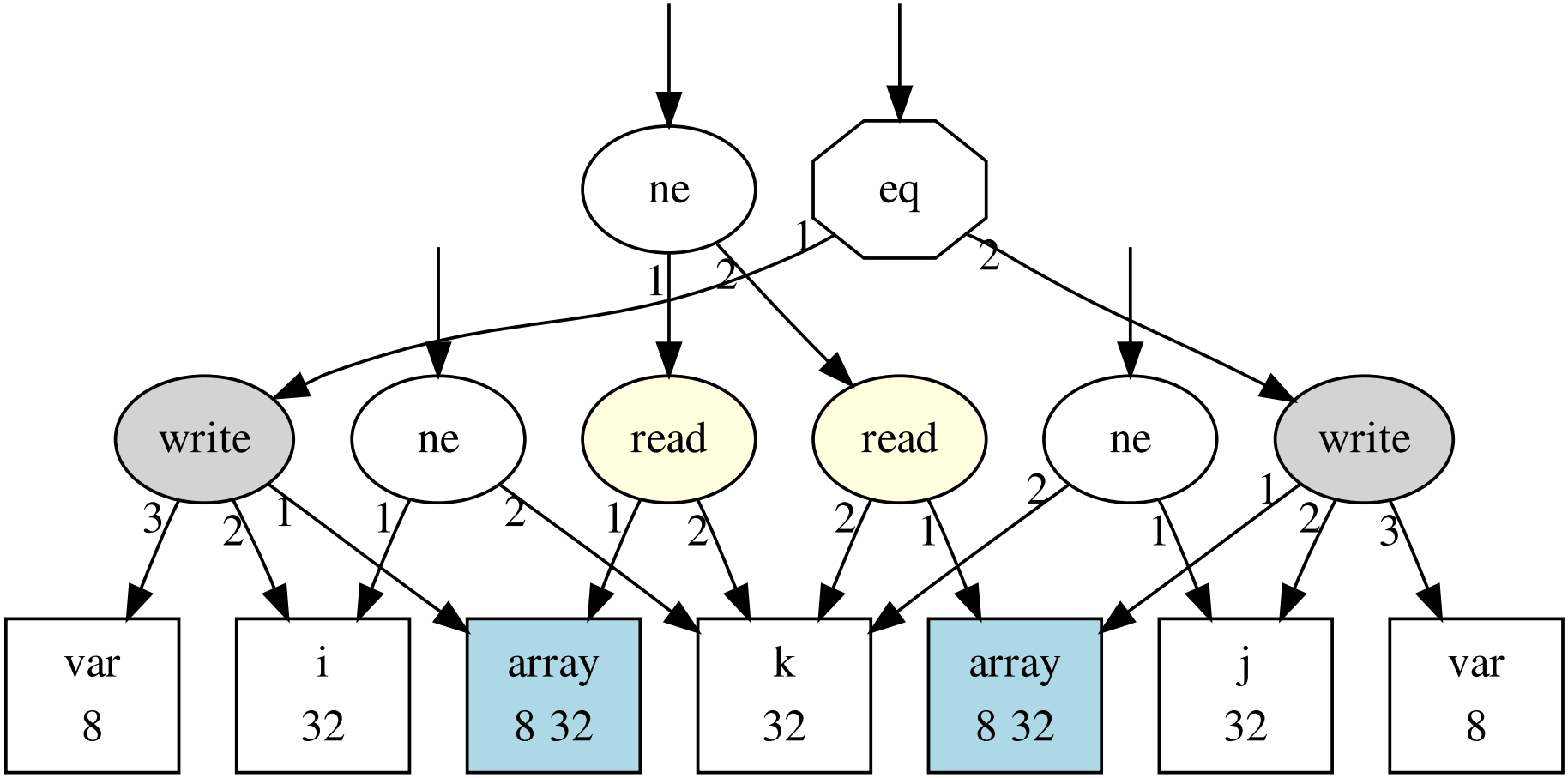
Adding Equalities on Arrays (2/2)

