



HAL
open science

Combining FEniCS and MFront Code Generation Tools for Nonlinear Mechanics

Jeremy Bleier, Thomas Helfer

► **To cite this version:**

Jeremy Bleier, Thomas Helfer. Combining FEniCS and MFront Code Generation Tools for Nonlinear Mechanics. SIAM Conference on Computational Science and Engineering (CSE21), Mar 2021, Fort Worth, Texas, United States. hal-03168914

HAL Id: hal-03168914

<https://enpc.hal.science/hal-03168914>

Submitted on 14 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Binding two code-generation tools

FEniCS : a code-generation software package for automated solution of **partial differential equations**

$$\int_{\Omega} \nabla \mathbf{u} : \nabla \mathbf{v} \, dx - \int_{\Omega} p \operatorname{div} \mathbf{v} \, dx + \int_{\Omega} q \operatorname{div} \mathbf{u} \, dx = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} \, dx$$

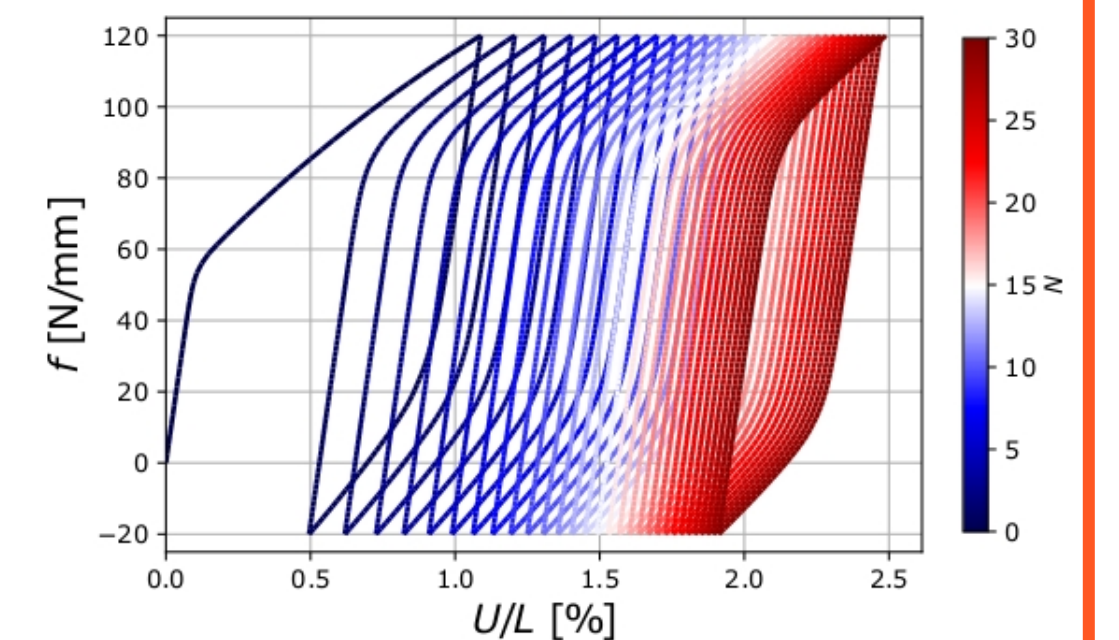
MFfront : a code-generation tool for complex **material constitutive laws**

$$\Delta \varepsilon, \sigma_n, \mathbf{Y}_n \rightarrow \boxed{\text{MFfront}} \rightarrow \sigma_{n+1}, \mathbf{Y}_{n+1}, \frac{\partial \sigma}{\partial \Delta \varepsilon}$$

Non-linear material constitutive laws

e.g. elasto-viscoplasticity, damage, ductile fracture usually formulated as a set of **non-linear equations** at the **material point level** => **implicit stress/strain relation, history-dependent**

