Lemmas on Demand for Lambdas

Mathias Preiner, Aina Niemetz and Armin Biere

Institute for Formal Models and Verification (FMV)
Johannes Kepler University, Linz, Austria
http://fmv.jku.at/

DIFTS Workshop 2013
October 19, 2013
Portland, OR, USA
Introduction
Why Lambdas?

**Theory of arrays** [McCarthy, 1962]

\[ A1 \quad i = j \rightarrow \text{read}(a, i) = \text{read}(a, j) \]  
(array congruence)

\[ A2 \quad i = j \rightarrow \text{read}(%20\text{write}(a, i, e), j) = e \]  
(read-over-write 1)

\[ A3 \quad i \neq j \rightarrow \text{read}(%20\text{write}(a, i, e), j) = \text{read}(a, j) \]  
(read-over-write 2)

**Limitations**

- array operations restricted to single array indices
- no efficient modeling of parallel array updates (e.g.: \texttt{memset}, \texttt{memcpy})

→ [Bryant et al., 2002] tackle limitations by using restricted $\lambda$-terms in UCLID
Theory of arrays [McCarthy, 1962]

A1  \(i = j \rightarrow \text{read}(a, i) = \text{read}(a, j)\) \hspace{1cm} (array congruence)

A2  \(i = j \rightarrow \text{read}(\text{write}(a, i, e), j) = e\) \hspace{1cm} (read-over-write 1)

A3  \(i \neq j \rightarrow \text{read}(\text{write}(a, i, e), j) = \text{read}(a, j)\) \hspace{1cm} (read-over-write 2)

Limitations

- array operations restricted to single array indices
- no efficient modeling of parallel array updates (e.g.: \textit{memset}, \textit{memcpy})

\(\rightarrow [\text{Bryant et al., 2002}]\) tackle limitations by using restricted \(\lambda\)-terms in UCLID
Lambdas as arrays

- **write**\((a, i, e)\):
  \[
  \lambda j \cdot \text{ite}(i = j, e, \text{read}(a, j))
  \]

- **memset**\((a, i, n, e)\):
  \[
  \lambda j \cdot \text{ite}(i \leq j \land j < i + n, e, \text{read}(a, j))
  \]

- **memcpy**\((a, b, i, k, n)\):
  \[
  \lambda j \cdot \text{ite}(k \leq j \land j < k + n, \text{read}(a, i + j - k), \text{read}(b, j))
  \]

- ... 

Further applications

- ordered data structures
- arbitrary functions
- SMT-LIB v2 macros
- ...
Lambdas as arrays

- **write** \((a, i, e)\): 
  \[
  \lambda j . \text{ite}(i = j, e, \text{read}(a, j))
  \]
- **memset** \((a, i, n, e)\): 
  \[
  \lambda j . \text{ite}(i \leq j \land j < i + n, e, \text{read}(a, j))
  \]
- **memcpy** \((a, b, i, k, n)\): 
  \[
  \lambda j . \text{ite}(k \leq j \land j < k + n, \text{read}(a, i + j - k), \text{read}(b, j))
  \]

Further applications
- ordered data structures
- arbitrary functions
- SMT-LIB v2 macros
...
Lambdas as arrays

- \texttt{write}(a, i, e):
  \[ \lambda j \cdot \text{ite}(i = j, e, \text{read}(a, j)) \]

- \texttt{memset}(a, i, n, e):
  \[ \lambda j \cdot \text{ite}(i \leq j \land j < i + n, e, \text{read}(a, j)) \]

- \texttt{memcpy}(a, b, i, k, n):
  \[ \lambda j \cdot \text{ite}(k \leq j \land j < k + n, \text{read}(a, i + j - k), \text{read}(b, j)) \]

- ... 

Further applications

- ordered data structures
- arbitrary functions
- SMT-LIB v2 macros
- ...
UCLID [Seshia, 2005]
- SMT solver using eager approach
- non-recursive $\lambda$-terms
- $\lambda$-terms used for modeling arrays and array operations (and more)

Lazy SMT solvers with lambda support:
- CVC4 [Barrett et al., 2011]
- Yices [Dutertre and de Moura, 2006]

Lambda handling in SMT solvers
- $\lambda$-terms treated as C-style macros
- eager elimination with $\beta$-reduction
- may result in a \textit{exponential blow-up} in formula size
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]

\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]

\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]

\[ \vdots \]

\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]

\[ L_k := \lambda x . g_k(x) \]
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]
\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]
\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]
\[ \vdots \]
\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]
\[ L_k := \lambda x . g_k(x) \]

\[ P(L_1(a)) \]
\[ \downarrow \]
\[ L_1 \]
\[ \downarrow \]
\[ L_2 \]
\[ \vdots \]
\[ \downarrow \]
\[ L_k \]

\(2^k\) instantiations of \(L_k\)

\[ \rightarrow \text{avoid with lazy lambda handling}\]
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]
\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]
\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]
\[ \vdots \]
\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]
\[ L_k := \lambda x . g_k(x) \]

\[ P(L_1(a)) \]
\[ \downarrow \]
\[ L_1 \]

\[ 2^k \] instantiations of \( L_k \)

\[ \rightarrow \text{avoid with lazy lambda handling} \]
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]

\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]

\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]

\[ \vdots \]

\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]

\[ L_k := \lambda x . g_k(x) \]
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]
\[ L_1 := \lambda x \cdot f_1(L_2(x), L_2(g_1(x))) \]
\[ L_2 := \lambda x \cdot f_2(L_3(x), L_3(g_2(x))) \]
\[ \vdots \]
\[ L_{k-1} := \lambda x \cdot f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]
\[ L_k := \lambda x \cdot g_k(x) \]

