The Unified Form Language

and Key Points on its Translation

Martin Sandve Alnæs

Center for Biomedical Computing,
Simula Research Laboratory

February 27th
SIAM CSE 2017

Overview of this talk

- Key concepts of the Unified Form Language (UFL)
- Generic remarks on what a UFL form compiler needs to do
- Specific algorithms of the UFLACS form compiler
- Some benchmarks

Mixed formulation of a Poisson problem

Find $(\sigma, u) \in W = V \times U$ s.t.

$$\int_\Omega \sigma \cdot \tau + u \, \nabla \cdot \tau + \nabla \cdot \sigma \, v \, \mathrm{d}x = \int_\Omega f \, v \, \mathrm{d}x, \qquad \forall (\tau, v) \in W$$

```
from fenics import *
mesh = UnitSquareMesh(150, 150)
cell = mesh.ufl_cell()
```

```
V = FiniteElement("BDM", cell, 1)
U = FiniteElement("DG", cell, 0)
W = FunctionSpace(mesh, V * U)
```

```
sigma, u = TrialFunctions(W)
tau,   v = TestFunctions(W)
f = Expression("exp(pow(x[0]+x[1],2))",
               degree=1)
```

```
a = (dot(sigma,tau)*dx + u*div(tau)*dx
     + div(sigma)*v*dx)
L = f*v*dx
```

```
w = Function(W)
solve(a == L, w)
plot(w.sub(1))
```

Topics

Key concepts of the Unified Form Language (UFL)

Generic remarks on what a UFL form compiler needs to do

Specific algorithms of the UFLACS form compiler

Some benchmarks

The UFL model of a variational form

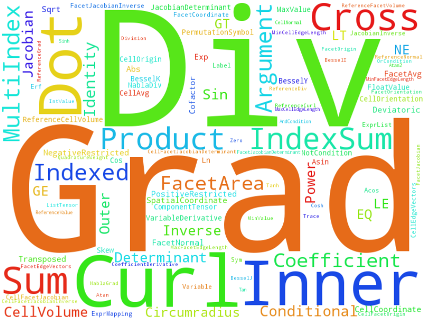
A `Form` is a sum of `Integral`s, where each integral is described by an integrand `Expr` and a `Measure` object.

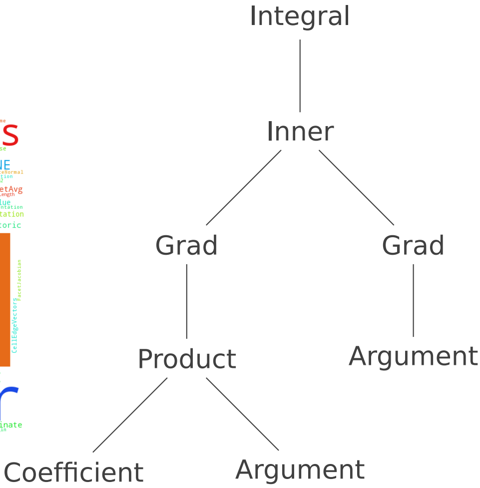
$$a(v, ...; w, ...) = \sum_k \int_{\Omega^k} f_k(v, ...; w, ...)\, d\mu_k$$

The geometric domain can be attached to the `Measure` or inferred from the integrand.

The expression language is the bulk of UFL

```
a = dot(grad(f*u),grad(v))*dx
```





The main categories of `Expr` types

- Terminal values (e.g. `SpatialCoordinate`, `Coefficient`)
- Computation (e.g. `Sum`, `Inner`, `IndexSum`)
- Derivatives (e.g. `Grad`, `Div`, `Curl`)
- Reshaping (e.g. `Transposed`, `Indexed`)

Every `Expr` node has tensor properties:

tensor shape, a tuple of free indices, and index dimensions

Assuming a 2 by 3 matrix expression A and **`Index`** objects `i`, `j`:

