# Generalized and Efficient Array Decision Procedures

FMCAD, Austin, 2009

Leonardo de Moura and Nikolaj Bjørner
Microsoft Research

# Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

Microsoft Research

# Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

Many Flavors:

SAT Solvers

**SMT Solvers**

First-order Theorem Provers

Computer Algebra Systems

Microsoft Research

# Satisfiability Modulo Theories (SMT)

**Is formula *F* satisfiable modulo theory *T* ?**

Microsoft Research

# Satisfiability Modulo Theories (SMT)

**Is formula *F* satisfiable modulo theory *T* ?**

Arithmetic,

Bit-vectors,

**Arrays**,

Inductive data-types,

….

Microsoft Research

# Satisfiability Modulo Theories (SMT)

Example:

**1>2**

Satisfiable if the symbols 1,2 and > are uninterpreted.

$|M| = \{ \bullet \}$

$M(1) = M(2) = \bullet$

$M(>) = \{ (\bullet, \bullet) \}$

Unsatisfiable modulo the theory arithmetic

Microsoft Research

# Satisfiability Modulo Theories (SMT)

$$b + 2 = c \ \text{ and } \ f(select(store(a,b,3), c-2) \neq f(c-b+1)$$

# Satisfiability Modulo Theories (SMT)

$$\boxed{b + 2 = c} \text{ and } f(select(store(a,b,\boxed{3}), \boxed{c-2}) \neq f(\boxed{c-b+1})$$

Arithmetic

# Satisfiability Modulo Theories (SMT)

$$b + 2 = c \ \ and \ \ f(\boxed{select}(\boxed{store}(a,b,3), c-2) \neq f(c-b+1)$$

Array Theory

# Satisfiability Modulo Theories (SMT)

$$b + 2 = c \;\; \text{and} \;\; \boxed{f(}select(store(a,b,3), c\text{-}2) \neq \boxed{f(}c\text{-}b+1)$$

Uninterpreted Functions

# Applications

Test case generation

Verifying Compilers

Predicate Abstraction

Invariant Generation

Type Checking

Model Based Testing

Microsoft Research

# Some Applications @ Microsoft

- The Spec# Programming System
- HAVOC
- ForμLa
- Hyper-V Microsoft Virtualization
- Terminator T-2
- VCC
- SLAM
- Yogi
- NModel
- Vigilante
- SpecExplorer
- Pex
- F7
- SAGE
- Prefix

Microsoft Research

# What is a Theory?

A theory $T$ is a set of first-order sentences.

$F$ is satisfiable modulo $T$

iff

$T \cup F$ is satisfiable.

Microsoft Research

# Array Theory

$$\forall a, i, v.\ select(store(a, i, v), i) = v$$

$$\forall a, i, \mathrm{j}, v\!:\ i = j \lor select(store(a, i, v), j) = \mathrm{select}(\mathrm{a}, j)$$

Microsoft Research

# Array Theory

$\forall a, i, v.\ select(store(a, i, v), i) = v$

$\forall a, i, j, v{:}\ \ i = j \vee select(store(a, i, v), j) = select(a, j)$

We say *store* is a **combinator**.

Microsoft Research

# Array Theory: a more familiar notation

$\forall a, i, v.\ select(store(a, i, v), i) = v$

$\forall a, i, j, v{:}\ \ i = j \vee select(store(a, i, v), j) = select(a, j)$

$\forall a, i, v.\ store(a, i, v)[i] = v$

$\forall a, i, j, v{:}\ \ i = j \vee store(a, i, v)[j] = a[i]$

Microsoft Research

# Why array theory is useful?

It is used to model the memory

in

Hardware/Software verification/analysis tools

Microsoft Research

# Extentional Array Theory

$$\forall a, b: (\forall i: a[i] = b[i]) \Rightarrow a = b$$

Microsoft® Research

# Arrays are actually “maps”

We have arrays from $T_1$ to $T_2$

$T_1$ does not need to be the Integers

# Models for arrays as "finite graphs"

$a = store(b, 0, 5)$, $b = store(c, 1, 10)$, $c[0] = 2$

$M(a) = \{ 0 \rightarrow 5, 1 \rightarrow 10, \text{else} \rightarrow 0 \}$

$M(b) = \{ 0 \rightarrow 2, 1 \rightarrow 10, \text{else} \rightarrow 0 \}$

$M(c) = \{ 0 \rightarrow 2, \text{else} \rightarrow 0 \}$

Microsoft Research

# A "Timeline" (Related Work)

1962 - McCarthy proposes the Basic Array Theory.

