A Write-Based Solver for SAT Modulo the Theory of Arrays

Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell and Albert Rubio

8th International Conference, FMCAD 2008
Portland, OR, USA
November 19th, 2008
Overview of the talk

- SAT Modulo Theories (SMT)
  - The Theory of Extensional Arrays
  - Solving SMT with DPLL($T$)

- Handling Arrays in SMT
  - Theory instantiation for Arrays
  - A new solver for the theory of Arrays

- Key points

- Experimental evaluation

- Conclusions
Overview of the talk

- SAT Modulo Theories (SMT)
  - The Theory of Extensional Arrays
  - Solving SMT with DPLL(T)

- Handling Arrays in SMT
  - Theory instantiation for Arrays
  - A new solver for the theory of Arrays

- Key points

- Experimental evaluation

- Conclusions
SAT Modulo Theories (SMT)

Some problems are more naturally expressed in other logics than propositional logic, e.g:

- Software verification needs reasoning about equality, arithmetic, data structures, ...

**SMT** consists of deciding the satisfiability of a **(ground)** FO formula with respect to a background theory

Example (Equality with Uninterpreted Functions – **EUF**):

\[ g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d \]

Wide range of applications:

- Predicate abstraction
- Model checking
- Equivalence checking
- Static analysis
- Scheduling
- ...

A Write-Based Solver for SAT Modulo the Theory of Arrays – p. 4
The Theory of Extensional Arrays

- This is a very common structure
The Theory of Extensional Arrays

This is a very common structure

Axiomatization of the Theory:

- **Read/Write Axioms**
  \[ i = j \implies \text{read}(\text{write}(a, i, x), j) = x \]
  \[ i \neq j \implies \text{read}(\text{write}(a, i, x), j) = \text{read}(a, j) \]

- **Extensionality**
  \[ \forall i. \text{read}(a, i) = \text{read}(b, i) \implies a = b \]
The Theory of Extensional Arrays

- This is a very common structure

Axiomatization of the Theory:

- **Read/Write Axioms**
  
  \[
  i = j \Rightarrow \text{read}(\text{write}(a, i, x), j) = x
  \]

  \[
  i \neq j \Rightarrow \text{read}(\text{write}(a, i, x), j) = \text{read}(a, j)
  \]

- **Extensionality**

  \[
  a \neq b \Rightarrow \exists i. \text{read}(a, i) \neq \text{read}(b, i)
  \]
The Theory of Extensional Arrays

- This is a very common structure

- Axiomatization of the Theory:
  - **Read/Write Axioms**
    \[ i = j \Rightarrow \text{read}(\text{write}(a, i, x), j) = x \]
    \[ i \neq j \Rightarrow \text{read}(\text{write}(a, i, x), j) = \text{read}(a, j) \]
  - **Extensionality**
    \[ a \neq b \Rightarrow \exists i. \text{read}(a, i) \neq \text{read}(b, i) \]

Combined with
Uninterpreted Functions, Linear Integer Arithmetic or Bit-vectors
This is a very common structure

Axiomatization of the Theory:

- **Read/Write Axioms**
  
  \[ i = j \implies \text{read}(\text{write}(a, i, x), j) = x \]
  
  \[ i \neq j \implies \text{read}(\text{write}(a, i, x), j) = \text{read}(a, j) \]

- **Extensionality**
  
  \[ a \neq b \implies \exists i. \text{read}(a, i) \neq \text{read}(b, i) \]

Combined with

Uninterpreted Functions, Linear Integer Arithmetic or Bit-vectors

**THIS TALK:** Quantifier-free formulas over Extensional Arrays
Solving SMT with DPLL($T$)

Methodology:

\[\text{read}(a, j) \neq \text{read}(b, i) \land (a = b \lor a = \text{write}(b, i, x)) \land \text{read}(a, i) \neq x \land j = i\]

SAT solver returns model \([1, 2, 4, 5]\)
Solving SMT with DPLL($T$)

Methodology:

\[\begin{align*}
\text{read}(a, j) \neq \text{read}(b, i) & \quad \land \quad (a = b \lor a = \text{write}(b, i, x)) \land \text{read}(a, i) \neq x \land j = i \\
\end{align*}\]

- SAT solver returns model [1, 2, 4, 5]
- Theory solver says $T$-inconsistent
Solving SMT with DPLL($T$)

Methodology:

\[
\begin{align*}
\text{read}(a, j) \neq \text{read}(b, i) & \land (a = b \lor a = \text{write}(b, i, x)) \land \text{read}(a, i) \neq x \land j = i \\
\end{align*}
\]

- SAT solver returns model [1, 2, 4, 5]
- Theory solver says $T$-inconsistent
- Send \{1, 2 \lor 3, 4, 5, \overline{1} \lor \overline{2} \lor \overline{4} \lor \overline{5}\} to SAT solver
Solving SMT with DPLL($T$)

