ThermophysicalModels library in OpenFOAM-2.3.x (or 2.4.x)

*How to implement a new thermophysical model*

Teaching within: CFD with OpenSource software (TME050)

Isabelle Choquet
isabelle.choquet@hv.se
University West

2015-09-28
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
### Equation of State — equationOfState

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adiabaticPerfectFluid</td>
<td>Adiabatic perfect gas equation of state</td>
</tr>
<tr>
<td>icoPolynomial</td>
<td>Incompressible polynomial equation of state, e.g. for liquids</td>
</tr>
<tr>
<td>perfectFluid</td>
<td>Perfect gas equation of state</td>
</tr>
<tr>
<td>incompressiblePerfectGas</td>
<td>Incompressible gas equation of state using a constant reference pressure. Density only varies with temperature and composition</td>
</tr>
<tr>
<td>rhoConst</td>
<td>Constant density equation of state</td>
</tr>
<tr>
<td>Ib</td>
<td>Basic thermophysical properties — thermo</td>
</tr>
<tr>
<td>----</td>
<td>-----------------------------------------</td>
</tr>
<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</th>
<th>transport</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>Name</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>basicMultiComponent-Mixture</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)

II

Ia, Ib & Ic for all species & suited mixing rules

(4/8)
Combines Ia, Ib, Ic & II Models available in OpenFOAM (see UserGuide)

<table>
<thead>
<tr>
<th>Thermophysical model — thermoModel</th>
<th>Combines Ia, Ib, Ic &amp; II</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>heheupsiReactionThermo</td>
<td>Calculates enthalpy for unburnt gas and combustion mixture</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Valid rhoThermo types are:</th>
<th>Ic</th>
<th>Ib</th>
<th>Ia</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>III</td>
<td>mixture</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>transport</td>
<td>therm</td>
<td>equationOfState</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>const</td>
<td>hConst</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>const</td>
<td>hConst</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>sutherland</td>
<td>janaf</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>homogeneousMixture</td>
<td>sutherland</td>
<td>janaf</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>inhomogeneousMixture</td>
<td>const</td>
<td>hConst</td>
</tr>
<tr>
<td>heRhoThermo</td>
<td>inhomogeneousMixture</td>
<td>const</td>
<td>hConst</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Models</th>
<th>Mixture Type</th>
<th>Solver</th>
<th>Equation Type</th>
<th>Specie</th>
<th>Internal Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhoThermo inhomogeneousMixture</td>
<td>sutherland janafe</td>
<td></td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>rhoThermo inhomogeneousMixture</td>
<td>sutherland janafe</td>
<td></td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleEnthalpy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>const hConst</td>
<td></td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>const hConst</td>
<td></td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>const hConst</td>
<td></td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>polynomial hPolynomial</td>
<td></td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>polynomial hPolynomial</td>
<td></td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>sutherland janafe</td>
<td></td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>sutherland janafe</td>
<td></td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo multiComponentMixture</td>
<td>sutherland janafe</td>
<td></td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>adiabaticPerfectFluid</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>adiabaticPerfectFluid</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>incompressiblePerfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>perfectFluid</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>perfectFluid</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>perfectGas</td>
<td>specie</td>
<td>sensibleInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>rhoConst</td>
<td>specie</td>
<td>sensitiveInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>const hConst</td>
<td></td>
<td>rhoConst</td>
<td>specie</td>
<td>sensitiveInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>kineticAr hKineticAr</td>
<td></td>
<td>rhoKineticAr</td>
<td>specie</td>
<td>sensitiveInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>polynomial hPolynomial</td>
<td></td>
<td>PengRobinsonGas</td>
<td>specie</td>
<td>sensitiveEnthalpy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>polynomial hPolynomial</td>
<td></td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensitiveInternalEnergy</td>
</tr>
<tr>
<td>rhoThermo pureMixture</td>
<td>polynomial hPolynomial</td>
<td></td>
<td>icoPolynomial</td>
<td>specie</td>
<td>sensitiveInternalEnergy</td>
</tr>
</tbody>
</table>
Models available in OpenFOAM (for a given solver)

