Implementation Guide:

Real-fluid thermophysicalModels: An OpenFOAM-based library for reacting flow simulations at high pressure

Danh Nam Nguyen, Ki Sung Jung, Jae Won Shim, Chun Sang Yoo

To cite this article: D. N. Nguyen, K. S. Jung, J. W. Shim, C. S. Yoo, Real-fluid thermophysicalModels: An OpenFOAM-based library for reacting flow simulations at high pressure, Computer Physics Communications (2021)(submitted).

Note: This document presents the implementation for only a set of widely used real-fluid models i.e., the Soave-Redlich-Kwong [1, 2] equation of state (EoS), Chung's model [3] for dynamic viscosity and thermal conductivity, mixture averaged model for mass diffusivity using Takahashi's correction [4] for binary diffusion coefficients at high pressure. For other models such as Peng-Robinson EoS [5] or the Standard Kinetic Theory model [6] for transport properties, readers are referred to *2-GuideForExtension.pdf* file.

Contents

1. A procedure for real-fluid models implementation

Figure 1 illustrates the class diagram of *thermophysicalModels* library in OpenFOAM 6.0 with sufficient classes associated with reacting flow simulations. The class diagram provides important features such as inheritance and the interface between classes which are necessary for code development of the *thermophysicalModels* library. In this diagram, the boxes with marked numbers denote the sample classes which can be one of the classes in side the box due to runtime selection mechanism. *ThermoType Block* and *MixtureType Block* contain all classes representing THERMPHYS models and mixture models. *ChemistryType Block* consists of chemistry models while *BasicType Block* involves base classes for types of system (e.g., *rho*-based or *psi*-based).

Since the interface between classes in *thermophysicalModels* is relatively complicated, we propose a simple procedure to implement real-fluid models as follows to minimize the debugging efforts during the implementation. Although the following proposed procedure is based on OpenFOAM-6, it can also be referred to implement real-fluid models in other version of OpenFOAM. The procedure has three main steps:

- Step 1: Create runtime selectable packages including real-fluid models;
- Step 2: Implement real-fluid thermophysical (THERMPHYS) models;
- Step 3: Modify related classes.

In the step 1, new classes representing to real-fluid models are created by copying from existing classes in the *thermophysicalModels* library without implementing any equations of real-fluid models into the code. In addition, some macros files need to be modified to create runtime selectable package for real-fluid models. The detail instruction about the step 1 is presented in Sec. 2. In the step 2, the source code of new class are modified to deliver equations of new models. The detail instruction about the step 2 is shown in Sec. 3. In step 3, source code of several classes are modified to be compatible with new added classes due to the use of templates, polymorphism, and inheritance in OpenFOAM. The detail instruction about the step 3 is presented in Sec. 4.