Methodology:

\[
\begin{align*}
\text{read}(a, j) \neq \text{read}(b, i) \quad &\land (a = b \lor a = \text{write}(b, i, x)) \quad \land \text{read}(a, i) \neq x \quad \land j = i \\
\end{align*}
\]

- SAT solver returns model $[1, 2, 4, 5]$
- Theory solver says $T$-inconsistent
- Send $\{1, 2 \lor 3, 4, 5, \overline{1} \lor \overline{2} \lor \overline{4} \lor \overline{5}\}$ to SAT solver
- SAT solver returns model $[1, \overline{2}, 3, 4, 5]$
Solving SMT with DPLL\((T)\)

Methodology:

\[
\text{read}(a, j) \neq \text{read}(b, i) \land (a = b \lor a = \text{write}(b, i, x)) \land \text{read}(a, i) \neq x \land j = i
\]

- **SAT solver** returns model \([1, 2, 4, 5]\)
- **Theory solver** says \(T\)-inconsistent
- Send \([1, 2 \lor 3, 4, 5, \overline{3} \lor \overline{4} \lor 5]\) to SAT solver
- **SAT solver** returns model \([1, \overline{2}, 3, 4, 5]\)
- **Theory solver** says \(T\)-inconsistent
Solving SMT with DPLL($T$)

Methodology:

\[
\underbrace{\text{read}(a,j) \neq \text{read}(b,i)}_{1} \land \underbrace{(a = b \lor a = \text{write}(b,i,x))}_{2} \land \underbrace{\text{read}(a,i) \neq x}_{4} \land j = i
\]

- SAT solver returns model $[1, 2, 4, 5]$
- Theory solver says $T$-inconsistent
- Send $\{1, 2 \lor 3, 4, 5, \overline{1} \lor \overline{2} \lor \overline{4} \lor \overline{5}\}$ to SAT solver
- SAT solver returns model $[1, \overline{2}, 3, 4, 5]$
- Theory solver says $T$-inconsistent
- SAT solver detects $\{1, 2 \lor 3, 4, 5, \overline{1} \lor \overline{2} \lor \overline{4} \lor \overline{5}, \overline{1} \lor \overline{3} \lor \overline{4} \lor \overline{5}\}$

UNSAT
Solving SMT with DPLL($T$)

Methodology:

\[
\begin{align*}
\text{read}(a, j) \neq \text{read}(b, i) \land \left( a = b \lor a = \text{write}(b, i, x) \right) \land \text{read}(a, i) \neq x \land j = i \\
\end{align*}
\]

- SAT solver returns model $[1, 2, 4, 5]$.
- Theory solver says $T$-inconsistent.
- Send $\{1, 2 \lor 3, 4, 5, \overline{1} \lor \overline{2} \lor \overline{4} \lor \overline{5}\}$ to SAT solver.
- SAT solver returns model $[1, \overline{2}, 3, 4, 5]$.
- Theory solver says $T$-inconsistent.
- SAT solver detects $\{1, 2 \lor 3, 4, 5, \overline{1} \lor \overline{2} \lor \overline{4} \lor \overline{5}, \overline{1} \lor \overline{3} \lor \overline{4} \lor \overline{5}\}$.
- UNSAT

Two components: Boolean engine $\text{DPLL}(X)$ + $T$-Solver.
Solving SMT with DPLL(\(T\)) (2)

Several optimizations for enhancing efficiency:

- Check \(T\)-consistency only of full prop. models (at a leaf)
Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full prop. models (at a leaf)
- Check $T$-consistency of partial assignment while being built
Solving SMT with DPLL($T$) (2)

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full prop. models (at a leaf)
- Check $T$-consistency of partial assignment while being built
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
Several optimizations for enhancing efficiency:

- Check \( T \)-consistency only of full propositional models (at a leaf).
- Check \( T \)-consistency of partial assignment while being built.
- Given a \( T \)-inconsistent assignment \( M \), add \( \neg M \) as a clause.
- Given a \( T \)-inconsistent assignment \( M \), identify a \( T \)-inconsistent subset \( M_0 \subseteq M \) and add \( \neg M_0 \) as a clause.
Solving SMT with DPLL($T$) (2)

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models (at a leaf)
- Check $T$-consistency of partial assignment while being built
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, identify a $T$-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- Upon a $T$-inconsistency, add clause and restart
Solving SMT with DPLL($T$) (2)

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models (at a leaf)
- Check $T$-consistency of partial assignment while being built
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, identify a $T$-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- Upon a $T$-inconsistency, add clause and restart
- Upon a $T$-inconsistency, bactrack to some point where the assignment was still $T$-consistent
Solving SMT with DPLL($T$) (2)

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full prop. models (at a leaf) –
- Check $T$-consistency of partial assignment while being built
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause –
- Given a $T$-inconsistent assignment $M$, identify a $T$-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- Upon a $T$-inconsistency, add clause and restart –
- Upon a $T$-inconsistency, bactrack to some point where the assignment was still $T$-consistent