Introduction
Eager Lambda Elimination Worst-Case

\[ P(L_1(a)) \]
\[ \downarrow \]
\[ L_1 \]
\[ \downarrow \]
\[ L_2 \]
\[ \vdots \]
\[ L_k \]

\[ 2^k \text{ instantiations of } L_k \]
→ avoid with lazy lambda handling
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]
\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]
\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]
\[ \vdots \]
\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]
\[ L_k := \lambda x . g_k(x) \]

\[ 2^k \text{ instantiations of } L_k \]

\[ \rightarrow \text{ avoid with lazy lambda handling} \]
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]

\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]

\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]

\[ \vdots \]

\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]

\[ L_k := \lambda x . g_k(x) \]

\[ P(L_1(a)) \]

\[ L_1 \]

\[ L_2 \quad L_2 \]

\[ \vdots \quad \vdots \quad \vdots \quad \vdots \]

\[ L_{k-1} \quad L_{k-1} \quad L_{k-1} \quad L_{k-1} \]

\[ L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \quad L_k \]

\[ 2^k \text{ instantiations of } L_k \]

\[ \rightarrow \text{ avoid with lazy lambda handling} \]
Example [Seshia, 2005]

\[ F := P(L_1(a)) \]

\[ L_1 := \lambda x . f_1(L_2(x), L_2(g_1(x))) \]

\[ L_2 := \lambda x . f_2(L_3(x), L_3(g_2(x))) \]

\[ \vdots \]

\[ L_{k-1} := \lambda x . f_{k-1}(L_k(x), L_k(g_{k-1}(x))) \]

\[ L_k := \lambda x . g_k(x) \]

\[ P(L_1(a)) \]

\[ L_1 \]

\[ L_2 \]

\[ \vdots \]

\[ L_{k-1} \]

\[ L_k \]

\[ 2^k \text{ instantiations of } L_k \]

→ avoid with lazy lambda handling
Boolector

- lazy SMT solver
- employs **lemmas on demand**
- supported theories:
  - fixed size bit vectors
  - arrays
- no quantifiers

**Old version (pre-lambda)**
- extensionality on arrays

**New version**
- \(\lambda\)-term support
- extensionality not supported (yet)
**Extensionality**

**Extensionality on arrays**

\[
\begin{align*}
a &= b \\
\Leftrightarrow \\
\forall i. \; \text{read}(a, i) &= \text{read}(b, i)
\end{align*}
\]

**Extensionality on lambdas**

\[
\begin{align*}
\lambda x \cdot \phi &= \lambda y \cdot \psi \\
\Leftrightarrow \\
\forall \bar{a}. \; (\lambda x \cdot \phi)(\bar{a}) &= (\lambda y \cdot \psi)(\bar{a})
\end{align*}
\]

**Quantifiers with extensionality on lambdas**

\[
\begin{align*}
\lambda x \cdot p(x) &= \lambda x \cdot \top \\
\Leftrightarrow \\
\forall x \cdot p(x)
\end{align*}
\]
Extensionality

Extensionality on arrays

\[ a = b \iff \forall i . \text{read}(a, i) = \text{read}(b, i) \]

Extensionality on lambdas

\[ \lambda \vec{x} . \phi = \lambda \vec{y} . \psi \iff \forall \vec{a} . (\lambda \vec{x} . \phi)(\vec{a}) = (\lambda \vec{y} . \psi)(\vec{a}) \]

Quantifiers with extensionality on lambdas

\[ \lambda x . p(x) = \lambda x . \top \iff \forall x . p(x) \]
Extensionality

Extensionality on arrays

\[ a = b \iff \forall i. \text{read}(a, i) = \text{read}(b, i) \]

Extensionality on lambdas

\[ \lambda \vec{x} . \phi = \lambda \vec{y} . \psi \iff \forall \vec{a}. (\lambda \vec{x} . \phi)(\vec{a}) = (\lambda \vec{y} . \psi)(\vec{a}) \]

Quantifiers with extensionality on lambdas

\[ \lambda x . p(x) = \lambda x . \top \iff \forall x . p(x) \]
Restrictions
- non-recursive
- non-extensional
- non-higher order functions

Lambdas in Boolector
- arrays represented as λ-terms\uninterpreted functions
  → no terms of sort array
  → uniform handling of arrays and λ-terms
- SMT-LIB v2 macros treated as curried λ-terms
- **lazy instantiation** of λ-terms
  → optional eager elimination
- new decision procedure DP$_\lambda$ for λ-terms
  → generalization of array decision procedure [Brummayer and Biere, 2009]
Partial \( \beta \)-reduction

- like \( \beta \)-reduction in \( \lambda \)-calculus
- \( \lambda \)-terms are expanded "function-wise"
- required for consistency checking in \( \text{DP}_\lambda \)
  \( \rightarrow \) considers current assignment

Full \( \beta \)-reduction

- eager elimination of \( \lambda \)-terms
- optional rewriting step

Given a DAG representing a \( \lambda \)-term . . .

1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial \( \beta \)-reduction: \( \lambda \bar{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p \)
Partial $\beta$-reduction

- like $\beta$-reduction in $\lambda$-calculus
- $\lambda$-terms are expanded "function-wise"
- required for consistency checking in $\text{DP}_\lambda$
  → considers current assignment

Full $\beta$-reduction

- eager elimination of $\lambda$-terms
- optional rewriting step

Given a DAG representing a $\lambda$-term...