R. Brummayer, FMV, JKU Linz

- We must ensure congruence on write values for equal writes (only necessary in a setting without congruence closure)
- 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

Example 2: Propagation upwards

R. Brummayer, FMV, JKU Linz



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

Extensional Experiments (1/2)

R. Brummayer, FMV, JKU Linz

			lemmas			*eager*		z3		cvc3	
benchmark	*S*	*P*	*R*	*T*	*M*	*T*	*M*	*T*	*M*	*T*	*M*
swapmem	s	2	16	0.1	0.6	**0.0**	**0.0**	0.1	5.0	554.1	18.2
swapmem	s	6	53	**0.4**	**1.2**	0.6	3.3	547.6	101.6	90.0	71.9
swapmem	s	8	37	**0.3**	**1.4**	1.5	5.4	TO	TO	333.8	175.8
swapmem	s	10	34	**0.5**	**1.7**	1.0	8.3	TO	TO	740.2	305.3
swapmem	s	12	64	**1.0**	**2.2**	3.5	11.2	TO	TO	TO	TO
swapmem	s	14	57	**1.6**	**2.4**	1.8	15.7	TO	TO	TO	TO
wchains	s	6	14	**0.0**	**0.0**	0.8	7.0	46.3	11.6	8.5	60.1
wchains	s	8	16	**0.0**	**0.0**	1.9	14.5	572.2	32.5	23.5	117.6
wchains	s	10	18	**0.1**	**1.3**	3.1	20.3	TO	TO	34.2	193.6
wchains	s	12	20	**0.1**	**1.4**	3.5	29.8	TO	TO	65.4	294.6
wchains	s	15	21	**0.1**	**1.5**	4.2	45.8	TO	TO	126.3	472.6
wchains	s	20	28	**0.3**	**2.0**	5.7	80.2	TO	TO	267.8	897.3
wchains	s	30	36	**0.6**	**2.8**	10.9	179.9	TO	TO	MO	MO
wchains	s	60	70	**2.3**	**4.8**	41.8	718.5	TO	TO	MO	MO
wchains	s	90	96	**4.8**	**6.8**	MO	MO	TO	TO	MO	MO
dubreva	u	2	19	0.1	0.6	**0.0**	**0.0**	0.5	5.0	TO	TO
dubreva	u	3	41	0.3	**0.7**	**0.1**	1.2	0.7	5.6	TO	TO
dubreva	u	4	239	**12.5**	**2.2**	**12.5**	6.5	TO	TO	TO	TO
dubreva	u	5	379	**27.4**	**3.7**	348.3	13.1	TO	TO	TO	TO
dubreva	u	6	1187	**322.2**	**12.4**	TO	TO	TO	TO	TO	TO
dubreva	u	7	1637	**663.1**	**18.2**	TO	TO	TO	TO	MO	MO

Extensional Experiments (2/2)