**THIS TALK:** obtain an Arr-solver that is incremental, backtrackable and produce inconsistency explanations
Solving SMT with DPLL($T$) (3)

Need of case analysis inside the $T$-Solver:

\[
\begin{cases}
  \text{write}(a, i, x) = \text{write}(b, j, y), \\
  \text{write}(c, i, x) \neq \text{write}(c, j, y), \\
  \text{read}(a, j) \neq y
\end{cases}
\]

It’s inconsistent, but we need a case analysis on $i = j$
Need of case analysis inside the $T$-Solver:

\[
\{ \text{write}(a,i,x) = \text{write}(b,j,y), \quad \text{write}(c,i,x) \neq \text{write}(c,j,y), \quad \text{read}(a,j) \neq y \}\]

It’s inconsistent, but we need a case analysis on $i = j$

Assume $i = j$:
- From 1 we infer $x = y$
- From 2 we infer $x \neq y$  
  Inconsistency
Solving SMT with DPLL($T$) (3)

Need of case analysis inside the $T$-Solver:

\[
\{ \text{write}(a, i, x) = \text{write}(b, j, y), \ \text{write}(c, i, x) \neq \text{write}(c, j, y), \ \text{read}(a, j) \neq y \} \]

It’s inconsistent, but we need a case analysis on $i = j$

- Assume $i = j$:
  - From 1 we infer $x = y$
  - From 2 we infer $x \neq y$  \hspace{1cm} \text{Inconsistency}

- Assume $i \neq j$:
  - From 1 we infer that $a$ at position $j$ has $y$
  - which contradicts 3  \hspace{1cm} \text{Inconsistency}
Solving SMT with DPLL($T$) (3)

Need of case analysis inside the $T$-Solver:

\[
\{ \text{write}(a, i, x) = \text{write}(b, j, y), \quad \text{write}(c, i, x) \neq \text{write}(c, j, y), \quad \text{read}(a, j) \neq y \} 
\]

It’s inconsistent, but we need a case analysis on $i = j$

- Assume $i = j$:
  - From 1 we infer $x = y$
  - From 2 we infer $x \neq y$ Inconsistency

- Assume $i \neq j$:
  - From 1 we infer that $a$ at position $j$ has $y$
  - which contradicts 3 Inconsistency

We use split-on-demand: case analysis done by the boolean engine
Overview of the talk

- SAT Modulo Theories (SMT)
  - The Theory of Extensional Arrays
  - Solving SMT with DPLL(\(T\))

- Handling Arrays in SMT
  - Theory instantiation for Arrays
  - A new solver for the theory of Arrays

- Key points
- Experimental evaluation
- Conclusions
Handling Arrays in SMT

There are basically two possibilities:

- Using theory instantiation
- Having an Arr-solver for DPLL(Arr)
Theory instantiation for Arrays

There is no explicit $T$-Solver for Arrays

Instead, have a Module that generate Lemmas

Lemmas are instances of the axioms of the theory

Add the Lemmas to the set of clauses used by the SAT engine.
Theory instantiation for Arrays

- There is no explicit $T$-Solver for Arrays
- Instead, have a Module that generate Lemmas
  Lemmas are instances of the axioms of the theory

Add the Lemmas to the set of clauses used by the SAT engine.

- Used in SMT solvers like Yices or Z3
- [Goel,Krstic&Fuch2008] studied completeness
Theory instantiation for Arrays

- There is no explicit \( T \)-Solver for Arrays
- Instead, have a Module that generate Lemmas
  - Lemmas are instances of the axioms of the theory
  - Add the Lemmas to the set of clauses used by the SAT engine.

- Used in SMT solvers like Yices or Z3
- [Goel, Krstic & Fuch 2008] studied completeness

- Positive: simple and easier to implement
- Negative: cannot use dedicated algorithms for the Theory
Theory instantiation for Arrays(2)

To see pros and cons

Consider a simpler theory: uninterpreted functions
Theory instantiation for Arrays(2)

To see pros and cons

Consider a simpler theory: uninterpreted functions

Using Theory Instantiation:
Generate Lemmas like

\[ a = b \Rightarrow f(a) = f(b) \]

if \( f \) is a function symbol and \( a \) and \( b \) are constants.
To see pros and cons

Consider a simpler theory: uninterpreted functions

- Using **Theory Instantiation**: Generate **Lemmas** like

  \[ a = b \Rightarrow f(a) = f(b) \]

  if \( f \) is a function symbol and \( a \) and \( b \) are constants.

- Having a **T-Solver**: Apply **congruence closure** on the set of equality literals.
Theory instantiation for Arrays(2)

To see pros and cons

Consider a simpler theory: uninterpreted functions

- Using Theory Instantiation:
  Generate Lemmas like
  \[ a = b \Rightarrow f(a) = f(b) \]
  if \( f \) is a function symbol and \( a \) and \( b \) are constants.