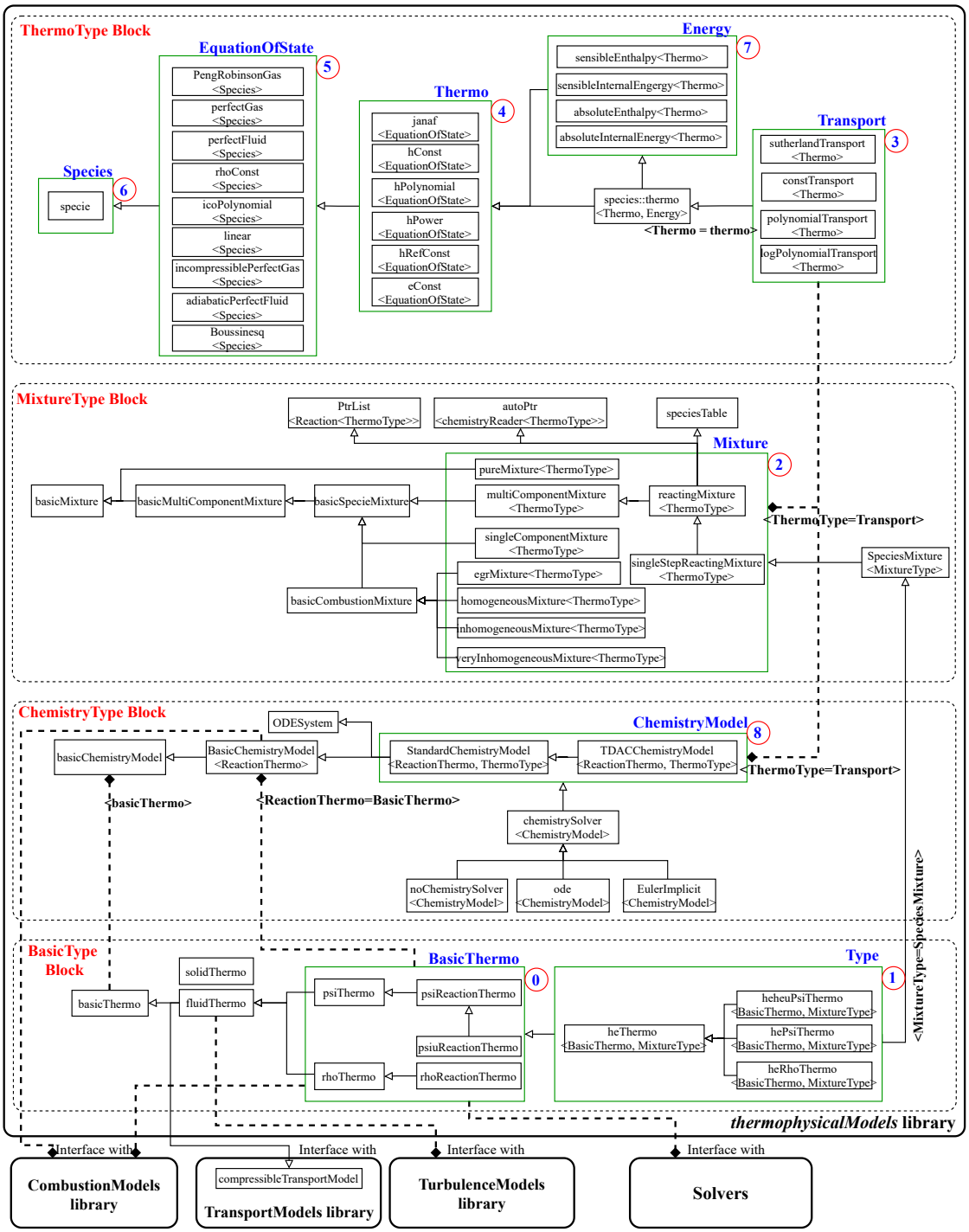

Figure 1: The class diagram of the *thermophysicalModels* library in OpenFOAM 6.0. The boxes with marked numbers denote sample classes. The arrow-line denotes the inheritance relationship in which the direction of arrow is from a subclass to its base class. A dashed line denotes a class-class or class-solver interface in which one class is used in another or in a solver.

Figure. 2 shows the class diagram of the new *thermophysicalModels* library of OpenFOAM-6 including real fluid models. In this library, several new classes are created (in yellow boxes) such as *rfSpecie*, *soaveRedlichKwong*, *rfJanafThermo*, and *chungTakaTransport* representing for species, soave-Redlich-Kwong (SRK) equation of state (EoS), real-fluid JANAF-based, and Chung's models, respectively. Note that mixture averaged mass diffusivity model with Takahashi correlation for binary diffusion coefficients is included in *chungTakaTransport* class. The *SRKchungTakaMixture* and *SRKchungTakaReactingMixture* are two new classes representing for mixture models which are implemented using a new algorithm proposed in our paper. The *SRKchungTakaStandardChemistryModel* is a new chemistry model class created by copying the original *StandardChemistryModel* class in OpenFOAM containing the interface with real fluid models. Furthermore, several existing classes in original *thermophysicalModels* library have also been modified (in gray boxes) to be compatible with new added classes.