1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial $\beta$-reduction: $\lambda\bar{x}[x_1\backslash a_1, \ldots, x_n\backslash a_n]_p$
Partial $\beta$-reduction
- like $\beta$-reduction in $\lambda$-calculus
- $\lambda$-terms are expanded "function-wise"
- required for consistency checking in $\text{DP}_\lambda$
  $\rightarrow$ considers current assignment

Full $\beta$-reduction
- eager elimination of $\lambda$-terms
- optional rewriting step

Given a DAG representing a $\lambda$-term . . .
1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial $\beta$-reduction: $\lambda\vec{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p$
Lambdas in Boolector

\(\beta\)-reduction Approaches

**Partial \(\beta\)-reduction**
- like \(\beta\)-reduction in \(\lambda\)-calculus
- \(\lambda\)-terms are expanded "function-wise"
- required for consistency checking in \(\text{DP}_\lambda\)
  \(\rightarrow\) considers current assignment

**Full \(\beta\)-reduction**
- eager elimination of \(\lambda\)-terms
- optional rewriting step

Given a DAG representing a \(\lambda\)-term …
1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial \(\beta\)-reduction: \(\lambda_{\bar{x}}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p\)
**Partial $\beta$-reduction**
- like $\beta$-reduction in $\lambda$-calculus
- $\lambda$-terms are expanded "function-wise"
- required for consistency checking in $\text{DP}_\lambda$
  → considers current assignment

**Full $\beta$-reduction**
- eager elimination of $\lambda$-terms
- optional rewriting step

Given a DAG representing a $\lambda$-term . . .
1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial $\beta$-reduction: $\lambda\bar{x}[x_1\backslash a_1, \ldots, x_n\backslash a_n]_p$
Lambdas in Boolector

\(\beta\)-reduction Approaches

**Partial \(\beta\)-reduction**
- like \(\beta\)-reduction in \(\lambda\)-calculus
- \(\lambda\)-terms are expanded ”function-wise”
- required for consistency checking in \(\text{DP}_\lambda\)
  \(\rightarrow\) considers current assignment

**Full \(\beta\)-reduction**
- eager elimination of \(\lambda\)-terms
- optional rewriting step

Given a DAG representing a \(\lambda\)-term . . .

1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial \(\beta\)-reduction: \(\lambda_{\overline{x}}[x_1\backslash a_1, \ldots, x_n\backslash a_n]_p\)
Lambdas in Boolector

\(\beta\)-reduction Approaches

**Partial \(\beta\)-reduction**
- like \(\beta\)-reduction in \(\lambda\)-calculus
- \(\lambda\)-terms are expanded "function-wise"
- required for consistency checking in \(\text{DP}_\lambda\)
  \(\rightarrow\) considers current assignment

**Full \(\beta\)-reduction**
- eager elimination of \(\lambda\)-terms
- optional rewriting step

Given a DAG representing a \(\lambda\)-term . . .
1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial \(\beta\)-reduction: \(\lambda\bar{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p\)
Lambdas in Boolector

\(\beta\)-reduction Approaches

**Partial \(\beta\)-reduction**
- like \(\beta\)-reduction in \(\lambda\)-calculus
- \(\lambda\)-terms are expanded "function-wise"
- required for consistency checking in \(\text{DP}_\lambda\)
  \(\Rightarrow\) considers current assignment

**Full \(\beta\)-reduction**
- eager elimination of \(\lambda\)-terms
- optional rewriting step

Given a DAG representing a \(\lambda\)-term …
1. perform DFS post-order traversal
2. consecutively assign arguments to parameters
3. rebuild terms with arguments instead of parameters

Our notation for partial \(\beta\)-reduction: \(\lambda\bar{x}[x_1 \atop a_1, \ldots, x_n \atop a_n]_p\)
Refinement loop

1. abstract input formula $\phi$ (bit vector skeleton)
   $\rightarrow$ introduce fresh bit vector variable for each function application
   $\rightarrow$ translate bit vector skeleton into prop. formula

2. let SAT solver ”guess” a solution
   $\rightarrow$ if SAT solver returns unsatisfiable, terminate with unsatisfiable

3. check if satisfying assignment is consistent w.r.t. $\phi$ ($\text{consistent}_\lambda$)
   $\rightarrow$ if check succeeds, terminate with satisfiable

4. if check fails, add lemma to refine formula abstraction ($\text{lemma}_\lambda$)

5. continue with 2
What to check?
Check whether current assignment $\sigma$ is spurious or not

Rules

- rule C: function congruence axiom EUF

\[
\forall \vec{a}, \vec{b}. \bigwedge_{i=1}^{n} \sigma(a_i) = \sigma(b_i) \rightarrow \sigma(f(\vec{a})) = \sigma(f(\vec{b}))
\]

\[
\ldots \rightarrow \sigma(v_{f(\vec{a})}) = \sigma(v_{f(\vec{b})})
\]

- rule B: abstraction variable consistency

\[
\sigma(v_{\lambda \vec{x}(a)}) = \sigma(\lambda \vec{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p)
\]

→ Optimization: rule P (see paper for more details)
What to check?
Check whether current assignment $\sigma$ is spurious or not

Rules

- rule C: function congruence axiom EUF

$$\forall \bar{a}, \bar{b}. \bigwedge_{i=1}^{n} \sigma(a_i) = \sigma(b_i) \rightarrow \sigma(f(\bar{a})) = \sigma(f(\bar{b}))$$

$$\cdots \rightarrow \sigma(v_{f(\bar{a})}) = \sigma(v_{f(\bar{b})})$$

- rule B: abstraction variable consistency

$$\sigma(v_{\lambda \bar{x}(a)}) = \sigma(\lambda \bar{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p)$$

→ Optimization: rule P (see paper for more details)
Lemmas on Demand for Lambdas
Consistency Checking in DPλ

What to check?
Check whether current assignment \( \sigma \) is spurious or not

Rules
- **rule C**: function congruence axiom EUF

\[
\forall \bar{a}, \bar{b}. \bigwedge_{i=1}^{n} \sigma(a_i) = \sigma(b_i) \rightarrow \sigma(f(\bar{a})) = \sigma(f(\bar{b}))
\]

\[
\ldots \rightarrow \sigma(v_{f(\bar{a})}) = \sigma(v_{f(\bar{b})})
\]

- **rule B**: abstraction variable consistency

\[
\sigma(v_{\lambda \bar{x}(a)}) = \sigma(\lambda \bar{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p)
\]

