Opening Keynote

Prepared for DConf 2016

Andrei Alexandrescu, Ph.D.

2016-05-04

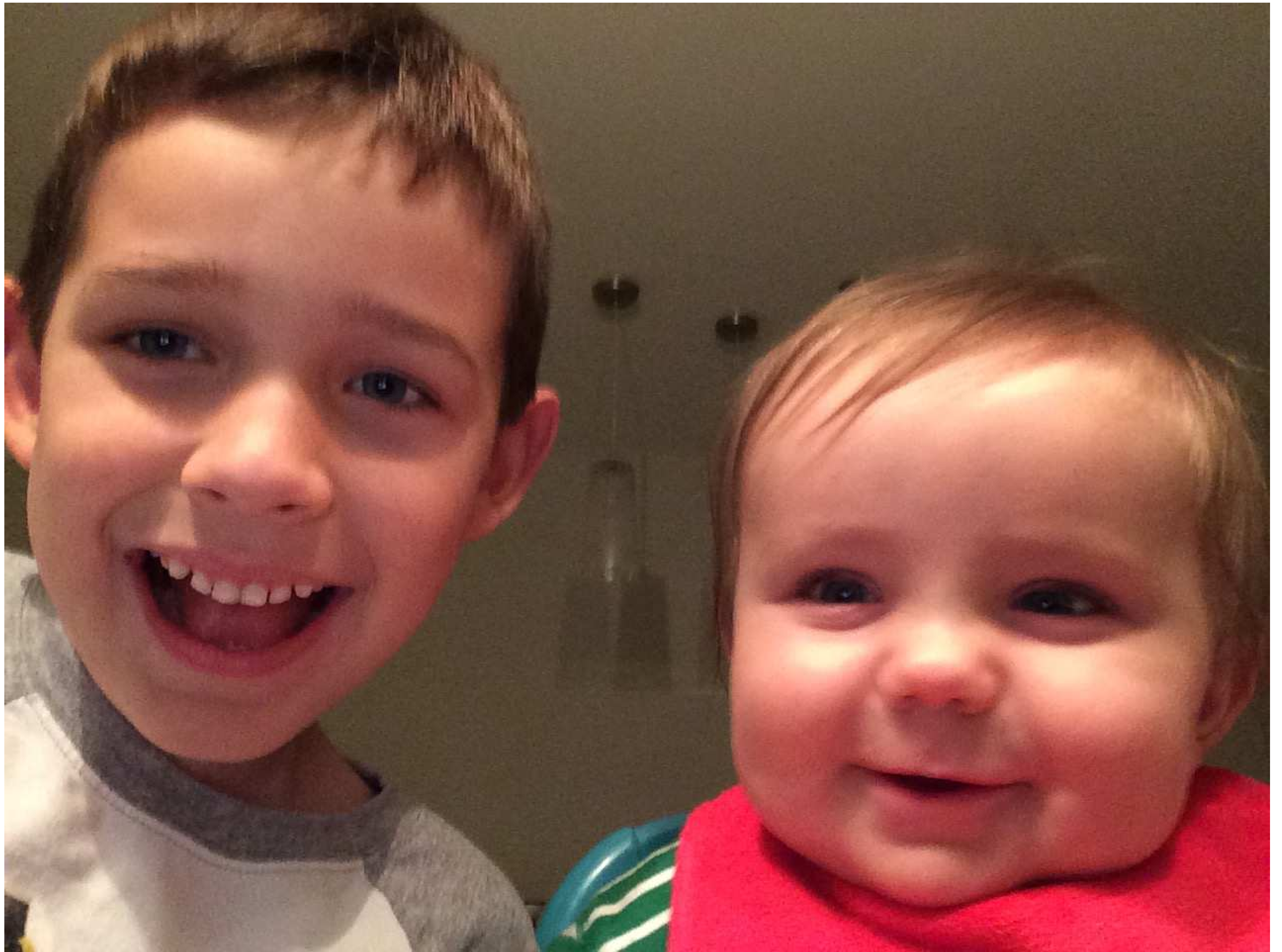
© 2016– Andrei Alexandrescu. Do not redistribute.

Welcome to DConf 2016!

© 2016– Andrei Alexandrescu. Do not redistribute.

Sup

© 2016– Andrei Alexandrescu. Do not redistribute.



First Five Minutes

- First five minutes become the next five years
- Out of the box experience
  - Website
  - Tutorials
  - Documentation
  - Libraries
- (Self-)Curating dub

© 2016– Andrei Alexandrescu. Do not redistribute.