New classes have been created:

```
// In thermophysicalModels/specie directory:
- specie/rfSpecie/rfSpecie
- specie/equationOfState/soaveRedlichKwong/soaveRedlichKwong
- specie/themo/rfJanaf/rfJanafThermo
- specie/transport/chungTaka/chungTakaTransport
// In thermophysicalModels/reaction directory:
- reactionThermo/mixtures/SRKchungTakaMixture
- reactionThermo/mixtures/SRKchungTakaReactingMixture
// In thermophysicalModels/chemistryModel directory:
- chemistryModel/chemistryModel/SRKchungTakaStandardChemistryModel
```

Classes have been modified:

```
// In thermophysicalModels/reactionThermo/mixtures directory:
- reactionThermo/mixtures/multiComponentMixture/multiComponentMixture
- reactionThermo/mixtures/singleComponentMixture/singleComponentMixture
- reactionThermo/mixtures/egrMixture/egrMixture
- reactionThermo/mixtures/homogeneousMixture/homogeneousMixture
```

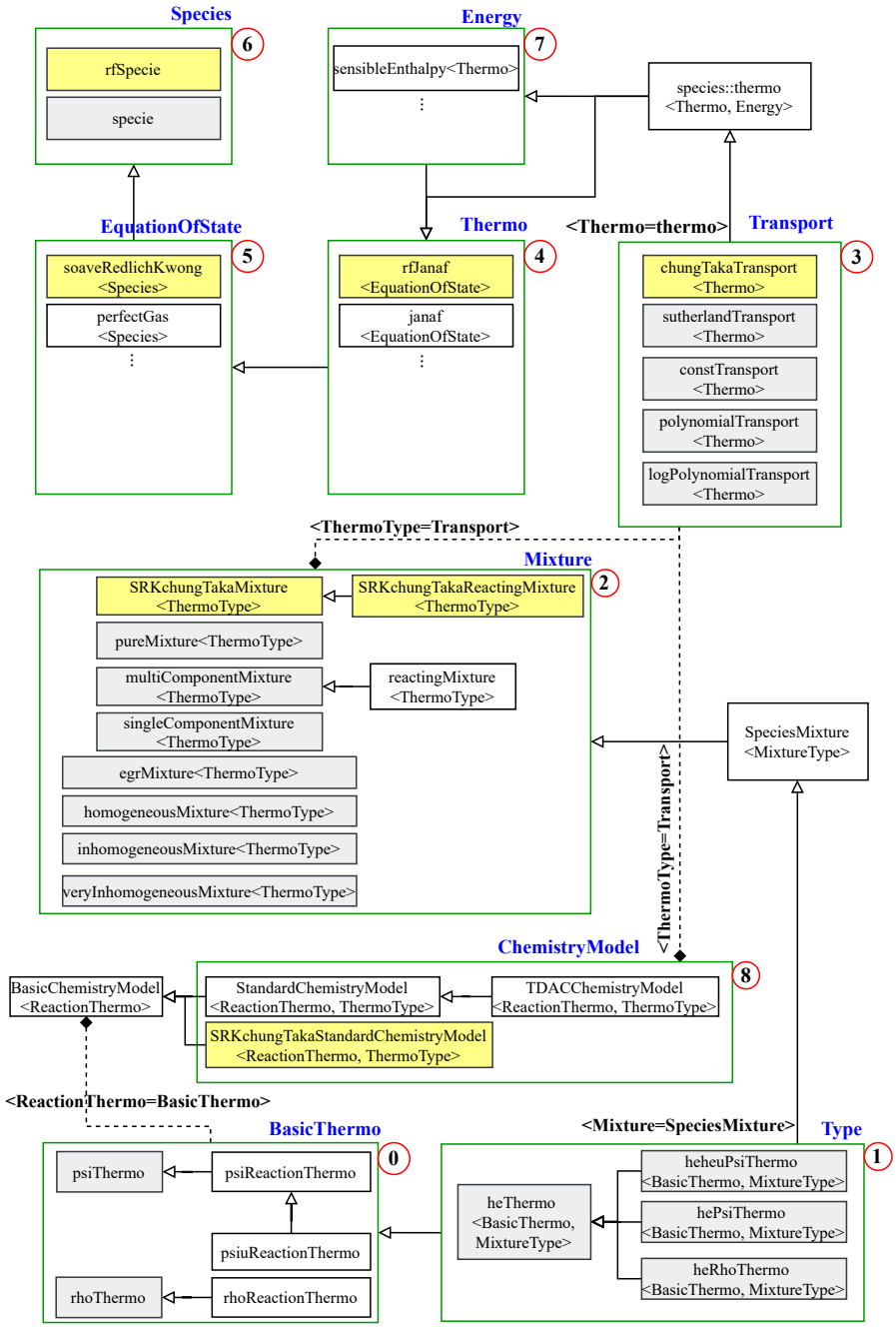

Figure 2: The class diagram of the real-fluid *thermophysicalModels* library in OpenFOAM-6. Yellow boxes are classes need to be created representing to real-fluid models. Gray boxes are existing classes in the original *thermophysicalModel* library need to be modified. The arrow-line denotes the inheritance relationship in which the direction of arrow is from a subclass to its base class. A dashed line denotes a class-class or class-solver interface in which one class is used in another or in a solver

```
- reactionThermo/mixtures/inhomogeneousMixture/inhomogeneousMixture
- reactionThermo/mixtures/veryInhomogeneousMixture/
   veryInhomogeneousMixture
- reactionThermo/psiuReactionThermo/heheuPsiThermo
// In thermophysicalModels/basic directory:
- basic/mixtures/pureMixture
- basic/heThermo/heThermo
- basic/rhoThermo/rhoThermo
- basic/rhoThermo/heRhoThermo
- basic/psiThermo/psiThermo
- basic/psiThermo/hePsiThermo
```

2. Create runtime selectable packages of real-fluid models

2.1. Making source files

Prepare a directory on your system, e.g., *yourDirectory*.

```
mkdir ~/OpenFOAM/yourDirectory
cd ~/OpenFOAM/yourDirectory/
```