\( \rightarrow \) Optimization: rule P (see paper for more details)
Lemmas on Demand for Lambdas
Consistency Checking in DP_{\lambda}

What to check?
Check whether current assignment \( \sigma \) is spurious or not

Rules

- **rule C**: function congruence axiom EUF

  \[
  \forall \bar{a}, \bar{b}. \bigwedge_{i=1}^{n} \sigma(a_i) = \sigma(b_i) \rightarrow \sigma(f(\bar{a})) = \sigma(f(\bar{b}))
  \]

  \[
  \ldots \rightarrow \sigma(v_{f(\bar{a})}) = \sigma(v_{f(\bar{b})})
  \]

- **rule B**: abstraction variable consistency

  \[
  \sigma(v_{\lambda \bar{x}}(a)) = \sigma(\lambda \bar{x} [x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p)
  \]

\[ \rightarrow \] Optimization: rule P (see paper for more details)
Lemmas on Demand for Lambdas
Consistency Checking in DPλ

What to check?
Check whether current assignment $\sigma$ is spurious or not

Rules

- rule C: function congruence axiom EUF

$$\forall \bar{a}, \bar{b}. \bigwedge_{i=1}^{n} \sigma(a_i) = \sigma(b_i) \rightarrow \sigma(f(\bar{a})) = \sigma(f(\bar{b}))$$

$$\ldots \rightarrow \sigma(v_{f(\bar{a})}) = \sigma(v_{f(\bar{b})})$$

- rule B: abstraction variable consistency

$$\sigma(v_{\lambda \bar{x}}(a)) = \sigma(\lambda \bar{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p)$$

$\rightarrow$ Optimization: rule P (see paper for more details)
Lemmas on Demand for Lambdas
Consistency Checking in $\text{DP}_\lambda$ (cont.)

**Algorithm** $\text{consistent}_\lambda$

- adaption of propagation algorithm in [Brummayer and Biere, 2009]

- associate each function application with resp. function
  → maintain hash table $\rho$ for every function

- for each *pair* of function applications in $\rho$ check rule C

- for each function application in $\rho$ check rule B ($\lambda$-terms only)

- if a *conflict* occurs, generate a lemma ($\text{lemma}_\lambda$)

- otherwise, current assignment $\sigma$ is *valid*
Violation of rule C

\[ s := g(a_1, \ldots, a_n), \; t := h(b_1, \ldots, b_n) \in \rho(f) \text{ violate rule } C \]

1. find propagation path \( p^s \) (\( p^t \)) from \( s \) (\( t \)) to \( f \)

2. collect all \( \text{ite} \) conditions \( c_0^s, \ldots, c_j^s \) (\( c_0^t, \ldots, c_j^t \)) on path \( p^s \) (\( p^t \)) that were \( \top \) under given assignment \( \sigma \)

3. collect all \( \text{ite} \) conditions \( c_0^s, \ldots, c_k^s \) (\( c_0^t, \ldots, c_m^t \)) on path \( p^s \) (\( p^t \)) that were \( \bot \) under given assignment \( \sigma \)

Lemma

\[
\bigwedge_{i=0}^{j} c_i^s \land \bigwedge_{i=0}^{k} \neg c_i^s \land \bigwedge_{i=0}^{l} c_i^t \land \bigwedge_{i=0}^{m} \neg c_i^t \land \bigwedge_{i=0}^{n} a_i = b_i \rightarrow s = t
\]
Lemmas on Demand for Lambdas
Lemma Generation

Violation of rule C
\[ s := g(a_1, \ldots, a_n), \ t := h(b_1, \ldots, b_n) \in \rho(f) \] violate rule C

1. find propagation path \( p^s \) (\( p^t \)) from \( s \) (\( t \)) to \( f \)

2. collect all \( \text{ite} \) conditions \( c_0^s, \ldots, c_j^s \) (\( c_0^t, \ldots, c_i^t \)) on path \( p^s \) (\( p^t \)) that were \( \top \) under given assignment \( \sigma \)

3. collect all \( \text{ite} \) conditions \( c_0^s, \ldots, c_k^s \) (\( c_0^t, \ldots, c_m^t \)) on path \( p^s \) (\( p^t \)) that were \( \bot \) under given assignment \( \sigma \)

Lemma
\[ \bigwedge_{i=0}^{j} c_i^s \land \bigwedge_{i=0}^{k} \neg c_i^s \land \bigwedge_{i=0}^{l} c_i^t \land \bigwedge_{i=0}^{m} \neg c_i^t \land \bigwedge_{i=0}^{n} a_i = b_i \rightarrow s = t \]

Prop. conditions s
Lemmas on Demand for Lambdas

Lemma Generation

Violation of rule C

\[ s := g(a_1, \ldots, a_n), \ t := h(b_1, \ldots, b_n) \in \rho(f) \] violate rule C

1. find propagation path \( p^s (p^t) \) from \( s (t) \) to \( f \)

2. collect all \( \text{ite} \) conditions \( c_{0}^s, \ldots, c_{j}^s \) \( (c_{0}^t, \ldots, c_{j}^t) \) on path \( p^s (p^t) \) that were \( \top \) under given assignment \( \sigma \)

3. collect all \( \text{ite} \) conditions \( c_{0}^s, \ldots, c_{k}^s \) \( (c_{0}^t, \ldots, c_{m}^t) \) on path \( p^s (p^t) \) that were \( \bot \) under given assignment \( \sigma \)

Lemma

\[
\bigwedge_{i=0}^{j} c_i^s \land \bigwedge_{i=0}^{k} \neg c_i^s \land \bigwedge_{i=0}^{l} c_i^t \land \bigwedge_{i=0}^{m} \neg c_i^t \land \bigwedge_{i=0}^{n} a_i = b_i \implies s = t
\]