- Having a T-Solver:
  Apply congruence closure on the set of equality literals.

  It’s not obvious what’s the best
Theory instantiation for Arrays(2)

To see pros and cons

Consider a simpler theory: uninterpreted functions

- Using Theory Instantiation:
  Generate Lemmas like

\[ a = b \Rightarrow f(a) = f(b) \]

if \( f \) is a function symbol and \( a \) and \( b \) are constants.

- Having a T-Solver:
  Apply congruence closure on the set of equality literals.

It’s not obvious what’s the best
We believe that the same happens with the Theory of Arrays
A new solver for the Theory of Arrays
A new solver for the Theory of Arrays

Existing Solver [Stump, Barrett, Dill & Levitt 2001]:

Based on the “read” operator

We call it Read-based:

write operators are translated into read operators.
A new solver for the Theory of Arrays

- Existing Solver [Stump, Barrett, Dill & Levitt 2001]:
  Based on the "read" operator
  We call it Read-based:
  write operators are translated into read operators.

- New approach:
  We call it Write-based:
  read operators are translated into write operators.
Read-based:

\[ a = \text{write}(b, i, x) \]
A new solver for the Theory of Arrays(2)

Read-based:

\[ a = \text{write}(b, i, x) \]

\[ \downarrow \]

is translated into
A new solver for the Theory of Arrays(2)

Read-based:

\[ a = \text{write}(b, i, x) \]
\[ \downarrow \]
\[ \text{read}(a, i) = x \]

is translated into
A new solver for the Theory of Arrays(2)

Read-based:

\[ a = \text{write}(b, i, x) \]

\[ \downarrow \]

\[ \text{read}(a, i) = x \]

\[ + \]

\[ a \simeq b \]

is translated into

???
A new solver for the Theory of Arrays(2)

Read-based:

\[ a = \text{write}(b, i, x) \]

\[ \downarrow \]

\[ \text{read}(a, i) = x \]

\[ + \]

\[ a =_{i} b \]

is translated into
equal except in \( i \)
A new solver for the Theory of Arrays(2)

Read-based:

\[ a = \text{write}(b, i, x) \]
\[ \downarrow \]
\[ \text{read}(a, i) = x \]
\[ + \]
\[ a =_{i} b \]

is translated into

equal except in \( i \)

Basically, ends up with uninterpreted functions plus this new theory of \( I \)-equality of arrays (which can be handled using theory instantiation)
A new solver for the Theory of Arrays(2)

- **Read-based:**

  \[ a = \text{write}(b, i, x) \]

  \[ \downarrow \]

  \[ \text{read}(a, i) = x \]

  \[ + \]

  \[ a =_i b \]

  is translated into

  equal except in \( i \)

- **Write-based:**

  \[ \text{read}(a, i) = x \]
A new solver for the Theory of Arrays(2)

- Read-based:

\[ a = \text{write}(b, i, x) \]

\[ \Downarrow \]

\[ \text{read}(a, i) = x \]

\[ + \]

\[ a =_i b \]

is translated into

equal except in \( i \)

- Write-based:

\[ \text{read}(a, i) = x \]

\[ \Downarrow \]
A new solver for the Theory of Arrays(2)

- **Read-based:**

  \[
  a = \text{write}(b, i, x) \\
  \Downarrow \\
  \text{read}(a, i) = x \\
  + \\
  a =_i b
  \]

  is translated into

  equal except in \(i\)

- **Write-based:**

  \[
  \text{read}(a, i) = x \\
  \Downarrow \\
  a = \text{write}(b, i, x)
  \]

  for some fresh \(b\)
A new solver for the Theory of Arrays(2)

- **Read-based:**

\[ a = \text{write}(b, i, x) \]

\[ \Downarrow \]

\[ \text{read}(a, i) = x \]

+ \[ a =_i b \]

is translated into

equal except in \( i \)

- **Write-based:**

\[ \text{read}(a, i) = x \]

\[ \Downarrow \]

\[ a = \text{write}(b, i, x) \]

for some fresh \( b \)

---

We follow the **Write-based** approach.
Set of literals:

\[
\begin{align*}
a &= \text{write}(b, j, x) \\
b &= \text{write}(c, i, y) \\
d &= \text{write}(e, i, y) \\
a &= d
\end{align*}
\]
A new solver for the Theory of Arrays (3)

Set of literals:

\[
\begin{align*}
    a &= \text{write}(b, j, x) \\
    b &= \text{write}(c, i, y) \\
    d &= \text{write}(c, i, y) \\
    a &= d
\end{align*}
\]

Representation:

\[
\begin{array}{ccc}
    a & j & x \\
    b & i & y \\
    c & & \\
    d & i & y \\
    e & & \\
\end{array}
\]

A Write-Based Solver for SAT Modulo the Theory of Arrays – p. 15
A new solver for the Theory of Arrays (3)

Set of literals:

\[ a = \text{write}(b, j, x) \]
\[ b = \text{write}(c, i, y) \]
\[ d = \text{write}(e, i, y) \]
\[ a = d \]

Which “writes” are relevant?