Create *src/thermophysicalModels* directory.

```
mkdir -p src/thermophysicalModels
```

Set an environment variable prescribing the path of the *src* directory.

```
echo "export␣LIB_REALFLUID_SRC=~/OpenFOAM/yourDirectory/src/" >> ~/.bashrc
source ~/.bashrc
```

Go to *thermophysicalModels* directory.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels
```

Copy *thermphysicalModels/specie* directory from original OpenFOAM into *thermophysicalModels* in your user directory.

```
cp -rf -p $WM_PROJECT_DIR/src/thermophysicalModels/specie .
```

Go to */yourDirectory/src/thermophysicalModels/specie* directory.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/specie
```

Create *rfSpecie* class by copying from *specie* class.

```
cp -rf specie rfSpecie
cd rfSpecie
mv specie.H rfSpecie.H
mv specie.C rfSpecie.C
mv specieI.H rfSpecieI.H
```

Open these files and replace *specie* by *rfSpecie*. To make sure you do not miss any thing you should use a command to automatically find and replace a string, for instance use the following command if you are using vim text editor.

```
vi specie.H
:%s/specie/rfSpecie/g
```

Note that do not change *specie* as the name of a dictionary in a constructor of *rfSpecie* class as follows:

```
37 // * * * * * * * * * * * * Constructors  * * * *  * * * * * * * * //
38
39 Foam::rfSpecie::rfSpecie(const dictionary& dict)
40 :
41     name_(dict.dictName()),
42     Y_(dict.subDict("specie").lookupOrDefault("massFraction", 1.0)),
43     molWeight_(readScalar(dict.subDict("specie").lookup("molWeight")))
44 {}
```

Do the same to create *soaveRedlichKwong*, *rfJanafThermo*, and *chungTakaTransport* classes from *perfectGas*, *janafThermo*, and *sutherlandTransport* classes inside *equationOfState*, *thermo*, and *transport* directories, respectively. For instance:

```
cd equationOfState
cp -rf perfectGas soaveRedlichKwong
cd soaveRedlichKwong
mv perfectGas.H soaveRedlichKwong.H
mv perfectGas.C soaveRedlichKwong.C
```

```
mv perfectGasI.H soaveRedlichKwongI.H
vi soaveRedlichKwong.H
:%s/perfectGas/soaveRedlichKwong/g
...
```