Scaling Up

- Many thanks for a great year!
  - Thanks for being here!
  - The Foundation is up!
- Roles: build czar, sysadmin, webmaster, social media, conference organizer...
- Enhance the “point of contact” approach

- Please get me fired!
- I’m in charge of too many things

© 2016– Andrei Alexandrescu. Do not redistribute.

Raising the Bar on Contributions

- The weird thing about “okay work”
- Reasons to NOT pull something:
  - “Porque no?”
  - Respected contributor
  - “That’s a lot of work”
  - One-liners and many-liners
  - Renames
  - Refactorings showing no clear improvement
    - Churn, illusion of progress

© 2016– Andrei Alexandrescu. Do not redistribute.

Raising the Bar on Contributions

- Reasons to pull something:
  - Adds value
  - Refactors for a net benefit
  - Fixes a bug “positively”
  - Makes code simpler
  - Makes code faster
  - Automates
  - Improves documentation

- talent $\times$ time $=$ win

© 2016– Andrei Alexandrescu. Do not redistribute.

Resource Management

© 2016– Andrei Alexandrescu. Do not redistribute.

Take the Garbage Out

Commit to making D great *with and without* a GC

© 2016– Andrei Alexandrescu. Do not redistribute.

Issues with RC

- Overall: RC a qualified success story in computing

- Make it work with `immutable`
- Make it safe
- Make it work with `class`es and value types
- Make it work lazily (COW)
  - Copies should not have arbitrary cost

© 2016– Andrei Alexandrescu. Do not redistribute.

Basics

- The C++ way: library smart pointers
  + Apply to any type
  - Apply to any type (unsafe)
  - Needs `mutable`
- Envisioned:
  - In-language support + library hooks
  - Attribute `@rc`
  - `opIncRef(uint)`, `opDecRef(uint)`

© 2016– Andrei Alexandrescu. Do not redistribute.

Safety

- Main issue: unaccounted `ref`s

```
struct RC { int a; ... }
void fun(ref RC obj, ref int x) {
  obj = RC();
  ... use x ...
}
void gun() {
  RC obj = ...;
  fun(obj, obj.f);
}
```

- Variant: `obj` is global
- Attack: Insert extra incs/decs for `ref` params
- Fuse successive incs/decs where possible
- Elide increments where possible

© 2016– Andrei Alexandrescu. Do not redistribute.

`immutable`

- Apparent contradiction!
  - `immutable`: “I’ll never change, honey”
  - RC: surreptitious change in metadata
- We considered revoking `immutable` rights for RC objects
- If only we had a means to:
  - allocate metadata along with each object
  - type metadata independently
  - access metadata in $O(1)$...

© 2016– Andrei Alexandrescu. Do not redistribute.

"Why do you completely discard external reference counting approach (i.e. storing refcount in GC/allocator internal data structures bound to allocated memory blocks)?"

– Dicebot

© 2016– Andrei Alexandrescu. Do not redistribute.

“Never ascribe to malice that which is adequately explained by incompetence.”

– Robert J. Hanlon

© 2016– Andrei Alexandrescu. Do not redistribute.

`AffixAllocator!`... `AffixAllocator!`...

- Part of `std.experimental.allocator` from day one
- `AffixAllocator!(GCAllocator, uint)`:
  - fronts each allocation with an extra `uint`
  - ...that's independently typed
  - Accessible in $O(1)$!

- Use this allocator for creating RC objects

© 2016– Andrei Alexandrescu. Do not redistribute.

```
import bigo;
```

© 2016– Andrei Alexandrescu. Do not redistribute.

“Big O” Notation

- Compact characterization of algorithms
- Growing importance in recent years
- Usually confined to documentation
- Pen and paper suffices for non-generic code

- Better: make it generic and a discoverable part of the API

© 2016– Andrei Alexandrescu. Do not redistribute.

Scaling Naming Conventions

- Nomenclature approaches don't scale
- `removeLinTime`, `removeLogTime`, `removeConstTime`
- Hierarchy of speeds: faster functions subsume slower ones
- Sometimes $O(\cdot)$ depends on 2+ parameters
- Helpless with HOFs
- Doesn't scale to large APIs

- We want to automate this

© 2016– Andrei Alexandrescu. Do not redistribute.

