ThermophysicalModels library in OpenFOAM-2.3.x

*How to implement a new thermophysical model*

Teaching within: CFD with OpenSource software (TME050)

Isabelle Choquet
isabelle.choquet@hv.se
University West

2014-09-16
Exemples of thermodynamic and transport properties:

- Density $\rho$
- Heat capacity $C_v$, $C_p$
- Internal energy $e$, enthalpy $h$
- Diffusivity $D, \alpha$ ...
- Viscosity $\mu$
- Thermal conductivity $\kappa$
- Electric conductivity

Depend on:

- Temperature $T$
- Pressure $P$
- Fluid (possibly solid) composition

Involved in heat transfer, compressible flow, multiphase problems, combustion, etc.
Content

- Thermophysical models available in OpenFOAM .......................... 4-11
- Other examples of thermophysical models .............................. 12-14
- Problem needing a new thermophysical model ..................... 15-30
  - Example
  - Heat transfer solvers in OpenFOAM, thermoFOAM
- Implement a new transport property ($\kappa$) .......................... 31-42
  - Import, declare, define, link to solver, run a case
- Implement a new equation of state (for $\rho$)
  and new thermodynamic properties ($h$ and $C_p$) .............. 43-58
  - Import, declare, define, link to solver, run a case
### Models available in OpenFOAM (see UserGuide)

<table>
<thead>
<tr>
<th>Equation of State — equationOfState</th>
</tr>
</thead>
<tbody>
<tr>
<td>adiabaticPerfectFluid</td>
</tr>
<tr>
<td>icoPolynomial</td>
</tr>
<tr>
<td>perfectFluid</td>
</tr>
<tr>
<td>incompressiblePerfectGas</td>
</tr>
<tr>
<td>rhoConst</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adiabatic perfect gas equation of state</td>
</tr>
<tr>
<td>Incompressible polynomial equation of state, <em>e.g.</em> for liquids</td>
</tr>
<tr>
<td>Perfect gas equation of state</td>
</tr>
<tr>
<td>Incompressible gas equation of state using a constant reference pressure.</td>
</tr>
<tr>
<td>Density only varies with temperature and composition</td>
</tr>
<tr>
<td>Constant density equation of state</td>
</tr>
</tbody>
</table>
### Basic thermophysical properties — thermo

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eConstThermo</td>
<td>Constant specific heat ( c_p ) model with evaluation of internal energy ( e ) and entropy ( s )</td>
</tr>
<tr>
<td>hConstThermo</td>
<td>Constant specific heat ( c_p ) model with evaluation of enthalpy ( h ) and entropy ( s )</td>
</tr>
<tr>
<td>hPolynomialThermo</td>
<td>( c_p ) evaluated by a function with coefficients from polynomials, from which ( h, s ) are evaluated</td>
</tr>
<tr>
<td>janafThermo</td>
<td>( c_p ) evaluated by a function with coefficients from JANAF thermodynamic tables, from which ( h, s ) are evaluated</td>
</tr>
</tbody>
</table>
Models available in OpenFOAM (see UserGuide)

<table>
<thead>
<tr>
<th>Transport properties — transport</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>constTransport</td>
<td>Constant transport properties</td>
</tr>
<tr>
<td>polynomialTransport</td>
<td>Polynomial based temperature-dependent transport properties</td>
</tr>
<tr>
<td>sutherlandTransport</td>
<td>Sutherland’s formula for temperature-dependent transport properties</td>
</tr>
</tbody>
</table>
Mixture properties — mixture

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pureMixture</td>
<td>General thermophysical model calculation for passive gas mixtures</td>
</tr>
<tr>
<td>homogeneousMixture</td>
<td>Combustion mixture based on normalised fuel mass fraction $b$</td>
</tr>
<tr>
<td>inhomogeneousMixture</td>
<td>Combustion mixture based on $b$ and total fuel mass fraction $f_t$</td>
</tr>
<tr>
<td>veryInhomogeneousMixture</td>
<td>Combustion mixture based on $b$, $f_t$ and unburnt fuel mass fraction $f_u$</td>
</tr>
<tr>
<td>basicMultiComponentMixture</td>
<td>Basic mixture based on multiple components</td>
</tr>
<tr>
<td>multiComponentMixture</td>
<td>Derived mixture based on multiple components</td>
</tr>
<tr>
<td>reactingMixture</td>
<td>Combustion mixture using thermodynamics and reaction schemes</td>
</tr>
<tr>
<td>egrMixture</td>
<td>Exhaust gas recirculation mixture</td>
</tr>
<tr>
<td>singleStepReactingMixture</td>
<td>Single step reacting mixture</td>
</tr>
</tbody>
</table>

Models available in OpenFOAM (see UserGuide)
Models available in OpenFOAM (see UserGuide)