Implementation should also return the **tangent operator**

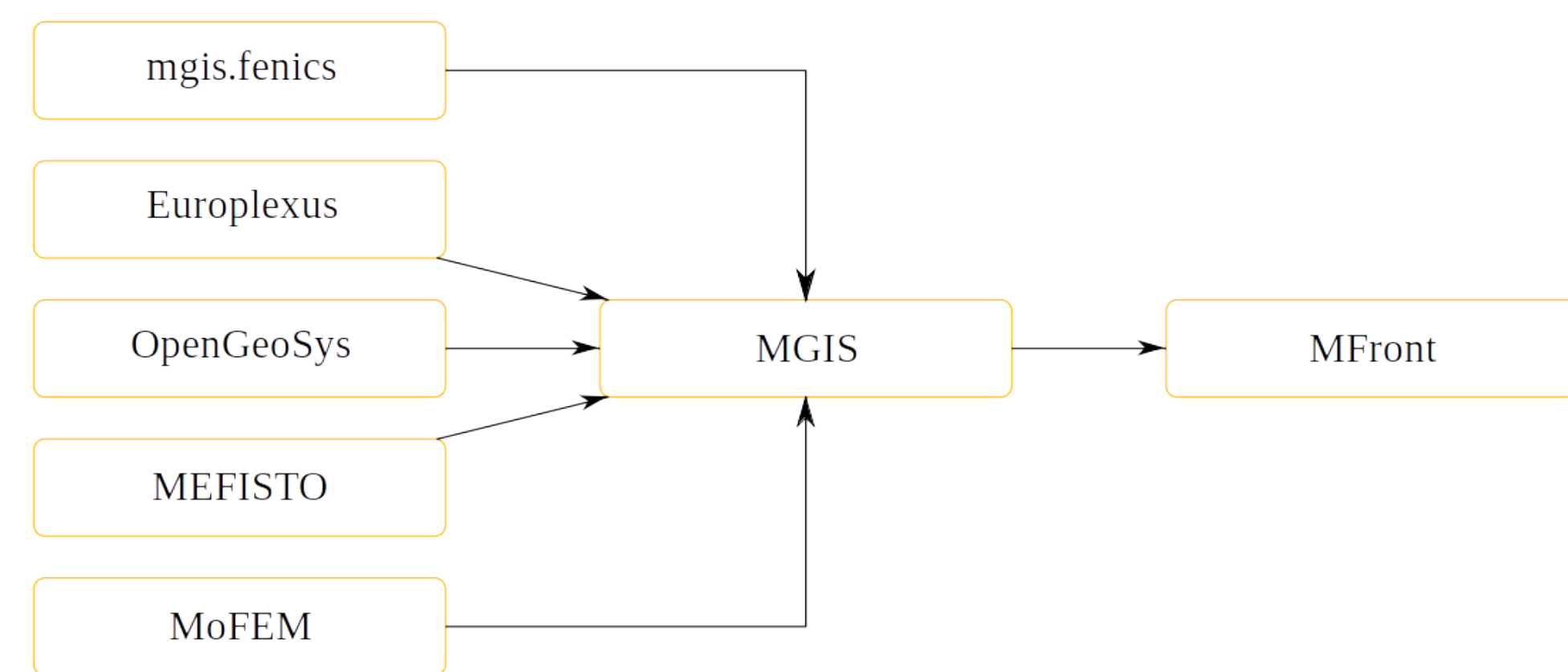


A Python package in the MGIS project

MFfront Generic Interface Support project

A generic interface between Mfront and (FE, FFT...) solvers

Bindings : C, Fortran, Python, Julia



within FEniCS

$$F(\mathbf{v}) = \int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) \cdot \delta \mathbf{g}(\mathbf{v}) \, dx - L(\mathbf{v}) = 0 \quad \forall \mathbf{v} \in V$$

$$a_{\text{tangent}}(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \delta \mathbf{g}(\mathbf{v}) \cdot \mathbb{T} \cdot \delta \mathbf{g}^j(\mathbf{u}) \, dx$$

Stress and **tangent blocks** represented by "Quadrature" functions => not optimal but not intrusive

The only metadata **not provided** by MGIS is how the gradients (e.g. strain) relate to the unknown fields (e.g. displacement)

The user is required to provide this link using UFL expressions (**registration**)

```
mat_prop = {"YoungModulus": E, "PoissonRatio": nu, "Hardening": H, "YieldStrength": sig0}
material = mf.MFrontNonlinearMaterial("src/libBehaviour.so",
                                       "IsotropicLinearHardeningPlasticity", hypothesis="plane_strain",
                                       material_properties=mat_prop)
u = Function(V, name="Displacement")
problem = mf.MFrontNonlinearProblem(u, material, quadrature_degree=2, bcs=bc)
problem.register_gradient("Strain", sym(grad(u)))
```

