Safety and correctness: essential for tensor languages?

Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

Dagstuhl Seminar 20111
Tensor Computations: Applications and Optimization
8–13 March 2020
Introduction
Tensor languages

- Inflation of tensor frameworks in recent years.
  - Each with its own tensor language

- tensor = (high-dimensional) array

- “Classical” examples of array languages:
  - APL, Fortran, Matlab, R, C++/Eigen, Python/Numpy
  - SaC, ML and Haskell with array libraries (e.g. Repa, Accelerate), Futhark, Lift

- Functional array languages enjoy type safety/memory safety properties.
  - E.g. absence of out-of-bounds accesses in well-typed programs

- Recent tensor languages are imperative.
  - Type safety? Correctness of implementation?
Need a formal specification for formal type-safety results.
- Tensor languages from the frameworks not formally specified.

Tell: an imperative Tensor Intermediate Language
- Common denominator for reasoning about imperative tensor languages
- Formal specification and type-safety in Coq
- No out-of-bounds accesses in well-typed Tell programs
- collective operations, aka. combinators

Formal specification also enables
- correctness proofs for implementations (compiler, runtime system...)
- rigorous and consistent development of tools.

Helps address the “DSL mess”.
Type discipline and memory safety
The TVM framework: example kernel

**TVM — Tensor Virtual Machine**

A = placeholder((m,h), name='A')
B = placeholder((n,h), name='B')
k = reduce_axis((0, h), name='k')
C = compute((m, n), lambda i, j:
             sum(A[k, i] * B[k, j], axis=k))

\[ C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj} \]

Segmentation fault or silent data corruption.

A = placeholder((h,m), name='A')
B = placeholder((h,n), name='B')
k = reduce_axis((0, h), name='k')
C = compute((m, n), lambda i, j:
             sum(A[k, i] * B[k, j], axis=k))

\[ C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj} \]
Syntax:

\[
\text{(program)} ::= \text{(alloc)* (stmt)*} \\
\text{(alloc)} ::= \text{alloc}\langle\text{id}\rangle : [i, \ldots, i] \\
\text{(stmt)} ::= \langle\text{id}\rangle = \langle\text{expr}\rangle \\
\text{(expr)} ::= \langle\text{id}\rangle | (\langle\text{expr}\rangle) \\
\text{ } | \text{add}\langle\text{expr}\rangle\langle\text{expr}\rangle | \text{mul}\langle\text{expr}\rangle\langle\text{expr}\rangle \\
\text{ } | \text{transp}\text{i i}\langle\text{expr}\rangle \text{| diag}\text{i i}\langle\text{expr}\rangle \\
\text{ } | \text{expa}\text{i i}\langle\text{expr}\rangle \text{| proj}\text{i i}\langle\text{expr}\rangle \\
\text{ } | \text{prod}\langle\text{expr}\rangle\langle\text{expr}\rangle \text{| red+}\text{i i}\langle\text{expr}\rangle \\
\text{ } | \text{conv}\text{i i}\langle\text{expr}\rangle
\]

- Tensor-valued variables are declared with alloc.
- Declaration assigns a type (shape) to the variable.
- Expressions are built from collective operations.

Theorem (type-safety):

If $\Gamma_{\text{allocs}} \vdash \text{allocs stmts} : \text{ok}$, then there exists a memory $\mu'$ such that

- $\langle\mu_{\text{allocs}}, \text{allocs stmts}\rangle \Downarrow \mu'$
- $\mu' \sim \mu_{\text{allocs}}$

Equivalence of memories:

$\mu_1 \sim \mu_2$ iff the memories $\mu_1$ and $\mu_2$ have the same domains.

(NA Rink, J Castrillon. ARRAY 2019)
Type/shape discipline

- TellL enforces a strict discipline for the types/shapes of tensors:
  - Every variable has to be declared with a specific shape
  - Operations will only work if argument shapes match exactly

- Contrast with Numpy/Tensorflow:
  - Leads to interesting effects during learning
  - One-hot encoding for MNIST labels:

```python
x = np.array(range(4))  # x.shape = (4,)
z = np.eye(4)  # z.shape = (4, 4)
x + z  # (x + z).shape = (4, 4)

y_train_HOT = np.array([np.eye(1, M=10, k=y) for y in y_train])  # y_train_HOT.shape = (60000, 1, 10)
# should be: (60000, 10)
# This works without error messages:
np.sum((y_pred - y_train_HOT)**2)
```

- 1’s prepended to the shape of x
- Broadcasting of the resulting first dimension of x
Outline

Towards provably correct compilation
The TACO framework: example kernels

**TACO – Tensor Algebra Compiler**
(F Kjolstad, S Kamil, S Chou, D Lugato, S Amarasinghe. OOPSLA 2017)

\[ A(i,j) = B(i,j,k) \times C(k) \]

\[ A_{ij} = \sum_{k=1}^{n} B_{ijk}C_{k} \]

\[ A(i,j) = B(i,j,k) \times C(i) \]

\[ A_{ij} = \left( \sum_{k=1}^{n} B_{ijk} \right)C_{i} \]

\[ A(i,j) = B(k,j,i) \times C(i) \]

\[ A_{ij} = \left( \sum_{k=1}^{n} B_{kji} \right)C_{i} \]

Compiler loops on this.
Why provably correct compilation?

- Reliable or trustworthy systems require correct compilation:
  - Compiler bugs can change the meaning of compiled programs.
  - Compromises safety and security.

- What are correctness requirements for tensor kernels or tensor-based applications?
  - Possible attack vectors when executed with concurrent processes (e.g. on multi-core CPUs)?

- Verification of programs and **program transformations** at source code level relies on correct compilation.

  e.g. loop transformations, optimizations

https://deepspec.org/main
In 2011, the Csmith tool found 202 bugs in LLVM, 79 bugs in GCC, 0 bugs in CompCert (middle-end).

(X Yang, Y Chen, E Eide, J Regehr. PLDI 2011)

For a correct DSL/TeIL compiler, could build on existing provably correct compilers

- CompCert (X Leroy. Communications of the ACM 2009)
- CakeML (R Kumar, MO Myreen, M Norrish, S Owens. POPL 2014)

How realistic is this? Is it worth the effort?
Potential performance of generated code

Matrix multiplication, quadratic matrices, single precision, 10 iterations:
(Intel Core i3-4160, 3.60GHz)

- gcc -ffast-math
- clang -ffast-math
- ccomp
- ocaml -unsafe
- ocaml -unsafe -unbox
- LAPACK

≈ 3 ... 10x
≈ 10x

Need a more convincing approach to correct compilation for tensor/array languages!
Questions
(instead of a summary)
Safety & correctness: how much should we care?

- Type safety for productivity in the tensor/machine learning domain:
  - How useful is static detection of simple type or shape errors?
  - What other errors or productivity issue can be caught by static analysis in tensor languages?

- Memory safety for safety & security:
  - How critical is this for tensor languages or frameworks?
    - e.g. erroneous predictions in safety-critical machine learning applications (automotive)
  - How can buffer overflows in machine learning or scientific applications be exploited?
    - Will we see attacks or exploits through machine learning applications?
    - e.g. in shared-memory or cloud environments

- Systems, compilers, and PL communities are increasingly presenting formally verified code as part of their research results.
  - How is code verified/validated in the scientific computing and machine learning domains?
Safety and correctness: essential for tensor languages?

Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

Dagstuhl Seminar 20111
Tensor Computations: Applications and Optimization
8–13 March 2020