### III Thermophysical model — thermoModel

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hePsiThermo</td>
<td>General thermophysical model calculation based on compressibility $\psi$</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>General thermophysical model calculation based on density $\rho$</td>
</tr>
<tr>
<td>psiReactionThermo</td>
<td>Calculates enthalpy for combustion mixture based on $\psi$</td>
</tr>
<tr>
<td>psiuReactionThermo</td>
<td>Calculates enthalpy for combustion mixture based on $\psi_u$</td>
</tr>
<tr>
<td>rhoReactionThermo</td>
<td>Calculates enthalpy for combustion mixture based on $\rho$</td>
</tr>
<tr>
<td>heheupsipsiReactionThermo</td>
<td>Calculates enthalpy for unburnt gas and combustion mixture</td>
</tr>
</tbody>
</table>

Combines Ia, Ib, Ic & II
Models available in OpenFOAM (for a given solver)

**Example:** go to tutorials/heatTransfer/buoyantSimpleFoam/buoyantCavity/constant in thermophysicalProperties change ”transport const” to ”transport dummy” and run the solver buoyantSimpleFoam; it returns the list of thermophysical models available for this solver:

Valid rhoThermo types are:

<table>
<thead>
<tr>
<th>type</th>
<th>mixture</th>
<th>transport</th>
<th>thermophysical</th>
<th>equationOfState</th>
<th>specie</th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>const</td>
<td>hConst</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>const</td>
<td>hConst</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>inhomogeneousMixture</td>
<td>const</td>
<td>hConst</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>inhomogeneousMixture</td>
<td>const</td>
<td>hConst</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
</tbody>
</table>

... (+ 2 following slides)
### Models available in OpenFOAM (for a given solver)

<table>
<thead>
<tr>
<th>Name of Model</th>
<th>Model Type</th>
<th>Constituents</th>
<th>Energy Model</th>
<th>Species</th>
<th>Internal Energy Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>heRhoThermo inhomogeneousMixture</td>
<td>sutherland, janaF</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo inhomogeneousMixture</td>
<td>sutherland, janaF</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>const, hConst</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>const, hConst</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>const, hConst</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>polynomial, hPolynomial</td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>polynomial, hPolynomial</td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>sutherland, janaF</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>sutherland, janaF</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>sutherland, janaF</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo multiComponentMixture</td>
<td>sutherland, janaF</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>adiabaticPerfectFluid</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>adiabaticPerfectFluid</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>perfectFluid</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>perfectFluid</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>rhoConst</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>const, hConst</td>
<td>rhoConst</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>kineticAr, hKineticAr</td>
<td>rhoKineticAr</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>polynomial, hPolynomial</td>
<td>PengRobinsonGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>polynomial, hPolynomial</td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
<td></td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>polynomial, hPolynomial</td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensibleEnthalpy, sensibleInternalEnergy</td>
<td></td>
</tr>
</tbody>
</table>
Models available in OpenFOAM (for a given solver)

<table>
<thead>
<tr>
<th>model</th>
<th>type</th>
<th>equation</th>
<th>species</th>
<th>property</th>
</tr>
</thead>
<tbody>
<tr>
<td>heRhoThermo pureMixture</td>
<td>polynomial</td>
<td>janaf</td>
<td>PengRobinsonGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>sutherland</td>
<td>hConst</td>
<td>PengRobinsonGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>sutherland</td>
<td>hConst</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>sutherland</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>sutherland</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>sutherland</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo pureMixture</td>
<td>sutherland</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>const</td>
<td>hConst</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>const</td>
<td>hConst</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>const</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>const</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>polynomial</td>
<td>hPolynomial</td>
<td>icoPolynomial</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>polynomial</td>
<td>hPolynomial</td>
<td>icoPolynomial</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo reactingMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo singleStepReactingMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo singleStepReactingMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>perfectGas</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>heRhoThermo veryInhomogeneousMixture</td>
<td>const</td>
<td>hConst</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo veryInhomogeneousMixture</td>
<td>const</td>
<td>hConst</td>
<td>perfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo veryInhomogeneousMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>incompressiblePerfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>heRhoThermo veryInhomogeneousMixture</td>
<td>sutherland</td>
<td>janaf</td>
<td>perfectGas</td>
<td>sensibleEnthalpy</td>
</tr>
</tbody>
</table>
Other example of thermophysical model: *phase change in solid state*

Heat capacity of Ti/6Al/4V as a function of temperature; ——, o, recommended values; Δ, Bros [3]; ■, Richardson [4].

Other example of thermophysical model: *thermal plasma*

Other example of thermophysical model: *thermal plasma*

![Graph showing specific heat at constant pressure for 50% Ar-50% metal vapour mixtures](image)

**Figure 1.** Specific heat at constant pressure for 50% Ar–50% metal vapour mixtures by mass at atmospheric pressure.

Example of problem needing a new thermophysical model:
thermal conduction in a high temperature argon gas

Governing equation

\[ \frac{\partial (\rho C_v T)}{\partial t} = \nabla (\kappa \nabla T) \]

with \( \rho \), \( C_v \) and \( \kappa \) function of \( T \) obtained from kinetic theory for \( T \in [200; 20000]K \)

*Rk: need a heatTransfer solver*

---

Example of problem needing a new thermophysical model: thermal conduction in a high temperature argon gas.