- rhoThermo pureMixture polynomial janaf PengRobinsonGas specie sensibleEnthalpy
- rhoThermo pureMixture sutherland hConst PengRobinsonGas specie sensibleEnthalpy
- rhoThermo pureMixture sutherland hConst incompressiblePerfectGas specie sensibleEnthalpy
- rhoThermo pureMixture sutherland hConst perfectGas specie sensibleInternalEnergy
- rhoThermo pureMixture sutherland hConst perfectGas specie sensibleInternalEnergy
- rhoThermo pureMixture sutherland janaf incompressiblePerfectGas specie sensibleEnthalpy
- rhoThermo pureMixture sutherland janaf incompressiblePerfectGas specie sensibleInternalEnergy
- rhoThermo pureMixture sutherland janaf perfectGas specie sensibleInternalEnergy
- rhoThermo pureMixture sutherland janaf perfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture const hConst incompressiblePerfectGas specie sensibleEnthalpy
- rhoThermo reactingMixture const hConst incompressiblePerfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture const hConst perfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture const hConst perfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture polynomial hPolynomial icoPolynomial specie sensibleInternalEnergy
- rhoThermo reactingMixture polynomial hPolynomial icoPolynomial specie sensibleInternalEnergy
- rhoThermo reactingMixture sutherland janaf incompressiblePerfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture sutherland janaf incompressiblePerfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture sutherland janaf perfectGas specie sensibleInternalEnergy
- rhoThermo reactingMixture sutherland janaf perfectGas specie sensibleInternalEnergy
- rhoThermo singleStepReactingMixture sutherland janaf perfectGas specie sensibleInternalEnergy
- rhoThermo singleStepReactingMixture sutherland janaf perfectGas specie sensibleInternalEnergy
- rhoThermo veryInhomogeneousMixture const hConst incompressiblePerfectGas specie sensibleEnthalpy
- rhoThermo veryInhomogeneousMixture const hConst perfectGas specie sensibleEnthalpy
- rhoThermo veryInhomogeneousMixture sutherland janaf incompressiblePerfectGas specie sensibleEnthalpy
- rhoThermo veryInhomogeneousMixture sutherland janaf perfectGas specie sensibleEnthalpy
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 by mass at atmospheric pressure.](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

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

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

\( Rk: \) need a heatTransfer solver

This equation of state \( \rho(T) \) needs to be implemented

Figure 2.2: Density of an argon gas versus temperature.

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

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

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

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 it:
  
  cd $WM_PROJECT_DIR
  
  cp -r --parents applications/solvers/heatTransfer/thermoFoam $WM_PROJECT_USER_DIR
  
  cd $WM_PROJECT_USER_DIR/applications/solvers/heatTransfer
  
  mv thermoFoam myThermoFoam
  
  cd myThermoFoam
  
  mv thermoFoam.C myThermoFoam.C

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

- Clean and compile
  
  wclean
  
  rm -r Make/linux* (to clean also the Debug version in the system)
  
  wmake
OpenFOAM solver heatTransfer/thermoFoam:

Parts of thermoFoam involving a thermophysical model

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

Info<< "Reading thermophysical properties\n" << endl;

autoPtr<rhoThermo> pThermo(rhoThermo::New(mesh));
rhoThermo& thermo = pThermo();
thermo.validate(args.executable(), "h", "e");

volScalarField rho
{
...

#include "rhoThermo.H"
...
#include "EEqn.H"
...
```

Create the object "thermo"

Include a thermophysic library; that object is used on the next slide.

Include an energy conservation equation (see next slide)
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() == "c"
    ? 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)

This total energy conservation equation is explained in the next slides
Energy conservation - total energy of flowing fluid: $\rho(\hat{h} + K)$

$$\frac{\partial}{\partial t} \left( \rho (\hat{h} + K) \right) + \nabla \cdot \left( \rho \mathbf{v} (\hat{h} + K) \right) - \frac{\partial p}{\partial t} - \nabla \cdot q = 0$$

where $q = \alpha \nabla \hat{h}$

- Enthalpy: $\hat{h} = \hat{e} + \frac{p}{\rho}$
- Kinetic energy: $K$
- Thermal diffusivity: $\alpha$

```cpp
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: \( \rho (\hat{h} + K) \)

\[
\frac{\partial}{\partial t} \left( \rho (\hat{h} + K) \right) + \nabla \cdot (\rho \mathbf{v} (\hat{h} + K)) - \frac{\partial p}{\partial t} - \nabla \cdot q = 0 \quad \text{where} \quad q = \alpha \nabla \hat{h}
\]
Energy conservation - total energy of flowing fluid: \( \rho (\hat{h} + K) \)

\[
\frac{\partial}{\partial t} \left( \rho (\hat{h} + K) \right) + \nabla \cdot \left( \rho \nu (\hat{h} + K) \right) - \frac{\partial p}{\partial t} - \nabla \cdot q = 0 \quad \text{where} \quad q = \alpha \nabla \hat{h}
\]

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

This is not true (as he is not “e”) so this part is not used while this term can be calculated.