Representation:

\[
\begin{array}{ccc}
\text{a} & \text{j} & \text{x} \\
\text{b} & \text{i} & \text{y} \\
\text{c} & & \\
\end{array}
= \begin{array}{cc}
\text{d} & \text{i} \\
\text{e} & \text{y} \\
\end{array}
\]
A new solver for the Theory of Arrays(3)

Set of literals:

\[ a = \text{write}(b, j, x) \]
\[ b = \text{write}(c, i, y) \]
\[ d = \text{write}(e, i, y) \]
\[ a = d \]

Which “writes” are relevant?

- if \( i = j \) then we need \( x = y \)
Set of literals:

\[ a = \text{write}(b, j, x) \]
\[ b = \text{write}(c, i, y) \]
\[ d = \text{write}(e, i, y) \]
\[ a = d \]

Which “writes” are relevant?

- if \( i = j \) then we need \( x = y \)
- if \( i \neq j \) we need \( e = \text{write}(e_1, j, x) \)
A new solver for the Theory of Arrays(3)

Set of literals:

\[ a = \text{write}(b, j, x) \]
\[ b = \text{write}(c, i, y) \]
\[ d = \text{write}(e, i, y) \]
\[ a = d \]

Which “writes” are relevant?

- if \( i = j \) then we need \( x = y \)
- if \( i \neq j \) we need \( e = \text{write}(e_1, j, x) \)
A new solver for the Theory of Arrays (3)

Set of literals:

\[ a = \text{write}(b, j, x) \]
\[ b = \text{write}(c, i, y) \]
\[ d = \text{write}(e, i, y) \]
\[ a = d \]

Representation:

\[
\begin{array}{c|c|c}
\hline
a & j & x \\
\hline
b & i & y \\
\hline
d & i & y \\
e & j & x \\
e_1 & & \\
\hline
\end{array}
\]

Which “writes” are relevant?

- if \( i = j \) then we need \( x = y \)
- if \( i \neq j \) we need \( e = \text{write}(e_1, j, x) \)
A new solver for the Theory of Arrays (3)

Set of literals:

\[ a = write(b, j, x) \]
\[ b = write(c, i, y) \]
\[ d = write(e, i, y) \]
\[ a = d \]

Which “writes” are relevant?

- If \( i = j \) then we need \( x = y \)
- If \( i \neq j \) we need \( e = write(e_1, j, x) \)

Recall: we may need splitting on \( i = j \)
Overview of the talk

- SAT Modulo Theories (SMT)
  - The Theory of Extensional Arrays
  - Solving SMT with DPLL($T$)

- Handling Arrays in SMT
  - Theory instantiation for Arrays
  - A new solver for the theory of Arrays

Key points

- Experimental evaluation
- Conclusions
Key points

There are three key points in our approach:
Key points

There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals
Key points

There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals

- **Delay negative witnesses introduction:**
Key points

There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals

- **Delay negative witnesses introduction:**
  Recall the extensionality axiom:

\[
 a \neq b \Rightarrow \exists i. \text{read}(a, i) \neq \text{read}(b, i)
\]
Key points

There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals

- **Delay negative witnesses introduction:**
  
  \[
  a \neq b \\
  \Downarrow \\
  a = \text{write}(a_1, ni, ne_1) \text{ and } b = \text{write}(b_2, ni, ne_2)
  \]

  where \( ni \) is a new index and \( ne_1 \) and \( ne_2 \) are fresh constants with \( ne_1 \neq ne_2 \)
Key points

There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals

- **Delay negative witnesses introduction:**

\[
a \neq b
\]

\[
\Downarrow
\]

\[
a = \text{write}(a_1, ni, ne_1) \text{ and } b = \text{write}(b_2, ni, ne_2)
\]

where \( ni \) is a new index and \( ne_1 \) and \( ne_2 \) are fresh constants with \( ne_1 \neq ne_2 \)

This name is a tribute to Monty Python’s “Ni knights” (check Google with “Knights who say Ni” for further details)

The relationship between them is that both Ni’s (the indexes and the Knights) introduce a lot of noise
There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals

- **Delay negative witnesses introduction:**
  Delay the introduction of “Ni’s” avoiding unnecessary case analysys
Key points

There are three key points in our approach:

- **Notion of solved form:**
  Early detection of satisfiable sets of literals

- **Delay negative witnesses introduction:**
  Delay the introduction of “Ni’s” avoiding unnecessary case analysis