This thermodynamic property $C_p(T)$ needs to be implemented.

This transport property $\kappa(T)$ needs to be implemented.

Figure 2.4: Thermal conductivity of an argon gas versus temperature.

Figure 2.5: Heat capacity at constant pressure of an argon gas versus temperature.

Example of problem needing a new thermophysical model:
thermal conduction in a high temperature argon gas

Why these peaks in $C_p$ and $\kappa$? Because the fluid is not a pure substance: its composition changes with T.

Temperature dependence of the equilibrium composition (species number densities) of an argon plasma at atmospheric pressure (starting from one mole of Ar at room temperature [21].)

Example of problem needing a new thermophysical model:
thermal conduction in a high temperature argon gas

From "Plasma arc welding simulation with openFOAM", M. Sass-Tisovskaya,
### Heat transfer and buoyancy-driven flows

<table>
<thead>
<tr>
<th>Solver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buoyantBoussinesqPimpleFoam</td>
<td>Transient solver for buoyant, turbulent flow of incompressible fluids</td>
</tr>
<tr>
<td>buoyantBoussinesqSimpleFoam</td>
<td>Steady-state solver for buoyant, turbulent flow of incompressible fluids</td>
</tr>
<tr>
<td>buoyantPimpleFoam</td>
<td>Transient solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer</td>
</tr>
<tr>
<td>buoyantSimpleFoam</td>
<td>Steady-state solver for buoyant, turbulent flow of compressible fluids</td>
</tr>
<tr>
<td>chtMultiRegionFoam</td>
<td>Combination of heatConductionFoam and buoyantFoam for conjugate heat transfer between a solid region and fluid region</td>
</tr>
<tr>
<td>chtMultiRegionSimpleFoam</td>
<td>Steady-state version of chtMultiRegionFoam</td>
</tr>
<tr>
<td>thermoFoam</td>
<td>Evolves the thermodynamics on a frozen flow field</td>
</tr>
</tbody>
</table>
OpenFOAM solver heatTransfer/thermoFoam:

- **Copy the solver in your user directory and rename:**
  
  ```shell
cd $WM_PROJECT_USER_DIR/applications/solvers/heatTransfer
  
  rk: if you do not yet have the heatTransfer directory under /solvers, create it (mkdir heatTransfer)
  
  cp -r $WM_PROJECT_DIR/applications/solvers/heatTransfer/thermoFoam .
  
  mv thermoFoam myThermoFoam
  
  cd myThermoFoam
  
  mv thermoFoam.C myThermoFoam.C
  ```

- **Modify Make/files to**
  
  ```shell
  myThermoFoam.C
  
  EXE = $(FOAM_USER_APPBIN)/myThermoFoam
  ```

- **Clean and compile**
  
  ```shell
  wclean
  
  wmake
  ```
OpenFOAM solver heatTransfer/thermoFoam:

Parts of thermoFoam involving a thermophysical model

```
createFields.H  
EEqn.H  
Make  
setAlphaEff.H  
thermoFoam.C
```

#include "rhoThermo.H"

#include "EEqn.H"

Include a thermophysic library

Include an energy conservation equation

Isabelle Choquet - 2014-09-16
OpenFOAM solver heatTransfer/thermoFoam:
Part of the energy conservation equation Eeqn.H

```c
volScalarField& he = thermo.he();
fvScalarMatrix Eeqn
{
    fvm::ddt(rho, he) + fvm::div(phi, he)
    + fvc::ddt(rho, K) + fvc::div(phi, K)
    + (he.name() == "S"
    ? fvc::div
    {
        fvc::absolute(phi/fvc::interpolate(rho), U),
        p,
        "div(phiv,p)"
    }
    : -dpdt
}
- fvm::laplacian(alphaEff, he)
==
    radiation->Sh(thermo)
    + fvOptions(rho, he)
};
```

"he" is either the specific internal energy or the specific enthalpy
(choice done when preparing a case, in the dictionary constant/thermophysicalProperties)
**Energy conservation** - total energy of flowing fluid: $\rho(\hat{h} + K)$ - in a control volume moving at $v_b$