Monolithic thermo-elastoplasticity

Generalized behaviour

displacement + temperature

```
1 @DSL DefaultGenericBehaviour;
2 @Behaviour HeatTransferPhaseChange;
3
4 @Gradient TemperatureGradient VT;
5 Vt.setGlossaryName("TemperatureGradient");
6
7 @Flux HeatFlux j;
8 j.setGlossaryName("HeatFlux");
9
10 @StateVariable real h;
11 h.setEntryName("Enthalpy"); //per unit of volume
12
13 @AdditionalTangentOperatorBlock djdT;
14 @AdditionalTangentOperatorBlock dh/dT;
```

$$\varepsilon^{\text{el}} = \varepsilon^{\text{to}} - \varepsilon^{\text{p}} - \varepsilon^{\text{th}}$$

$$\boldsymbol{\sigma} = \lambda \operatorname{tr}(\varepsilon^{\text{el}}) \mathbf{I} + 2\mu \varepsilon^{\text{el}}$$

$$s = \frac{C_{\varepsilon}}{T_{\text{ref}}}(T - T_{\text{ref}}) + \frac{\kappa}{\rho} \operatorname{tr}(\varepsilon^{\text{el}})$$

$$\mathbf{j} = -k \nabla T$$

$$\int_{\Omega} \boldsymbol{\sigma}_{n+1} : \nabla^s \hat{\mathbf{u}} \, dx = \int_{\partial \Omega} \mathbf{T} \cdot \hat{\mathbf{u}} \, dS \quad \forall \hat{\mathbf{u}}$$

$$\int_{\Omega} (\rho T_{\text{ref}} \frac{s_{n+1} - s_n}{\Delta t} - \mathbf{j}_{n+1}) \hat{\mathbf{T}} \, dx = - \int_{\partial \Omega} q \hat{\mathbf{T}} \, dS \quad \forall \hat{\mathbf{T}}$$

```
Vue = VectorElement("CG", mesh.ufl_cell(), 2) # displacement finite element
Vte = FiniteElement("CG", mesh.ufl_cell(), 1) # temperature finite element
V = FunctionSpace(mesh, MixedElement([Vue, Vte]))

v = Function(V)
(u, Theta) = split(v)
problem = mf.MFrontNonlinearProblem(v, material, quadrature_degree=2, bcs=bcs)
problem.register_gradient("Strain", sym(grad(u)))
problem.register_gradient("TemperatureGradient", grad(Theta))
problem.register_external_state_variable("Temperature", Theta + Tref)
```

```
sigsig = problem.get_flux("Stress")
j = problem.get_flux("HeatFlux")
s = problem.get_state_variable("EntropyPerUnitOfMass")
prproblem.initialize()

s_s_old = s.copy(deepcopy=True)
v_v_ = TestFunction(V)
u_u_, T_ = split(v_v_) # Displacement and temperature test functions
eps_eps_ = problem.gradients["Strain"].variation(v_v_)
mech_resid = dot(sig, eps_) * problem.dx
th_thermal_resid = (rho * Tref * (s - s_old) / dt + T_ - dot(j, grad(T_))) * problem.dx
prproblem_residual = mech_resid + thermal_resid
prproblem.compute_tangent_form()
```