- **Produce better (shorter) explanations:**
  Using specialized mechanisms that take into account the knowledge about the theory of Arrays
Key points: Solved forms

There are several solved situations

Three particular examples (see paper for general definition):
Key points: Solved forms

There are several solved situations

Three particular examples (see paper for general definition):

\[
\begin{align*}
\text{write}(a, i, x) &= \text{write}(b, j, y) \\
\text{if } i &= j, \ x = y \text{ and } a \text{ and } b \text{ are different free constants.}
\end{align*}
\]
Key points: Solved forms

There are several solved situations

Three particular examples (see paper for general definition):

- \( \text{write}(a, i, x) = \text{write}(b, j, y) \)
  
  if \( i = j, x = y \) and \( a \) and \( b \) are different free constants.

- \( \text{write}(a, i, x) \neq \text{write}(b, j, y) \)
  
  if we don’t have \( i = j \) and \( b \) is a free constant.
Key points: Solved forms

There are several solved situations

Three particular examples (see paper for general definition):

1. \( \text{write}(a, i, x) = \text{write}(b, j, y) \)
   if \( i = j, x = y \) and \( a \) and \( b \) are different free constants.

2. \( \text{write}(a, i, x) \neq \text{write}(b, j, y) \)
   if we don’t have \( i = j \) and \( b \) is a free constant.

3. \( \text{write}(a, i, x) \neq \text{write}(b, i, y) \)
   if we have neither \( x = y \) nor \( x \neq y \).
Key points: Solved forms(2)

We can complete our partial model as follows:
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \[ \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1. \]
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \[ \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1. \]

- **Arrays:** assume there is a value \( d \) different from all others.
  \[ \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d \]
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1. \)

- **Arrays:** assume there is a value \( d \) different from all others.
  \( \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d \)

- \( \text{write}(a, i, x) = \text{write}(b, j, y) \)
  if \( i = j, x = y \) and \( a \) and \( b \) are different free constants.
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \(\forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1.\)

- **Arrays:** assume there is a value \(d\) different from all others.
  \(\forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d\)

- \(\text{write}(a, i, x) = \text{write}(b, j, y)\)
  if \(i = j, x = y\) and \(a\) and \(b\) are different free constants.
  Since \(a\) and \(b\) are free constants they have the same interpretation in the model.
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1. \)

- **Arrays:** Assume there is a value \( d \) different from all others.
  \( \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d \)

- **write** \((a, i, x) = \text{write}(b, j, y)\)
  if \( i = j, x = y \) and \( a \) and \( b \) are different free constants.
  Since \( a \) and \( b \) are free constants they have the same interpretation in the model.
  which satisfies the literal
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model} \)
  \( \text{we take } v_2 \neq v_1 \).

- **Arrays:**
  \( \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model} \)
  \( \text{we take } A[i] = d \)

- \( \text{write}(a, i, x) \neq \text{write}(b, j, y) \)
  \( \text{if we don’t have } i = j \text{ and } b \text{ is a free constant.} \)
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \) and \( v_2 \), if neither \( v_1 = v_2 \) nor \( v_2 \neq v_1 \) in the partial model we take \( v_2 \neq v_1 \).

- **Arrays:** assume there is a value \( d \) different from all others.
  \( \forall \) array \( A \), if \( A[i] \) is not defined for some \( i \) in the partial model we take \( A[i] = d \)

  \( \text{write}(a,i,x) \neq \text{write}(b,j,y) \)
  if we don’t have \( i = j \) and \( b \) is a free constant.
  Since we don’t have \( i = j \)
  we take \( i \neq j \) in the model and
We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1. \)

- **Arrays:** assume there is a value \( d \) different from all others.
  \( \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d \)

- **write \((a, i, x) \neq write(b, j, y)\)**
  if we don’t have \( i = j \) and \( b \) is a free constant.
  Since we don’t have \( i = j \)
  we take \( i \neq j \) in the model and
  since \( b \) is free constant
  we take \( b[i] = d \neq x \) in the model
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \) and \( v_2 \), if neither \( v_1 = v_2 \) nor \( v_2 \neq v_1 \) in the partial model we take \( v_2 \neq v_1 \).

- **Arrays:** assume there is a value \( d \) different from all others.
  \( \forall \) array \( A \), if \( A[i] \) is not defined for some \( i \) in the partial model we take \( A[i] = d \)

- \( \text{write}(a, i, x) \neq \text{write}(b, j, y) \)
  if we don’t have \( i = j \) and \( b \) is a free constant.
  Since we don’t have \( i = j \)
  we take \( i \neq j \) in the model and
  since \( b \) is free constant
  we take \( b[i] = d \neq x \) in the model
  which satisfies the literal
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \[
  \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1.
  \]

- **Arrays:** assume there is a value \( d \) different from all others.
  \[
  \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d
  \]

  \[
  \text{write}(a, i, x) \neq \text{write}(b, i, y)
  \]
  if we have neither \( x = y \) nor \( x \neq y \).
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:** 
  \( \forall v_1 \text{ and } v_2, \) if neither \( v_1 = v_2 \) nor \( v_2 \neq v_1 \) in the partial model we take \( v_2 \neq v_1 \).