Enthalpy: $\hat{h} = \hat{e} + \frac{p}{\rho}$

Kinetic energy: $K$

```plaintext
fvm::ddt(rho, he) + fvm::div(phi, he) + fvc::ddt(rho, K) + fvc::div(phi, K) + 
{
    he.name() == "e" ? fvc::div 
    {
        fvc::absolute(phi/fvc::interpolate(rho), U), 
        p, 
        "div(phiv,p)"
    }
    : -dpdt 
} - fvm::laplacian(alphaEff, he)
```
Energy conservation - total energy of flowing fluid: $\rho(h + K)$ - in a control volume moving at $v_b$

$$\frac{d}{dt} \int_V \rho \left( h + K \right) dV + \int_S \rho \left( h + K \right) (v - v_b) \cdot ds - \frac{d}{dt} \int_V pdV - \int_S p(v - v_b) \cdot ds = - \int_S q \cdot ds - \int_S (p v + \tau \cdot v) \cdot ds$$
Energy conservation - total energy of flowing fluid: $\rho(\hat{h} + K)$ - in a control volume moving at $\mathbf{v}_b$ (11/16)

\[
\frac{d}{dt} \int_V \rho (\hat{h} + K) \, dV + \int_S \rho (\hat{h} + K) (\mathbf{v} - \mathbf{v}_b) \cdot d\mathbf{s} = \frac{d}{dt} \int_V p \, dV - \int_S p (\mathbf{v} - \mathbf{v}_b) \cdot d\mathbf{s} - \int_S \mathbf{q} \cdot d\mathbf{s} - \int_S (\rho \mathbf{v} + \tau \cdot \mathbf{v}) \cdot d\mathbf{s}
\]

If “he” is the specific enthalpy “$\hat{h}$”

\[
fvm::ddt(rho, he) + fvm::div(phi, he)
+ fvc::ddt(rho, K) + fvc::div(phi, K)
+ (\text{If } he\text{ name() == "e" then finally fvc::div})
\]

This is not true (as he is not “e”)

so this part is not used

while this term is calculated

Isabelle Choquet - 2014-09-16 25
Energy conservation - total energy of flowing fluid: $\rho (\hat{h} + K)$ - in a control volume moving at $v_b$.

\[
\frac{d}{dt} \int_V \rho (\hat{h} + K) \, dV + \int_{\partial V} \rho (\hat{h} + K) (v - v_b) \cdot ds = \frac{d}{dt} \int_{\partial V} pdV - \int_{\partial V} p (v - v_b) \cdot ds = - \int_{\partial V} q \cdot ds - \int_{\partial V} (pv + \tau \cdot v) \cdot ds
\]

If "he" is the specific internal energy $\hat{\epsilon}$, then:

\[
\hat{h} = \hat{\epsilon} + \frac{p}{\rho}
\]

This is true

If $he$ name() == "e"?

\[+ fvc::ddt(rho, K) + fvc::div(phi, K)\]

This term is calculated

While this part is not used

Isabelle Choquet - 2014-09-16
Energy conservation - total energy of flowing fluid: $\rho (\hat{h} + K)$ - in a control volume moving at $v_b$ (13/16)

$$\frac{d}{dt} \int_V \rho (\hat{h} + K) \, dV + \int_S \rho (\hat{h} + K) (v - v_b) \cdot ds - \frac{d}{dt} \int_V \rho \, dV - \int_S p (v - v_b) \cdot ds = - \int_S q \cdot ds - \int_S (p v + \tau \cdot v) \cdot ds$$

$$\hat{h} = \hat{e} + \frac{p}{\rho}$$

```
fvm::ddt(rho, he) + fvm::div(phi, he) + fvc::ddt(rho, K) + fvc::div(phi, K)
+ (
    he.name() == "\text{e}"
? fvc::div
{
    fvc::absolute(phi/fvc::interpolate(rho), U),
    p,
    "div(phiv,p)"
}
: -dpdt
- fvm::laplacian(alphaEff, he)
```
\textbf{alphaEff} (the thermal diffusivity) is made of 2 contributions:

\[ \text{alphaEff} = \text{alpha laminar} + \text{alpha turbulent} \]

It is set in solver/heatTransfer/thermoFoam/setAlphaEff.H

\textit{via the turbulence library} : src/turbulenceModels

the ”turbulenceModels” library is linked to the thermo library src/thermophysicalModels

\textcolor{red}{so if the executables are renamed in a new thermophysical library and the library is compiled, they need to be renamed when they appear in the Make/options of the turbulenceModels library and the turbulenceModels library needs also to be compiled (in $WM\_PROJECT\_USER\_DIR$)}

\textbf{alpha turbulent} is defined in src/turbulenceModels

\textbf{alpha laminar} is defined in src/thermophysicalModels (as transport property)
What do we need to implement?

In EEqn.H the heat flux is written:

\[ q = - \alpha_{\text{Eff}} \nabla he \]

where the laminar part of \( \alpha_{\text{Eff}} \) is either:

\[ \kappa / C_p \] if "he" represents the specific enthalpy \( h \)  
(we will work with this case and from now assume \( he=h \))
or

\[ \kappa / C_v \] if "he" represents the specific internal energy \( e \)

So we need to implement the thermal conductivity \( \kappa = \kappa(T) \) - plotted slide 16 (right).

the density, and the specific heat capacity, respectively plotted slide 15 and 16, so that \( C_p = \rho(T) \cdot c_p(T) \).