Prop. conditions t
Lemmas on Demand for Lambdas

Lemma Generation

Violation of rule C

\[ s := g(a_1, \ldots, a_n), \quad t := h(b_1, \ldots, b_n) \in \rho(f) \text{ violate rule C} \]

1. find propagation path \( p^s \) (\( p^t \)) from \( s \) (\( t \)) to \( f \)

2. collect all \( \text{ite} \) conditions \( c_0^s, \ldots, c_j^s \) (\( c_0^t, \ldots, c_i^t \)) on path \( p^s \) (\( p^t \)) that were \( \top \) under given assignment \( \sigma \)

3. collect all \( \text{ite} \) conditions \( c_0^s, \ldots, c_k^s \) (\( c_0^t, \ldots, c_m^t \)) on path \( p^s \) (\( p^t \)) that were \( \bot \) under given assignment \( \sigma \)

Lemma

\[
\bigwedge_{i=0}^{j} c_i^s \land \bigwedge_{i=0}^{k} \neg c_i^s \land \bigwedge_{i=0}^{l} c_i^t \land \bigwedge_{i=0}^{m} \neg c_i^t \land \bigwedge_{i=0}^{n} a_i = b_i \rightarrow s = t
\]

function congruence
Violation of rule B

$s := \lambda \overline{y}(a_1, \ldots, a_n) \in \rho(\lambda \overline{x}), \; t := \lambda \overline{x}[x_1 \setminus a_1, \ldots, x_n \setminus a_n]_p$ violates rule B

1. collect conditions $c^s_0, \ldots, c^s_j, c^s_0, \ldots, c^s_k$ as before
2. collect all *ite* conditions $c^t_0, \ldots, c^t_i$ that evaluated to $\top$ under given assignment $\sigma$ while obtaining $t$
3. collect all *ite* conditions $c^t_0, \ldots, c^t_m$ that evaluated to $\bot$ under given assignment $\sigma$ while obtaining $t$

Lemma

$$\left( \bigwedge_{i=0}^{j} c^s_i \land \bigwedge_{i=0}^{k} \neg c^s_i \land \bigwedge_{i=0}^{l} c^t_i \land \bigwedge_{i=0}^{m} \neg c^t_i \right) \rightarrow s = t$$
Lesmas on Demand for Lambdas
Lemma Generation (cont.)

Violation of rule B
\[ s := \lambda y (a_1, \ldots, a_n) \in \rho (\lambda x), \ t := \lambda x [x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p \text{ violates rule B} \]

1. collect conditions \( c_0^s, \ldots, c_j^s, c_0^t, \ldots, c_k^s \) as before
2. collect all \( \text{ite} \) conditions \( c_0^t, \ldots, c_i^t \) that evaluated to \( \top \) under given assignment \( \sigma \) while obtaining \( t \)
3. collect all \( \text{ite} \) conditions \( c_0^t, \ldots, c_i^t \) that evaluated to \( \bot \) under given assignment \( \sigma \) while obtaining \( t \)

Lemma
\[
\bigwedge_{i=0}^{j} c_i^s \land \bigwedge_{i=0}^{k} \neg c_i^s \land \bigwedge_{i=0}^{l} c_i^t \land \bigwedge_{i=0}^{m} \neg c_i^t \rightarrow s = t
\]

Prop. conditions \( s \)
Lemmas on Demand for Lambdas
Lemma Generation (cont.)

Violation of rule B
\( s := \lambda \bar{y}(a_1, \ldots, a_n) \in \rho(\lambda \bar{x}), \ t := \lambda \bar{x}[x_1 \backslash a_1, \ldots, x_n \backslash a_n]_p \) violates rule B

1. collect conditions \( c_0^s, \ldots, c_j^s, c_0^s, \ldots, c_k^s \) as before

2. collect all \( \text{ite} \) conditions \( c_0^t, \ldots, c_i^t \) that evaluated to \( \top \) under given assignment \( \sigma \) while obtaining \( t \)

3. collect all \( \text{ite} \) conditions \( c_0^t, \ldots, c_m^t \) that evaluated to \( \bot \) under given assignment \( \sigma \) while obtaining \( t \)

Lemma

\[
\bigwedge_{i=0}^{j} c_i^s \wedge \bigwedge_{i=0}^{k} \neg c_i^s \wedge \bigwedge_{i=0}^{l} c_i^t \wedge \bigwedge_{i=0}^{m} \neg c_i^t \rightarrow s = t
\]

Eval. conditions \( t \)
Lemmas on Demand for Lambdas
Lemma Generation (cont.)

Violation of rule B
$s := \lambda \bar{y}(a_1, \ldots, a_n) \in \rho(\lambda \bar{x}), \ t := \lambda \bar{x}[x_1\backslash a_1, \ldots, x_n\backslash a_n]_p$ violates rule B

1. collect conditions $c_0^s, \ldots, c_j^s, c_0^s, \ldots, c_k^s$ as before
2. collect all $ite$ conditions $c_0^t, \ldots, c_j^t$ that evaluated to $\top$ under given assignment $\sigma$ while obtaining $t$
3. collect all $ite$ conditions $c_0^t, \ldots, c_m^t$ that evaluated to $\bot$ under given assignment $\sigma$ while obtaining $t$

Lemma
\[
\bigwedge_{i=0}^{j} c_i^s \land \bigwedge_{i=0}^{k} \neg c_i^s \land \bigwedge_{i=0}^{l} c_i^t \land \bigwedge_{i=0}^{m} \neg c_i^t \rightarrow s = t
\]
abstr. variable consistency
Experiments
Overview

3 benchmark categories

- **crafted**: benchmarks with SMT-LIB v2 macros
- **SMT’12**: all non-extensional QF_AUFBV benchmarks used in SMT competition 2012
- **application**: instantiation benchmarks \(^1\) [Falke et al., 2013] generated with LLBMC (with and without \(\lambda\)-terms as arrays)