R. Brummayer, FMV, JKU Linz

			lemmas			*eager*		z3		cvc3	
benchmark	*S*	*P*	*R*	*T*	*M*	*T*	*M*	*T*	*M*	*T*	*M*
swapmem	u	2	13	**0.1**	**0.7**	**0.1**	0.8	1.2	5.5	6.1	9.0
swapmem	u	4	25	**1.1**	**1.0**	3.0	2.0	5.2	7.1	159.6	49.8
swapmem	u	6	37	**3.2**	**1.2**	27.9	4.2	16.1	8.6	TO	TO
swapmem	u	8	49	**9.0**	**1.5**	129.0	8.3	20.7	10.1	TO	TO
swapmem	u	10	61	**18.3**	**2.0**	411.2	15.4	30.9	11.0	TO	TO
swapmem	u	12	73	**34.7**	**2.4**	TO	TO	62.1	13.3	TO	TO
swapmem	u	14	85	**63.4**	**2.8**	TO	TO	102.7	19.4	TO	TO
wchains	u	2	17	**0.0**	**0.0**	**0.0**	**0.0**	TO	TO	4.1	5.8
wchains	u	4	33	**0.1**	**0.9**	1.7	3.8	10.5	16.4	TO	TO
wchains	u	6	49	**0.4**	**1.1**	13.5	8.2	1.9	5.5	MO	MO
wchains	u	8	65	**0.6**	**1.3**	54.8	15.0	848.0	725.1	MO	MO
wchains	u	10	81	**1.1**	**1.5**	158.2	23.3	59.7	14.6	MO	MO
wchains	u	12	97	**2.4**	**1.8**	333.5	34.3	93.7	22.4	MO	MO
wchains	u	14	113	**2.9**	**1.9**	588.2	46.7	19.3	6.9	MO	MO
wchains	u	16	129	**4.8**	**2.1**	TO	TO	35.7	8.1	MO	MO
wchains	u	18	145	**6.6**	**2.3**	TO	TO	209.8	26.2	MO	MO

Non-Extensional Experiments

R. Brummayer, FMV, JKU Linz

		lemmas			*eager*		z3		cvc3		stp	
benchmark	*S*	*R*	*T*	*M*	*T*	*M*	*T*	*M*	*T*	*M*	*T*	*M*
bubsort4	u	10	0.9	**0.8**	**0.2**	**0.8**	TO	TO	TO	TO	0.3	7.6
bubsort6	u	23	4.6	**1.1**	**2.3**	1.5	TO	TO	TO	TO	19.2	10.3
bubsort8	u	40	66.1	**1.8**	**15.8**	2.7	TO	TO	TO	TO	TO	TO
bubsort10	u	61	**461.4**	**3.1**	514.0	10.1	TO	TO	TO	TO	TO	TO
bubsort12	u	-	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
selsort2	u	8	0.2	**0.6**	**0.1**	0.7	TO	TO	TO	TO	0.2	7.3
selsort3	u	21	4.2	**0.8**	**0.8**	1.1	TO	TO	TO	TO	1.5	9.8
selsort4	u	42	35.6	**1.2**	**5.4**	1.7	TO	TO	TO	TO	14.6	16.9
selsort5	u	66	216.8	**2.3**	**34.7**	3.1	TO	TO	TO	TO	121.7	37.4
selsort6	u	-	TO	TO	**407.2**	**7.6**	TO	TO	TO	TO	TO	TO
selsort7	u	-	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
grep0084	s	191	31.1	**27.4**	23.7	142.7	81.1	73.2	TO	TO	**15.3**	124.4
grep0095	s	134	**24.6**	**26.3**	26.6	153.7	101.5	78.6	TO	TO	187.0	550.4
grep0106	s	300	41.6	**30.6**	**33.5**	164.9	103.2	88.0	TO	TO	186.2	510.2
grep0117	s	205	35.1	**29.9**	**34.7**	176.4	149.9	90.2	TO	TO	139.9	412.1
grep0777	s	474	**481.4**	**85.1**	MO	MO	TO	TO	TO	TO	MO	MO
testcase15	s	0	99.5	**457.9**	98.0	458.8	**32.4**	594.6	MO	MO	TO	TO
testcase16	s	0	73.4	**476.5**	73.4	**476.5**	**33.3**	627.4	MO	MO	TO	TO
testcase20	s	-	TO	TO	MO	MO	TO	TO	MO	MO	TO	TO
tbnail-sp1	s	0	**58.9**	**1173.7**	62.3	1178.2	MO	MO	MO	MO	TO	TO
610dd9dc	s	319	72.1	**12.5**	33.6	148.2	TO	TO	MO	MO	**21.6**	303.9

Conclusion

R. Brummayer, FMV, JKU Linz

- 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