But this is not sufficient since the conservative variable in EEqn.H is the specific enthalpy \( h \) while the thermodynamic and transport properties depend on another variable: the temperature \( T \).

So we also need to determine \( T \) from \( h \). This is done solving (with an iterative procedure already implemented in OpenFAOM) the equation of state

\[ \Delta h = \int_{T_{\text{ref}}}^{T} c_p(T) \, dT \]

It implies that we also need to implement the specific enthalpy \( h = h(T) \).
This implementation can be done in 2 parts

1. First, the implementation of the new transport property $\kappa$ (the thermal conductivity) for a temperature range from 200K to 20kK. It will be implemented, compiled and tested.

   Rk. The resultant thermophysical model will not be consistent from a physical point of view (as it will be associated with constant enthalpy, constant specific heat, ...). But it runs and has the advantage of allowing splitting the implementation work in smaller parts (easier to debug).

2. Implement the new thermodynamic properties $C_p$ and $h$ and the new equation of state $\rho$.
   These must be implemented together to be able to derive the temperature $T$ from $h(T)$ using $c_p(T)$.

   Rk. The resultant thermophysical model will then be consistent from a physical point of view. It can be observed that the consistent model runs faster than the non-consistent model implemented in the 1st part.
Implement a new transport property K: main steps

Step 0: import and rename suited parts of the library "thermophysicalModels"

Step 1: declare the new transport property model (see Ic)*

Step 2: define the new transport property model (see Ic)*

Step 3: declare the new thermophysical model (see III)**

Step 4: link the new thermophysical model to the solver

Step 5: call the new thermophysical model in a test case

*Slide 6 **Slide 7
Implement a new transport property K: step 0

Prepare your library myThermophysicalModels/mySpecie

- Copy the folder “specie” of the library in your user directory and rename it:
  cd $WM_PROJECT_USER_DIR/src/myThermophysicalModels
  rk: if you do not yet have the directory myThermophysicalModels under /src, create it with mkdir
  cp -r $WM_PROJECT_DIR/src/thermophysicalModels/specie .
  mv specie mySpecie
  cd mySpecie

- Modify the Make/files to:
  mySpecie.C
  LIB = $(FOAM_USER_LIBBIN)/libspecie

- Clean & compile
  wclean
  wmake libso

Now the executable is in your working space, and your ”libspecie” will be accessed in priority (instead of the OpenFOAM executable in $FOAM_LIBBIN, even if the name is the same).

Then this name of executable can be kept unchanged. Doing so, no need to import the turbulence library, no need to rename its links (in Make/options) to your thermo library, and no need to recompile the turbulence library (See slide 28): the $FOAM_LIBBIN turbulence library will link to your own thermo library.
Implement a new transport property \( K \): step 0

Prepare your library myThermophysicalModels/myBasic

- Copy the folder “basic” of the library in your user directory and rename it:

```plaintext
cd $WM_PROJECT_USER_DIR/src/myThermophysicalModels
cp -r $WM_PROJECT_DIR/src/thermophysicalModels/basic .
mv basic myBasic
cd myBasic
```

- Modify Make/files to:

```plaintext
myBasic.C
LIB = $(FOAM_USER_LIBBIN)/libfluidThermophysicalModels
```

Similar to the previous slide
Implement a new transport property K: step 0

- Modify the Make/options file to

```plaintext
EXE_INC = \\
-$(LIB_SRC)/finiteVolume/lnInclude \\
-$(WM_PROJECT_USER_DIR)/src/myThermophysicalModels/mySpecie/lnInclude \\
-$(LIB_SRC)/meshTools/lnInclude

LIB_LIBS = \\
-L$(FOAM_USER_LIBBIN) \\
-lfiniteVolume \\
-lSpecie
```

(since myBasic needs to be linked to mySpecie at compilation)

- Clean the dependencies and compile

  `wclean`
  `wmake libso`

(1) Gives the path to access your own files located in mySpecie

(2) Indicates that the compilator must 1st look in your own working space (in $FOAM_USER_LIBBIN) to pick the libraries listed below. If not found there (ex. may be lfiniteVolume is not in your space $FOAM_USER_LIBBIN) the compilator will next look in the OpenFOAM space (in $FOAM_LIBBIN). If still not found it will complain.

(3) Can check the path used by reading the messages written on the screen during compilation (see next slide)
Here it can be checked that the library “finiteVolume” is picked in
/home/…/OpenFOAM2.3.x/src
(so in $WM_PROJECT_DIR/src) since it does not exist in
$WM_PROJECT_USER_DIR/src

Same for meshTools

There it can be checked that “myThermophysicalModels/my Specie” is picked in
/home/…/isabelle-2.3.x/src
(so in $WM_PROJECT_USER_DIR/src) and not in
$WM_PROJECT_DIR

How to check that the compiler links the desired libraries?
Implement a new transport property K: step 1