Note that change *specie* into *rfSpecie* inside the *rfJanafThermo* and *chungTransport* classes since *rfSpecie* class is their base class. Do not confuse between *specie* (class's name) and *Specie* (name of type in templates), and the names at runtime of real-fluid models should be specified correctly in the *.H files as follows:

```
/* In rfSpecie.H file: */
//- Runtime type information
ClassName("rfSpecie");

/* In soaveRedlichKwong.H file: */
//- Return the instantiated type name
static word typeName()
{
    return "soaveRedlichKwong<" + word(Specie::typeName_()) + '>';
}

/* In rfJanafThermo.H file: */
static word typeName()
{
    return "rfJanaf<" + EquationOfState::typeName() + '>';
}

/* In chungTransport.H file: */
static word typeName()
{
    return "chungTaka<" + Thermo::typeName() + '>';
}
```

2.2. Making runtime selectable thermophysical model package

In OpenFoam, the THERMPHYS models are used in a reacting flow simulation by using runtime selection mechanism such that the *thermoType* dictionary is specified by user at runtime in the *constant/thermophysicalProperties* file as:

```
thermoType
{
    type            hePsiThermo;      //(1) type of system
    mixture         reactingMixture;  //(2) mixture model
    transport       sutherland;       //(3) transport model
    thermo          janaf;            //(4) thermodynamic model
    energy          sensibleEnthalpy; //(7) energy type model
    equationOfState perfectGas;       //(5) equation of state model
    specie          specie;           //(6) species model
}
```

A set of THERMPHYS models like that can be referred as a runtime *thermo-package*. Our goal is create four *thermo-packages* including real-fluid THERMPHYS models in which the type of system can be either *hePsiThermo* (*psi*-based) or *heRhoThermo* (*rho*-based) and the energy type can be either *sensibleEnthalpy* or *sensibleInternalEnergy* as following:

```
thermoType
{
    type            hePsiThermo;        //or "heRhoThermo"
    mixture         SRKchungTakaReactingMixture;
    transport       chungTaka;
    thermo          rfJanaf;
    energy          sensibleEnthalpy;   //or "sensibleInternalEnergy"
    equationOfState soaveRedlichKwong;
    specie          rfSpecie;
}
```

The structure of *thermophysicalModels* library in OpenFOAM is relatively complicated since it is built based on a runtime selection mechanism using dynamic polymorphism, class templates, and macros. This instruction covers only models related to reacting flow

simulations.

2.2.1. Making interface inside thermophysicalModels library

Copy the following directories from original OpenFOAM into *thermophysicalModels* in your directory.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels
cp -rf -p $WM_PROJECT_DIR/src/thermophysicalModels/basic .
cp -rf -p $WM_PROJECT_DIR/src/thermophysicalModels/reactionThermo .
cp -rf -p $WM_PROJECT_DIR/src/thermophysicalModels/chemistryModel .
```

Then modify these following files in order.

```
// =============== The list files ======================================
// In thermophysicalModels/specie directory
1. thermophysicalModels/specie/Make/files
2. thermophysicalModels/specie/include/thermoPhysicsTypes.H
3. thermophysicalModels/specie/include/reactionTypes.H
4. thermophysicalModels/specie/reaction/reactions/makeReaction.H
5. thermophysicalModels/specie/reaction/reactions/makeReactions.C
//--> compile specie to make specie.so

// In thermophysicalModels/basic directory
1. thermophysicalModels/basic/Make/files (+ options)
2. thermophysicalModels/basic/psiThermo/psiThermos.C
//--> compile basic to make fluidThermophysicalModels.so

// In thermophysicalModels/reactionThermo directory
1. thermophysicalModels/reactionThermo/Make/files (+ options)
2. thermophysicalModels/reactionThermo/psiReactionThermo/
   psiReactionThermos.C
3. thermophysicalModels/reactionThermo/rhoReactionThermo/
   rhoReactionThermos.C
4. thermophysicalModels/reactionThermo/chemistryReaders/chemistryReader/
   makeChemistryReaders.C
//--> compile reactionThermo to make reactionThermophysicalModels.so
```