1968 - Kaplan solves the satisfiability problem.

1981 - Nelson propose a simple procedure based on (lazy) instantiation (PhD thesis).

2001 - Stump, Barrett, Dill and Levitt propose a procedure for extentional arrays.

2005 - Lazy instantiation is used in Yices
(it wins all array divisions in SMT-COMP from 2005 - 2007).

2005 - Kapur and Zarba propose the reduction approach
(many array-like theories are described).

2006 - Bradley, Manna and Sipma propose a procedure for a rich decidable array fragment.

Microsoft Research

# A “Timeline” (Related Work)

2008 - Goel, Krstic and Fuchs formalize the lazy instantiation approach.

2008 - Bofill, Nieuwenhuis, Oliveras, Rodriguez-Carbonell and Rubio propose the store-reduction approach

“Model-Based” approaches:

2007 - Ganesh and Dill, “a decision procedure for bitvectors and arrays”, CAV’07

2008 - Brummayer and Biere, “lemmas on demand for the extentional theory of arrays”, SMT’08

# A “Timeline” (Related Work)

“Rewrite-Based” approaches:

2002 - Lynch and Morawska, “Automatic Decidability”, LICS

2005 - Armando, Bonacina, Ranise and Schulz propose the rewrite based approach.

Arrays in hardware verification:

1994 - Burch and Dill, “Automatic Verification of pipelined microprocessor control”, CAV

2006 - Manolios, Srinivasan, Vroon, “Automatic memory reductions for RTL model verification”, ICCAD

More relevant work can be found in our paper…

# Naïve instantiation

Recipe: Given a formula F

1) Collect all array terms in F
2) Collect all indices in F
3) Instantiate array axioms using 1 and 2

   $F' = F \cup \text{Instances}$

4) Execute EUF solver on F'

Array theory is a local theory extension.

Microsoft Research

# Naïve instantiation: Example

$a = store(b, i, v), a[j] \neq v, c[k] = v, i = j$

array terms: $a, b, store(b, i, v), c$

indices: $i, j, k$

Microsoft Research

# Naïve instantiation: Example

$a = store(b, i, v), a[j] \neq v, c[k] = v, i = j$

array terms: $a, b, store(b, i, v), c$

indices: $i, j, k$

Instances:

$store(a, i, v)[i] = v, store(a, j, v)[j] = v, \ldots$

$i = j \vee store(a, i, v)[j] = \mathrm{a[i]}, \ldots$

Problem: Many useless instances!

Microsoft Research

# Naïve instantiation: Example

$a = store(b, i, v), a[j] \neq v, c[k] = v, i = j$

array terms: $a, b, store(b, i, v), c$

indices: $i, j, k$

Lazy instantiation: select a small subset of instances. (more later)

Instances:

$store(a, i, v)[i] = v$, store($a, j, v$ … $v$, …

$i = j \vee store(a, i, v)[j] =$ a[i], …

Problem: Many useless instances!

# Our contributions

A generalization of the Array theory

CAL: Combinatory Array Logic

New filters for minimizing the number of instances

A simple architecture for non-stably infinite theories

We want arrays of bit-vectors.

# CAL: Combinatory Array Logic

$\forall v, i\colon K(v)[i] = v$

$\forall a_1, \ldots, a_n, i\colon map_f(a_1, \ldots, a_n)[i] = f(a_1[i], \ldots, a_n[i])$

# CAL: Combinatory Array Logic

Suggested by Stump, Barrett, Dill, Levitt
Their procedure works for
infinite-domain satisfiability.