Declare the new transport model in user src/myThermophysicalModels/mySpecie

- In mySpecie/include/thermoPhysicsTypes.H add the following lines

```cpp
#include "kineticArTransport.H"

typedef
kineticArTransport
<
    species::thermo
    <
        hConstThermo
        <
            perfectGas<specie>
        >,
        sensibleEnthalpy
    >
> kineticArGasHThermoPhysics;
```

To access the new transport model

Name given to the new transport model

Name given to the new thermophysical model
Implement a new transport property K: step 2

Define the new transport model in user src/myThermophysicalModels/mySpecie

- Copy and rename an existing model:
  - cd transport
  - cp –r const kineticAr
  - cd kineticAr
  - mv constTransport.C kineticArTransport.C
  - mv constTransport.H kineticArTransport.H
  - mv constTransportI.H kineticArTransportI.H

  Open the files one by one and replace “constTransport” (NOT just “const” !) with “kineticArTransport”

  Update the “instantiated type name” in kineticArTransport.H

  So look for “instantiated” and below (only there!) replace “return “const”” with “return “kineticAr””

- Clean the dependencies (“wclean”) and compile mySpecie (“wmake libso”)
Implement a new transport property $K$: step 2

- **Open** kinticArTransportI.H

```cpp
// Thermal conductivity changed from constant to tabulated data table
template<class Thermo>
inline Foam::scalar Foam::kineticArTransport<Thermo>::kappa
{
    const scalar p,
    const scalar T
} const
{
    // original version:
    //return this->Cpv(p, T)*mu(p, T)*rPr_;  

    // new version for argon plasma:
    // Thermal conductivity kappa [W/(m.K)] function of T, for Argon plasma,
    // tabulated for T from T0=200K to 20000K
    // with tabulation interval of dT=100K
    int i_index;
    scalar dT=100;
    ...
    return kappa_T_Argon;

    // end of kappa version implemented for argon plasma
```

Comment the original model

thermal conductivity

insert the new model provided in the file Ar_Data_ThermalConduct
Implement a new transport property K: step 2

- Clean the dependencies ("wclean") and compile mySpecie ("wmake libso")

```cpp
// Thermal diffusivity for enthalpy [kg/ms]
//
template<class Thermo>
inline Foam::scalar Foam::kineticArTransport<Thermo>::alphah
{
    const scalar p,
    const scalar T
}
const
{
    // original version (with Pr constant):
    //return mu(p, T)*rPr_;

    // new version for argon plasma (since Pr is not constant):
    // Pr = mu(p,T)*Cp(p,T)/kappa(p,T)
    // mu(p,T)/Pr = kappa(p,T)/Cp(p,T)
    return kappa(p,T)/this->Cpv(p,T);

    // end of alpha version implemented for argon plasma
}
```

Comment the original model

Write the new model
Declare the new thermophysical model in user src/myThermophysicalModels/myBasic

- In myBasic/rhoThermo/rhoThermos.C add the following lines

```c
#include "kineticArTransport.H"

makeThermo
(
    rhoThermo,  // New combination of Ia, Ib, Ic, II and III (see slides 3 to 7) defining a new thermophysical model
    heRhoThermo,
    pureMixture,
    kineticArTransport,
    sensibleEnthalpy,
    hConstThermo,
    perfectGas,
    specie
);
```

- Clean the dependencies (“wclean”) and compile myBasic (“wmake libso”)
Implement a new transport property $K$: step 4

Link the new thermophysicalModel library to the solver

- In `myThermoFoam/Make/options` do the following changes to access the new library

  ```bash
  -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
  -I$(WM_PROJECT_USER_DIR)/src/myThermophysicalModels/myBasic/lnInclude \
  EXE_LIBS = \n    -lfiniteVolume \
  EXE_LIBS = \n    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
  ```

- Clean the dependencies (wclean) and compile `myThermoFoam.C` (wmake)
Implement a new transport property K: step 5

Call the new thermophysicalModel in a test case

- Use the test case provided: blockThermoFoamCase.tgz
- Run this case with the solver thermoFoam (the original one)
- Copy blockThermoFoam to blockNewThermoFoam and clean
- Update constant/thermophysicalProperties to

```plaintext
thermoType
{
    type heRhoThermo;
    mixture pureMixture;
    transport kineticAr; // new model
    //transport const;
    thermo hConst;
    equationOfState perfectGas;
    specie specie;
    energy sensibleEnthalpy;
}
```

- Run this case with the solver myThermoFoam linked to the new thermophysical library
- Compare the results (for instance doing a plot along a suited line you can choose)
This implementation can be done in 2 parts

1. First, the implementation of the new transport property $\kappa$ (the thermal conductivity) for a temperature range from 200K to 20kK. It will be implemented, compiled and tested.

   Rk. The resultant thermophysical model will not be consistent from a physical point of view (as it will be associated with constant enthalpy, constant specific heat, …). But it runs and has the advantage of allowing splitting the implementation work in smaller parts (easier to debug).