Math	UFL	Shape	Free indices	Index dimensions
A	A	(2, 3)	()	()
A_{00}	A[0,0]	()	()	()
A_{i0}	A[i,0]	()	(i,)	(2,)
A_{0i}	A[0,i]	()	(i,)	(3,)
A_{ij}	A[i,j]	()	(i,j)	(2,3)
A_{ji}	A[j,i]	()	(i,j)	(3,2)
$e_0 \cdot A$	A[0,:]	(3,)	()	()
$e_i \cdot A$	A[i,:]	(3,)	(i,)	(2,)

Example: tensor algebra and index notation

– equivalent expressions using tensor and index notation

$$u : x \mapsto R^d, \qquad v : x \mapsto R^d, \qquad M : x \mapsto R^{d,d}. \tag{1}$$

$$a_1(u, v; M) = \int_\Omega (\mathrm{grad}\, u \cdot M) : \mathrm{grad}\, v \,\mathrm{d}x, \tag{2}$$

$$a_2(u, v; M) = \int_\Omega (M^T \nabla u) : \nabla v \,\mathrm{d}x, \tag{3}$$

$$a_3(u, v; M) = \int_\Omega M_{ij} u_{k,i} v_{k,j} \,\mathrm{d}x \tag{4}$$

```
a1 = inner(dot(grad(u), M), grad(v))*dx
a2 = inner(M.T*nabla_grad(u), nabla_grad(v))*dx
a3 = M[i,j] * u[k].dx(i) * v[k].dx(j) * dx
```

Variational forms can be manipulated using e.g. partial evaluation or Gateaux differentiation

Consider the example bilinear form

```
a = dot(grad(f*u),grad(v))*dx
```

With this you can f.ex.

- Replace a coefficient function with another expression
  `replace(a, { f: g }) == dot(grad(g*u),grad(v))*dx`
- Construct the action of a bilinear form on a coefficient
  `action(a, g) == dot(grad(f*g),grad(v))*dx`
- Compute the derivative of a form or functional
  `derivative(a, u, du)`

Topics

Key concepts of the Unified Form Language (UFL)

Generic remarks on what a UFL form compiler needs to do

Specific algorithms of the UFLACS form compiler

Some benchmarks

UFL contains algorithms for form compiler preprocessing

Including but not limited to:

- Integrals are joined by subdomain ($\int_\Omega f + \int_{\Omega_0} g \rightarrow \int_{\Omega - \Omega_0} f + \int_{\Omega_0} f + g$)
- High level types are rewritten to index notation ($A : B \rightarrow A_{ij} B_{ij}$)
- Automatic differentiation is applied ($\nabla(cf + g) \rightarrow c\nabla f + \nabla g$)
- Restrictions are propagated to terminals ($(cv)^+ \rightarrow c^+ v^+$)
- Rewriting geometric quantities (next slide)

Symbolic geometric quantities can be rewritten in terms of the Jacobian

- Change of coordinates to reference cell integral:

$$\int f(x)\,dx \rightarrow \int F(X)|J|\,dX \tag{5}$$

- Application of symbolic Piola mappings:

$$v \rightarrow J^{-T}V, \qquad u \rightarrow \frac{1}{\det J}JU \tag{6}$$

- Lowering of abstractions of various cell geometry

$$n \rightarrow J^{-T}N, \qquad |f| \rightarrow \det\left(J\frac{dX}{dX^f}\right)|F| \tag{7}$$

Form compilers need to translate any *modified terminals* to the target framework

- A *modified terminal* a **`Terminal`** with a select set of operators optionally applied:
  - **`ReferenceValue`**
  - **`Grad`** or **`ReferenceGrad`** (any number)
  - **`CellAvg`** or **`FacetAvg`**
  - **`Restricted`** (obligatory where relevant)
  - **`Indexed`** with fixed indices

Examples: $v \in V_h$, ∇v, v^-, ∇v^+, $(\nabla v)^+_{01}$.