Related Work

- Java: initially complexity-oblivious APIs
  - Later: `RandomAccess` “marker” interface
- STL carefully specifies complexity
  - Archetypal examples: `push_front`, `push_back`, `operator[]`
  - Syntax complexity follows algo complexity
  - Undecided on “best effort” vs. “present or not”, e.g. `distance`

© 2016– Andrei Alexandrescu. Do not redistribute.

Loosely Related Work

- $O(\cdot)$ analysis part of typechecking (Marion 2011)
- Automated Higher-Order Complexity Analysis (Benzinger 2004)
- Monotonic State (Pilkiewicz et al 2011)

© 2016– Andrei Alexandrescu. Do not redistribute.

Here

- User:
  - Introduces annotations
  - Defines composition
- Framework:
  - Provides algebra
  - Propagates attributes
  - Calculates composition
    - Notably for higher-order functions
  - Makes result available by static introspection

© 2016– Andrei Alexandrescu. Do not redistribute.

Synopsis

```
// Generic doubly-linked list of E
struct DoublyLinkedList(E) {
  ...
  // Complexity is O(1)
  void insertFront(E x) @O(1);
}

// Generic contiguous array of E
struct Array(E) {
  ...
  // Complexity is O(n) in the first argument (this)
  void insertFront(E x) @O("n");
}
```

© 2016– Andrei Alexandrescu. Do not redistribute.

Synopsis

```
// Complexity of insertFrontMany is the complexity of
// C.insertFront multiplied by the size of the second
// argument.
void insertFrontMany(C, E)(ref C container, E[] items)
@(complexity!(C.insertFront) * O("n2")) {
  foreach (item; items) {
    c.insertFront(item);
  }
}
```

© 2016– Andrei Alexandrescu. Do not redistribute.

Introspection

```
static assert(
  complexity!(insertFrontMany!MyC) <=
    O("n2") * log(O("n1")),
  "Too high complexity for insertFrontMany.");
```

© 2016– Andrei Alexandrescu. Do not redistribute.

Conventions

- Unannotated functions are considered `O(1)`
- `"nk"` for the kth parameter
- `this` is the first parameter
- `"n"` if only one parameter of interest

© 2016– Andrei Alexandrescu. Do not redistribute.

Algebra

- Credit: Timon Gehr

$$C \triangleq O\left(\sum_i \prod_j v_{ij}^{p_{ij}} \log^{l_{ij}} v_{ij}\right)$$

- p_{ij}, l_{ij} positive, $p_{ij} + l_{ij} > 0$
- $\log n$, n, $n \log n$, $\sqrt{n_1} + n_2 \log n_2$, $n_2 \log n_1$ ...

© 2016– Andrei Alexandrescu. Do not redistribute.

Normal Form & Partial Order

- Normal form for terms: most compact form
- $A \leq A'$ immediate (compare vars and powers)
- $T \leq T'$ iff for each atom A in T there's an atom A' in T' with $A \leq A'$
- $C \leq C'$ iff for each term T in C there's a term T' in C' with $T \leq T'$

- Normal form for complexities: no terms are ordered by $\leq$

© 2016– Andrei Alexandrescu. Do not redistribute.

Operations on Complexities

- Comparison for equality and $\leq$
- Addition (add, then normalize)
- Multiplication (multiply, then normalize)
- Normalization keeps only the fastest-growing terms: $O\left(n+\sqrt{m}+m\log n\right)+O\left(m^2+\log n\right)$ is $O\left(n+m^2+m\log n\right)$

- $\log$ just a bit trickier (can't express $\log(n_1+n_2)$)

© 2016– Andrei Alexandrescu. Do not redistribute.

$\log$ for Complexities

- Rely on a simple approximation

$$\log\left(\sum_i \prod_j v_{ij}^{p_{ij}} \log^{l_{ij}} v_{ij}\right) \triangleq \sum_i \sum_{j, p_{ij} > 0} \log v_{ij} \quad (1)$$

- $\log \log$ very slow growing, ignore
- $\log(a + b) \leq \log(ab)$

© 2016– Andrei Alexandrescu. Do not redistribute.

Implementation

- Operator overloading (==, <=, +, ∗)
- Pivotal use of compile-time evaluation
  - Perfect match with attribute expressions
- Run-time computation automatically available
- Sweet spot between convenience and complexity (sic)

- Coming soon!

© 2016– Andrei Alexandrescu. Do not redistribute.

One Last Thing

© 2016– Andrei Alexandrescu. Do not redistribute.