```
// In thermophysicalModels/chemistryModel directory
1. thermophysicalModels/chemistryModel/chemistryModel/BasicChemistryModel/
   BasicChemistryModels.C
2. thermophysicalModels/chemistryModel/chemistryModel/basicChemistryModel/
   basicChemistryModelTemplates.C
  //new created files
3. thermophysicalModels/chemistryModel/chemistrySolver/chemistrySolver/
   makeRealFluidChemistrySolverTypes.H
4. thermophysicalModels/chemistryModel/chemistrySolver/chemistrySolver/
   makeRealFluidChemistrySolvers.C
5. thermophysicalModels/chemistryModel/Make/files (+ options)
//--> compile chemistryModel to make chemistryModel.so
// ======================================================================
```

a. Modification in */yourDirectory/src/thermophysicalModels/specie* directory:

In *Make/files* file, add *rfSpecie.C* into the list of source files and change the position to save the binary file, *specie.so*, at user directory.

```
...
rfSpecie/rfSpecie.C                  // add new source file
...
LIB = $(FOAM_USER_LIBBIN)/libspecie  // save at user directory
```

In *include/thermoPhysicsTypes.H* file, include the header files of new classes.

```
...
#include "rfSpecie.H"
#include "soaveRedlichKwong.H"
#include "rfJanafThermo.H"
#include "chungTakaTransport.H"
```

Then make the shorthand names of combinations for set of real-fluid models.

```
typedef
chungTakaTransport
<
```

```
    species::thermo
    <
        rfJanafThermo
        <
            soaveRedlichKwong<rfSpecie>
        >,
        sensibleEnthalpy
    >
>
chungTakaRealJsrkHThermoPhysics;
```

and

```
typedef
chungTakaTransport
<
    species::thermo
    <
        rfJanafThermo
        <
            soaveRedlichKwong<rfSpecie>
        >,
        sensibleInternalEnergy
    >
>
chungTakaRealJsrkEThermoPhysics;
```

Note that the these two shorthand names *chungTakaRealJsrkHThermoPhysics* and *chungTakaRealJsrkEThermoPhysics* will be used many times later on. We will refer these two combinations are two THERMPHYS types.

In *include/reactionTypes.H* file, create new alias of reaction types based on two THERMPHYS types defined in *thermophysicalTypes.H* file as follows.

```
...
typedef Reaction<chungTakaRealJsrkHThermoPhysics>
   chungTakaRealJsrkHReaction;
```

```
...
typedef Reaction<chungTakaRealJsrkEThermoPhysics>
    chungTakaRealJsrkEReaction;
...
```

In *reaction/reactions/makeReaction.H* file, include the header files of new classes.

```
...
#include "chungTakaTransport.H"
#include "rfJanafThermo.H"
#include "soaveRedlichKwong.H"
```

In *reaction/reactions/makeReactions.C* file, create new *makeReactions* macros as follows.

```
...
makeReactions(chungTakaRealJsrkHThermoPhysics, chungTakaRealJsrkHReaction)
...
makeReactions(chungTakaRealJsrkEThermoPhysics, chungTakaRealJsrkEReaction)
...
```

Compile to make *specie.so*.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/specie
wclean
wmake libso
```

If errors occur, check carefully your steps again to make sure you do not miss any thing.

b. Modification in */yourDirectory/src/thermophysicalModels/basic* directory:

In *Make/files* file, change the position to save the binary file, *fluidThermophysicalModels.so*, at user directory.

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

In *Make/options* file, change the path to include the header files of *specie.o* library as follows:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
```

```
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalProperties/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude

LIB_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lcompressibleTransportModels \
    -lspecie \
    -lthermophysicalProperties \
    -lfiniteVolume \
    -lmeshTools
```

Compile to make *fluidThermophysicalModels.so*.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/basic
wclean
wmake libso
```

If errors occur, check carefully your steps again to make sure you do not miss any thing.

c. Modification in *yourDirectory/src/thermophysicalModels/reactionThermo* directory:

Go to */yourDirectory/src/thermophysicalModels/reactionThermo/mixtures* directory.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/reactionThermo/
   mixtures
```

Create *SRKchungTakaMixture* class by copying from *multiComponentMixture* class.