Topics

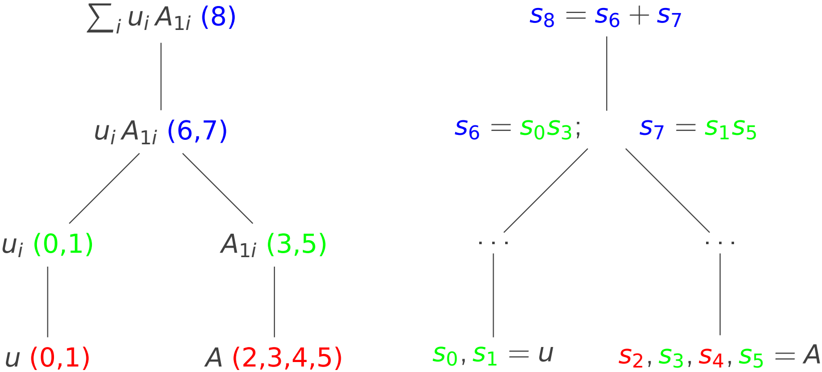
Key concepts of the Unified Form Language (UFL)

Generic remarks on what a UFL form compiler needs to do

Specific algorithms of the UFLACS form compiler

Some benchmarks

First pass: Scalar value numbering



A simpler scalar expression graph is created for

$$s_8 = u_0 A_{10} + u_1 A_{11} \tag{8}$$

Second pass: Form argument factorization

With a single pass over the new scalar graph, the integrand is factorized to a sum of monomials

$$a(u, v) = \int_T \sum_k f_k \, D_k^0 u \, D_k^1 v \, dx \qquad (9)$$

where f_k is an arbitrary scalar expression and $D_k^1 v$ is a component or derivative of the test function v.

Example: Considering the 1D form

$$a(u, v) = \int_T (\alpha u) v + (K u')(K v') \, dx, \qquad (10)$$

the factorized form is

$$a(u, v) = \int_T \alpha (u v) + (K K)(u' v') \, dx. \qquad (11)$$

Third pass: Classify monomial factors

Defining $\hat{u} = D^0 u$, $\hat{v} = D^1 v$, each integrated monomial is a matrix with structure $B_{ij} = \int_T f\,\hat{u}_i\,\hat{v}_j\,dx$.

- If f is cellwise constant, preintegration is possible:

$$P_{ij} = \int_T \hat{u}_i\,\hat{v}_j\,dx, \qquad B_{ij} = f\,P_{ij}. \tag{12}$$

- If both $\hat{u}$ *and* $\hat{v}$ are cellwise constant, can integrate f at runtime and then scale B:

$$F = \int_T f\,dx, \qquad B_{ij} = F\,\hat{u}_i\,\hat{v}_j. \tag{13}$$

- If $\hat{u}$ (*or* $\hat{v}$) is cellwise constant: the vector $f\hat{u}$ can be integrated runtime.

$$R_i = \int_T f\,\hat{u}_i\,dx, \qquad B_{ij} = f\,R_i\,\hat{v}_j. \tag{14}$$

Topics

Key concepts of the Unified Form Language (UFL)

Generic remarks on what a UFL form compiler needs to do

Specific algorithms of the UFLACS form compiler

Some benchmarks

Benchmarks: a couple of nonlinear problems

ns	uflacs	quadrature	quadrature -O
Hyperelasticity	268	8656	1520
Cahn Hillard	460	3753	3225

- “tensor”, “quadrature”, “uflacs”, and “tsfc” are representations or approaches to code generation in FFC.
- “tensor” representation in ffc does not handle the above equations.
- “tsfc” is not included in these benchmarks due to lack of time.
- Due to the same lack of time, please take these benchmarks with a grain of salt.

Benchmarks: some simpler problems

ns	uflacs	quadrature	quadrature -O	tensor
Mass q=1	34	33	38	29
Mass q=2	45	664	493	56
Mass q=3	113	8023	6252	137
Stiffness q=1	63	66	75	54
Stiffness q=2	280	984	2155	109
Stiffness q=3	1197	13036	35765	404
Stokes	1121	50189	7636	805
Helmholtz	76	158	230	56

Questions?

- `martinal@simula.no`

- `https://fenicsproject.org`
- `https://bitbucket.org/fenics-project`
- `https://fenics.readthedocs.io`
- `https://fenicsproject.org/tutorial`

Alnæs, Logg, Ølgaard, Rognes, Wells, *Unified Form Language: A domain-specific language for weak formulations of partial differential equations*, `http://arxiv.org/abs/1211.4047`