$\forall v, i: K(v)[i] = v$

$\forall a_1, \ldots, a_n, i: map_f(a_1, \ldots, a_n)[i] = f(a_1[i], \ldots, a_n[i])$

# CAL: Combinatory Array Logic

$\forall v, i$: $K(v)[i] = v$

$\forall a_1, \ldots, a_n, i$: $map_f(a_1, \ldots, a_n)[i] = f(a_1[i], \ldots, a_n[i])$

"Family" of combinators.
We can instantiate it with any $f$.

Microsoft Research

# $map_f$ is the pointwise function application

$$map_f(\; \boxed{\;\ldots \mid v_1 \mid v_2 \mid v_3 \mid v_4 \mid v_5 \mid \ldots\;}\;,\; \boxed{\;\ldots \mid w_1 \mid w_2 \mid w_3 \mid w_4 \mid w_5 \mid \ldots\;}\;)$$

$$=$$

$$\boxed{\;\ldots \mid f(v_1,w_1) \mid f(v_2,w_2) \mid f(v_3,w_3) \mid f(v_4,w_4) \mid f(v_5,w_5) \mid \ldots\;}$$

# CAL is powerful: Sets as arrays

Set of T as an Array from T to Boolean

| | | |
|---|---|---|
| $\emptyset$ | $\equiv$ | $K(false)$ |
| $\{a\}$ | $\equiv$ | $store(\emptyset, a, true)$ |
| $a \in S$ | $\equiv$ | $S[a]$ |
| $S_1 \cup S_2$ | $\equiv$ | $map_{\vee}(S_1, S_2)$ |
| $S_1 \cap S_2$ | $\equiv$ | $map_{\wedge}(S_1, S_2)$ |

Microsoft Research

# CAL is powerful: Sets as arrays

Set of T as an Array from T to Boolean

$$\begin{array}{lcl}
\emptyset & \equiv & K(false) \\
\{a\} & \equiv & store(\emptyset, a, true) \\
a \in S & \equiv & S[a] \\
S_1 \cup S_2 & \equiv & map_{\vee}(S_1, S_2) \\
S_1 \cap S_2 & \equiv & map_{\wedge}(S_1, S_2)
\end{array}$$

But not cardinality $|S|$, power-set, ...

# CAL is powerful: Bags as arrays

Bag of T as an Array from T to Integer

$$\emptyset \equiv K(0)$$

$$\{a\} \equiv store(\emptyset, a, 1)$$

$$mult(a, B) \equiv B[a]$$

$$B_1 \oplus B_2 \equiv map_+(B_1, B_2)$$

$$B_1 \prod B_2 \equiv map_{\min}(B_1, B_2)$$

Microsoft Research

# CAL is powerful: a multiplexer

$$map_{ite}\left(\begin{array}{|c|c|c|c|c|c|c|}\hline \dots & T & F & T & T & F & \dots \\ \hline\end{array}\;,\right.$$

$$\begin{array}{|c|c|c|c|c|c|c|}\hline \dots & \boldsymbol{v_1} & v_2 & \boldsymbol{v_3} & \boldsymbol{v_4} & v_5 & \dots \\ \hline\end{array}\;,$$

$$\left.\begin{array}{|c|c|c|c|c|c|c|}\hline \dots & w_1 & \boldsymbol{w_2} & w_3 & w_4 & \boldsymbol{w_5} & \dots \\ \hline\end{array}\right)$$

$$=$$

$$\begin{array}{|c|c|c|c|c|c|c|}\hline \dots & \boldsymbol{v_1} & \boldsymbol{w_2} & \boldsymbol{v_3} & \boldsymbol{v_4} & \boldsymbol{w_5} & \dots \\ \hline\end{array}$$

# Core solver

Support for equality and uninterpreted functions (EUF)
Set of strongly disjoint theories (more later)
Clauses and literals
Boolean terms