- **Arrays:** assume there is a value \( d \) different from all others. 
  \( \forall \) array \( A, \) if \( A[i] \) is not defined for some \( i \) in the partial model we take \( A[i] = d \)

\[ \text{write}(a, i, x) \neq \text{write}(b, i, y) \]
if we have neither \( x = y \) nor \( x \neq y \).

since we have neither \( x = y \) nor \( x \neq y \)
we take \( x \neq y \) in the model
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \[ \forall v_1 \text{ and } v_2, \text{ if neither } v_1 = v_2 \text{ nor } v_2 \neq v_1 \text{ in the partial model we take } v_2 \neq v_1. \]

- **Arrays:** assume there is a value \( d \) different from all others.
  \[ \forall \text{ array } A, \text{ if } A[i] \text{ is not defined for some } i \text{ in the partial model we take } A[i] = d \]

- \( \text{write}(a, i, x) \neq \text{write}(b, i, y) \)
  if we have neither \( x = y \) nor \( x \neq y \).
  since we have neither \( x = y \) nor \( x \neq y \)
  we take \( x \neq y \) in the model
  which satisfies the literal
Key points: Solved forms(2)

We can complete our partial model as follows:

- **Indexes and values:**
  \( \forall v_1 \text{ and } v_2, \) if neither \( v_1 = v_2 \) nor \( v_2 \neq v_1 \) in the partial model we take \( v_2 \neq v_1 \).

- **Arrays:** assume there is a value \( d \) different from all others.
  \( \forall \) array \( A \), if \( A[i] \) is not defined for some \( i \) in the partial model we take \( A[i] = d \).

We have several inference rules that transform literals NOT in solved form until they are (see paper for details).
Key points: Delay Ni’s introduction

Consider the following negative literal:

\[
\begin{array}{|c|c|}
\hline
a1 & i1 & x1 \\
\hline
a2 & i2 & x2 \\
\hline
a3 & \text{ } & \text{ } \\
\hline
\end{array} \neq
\begin{array}{|c|c|}
\hline
b1 & i2 & y2 \\
\hline
b2 & i1 & x1 \\
\hline
b3 & \text{ } & \text{ } \\
\hline
\end{array}
\]

With: \( i_1 \neq i_2 \land x_2 \neq y_2 \)
Key points: Delay Ni’s introduction

Consider the following negative literal:

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>i1</th>
<th>x1</th>
<th></th>
<th>b1</th>
<th>i2</th>
<th>y2</th>
<th>≠</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>i2</td>
<td>x2</td>
<td></td>
<td>b2</td>
<td>i1</td>
<td>x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td></td>
<td></td>
<td></td>
<td>b3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With: \( i_1 \neq i_2 \land x_2 \neq y_2 \)

There is no need to add any new index \( ni \)
Avoiding case analysis between \( ni \) and the other indexes.
Consider the following negative literal:

\[
\begin{array}{c|cc}
\text{a1} & \text{i1} & \text{x1} \\
\text{a2} & \text{i2} & \text{x2} \\
\text{a3} & & \\
\text{b1} & \text{i2} & \text{x2} \\
\text{b2} & \text{i1} & \text{x1} \\
\text{b3} & & \\
\end{array}
\]

With: $i_1 \neq i_2$
Key points: Delay Ni’s introduction(2)

Consider the following negative literal:

\[
\begin{array}{|c|c|}
\hline
a1 & i1 & x1 \\
\hline
a2 & i2 & x2 \\
\hline
a3 & \multicolumn{2}{|c|}{\text{\ }} \\
\hline
\end{array} \quad \neq \quad \begin{array}{|c|c|}
\hline
b1 & i2 & x2 \\
\hline
b2 & i1 & x1 \\
\hline
b3 & \multicolumn{2}{|c|}{\text{\ }} \\
\hline
\end{array}
\]

With: \( i_1 \neq i_2 \)

We have to add a new index \( ni \), but we add it at the end.

\[
a_3 = \text{write}(a_4, ni, ed_1) \land b_3 = \text{write}(b_4, ni, ed_2)
\]

with \( ed_1 \neq ed_2, ni \neq i_1 \) and \( ni \neq i_2 \)
Key points: Delay Ni’s introduction(2)

Consider the following negative literal:

\[
\begin{array}{c|c|c}
\hline
a_1 & i_1 & x_1 \\
\hline
a_2 & i_2 & x_2 \\
\hline
a_3 & i_1 & x_2 \\
\hline
\end{array}
\]

\[\neq\]

\[
\begin{array}{c|c|c}
\hline
b_1 & i_2 & x_2 \\
\hline
b_2 & i_1 & x_1 \\
\hline
b_3 & i_1 & x_1 \\
\hline
\end{array}
\]