2. Implement the new thermodynamic properties $C_p$ and $h$ and the new equation of state $\rho$.
   These must be implemented together since the aim is to be able to derive the temperature $T$ from $h(T)$ and to calculate the heat capacity $C_p = \rho(T) \cdot c_p(T)$

   Rk. The resultant thermophysical model will then be consistent from a physical point of view.
   It can also be observed that the consistent model runs faster than the non-consistent model implemented in the 1st step

Part 1 is done. We now start this 2nd part
Implement new thermodynamic properties and a new equation of state

**Step 1**: declare (see Ia, b) the new thermodynamic properties and the new equation of state

**Step 2**: define (see Ia) the new equation of state

**Step 3**: define (see Ib) the new thermodynamic properties

**Step 4**: declare (see III) the new thermophysical model

**Step 5**: link the new thermophysical model to the solver

**Step 6**: call the new thermophysical model in a test case and run

#See slide 4  *See slide 5  **See slide 7
Implement new thermodynamic properties and equation of state: step 1

Declare the new models in user src/myThermophysicalModels/mySpecie

- In mySpecie/include/thermoPhysicsTypes.H

  add

  ```cpp
  #include "hKineticArThermo.H"
  #include "rhoKineticAr.H"
  ```

  and modify

  ```cpp
typedef kineticArTransport <
    species::thermo <
      hConstThermo <
        perfectGas<specie>,
      >, sensibleEnthalpy
    >
  > kineticArGasHThermoPhysics;
  ```

  to

  ```cpp
typedef kineticArTransport <
    species::thermo <
      hKineticArThermo <
        >, sensibleEnthalpy
    >
  > kineticArGasHThermoPhysics;
  ```

  To access the new thermodynamic model & the new equation of state

  Name given to the new thermodynamic model & equation of state
Implement new thermodynamic properties and equation of state: step 2

Define a new equation of state in in user src/myThermophysicalModels/mySpecie

- Copy and rename an existing model:
  cd equationOfState
  cp –r perfectGas rhoKineticAr
  cd rhoKineticAr
  mv perfectGas.C rhoKineticAr.C
  mv perfectGas.H rhoKineticAr.H
  mv perfectGasI.H rhoKineticArI.H
  open the files one by one and replace 
  "perfectGas" with "rhoKineticAr"

  update the "instantiated type name" in rhoConstAr.H
  so look for "instantiated" and below replace "return "perfectGas<" " with " return "rhoKineticAr<" "

Prepare the structure
Implement new thermodynamic properties and equation of state: step 2

- Open `rhoKineticArI.H` and do the following modifications:

```c++
template<class Specie>
inline Foam::scalar Foam::rhoKineticAr<Specie>::rho(scalar p, scalar T) const
{
    //old model
    //return p/(this->R()*T);

    int i_index;
    scalar dT=100;
    scalar T0=200;
    scalar Temp_Argon;
    scalar rho_T_Argon;
    ...
    return rho_T_Argon;

    //new model
    // Density [kg/m^3] function of T, for Argon plasma,
    // tabulated for T from T0=200K to 20000K
    // with tabulation interval of dT=100K

density
    }
```

Comment the original model

Insert the new model provided in the file `density_Ar_Data`
Implement new thermodynamic properties and equation of state: step 2

- Modify also

```cpp
template<class Specie>
inline Foam::scalar Foam::rhoKineticAr<Specie>::psi(scalar, scalar T) const
{
    // old model
    // return 1.0/(this->R()*T);

    // new model
    Info << " psi should not be used with the rhoKineticAr model " << endl;
    return 0.0;
}
```

Comment the original model (ideal gas)

Write the new model

*Rk*: psi is set to zero since the plasma model implemented here is mechanically incompressible, and thermally expansible: \( \rho(P, T) = \rho(T) \).
Implement new thermodynamic properties and equation of state: step 2

- Modify also

```cpp
template<class Specie>
inline Foam::scalar Foam::rhoKineticAr<Specie>::Z(scalar, scalar) const
{
    // old model
    //return 1.0;

    // new model
    Info << " Z should not be used with the rhoKineticAr model " << endl;
    return 0.0;
}
```

- Comment the original model (ideal gas)
- Compressibility factor
- Write the new model
Implement new thermodynamic properties and equation of state: step 2

- Modify also

```cpp
template<class Specie>
inline Foam::scalar Foam::rhoKineticAr<Specie>::cpMcv(scalar, scalar) const
{
    // old model
    //return this->RR;

    //new model
    Info << " cpMcv should not be used with the rhoKineticAr model " << endl;
    return 0.0;
}
```

- Clean and compile mySpecie
  - wclean
  - wmake libso
Imaginary new thermodynamic properties and equation of state: step 3

Define the new properties \( h, C_p \) in user src/myThermophysicalModels/mySpecie

- Copy and rename an existing model:
  
  ```
  cd thermo
  cp –r hConst hKineticAr
  cd hKineticAr
  mv hConstThermo.C hKineticArThermo.C
  mv hConstThermo.H hKineticArThermo.H
  mv hConstThermoI.H hKineticArThermoI.H
  ```