$a \equiv t$ – $a$ is a name for the term $t$
$a$: σ – $a$ has sort σ
$a \sim b$ – $a$ and $b$ are equal in the current context

$$\frac{w_1 \equiv f(v_1, \ldots, v_n),\ w_2 \equiv f(v'_1, \ldots, v'_n),\ v_1 \sim v'_1, \ldots, v_n \sim v'_n}{w_1 \simeq w_2}$$

Microsoft Research

# Array Saturation Rules
## (this is not new)

$$\text{idx}\ \frac{a \equiv store(b,i,v)}{a[i] \simeq v}$$

$$\Downarrow\ \frac{a \equiv store(b,i,v), \quad w \equiv a'[j], \quad a \sim a'}{i \simeq j \vee a[j] \simeq b[j]}$$

$$\Uparrow\ \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

$$\text{ext}\ \frac{a\colon(\sigma \Rightarrow \tau), \quad b\colon(\sigma \Rightarrow \tau)}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

$a \sim b$ – *a* and *b* are equal in the current context

$a \equiv t$ – *a* is a name for the term *t*

$a$:$(\sigma \Rightarrow \tau)$ – *a* is an array from σ to τ

# Bottlenecks

$$\text{ext}\ \frac{a\colon(\sigma \Rightarrow \tau), \quad b\colon(\sigma \Rightarrow \tau)}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Extensionality is applied to every pair of array constants.

$$\Uparrow\ \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

Upwards propagation distributes index over all modifications of same array.

Microsoft Research

# Bottlenecks: simple "tricks"

$$\text{ext}\ \frac{a:(\sigma \Rightarrow \tau), \quad b:(\sigma \Rightarrow \tau)}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Extensionality is applied to every pair of array constants.

Delay the application of ext and $\Uparrow$.

Only works for unsatisfiable instances.

$$\Uparrow\ \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

Upwards propagation distributes index over all modifications of same array.

Microsoft Research

# Bottlenecks: simple "tricks"

Ignore "congruent" axiom instances

$$i \simeq j \vee a[j] \simeq b[j]$$

$$i' \simeq j' \vee a'[j'] \simeq b'[j']$$

$$a \sim a',\ b \sim b',\ i \sim i',\ \text{and}\ \ j \sim j'$$

# Bottlenecks

$$\text{ext}\ \frac{a\colon(\sigma \Rightarrow \tau), \quad b\colon(\sigma \Rightarrow \tau)}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Extensionality is applied to every
pair of array constants.

$$\text{ext}_{\not\simeq}\ \frac{p \equiv a \simeq b, \quad \Gamma(p) = \text{false}}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

$$\text{ext}_r\ \frac{a\colon(\sigma \Rightarrow \tau), \quad b\colon(\sigma \Rightarrow \tau), \quad \{a, b\} \subseteq \text{foreign}}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Restrict to constants asserted to be different or foreign.
We say *a* is foreign if there is *b* s.t. $a \sim b$ and
*b* is the argument of an uninterpreted function symbol.

Microsoft Research

# Why do we need $\text{ext}_r$?

$$\text{ext}_{\not\simeq}\ \frac{p \equiv a \simeq b, \quad \Gamma(p) = \text{false}}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

$$\text{ext}_r\ \frac{a\!:(\sigma \Rightarrow \tau), \quad b\!:(\sigma \Rightarrow \tau), \quad \{a, b\} \subseteq \text{foreign}}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Example:

a = store(b, i, v), b[i] = v, f(a) ≠ f(b)

# Another optimization…

We do not need to add the extensionality axiom for (*a*,*b*) if they are already known to be disequal.

*Definition 9 (Already Disequal)* Given a state $\Gamma$, $(a, b) \in$ already-diseq iff there are two definitions $v_1 \equiv a_1[i_1]$ and $v_2 \equiv a_2[i_2]$ in $\Gamma$ such that $v_1 \not\sim v_2$, $a \sim a_1$, $b \sim b_1$, and $i_1 \sim i_2$.

# Another optimization…

We do not need to add the extensionality axiom for (*a*,*b*) if they are already known to be disequal.