SMT Solvers

- **Boolector**: with DP\(_\lambda\)
- **Boolector\(_{nop}\)**: with DP\(_\lambda\), but without rule P
- **Boolector\(_{\beta}\)**: with eager \(\lambda\)-term elimination
- **Boolector\(_{sc12}\)**: version submitted to SMT competition 2012
- **CVC4 1.2**, MathSAT 5.2.6, SONOLAR 2013-05-15, STP 1673 (svn revision), Z3 4.3.1

Machine Setup: 2.83Ghz Intel Core 2 Quad, 8GB memory, Ubuntu 12.04.2

\(^1\)http://llbmc.org/files/downloads/vstte-2013.tgz
## Experiments
### Category: crafted

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time $[10^3\text{s}]$</th>
<th>Space [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>macro blow-up</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>24.2</td>
<td>9.4</td>
</tr>
<tr>
<td>Boolector$_{nop}$</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>18.2</td>
<td>8.4</td>
</tr>
<tr>
<td>Boolector$\beta$</td>
<td>28</td>
<td>49</td>
<td>23</td>
<td>91.5</td>
<td>160.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.7</td>
<td>551.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>51</td>
<td>2</td>
<td>47</td>
<td>64.6</td>
<td>395.0</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>26</td>
<td>74</td>
<td>0</td>
<td>90.2</td>
<td>1.7</td>
</tr>
<tr>
<td>Z3</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.0</td>
<td>552.2</td>
</tr>
<tr>
<td><strong>fisher-yates SAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>14.0</td>
<td>7.5</td>
</tr>
<tr>
<td>Boolector$_{nop}$</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>17.3</td>
<td>7.0</td>
</tr>
<tr>
<td>Boolector$\beta$</td>
<td>6</td>
<td>1</td>
<td>11</td>
<td>15.0</td>
<td>76.4</td>
</tr>
<tr>
<td>CVC4</td>
<td>5</td>
<td>1</td>
<td>12</td>
<td>15.7</td>
<td>83.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>14.7</td>
<td>17.3</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>14</td>
<td>1</td>
<td>18.1</td>
<td>6.9</td>
</tr>
<tr>
<td>Z3</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>14.8</td>
<td>0.2</td>
</tr>
<tr>
<td><strong>fisher-yates UNSAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>5</td>
<td>13</td>
<td>1</td>
<td>17.4</td>
<td>7.1</td>
</tr>
<tr>
<td>Boolector$_{nop}$</td>
<td>4</td>
<td>14</td>
<td>1</td>
<td>18.2</td>
<td>6.9</td>
</tr>
<tr>
<td>Boolector$\beta$</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>12.1</td>
<td>72.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>3</td>
<td>4</td>
<td>12</td>
<td>19.2</td>
<td>82.1</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>15.9</td>
<td>14.7</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>19.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Z3</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>11.2</td>
<td>2.2</td>
</tr>
</tbody>
</table>

**Limits:** time: 1200s, memory: 7GB  
**Penalty:** TO: +1200s, MO: +1200s, +7GB
### Experiments

**Category:** crafted

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [10³ s]</th>
<th>Space [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>macro blow-up</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>24.2</td>
<td>9.4</td>
</tr>
<tr>
<td>Boolector&lt;sub&gt;nop&lt;/sub&gt;</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>18.2</td>
<td>8.4</td>
</tr>
<tr>
<td>Boolector&lt;sub&gt;β&lt;/sub&gt;</td>
<td>28</td>
<td>49</td>
<td>23</td>
<td>91.5</td>
<td>160.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.7</td>
<td>551.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>51</td>
<td>2</td>
<td>47</td>
<td>64.6</td>
<td>395.0</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>26</td>
<td>74</td>
<td>0</td>
<td>90.2</td>
<td>1.7</td>
</tr>
<tr>
<td>Z3</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.0</td>
<td>552.2</td>
</tr>
<tr>
<td><strong>fisher-yates SAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>14.0</td>
<td>7.5</td>
</tr>
<tr>
<td>Boolector&lt;sub&gt;nop&lt;/sub&gt;</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>17.3</td>
<td>7.0</td>
</tr>
<tr>
<td>Boolector&lt;sub&gt;β&lt;/sub&gt;</td>
<td>6</td>
<td>1</td>
<td>11</td>
<td>15.0</td>
<td>76.4</td>
</tr>
<tr>
<td>CVC4</td>
<td>5</td>
<td>1</td>
<td>12</td>
<td>15.7</td>
<td>83.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>14.7</td>
<td>17.3</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>14</td>
<td>1</td>
<td>18.1</td>
<td>6.9</td>
</tr>
<tr>
<td>Z3</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>14.8</td>
<td>0.2</td>
</tr>
<tr>
<td><strong>fisher-yates UNSAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>5</td>
<td>13</td>
<td>1</td>
<td>17.4</td>
<td>7.1</td>
</tr>
<tr>
<td>Boolector&lt;sub&gt;nop&lt;/sub&gt;</td>
<td>4</td>
<td>14</td>
<td>1</td>
<td>18.2</td>
<td>6.9</td>
</tr>
<tr>
<td>Boolector&lt;sub&gt;β&lt;/sub&gt;</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>12.1</td>
<td>72.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>3</td>
<td>4</td>
<td>12</td>
<td>19.2</td>
<td>82.1</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>15.9</td>
<td>14.7</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>19.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Z3</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>11.2</td>
<td>2.2</td>
</tr>
</tbody>
</table>