Logarithmic strain plasticity

Hencky strain measure + elasto-plastic behaviour

$$H = \frac{1}{2} \log(\mathbf{F}^T \cdot \mathbf{F})$$

$$H = H^e + H^p$$

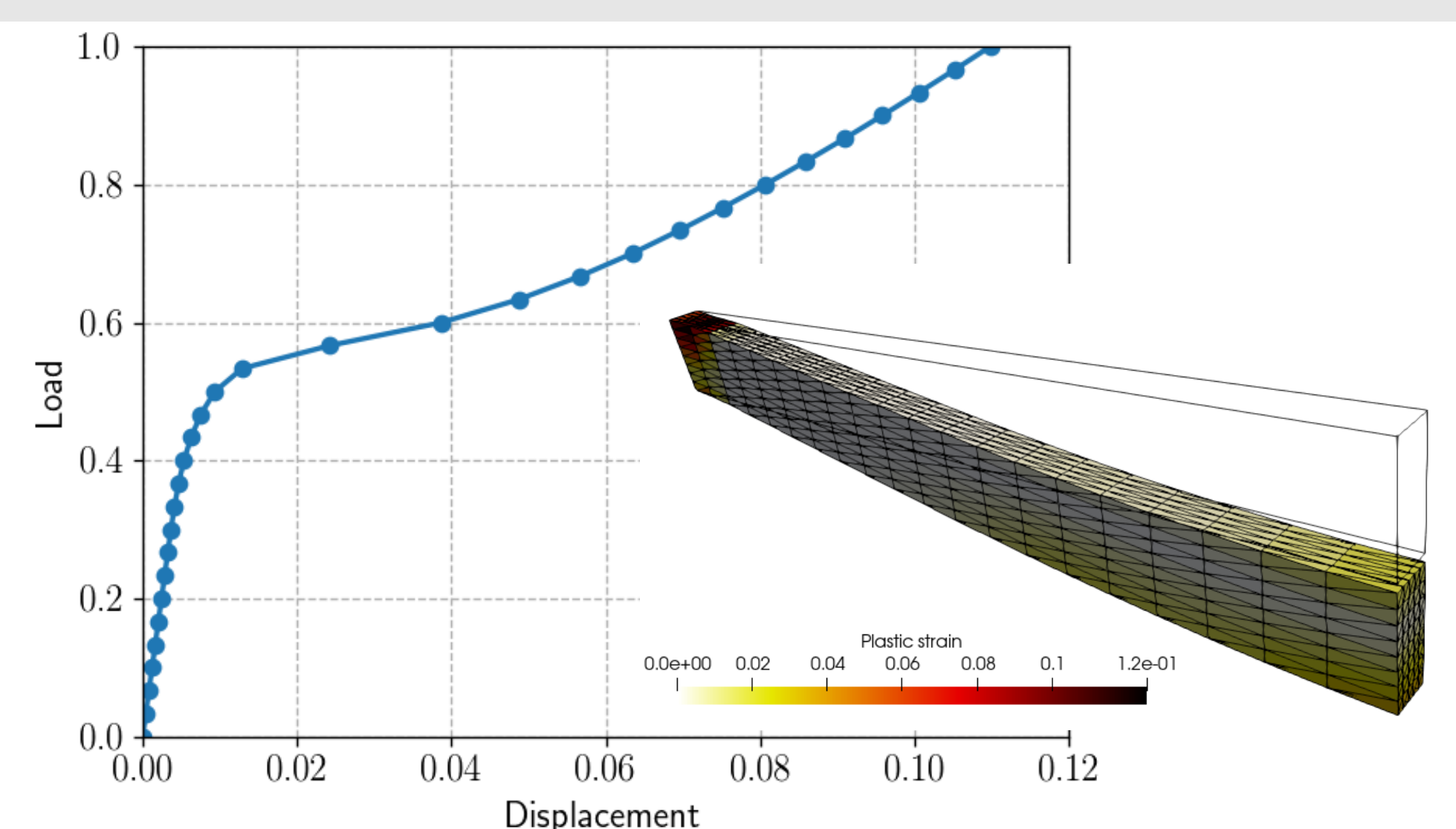
$$\mathbf{T} = \mathbb{C} : H^e$$

$$f(\mathbf{T}) = \|\mathbf{T}\| - (\sigma_0 + H^p) \leq 0$$

$$\dot{H}^p = \dot{\rho} \frac{\mathbf{T}}{\|\mathbf{T}\|}, \quad \dot{\rho} \geq 0$$

```
1 @DSL Implicit;
2
3 @Behaviour LogarithmicStrainPlasticity;
4
5 @StrainMeasure Hencky;
6
7 @Brick StandardElastoViscoPlasticity{
8   stress_potential: "Hooke" {
9     young_modulus: 210e9,
10    poisson_ratio: 0.3
11   },
12   inelastic_flow: "Plastic" {
13     criterion: "Mises",
14     isotropic_hardening: "Linear" {H: 500e6,
15                                   Ro: 250e6}
16   }
17 };
```

```
material = mf.MFrontNonlinearMaterial("./src/libBehaviour.so", "LogarithmicPlasticity")
problem = mf.MFrontNonlinearProblem(u, material, bcs=bc)
problem.set_loading(dot(selfweight, u)*dx)
problem.register_gradient("DeformationGradient", Identity(3)+grad(u))
```



Conclusion

Working binding between FEniCS and MFfront

Complex behaviours with very limited syntax, extension to generalized behaviours

Perspectives: better solution with dolfin-x, ExternalOperator ?