*Definition 9 (Already Disequal)* Given a state $\Gamma$, $(a, b) \in$ already-diseq iff there are two definitions $v_1 \equiv a_1[i_1]$ and $v_2 \equiv a_2[i_2]$ in $\Gamma$ such that $v_1 \not\sim v_2$, $a \sim a_1$, $b \sim b_1$, and $i_1 \sim i_2$.

Typo in the paper!
Should be $b_1$

# Why is $\Uparrow$ expensive?

$$\Uparrow \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

Scenario from software verification

Bunch of facts about the initial state of the heap

$a_0[i_0] = v_0,\ a_0[i_1] = v_1, a_0[i_2] = v_2, \ldots$

Perform a series of updates

$a_1 = store(a_0, j_1, w_1), a_2 = store(a_1, j_2, w_2), \ldots$

Check some property on the final heap

$a_n[k] \neq v$

Microsoft Research

# Why do we need $\Uparrow$?

$store(a, i, v_1) = store(b, i, v_2)$, $i \neq k$, $a[k] \neq b[k]$

*Definition 10 (Linearity)* Given a state $\Gamma$, the set non-linear of *non-linear constants* is the least set such that:

1. $a_1 \equiv store(b_1, i_1, v_1)$, $a_2 \equiv store(b_2, i_2, v_2)$, $a_1$ is not $a_2$ and $a_1 \sim a_2$ implies $\{a_1, a_2\} \subseteq$ non-linear,
2. $a \equiv store(b, i, v)$ and $a \in$ non-linear implies $b \in$ non-linear,
3. $a \in$ non-linear and $a \sim b$ implies $b \in$ non-linear.

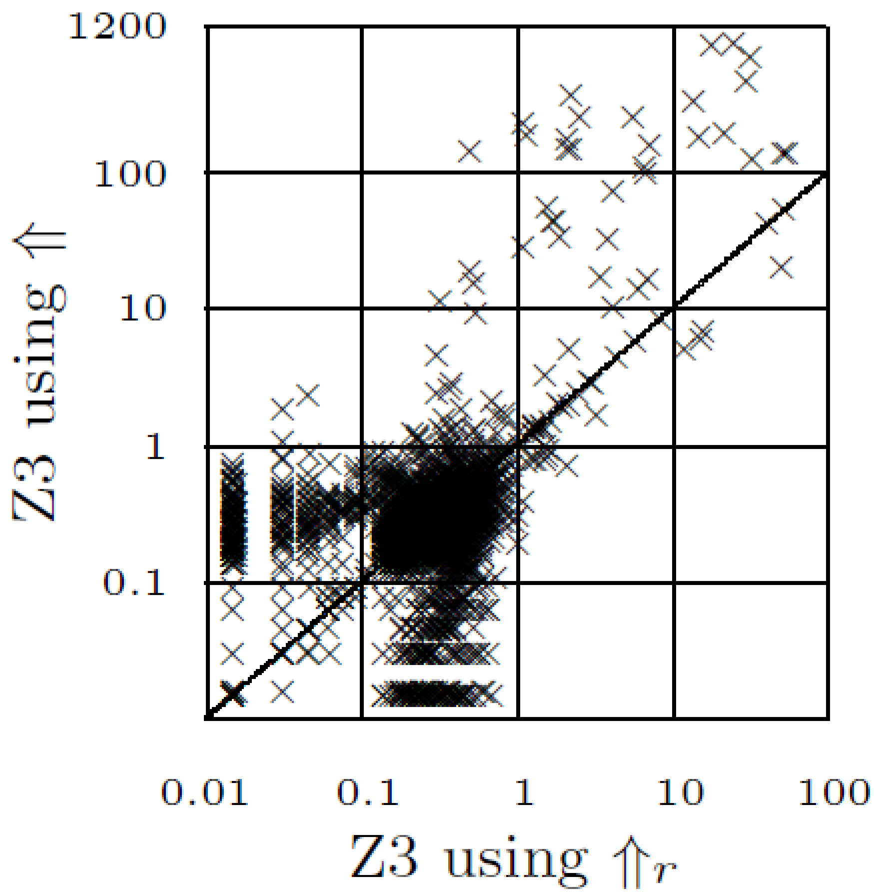
We say $a$ is *linear* if $a \notin$ non-linear.