With: \( i_1 \neq i_2 \)

We have to add a new index \( n_i \), but we add it at the end.

\[
a_3 = \text{write}(a_4, n_i, ed_1) \land b_3 = \text{write}(b_4, n_i, ed_2)
\]

with \( ed_1 \neq ed_2 \), \( n_i \neq i_1 \) and \( n_i \neq i_2 \)
Key points: Shorter explanations

Consider the following inconsistent literal with $i_1 \neq i_3 \land i_2 \neq i_3 \land i_1 \neq i_2$:

\[
\begin{array}{|c|c|c|}
\hline
a1 & i1 & x1 \\
\hline
a2 & i2 & x2 \\
\hline
a3 & i3 & x3 \\
\hline
\end{array}
\neq
\begin{array}{|c|c|c|}
\hline
b1 & i3 & x3 \\
\hline
b2 & i1 & x1 \\
\hline
b3 & i2 & x2 \\
\hline
\end{array}
\]

Inconsistency explanation: $a_1 \neq b_1 \land i_1 \neq i_3 \land i_2 \neq i_3 \land i_1 \neq i_2$
Key points: Shorter explanations

Consider the following inconsistent literal with \( i_1 \neq i_3 \land i_2 \neq i_3 \land i_1 \neq i_2 \):

\[
\begin{array}{ccc}
\text{a}_1 & i_1 & x_1 \\
\text{a}_2 & i_2 & x_2 \\
\text{a}_3 & i_3 & x_3 \\
\text{c} & & \\
\end{array} \not= \\
\begin{array}{ccc}
\text{b}_1 & i_3 & x_3 \\
\text{b}_2 & i_1 & x_1 \\
\text{b}_3 & i_2 & x_2 \\
\text{c} & & \\
\end{array}
\]

Inconsistency explanation: \( a_1 \neq b_1 \land i_1 \neq i_3 \land i_2 \neq i_3 \land i_1 \neq i_2 \)
Key points: Shorter explanations

Consider the following inconsistent literal with \( i_1 \neq i_3 \land i_2 \neq i_3 \land i_1 \neq i_2 \):

\[
\begin{array}{|c|c|}
\hline
a_1 & i_1 \neq x_1 \\
\hline
a_2 & i_2 \neq x_2 \\
\hline
a_3 & i_3 \neq x_3 \\
\hline
\end{array}
\quad \neq 
\begin{array}{|c|c|}
\hline
b_1 & i_3 \neq x_3 \\
\hline
b_2 & i_1 \neq x_1 \\
\hline
b_3 & i_2 \neq x_2 \\
\hline
\end{array}
\]

Inconsistency explanation: \( a_1 \neq b_1 \land i_1 \neq i_3 \land i_2 \neq i_3 \land i_1 \neq i_2 \)
Overview of the talk

- SAT Modulo Theories (SMT)
  - The Theory of Extensional Arrays
  - Solving SMT with DPLL($T$)

- Handling Arrays in SMT
  - Theory instantiation for Arrays
  - A new solver for the theory of Arrays

- Key points

- Experimental evaluation

- Conclusions
Experimental evaluation

Setting used: SMT-LIB benchmarks 2007, 300 sec.

<table>
<thead>
<tr>
<th></th>
<th>YICES 1.0.10</th>
<th>YICES 1.0</th>
<th>Z3 0.1</th>
<th>CVC3 1.2</th>
<th>BARCELOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tot</td>
<td>Max</td>
<td>Tot</td>
<td>Max</td>
<td>Tot</td>
</tr>
<tr>
<td>array_ben</td>
<td>52</td>
<td>42</td>
<td>69</td>
<td>52</td>
<td>21</td>
</tr>
<tr>
<td>cvc</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>qlock2</td>
<td>49</td>
<td>5</td>
<td>50</td>
<td>6</td>
<td>114</td>
</tr>
<tr>
<td>storecomm</td>
<td>35</td>
<td>0.1</td>
<td>41</td>
<td>0.1</td>
<td>37</td>
</tr>
<tr>
<td>storeinv</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.1</td>
<td>8</td>
</tr>
<tr>
<td>swap</td>
<td>970</td>
<td>130</td>
<td>581</td>
<td>60</td>
<td>1431</td>
</tr>
</tbody>
</table>

SMT competition 2008 results.

- **QF_AX:** Barcelogic winner. Z3.2 second. NO Timeouts.
- **QF_AUFLIA:** Z3.2 winner. Barcelogic second. NO Timeouts.
Conclusions

- Our solver is intuitive and still competitive.
- Completely different from previous approaches.
- Observation: there is no unique best approach.

  The more approaches we have the better

- Need of new hard benchmarks to compare and improve.
Thank you!