In fact in thermoFOAM \( dpdt \) (created in createFields) it is set to zero since this is a frozen field
Energy conservation - total energy of flowing fluid: \( \rho (\hat{h} + K) \) - in a control volume moving at \( \mathbf{v}_b \)

\[
\frac{\partial}{\partial t} \left( \rho (\hat{h} + K) \right) + \nabla \cdot (\rho \mathbf{v} (\hat{h} + K)) - \frac{\partial p}{\partial t} - \nabla \cdot \mathbf{q} = 0
\]

where \( \mathbf{q} = \alpha \nabla \hat{h} \)

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

If "\( \hat{e} \)" is the specific internal energy \( \hat{e} \) this is true so this term can be calculated

while this part is not used

\[
\begin{align*}
\text{fvm::ddt(rho, he)} & + \text{fvm::div(phi, he)} \\
& + \text{fvc::ddt(rho, K)} + \text{fvc::div(phi, K)} \\
& + ( \begin{cases}
\text{he.name()} == "\text{c}" \\
? \text{fvc::div} \left( \begin{cases}
\text{fvc::absolute(phi/fvc::interpolate(rho), U), p,} \\
"\text{div(phiv,p)}"
\end{cases} \right)
\end{cases}
\end{align*}
\]

This is true

so this term can be calculated

while this part is not used
Energy conservation - total energy of flowing fluid: $\rho(\hat{h} + K)$ - in a control volume moving at $\mathbf{v}_b$ 

$$\frac{\partial}{\partial t} \left( \rho (\hat{h} + K) \right) + \nabla \cdot \left( \rho \mathbf{v} (\hat{h} + K) \right) - \frac{\partial p}{\partial t} - \nabla \cdot q = 0 \quad \text{where} \quad q = \alpha \nabla \hat{h}$$

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

**Kinetic energy:** $K$

**Thermal diffusivity:** $\alpha$
alphaEff (the thermal diffusivity) is made of 2 contributions:

\[
\alpha_{\text{Eff}} = \alpha_{\text{laminar}} + \alpha_{\text{turbulent}}
\]

It is set in solver/heatTransfer/thermoFoam/setAlphaEff.H via the turbulence library : src/turbulenceModels

the ”turbulenceModels” library is linked to the thermo library src/thermophysicalModels so if the file names of thermophysical library(*.so) are changes, also the turbulenceModels need to be recompiled with links to the new library names

alpha turbulent is defined in src/turbulenceModels

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

The heat flux is: \( q = -\alpha \nabla he \)
where the laminar part of \( \alpha \) is either:

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

or

\[ \frac{\kappa}{C_v} \] 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 termodynamic 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 OpenFOAM) the equation of state

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

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**: copy 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
Prepare your library thermophysicalModels/specie

- Copy the folder “specie” of the library in your user directory:
  ```
  foam
  cp -r --parents src/thermophysicalModels/specie $WM_PROJECT_USER_DIR
  cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
  cd specie
  ```

- Modify the Make/files to:
  ```
  LIB = $(FOAM_USER_LIBBIN)/libspecie
  ```

- Clean & compile
  You should be in the directory specie
  ```
  wclean lib
  rm -r Make/kinux*
  wmake libso
  ```

Now the executable is in your working space, and your “libspectie” 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 thermophysicalModels/basic

- **Copy the folder “basic” of the library in your user directory and rename it:**
  
  ```
  foam
  cp -r --parents src/thermophysicalModels/basic $WM_PROJECT_USER_DIR
  cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
  cd basic
  ```

- **Modify Make/files to:**

  ```
  LIB = $(FOAM_USER_LIBBIN)/libfluidThermophysicalModels
  ```