# Restricting $\Uparrow$

$$\Uparrow \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

$$\Uparrow_r \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b', \quad \boxed{b \in \text{non-linear}}}{i \simeq j \vee a[j] \simeq b[j]}$$

# Effect on Benchmarks




Microsoft Research

# Saturating CAL

$$\mathsf{K}\Downarrow\ \frac{a \equiv K(v), \quad w \equiv a'[j], \quad a \sim a'}{a[j] \simeq v}$$

$$\mathsf{map}\Downarrow\ \frac{a \equiv map_f(b_1,\ldots,b_n), \quad w \equiv a'[j], \quad a \sim a'}{a[j] \simeq f(b_1[j],\ldots,b_n[j])}$$

$$\mathsf{map}\Uparrow\ \frac{\begin{array}{c} a \equiv map_f(b_1,\ldots,b_n), \quad w \equiv b'_k[j], \\ b_k \sim b'_k, \text{ for some } \ k \in \{1,\ldots,n\} \end{array}}{a[j] \simeq f(b_1[j],\ldots,b_n[j])}$$

$$\epsilon\not\simeq\ \frac{v \equiv a[i], \quad i:\sigma, \quad i \text{ is not } \epsilon_\sigma}{\epsilon_\sigma \not\simeq i} \qquad \epsilon\delta\ \frac{a:(\sigma \Rightarrow \tau)}{a[\epsilon_\sigma] \simeq \delta_a}$$

Microsoft Research

# Saturating CAL

$$\epsilon_{\not\simeq}\ \frac{v \equiv a[i], \quad i\!:\!\sigma, \quad i \text{ is not } \epsilon_\sigma}{\epsilon_\sigma \not\simeq i}$$

Potentially unsound if *F* only has models M where M(σ) is finite.

$$\text{blast}\ \frac{a\!:\!(\sigma \Rightarrow \tau), \quad \mathsf{size}(\sigma) = k}{a[\sigma_1] \simeq \delta_{a,1}, \ \ldots, \ a[\sigma_k] \simeq \delta_{a,k}}$$

Microsoft Research

# Saturating CAL

We also have a restricted version of map$\Uparrow$ using linear stratification (see paper for details).

$$a \simeq map_{ite}(a, b, c) \;\wedge\; b[j] \simeq \bot \;\wedge\; c[j] \simeq \top$$

Default-value extension (new theory symbol δ), and alternative for $\epsilon_{\not\simeq}$ and $\epsilon\delta$

$$\text{U}\delta \; \frac{a \equiv store(b, i, v)}{\delta(a) \simeq \delta(b)} \qquad \text{K}\delta \; \frac{a \equiv K(v)}{\delta(a) \simeq v}$$

$$\text{map}\delta \; \frac{a \equiv map_f(b_1, \ldots, b_n)}{\delta(a) \simeq f(\delta(b_1), \ldots, \delta(b_n))}$$

Microsoft Research

# Theory combination in Z3

**Efficient Core**

Strongly disjoint theories + Uninteprеted functions

Strongly disjoint theory ≡ Sort disjoint

Examples: Arithmetic, Bitvectors and Booleans

$$f(\top) \simeq w \ \wedge \ f(\bot) \simeq w \ \wedge \ f(v) \not\simeq w$$

All other theories are reduced to this core.

Not covered today: inductive datatypes.

Microsoft Research

# Conclusion

Arrays are useful in practice.

They are used in many verification tools at Microsoft.

CAL is a useful extension of the array theory.

Simple combination architecture.
Efficient and easy to implement.

# Conclusion

Arrays are useful in practice.

They are used in many verification tools at Microsoft.

CAL is a useful extension of the array theory.

Simple combination architecture.
Efficient and easy to implement.

Thank You!

Microsoft Research