**Limits:** time: 1200s, memory: 7GB

**Penalty:** TO: +1200s, MO: +1200s, +7GB
<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time $[10^3\text{s}]$</th>
<th>Space $[\text{GB}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>macro blow-up</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>24.2</td>
<td>9.4</td>
</tr>
<tr>
<td>Boolector$_{nop}$</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>18.2</td>
<td>8.4</td>
</tr>
<tr>
<td>Boolector$_{\beta}$</td>
<td>28</td>
<td>49</td>
<td>23</td>
<td>91.5</td>
<td>160.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.7</td>
<td>551.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>51</td>
<td>2</td>
<td>47</td>
<td>64.6</td>
<td>395.0</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>26</td>
<td>74</td>
<td>0</td>
<td>90.2</td>
<td>1.7</td>
</tr>
<tr>
<td>Z3</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.0</td>
<td>552.2</td>
</tr>
<tr>
<td><strong>fisher-yates SAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>14.0</td>
<td>7.5</td>
</tr>
<tr>
<td>Boolector$_{nop}$</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>17.3</td>
<td>7.0</td>
</tr>
<tr>
<td>Boolector$_{\beta}$</td>
<td>6</td>
<td>1</td>
<td>11</td>
<td>15.0</td>
<td>76.4</td>
</tr>
<tr>
<td>CVC4</td>
<td>5</td>
<td>1</td>
<td>12</td>
<td>15.7</td>
<td>83.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>14.7</td>
<td>17.3</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>14</td>
<td>1</td>
<td>18.1</td>
<td>6.9</td>
</tr>
<tr>
<td>Z3</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>14.8</td>
<td>0.2</td>
</tr>
<tr>
<td><strong>fisher-yates UNSAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>5</td>
<td>13</td>
<td>1</td>
<td>17.4</td>
<td>7.1</td>
</tr>
<tr>
<td>Boolector$_{nop}$</td>
<td>4</td>
<td>14</td>
<td>1</td>
<td>18.2</td>
<td>6.9</td>
</tr>
<tr>
<td>Boolector$_{\beta}$</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>12.1</td>
<td>72.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>3</td>
<td>4</td>
<td>12</td>
<td>19.2</td>
<td>82.1</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>15.9</td>
<td>14.7</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>19.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Z3</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>11.2</td>
<td>2.2</td>
</tr>
</tbody>
</table>

**Limits:** time: 1200s, memory: 7GB  
**Penalty:** TO: +1200s, MO: +1200s, +7GB
## Experiments

**Category:** crafted

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [10^3s]</th>
<th>Space [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>macro blow-up</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>24.2</td>
<td>9.4</td>
</tr>
<tr>
<td>Boolector(_nop)</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>18.2</td>
<td>8.4</td>
</tr>
<tr>
<td>Boolector(_\beta)</td>
<td>28</td>
<td>49</td>
<td>23</td>
<td>91.5</td>
<td>160.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.7</td>
<td>551.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>51</td>
<td>2</td>
<td>47</td>
<td>64.6</td>
<td>395.0</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>26</td>
<td>74</td>
<td>0</td>
<td>90.2</td>
<td>1.7</td>
</tr>
<tr>
<td>Z3</td>
<td>21</td>
<td>0</td>
<td>79</td>
<td>95.0</td>
<td>552.2</td>
</tr>
<tr>
<td><strong>fisher-yates SAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>7</td>
<td>10</td>
<td>1</td>
<td>14.0</td>
<td>7.5</td>
</tr>
<tr>
<td>Boolector(_nop)</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>17.3</td>
<td>7.0</td>
</tr>
<tr>
<td>Boolector(_\beta)</td>
<td>6</td>
<td>11</td>
<td>1</td>
<td>15.0</td>
<td>76.4</td>
</tr>
<tr>
<td>CVC4</td>
<td>5</td>
<td>1</td>
<td>12</td>
<td>15.7</td>
<td>83.6</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>14.7</td>
<td>17.3</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>14</td>
<td>1</td>
<td>18.1</td>
<td>6.9</td>
</tr>
<tr>
<td>Z3</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>14.8</td>
<td>0.2</td>
</tr>
<tr>
<td><strong>fisher-yates UNSAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolector</td>
<td>5</td>
<td>13</td>
<td>1</td>
<td>17.4</td>
<td>7.1</td>
</tr>
<tr>
<td>Boolector(_nop)</td>
<td>4</td>
<td>14</td>
<td>1</td>
<td>18.2</td>
<td>6.9</td>
</tr>
<tr>
<td>Boolector(_\beta)</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>12.1</td>
<td>72.0</td>
</tr>
<tr>
<td>CVC4</td>
<td>3</td>
<td>4</td>
<td>12</td>
<td>19.2</td>
<td>82.1</td>
</tr>
<tr>
<td>MathSAT</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>15.9</td>
<td>14.7</td>
</tr>
<tr>
<td>SONOLAR</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>19.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Z3</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>11.2</td>
<td>2.2</td>
</tr>
</tbody>
</table>

**Limits:** time: 1200s, memory: 7GB

**Penalty:** TO: +1200s, MO: +1200s, +7GB
Experiments
Category: SMT'12

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [10^3 s]</th>
<th>Space [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolector</td>
<td>139</td>
<td>10</td>
<td>0</td>
<td>19.9</td>
<td>14.8</td>
</tr>
<tr>
<td>Boolector_{nop}</td>
<td>134</td>
<td>15</td>
<td>0</td>
<td>26.3</td>
<td>14.5</td>
</tr>
<tr>
<td>Boolector_β</td>
<td>137</td>
<td>11</td>
<td>1</td>
<td>21.5</td>
<td>22.7</td>
</tr>
<tr>
<td>Boolector_{sc12}</td>
<td>140</td>
<td>9</td>
<td>0</td>
<td>15.9</td>
<td>10.3</td>
</tr>
</tbody>
</table>

- Boolector solves 5 instances that Boolector_β couldn’t
- Boolector_β solves 3 instances that Boolector couldn’t
- combined they solve 2 instances that Boolector_{sc12} couldn’t

Limits: time: 1200s, memory: 7GB
Penalty: TO: +1200s, MO: +1200s, +7GB
### Experiments