```
cp -rf multiComponentMixture SRKchungTakaMixture
cd SRKchungTakaMixture
mv multiComponentMixture.H SRKchungTakaMixture.H
mv multiComponentMixture.C SRKchungTakaMixture.C
```

Open these files and replace *multiComponentMixture* by *SRKchungTakaMixture*. To make sure you do not miss any thing you should use a command to automatically find and replace a string, for instance use the following command if you are using vim text editor.

```
vi SRKchungTakaMixture.H
```

```
:%s/multiComponentMixture/SRKchungTakaMixture/g
```

Create *SRKchungTakaReactingMixture* class by copying from *reactingMixture* class.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/reactionThermo/
   mixtures
cp -rf reactingMixture SRKchungTakaReactingMixture
cd SRKchungTakaReactingMixture
mv reactingMixture.H SRKchungTakaReactingMixture.H
mv reactingMixture.C SRKchungTakaReactingMixture.C
```

Open these files and replace *reactingMixture* and *multiComponentMixture* by *SRKchungTakaReactingMixture* and *SRKchungTakaMixture*, respectively. To make sure you do not miss any thing you should use a command to automatically find and replace a string, for instance use the following command if you are using vim text editor.

```
vi SRKchungTakaReactingMixture.H
:%s/reactingMixture/SRKchungTakaReactingMixture/g
:%s/multiComponentMixture/SRKchungTakaMixture/g
```

In *Make/files* file, change the position to save the object file, *reactionThermophysicalModels.so*, at user directory.

```
...
LIB = $(FOAM_USER_LIBBIN)/libreactionThermophysicalModels
```

In *Make/options* file, change the path to include the header files of *specie.so* and *fluidThermophysicalModels.so* libraries as follows:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/solidSpecie/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude

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

```
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lspecie \
    -lsolidSpecie \
    -lfiniteVolume
```

In *psiReactionThermo/psiReactionThermos.C*, include the header files of new classes and then create new *makeThermoPhysicsReactionThermos* macros as follows:

```
...
#include "rfSpecie.H"
#include "soaveRedlichKwong.H"
#include "rfJanafThermo.H"
#include "chungTakaTransport.H"
#include "SRKchungTakaMixture.H"
#include "SRKchungTakaReactingMixture.H"
...
// real fluid mixture thermo for sensible enthalpy
makeThermoPhysicsReactionThermos
(
    psiThermo,
    psiReactionThermo,
    hePsiThermo,
    SRKchungTakaMixture,
    chungTakaRealJsrkHThermoPhysics
);

// real fluid mixture thermo for internal energy
makeThermoPhysicsReactionThermos
(
    psiThermo,
    psiReactionThermo,
    hePsiThermo,
    SRKchungTakaMixture,
    chungTakaRealJsrkEThermoPhysics
```

```
);

// real fluid mixture reaction thermo for sensible enthalpy
makeThermoPhysicsReactionThermos
(
    psiThermo,
    psiReactionThermo,
    hePsiThermo,
    SRKchungTakaReactingMixture,
    chungTakaRealJsrkHThermoPhysics
);

// real fluid mixture reaction thermo for internal energy
makeThermoPhysicsReactionThermos
(
    psiThermo,
    psiReactionThermo,
    hePsiThermo,
    SRKchungTakaReactingMixture,
    chungTakaRealJsrkEThermoPhysics
);
...
```