  Open the files one by one and replace
  
  “hConstThermo” with “hKineticArThermo”

  Update the “instantiated type name” in hKineticArThermo.H
  
  ```
  so look for “instantiaded” and below replace “return “hConst<” “
  with “ return “hKineticAr<” “
  ```
Open hKineticArI.H and do the following modifications:

```cpp
template<class equationOfState>
inline Foam::scalar Foam::hKineticArThermo<equationOfState>::cp
(
    const scalar p,
    const scalar T
) const
{
    // original model
    //return Cp_;

    // New model:
    // heat capacity at constant pressure [J/(kmol.K)] function of T,
    // for Argon plasma,
    // tabulated for T from T0=200K to 20000K
    // with tabulation interval of dT=100K
    int i_index;
    scalar dT=100;
    ...
    return Cp_T_Argon*this->W();

    // end of cp version implemented for argon plasma

    Heat capacity at constant pressure
    Comment the original model
    insert the new model provided in the file heatCapacity_Cp_Data
```
Implement new thermodynamic properties and equation of state: step 3

- Modify also

```cpp
template<class equationOfState>
inline Foam::scalar Foam::hKineticArThermo<equationOfState>::ha
{
    const scalar p, const scalar T
}
{
    // original model
    //return Cp_*T + Hf;

    // enthalpy [J/kg] function of T, for Argon plasma,
    // tabulated for T from T0=200K to 20000K
    // with tabulation interval of dT=100K
    int i_index;
    scalar dT=100;
    ...
    return h_T_Argon*this->W();
    // end of h version implemented for argon plasma
}
```

absolute enthalpy
Hf is the enthalpy of formation

Comment the original model

insert the new model provided in the file enthalpy_Data.
Rk. The reference temperature was set so that Hf is zero.
Implement new thermodynamic properties and equation of state: step 3

- Modify also

```cpp
template<class equationOfState>
inline Foam::scalar Foam::hKineticArThermo<equationOfState>::hs
(
    const scalar p, const scalar T
)
const
{
    // original model
    //return Cp_*T;
    return ha(p,T)-hc();
}

sensible enthalpy

Comment the original constant model

Write the new temperature dependent model.
Rk. As the non constant thermodynamic models in openFoam depend on both pressure p and temperature, we write ha(p,T) although p is not used.
Implement new thermodynamic properties and equation of state: step 3

- **Modify also**

```cpp
template<class equationOfState>
inline Foam::scalar Foam::hKineticArThermo<equationOfState>::hc() const
{
    // original model
    //return Hf;
    return 0.;
}
```

- **Clean (wclean) and compile mySpecie (wmake libso)**

  - Comment the original model
  - Write the new model

  Rk. Here the plasma is considered as one-fluid. The ionization reactions were accounted for when tabulating the absolute enthalpy $h_a$. For a one-fluid model $h_c$ is the enthalpy of formation. Here it is set to zero (see the remark on slide 53).
Implement new thermodynamic properties and equation of state: step 4

Declare the new thermophysical model in user src/myThermophysicalModels/myBasic

- In myBasic/rhoThermo/rhoThermos.C add the following lines

```c
#include "hKineticArThermo.H"
#include "rhoKineticAr.H"
```

...and change...

```c
makeThermo
(
    rhoThermo,
    heRhoThermo,
    pureMixture,
    kineticArTransport,
    sensibleEnthalpy,
    hConstThermo,
    perfectGas,
    specie
);
```

...to:

```c
makeThermo
(
    rhoThermo,
    heRhoThermo,
    pureMixture,
    kineticArTransport,
    sensibleEnthalpy,
    hKineticArThermo,
    rhoKineticAr,
    specie
);
```

- Clean (wclean) and compile myBasic (wmake libso)
Implement new thermodynamic properties and equation of state: step 5

Link the new thermophysical model to the solver

- The update of *myThermoFoam/Make/options*
  needed to access the new thermophysical library
  has already been done when implementing the thermal conductivity $\kappa$ (see slide 41)

So nothing else needs to be done here (unless you decided to rename *myBasic* in part 2;
in that case see slide 41).
Implement new thermodynamic properties and equation of state: step 6

*Call the new thermophysical model in a test case and run*

- Copy `blockThermoFoam` to `blockKineticArThermoFoam` and clean (`wclean`)

- Update `constant/thermophysicalProperties` to

```
thermoType
{
    type heRhoThermo;
    mixture pureMixture;
    transport kineticAr; // new model
    //transport const;
    thermo hKineticAr; // new model
    //thermo hConst;
    equationOfState rhoKineticAr; // new model
    //equationOfState perfectGas;
    specie specie;
    energy sensibleEnthalpy;
}
```

- Run this case with the solver `MyThermoFoam` (now linked to your new thermophysical library)

- Compare the results with the previous results (doing a plot along a suited line you can choose)