**Category:** application

<table>
<thead>
<tr>
<th>Instantiation</th>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [s]</th>
<th>Space [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolector</td>
<td>37</td>
<td>8</td>
<td>0</td>
<td>576</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>Boolector\textsubscript{nop}</td>
<td>35</td>
<td>10</td>
<td>0</td>
<td>673</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>Boolector\textsubscript{β}</td>
<td>44</td>
<td>1</td>
<td>0</td>
<td>138</td>
<td>961</td>
<td></td>
</tr>
<tr>
<td>Boolector\textsubscript{sc12}</td>
<td>39</td>
<td>6</td>
<td>0</td>
<td>535</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>STP</td>
<td>44</td>
<td>1</td>
<td>0</td>
<td>141</td>
<td>3814</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\lambda^2) benchmarks</th>
<th>Boolector</th>
<th>37</th>
<th>8</th>
<th>0</th>
<th>594</th>
<th>236</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lambda^2) benchmarks</td>
<td>Boolector\textsubscript{nop}</td>
<td>35</td>
<td>10</td>
<td>0</td>
<td>709</td>
<td>166</td>
</tr>
<tr>
<td>(\lambda^2) benchmarks</td>
<td>Boolector\textsubscript{β}</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>52</td>
<td>676</td>
</tr>
<tr>
<td>(\lambda^2) benchmarks</td>
<td>Boolector\textsubscript{sc12}</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\lambda^2) benchmarks</td>
<td>STP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Limits:** time: 60s, memory: 7GB

**Penalty:** TO: +60s, MO: +60s, +7GB

\(^2\text{lambda benchmarks kindly provided by Carsten Sinz et. al.}\)
# Experiments

**Category:** application

<table>
<thead>
<tr>
<th></th>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [s]</th>
<th>Space [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boolector</td>
<td>37</td>
<td>8</td>
<td>0</td>
<td>576</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>Boolector\textsubscript{nop}</td>
<td>35</td>
<td>10</td>
<td>0</td>
<td>673</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>Boolector\textsubscript{β}</td>
<td>44</td>
<td>1</td>
<td>0</td>
<td>138</td>
<td>961</td>
</tr>
<tr>
<td></td>
<td>Boolector\textsubscript{sc\textsubscript{12}}</td>
<td>39</td>
<td>6</td>
<td>0</td>
<td>535</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>STP</td>
<td>44</td>
<td>1</td>
<td>0</td>
<td>141</td>
<td>3814</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [s]</th>
<th>Space [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boolector</td>
<td>37</td>
<td>8</td>
<td>0</td>
<td>594</td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>Boolector\textsubscript{nop}</td>
<td>35</td>
<td>10</td>
<td>0</td>
<td>709</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>Boolector\textsubscript{β}</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>52</td>
<td>676</td>
</tr>
<tr>
<td></td>
<td>Boolector\textsubscript{sc\textsubscript{12}}</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>STP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Limits:** time: 60s, memory: 7GB

**Penalty:** TO: +60s, MO: +60s, +7GB

\textsuperscript{2}\textit{lambda} benchmarks kindly provided by Carsten Sinz et. al.
### Experiments

**Category:** application

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [s]</th>
<th>Space [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolector</td>
<td>37</td>
<td>8</td>
<td>0</td>
<td>576</td>
<td>235</td>
</tr>
<tr>
<td>Boolector_nop</td>
<td>35</td>
<td>10</td>
<td>0</td>
<td>673</td>
<td>196</td>
</tr>
<tr>
<td>Boolector_β</td>
<td>44</td>
<td>1</td>
<td>0</td>
<td>138</td>
<td>961</td>
</tr>
<tr>
<td>Boolector_sc12</td>
<td>39</td>
<td>6</td>
<td>0</td>
<td>535</td>
<td>308</td>
</tr>
<tr>
<td>STP</td>
<td>44</td>
<td>1</td>
<td>0</td>
<td>141</td>
<td>3814</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>TO</th>
<th>MO</th>
<th>Time [s]</th>
<th>Space [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolector_nop</td>
<td>35</td>
<td>10</td>
<td>0</td>
<td>709</td>
<td>166</td>
</tr>
<tr>
<td>Boolector_β</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>52</td>
<td>676</td>
</tr>
<tr>
<td>Boolector_sc12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>STP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Limits:** time: 60s, memory: 7GB

**Penalty:** TO: +60s, MO: +60s, +7GB

\(^2\)\textit{lambda} benchmarks kindly provided by Carsten Sinz et. al.
Conclusion

• DP_\lambda, a decision procedure for non-recursive, non-extensional \lambda-terms
  \rightarrow consistent_\lambda, lemma_\lambda

• experimental results look promising
  \rightarrow category application demonstrates potential of native \lambda-term support

• still room for improvements
  \rightarrow optimization of \beta-reduction
  \rightarrow no \lambda-term specific rewriting yet

Future Work

• rewriting rules for \lambda-terms
• better \beta-reduction implementation
• various \beta-reduction strategies
• extensionality on \lambda-terms
• quantifiers
Lemmas on Demand for Lambdas

Mathias Preiner, Aina Niemetz and Armin Biere

Institute for Formal Models and Verification (FMV)
Johannes Kepler University, Linz, Austria
http://fmv.jku.at/

DIFTS Workshop 2013
October 19, 2013
Portland, OR, USA
References

CVC4.
In CAV, volume 6806 of LNCS, pages 171–177. Springer.

Lemmas on Demand for the Extensional Theory of Arrays.
JSAT, 6(1-3):165–201.

Modeling and Verifying Systems Using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions.
In CAV, volume 2404 of LNCS, pages 78–92. Springer.

The Yices SMT solver.

Extending the Theory of Arrays: memset, memcpy, and Beyond.

Towards a Mathematical Science of Computation.
In IFIP Congress, pages 20–50.