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

- Modify the Make/options file to (since basic needs to be linked to specie at compilation)

```
EXE_INC = \n   -I$(LIB_SRC)/finiteVolume/lnInclude \n   -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/lnInclude 
   -I$(LIB_SRC)/meshTools/lnInclude

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

- Clean the dependencies and compile
You should be in the directory basic
wclean lib
rm -r Make/linux*
wmake libso

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

(2) Indicates that the compiler 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 compiler 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

There it can be checked that “myThermophysicalModels/spe
cie” is picked in 
/home/…/isabelle-2.3.x/src

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/thermophysicalModels/specie

- In specie/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/thermophysicalModels/specie

- **Copy and rename an existing model:** *(Prepare the structure, you should be in specie)*
  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 and compile specie**
wclean lib
rm –r Make/linux*
wmake libso
Go in the directory kineticAr and open kineticArTransportI.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
```

- thermal conductivity
- Comment the original model
- insert the new model provided in the file Ar_Data_ThermalConduct
Implement a new transport property K: step 2

- Clean the dependencies ("wclean lib" and "rm –r Make/linux*") 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
}
```

39
Declare the new thermophysical model in user src/thermophysicalModels/basic

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

```c
#include "kineticArTransport.H"
...
makeThermo
{
    rhoThermo,
    heRhoThermo,
    pureMixture,
    kineticArTransport,
    sensibleEnthalpy,
    hConstThermo,
    perfectGas,
    specie
};
```

- Clean the dependencies (“wclean lib” and “rm –r Make/linux* ”)
- Compile myBasic (“wmake libso”)

New combination of Ia, Ib, Ic, II and III (see slides 3 to 7) defining a new thermophysical model
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

```plaintext
-I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \n-I$(WM_PROJECT_USER_DIR)/thermophysicalModels/basic/lnInclude 

EXE_LIBS = \n    -lfiniteVolume 
    -lfiniteVolume 

EXE_LIBS = \n    -L$(FOAM_USER_LIBBIN) 
    -lfiniteVolume 
```

Check that the solver is using your library:

```
lld `which thermoFoam` | grep specie
```

```
lld `which thermoFoam` | grep specie
```
Implement a new transport property $K$: step 5

**Call the new thermophysicalModel in a test case**

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

```plaintext
 thermoType
 {
   type rhoThermo;
   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: do a plotOverLine of temperature for both cases
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 $C_p = \rho(T) c_p(T)$

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
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 thermophysical model in user src/thermophysicalModels/basic

- In specie/include/thermoPhysicsTypes.H add in the header

```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 <
            rhoKineticAr<specie>,
        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 model in user src/thermophysicalModels/specie

- **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 perfectGastI.H rhoKineticArI.H
  - open the files one by one and replace “perfectGas” with “rhoKineticAr”

*Prepare the structure*
Open `rhoKineticArI.H` and do the following modifications:

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

    //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

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

- Comment the original model
- Insert the new model provided in the file `density_Ar_Data`
Comment the original model (ideal gas)

\[ \rho \frac{P}{T} = \rho(T) \]  \hspace{6cm} (6/16)

Modify also

Implement new thermodynamic properties and equation of state: step 2

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

    // new model
    // psi should not be used with the rhoKineticAr model
    return 0.0;
}
\end{verbatim}

Write the new model

Rk: psi is set to zero since the plasma model implemented here is mechanically incompressible, and thermaly 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
    // Z should not be used with the rhoKineticAr model
    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
    // cpMcv should not be used with the rhoKineticAr mode
    return 0.0;
}
```

Comment the original model

write the new model
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 “instantiated” and below replace “return “hConst<” “ with “ return “hKineticAr<” “
```
Implement new thermodynamic properties and equation of state: step 2

- 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
} const
{
    // 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 $H_f$ 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 $H_f$ is zero.
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();
}
```

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

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

chemical enthalpy

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 ha. For a one-fluid model hc is the enthalpy of formation. Here it is set to zero (see the remark on slide 53).

- Clean and compile

  wclean lib
  rm –r Make/linux*
  wmake libso
Implement new thermodynamic properties and equation of state: step 4

 Declare the new thermophysical model in user src/thermophysicalModels/basic

- In basic/rhoThermo/rhoThermos.C add the following lines in the header

```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 and compile

```bash
wclean lib
rm –r Make/linux*
wmake libso
```
Implement new thermodynamic properties and equation of state: step 6

Call the new thermophysical model in a test case and run

- Copy `blockThermoFoamCase` to `blockKineticArThermoFoamCase` and clean (`wclean`)
- Update `constant/thermophysicalProperties` to

```cpp
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: do a `plotOverLine` of temperature for both cases