In *rhoReactionThermo/rhoReactionThermos.C*, include the header files of new classes and then create new *makeThermoPhysicsReactionThermos* macros as follows:

```
...
#include "rfSpecie.H"
#include "soaveRedlichKwong.H"
#include "rfJanafThermo.H"
#include "chungTakaTransport.H"
#include "SRKchungTakaMixture.H"
#include "SRKchungTakaReactingMixture.H"
...
// real fluid mixture thermo for internal energy
```

```
makeThermoPhysicsReactionThermos
(
    rhoThermo,
    rhoReactionThermo,
    heRhoThermo,
    SRKchungTakaMixture,
    chungTakaRealJsrkEThermoPhysics
);

// real fluid mixture thermo for sensible enthalpy
makeThermoPhysicsReactionThermos
(
    rhoThermo,
    rhoReactionThermo,
    heRhoThermo,
    SRKchungTakaMixture,
    chungTakaRealJsrkHThermoPhysics
);

// real fluid mixture reaction thermo for internal energy
makeThermoPhysicsReactionThermos
(
    rhoThermo,
    rhoReactionThermo,
    heRhoThermo,
    SRKchungTakaReactingMixture,
    chungTakaRealJsrkEThermoPhysics
);

// real fluid mixture reaction thermo for sensible enthalpy
makeThermoPhysicsReactionThermos
(
    rhoThermo,
    rhoReactionThermo,
```

```
    heRhoThermo,
    SRKchungTakaReactingMixture,
    chungTakaRealJsrkHThermoPhysics
);
...
```

In *chemistryReaders/chemistryReader/makeChemistryReaders.C* file, create new *makeChemistryReader* and *makeChemistryReaderType* macros as follows:

```
...
makeChemistryReader(chungTakaRealJsrkHThermoPhysics);
makeChemistryReaderType(foamChemistryReader,
   chungTakaRealJsrkHThermoPhysics);
...
makeChemistryReader(chungTakaRealJsrkEThermoPhysics);
makeChemistryReaderType(foamChemistryReader,
   chungTakaRealJsrkEThermoPhysics);
...
```

Compile to make *reactionThermophysicalModels.so*.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/reactionThermo
wclean
wmake libso
```

If errors occur, check carefully your steps again to make sure you do not miss any thing.

d. Modification in */yourDirectory/src/thermophysicalModels/chemistryModel* directory:

Create *SRKchungTakaStandardChemistryModel* class by copying from *StandardChemistryModel* class.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/chemistryModel/
   chemistryModel/
cp -rf StandardChemistryModel SRKchungTakaStandardChemistryModel
cd SRKchungTakaStandardChemistryModel
mv StandardChemistryModel.H SRKchungTakaStandardChemistryModel.H
mv StandardChemistryModel.C SRKchungTakaStandardChemistryModel.C
mv StandardChemistryModelI.H SRKchungTakaStandardChemistryModelI.H
```

Open these files and replace *StandardChemistryModel* and *reactingMixture* by *SRKchungTakaStandardChemistryModel* and *SRKchungTakaReactingMixture*, respectively. To make sure you do not miss any thing you should use a command to automatically find and replace a string, for instance use the following command if you are using vim text editor.

```
vi SRKchungTakaStandardChemistryModel.H
:%s/StandardChemistryModel/SRKchungTakaStandardChemistryModel/g
:%s/reactingMixture/SRKchungTakaReactingMixture/g
```

It is of importance to note that the name of this class in runtime is defined in the *SRKchungTakaStandardChemistryModel.H* file as follows:

```
//- Runtime type information
TypeName("SRKchungTakaStandard");
```

In *chemistryModel/chemistryModel/BasicChemistryModel/BasicChemistryModels.C* file, include the header files of new classes and then create new *makeChemistryModelType* macros as follows:

```
#include "SRKchungTakaStandardChemistryModel.H"
...
makeChemistryModelType
(
    SRKchungTakaStandardChemistryModel,
    psiReactionThermo,
    chungTakaRealJsrkHThermoPhysics
);

makeChemistryModelType
(
    SRKchungTakaStandardChemistryModel,
    rhoReactionThermo,
    chungTakaRealJsrkHThermoPhysics
);

makeChemistryModelType
(
```

```
    SRKchungTakaStandardChemistryModel,
    psiReactionThermo,
    chungTakaRealJsrkEThermoPhysics
);

makeChemistryModelType
(
    SRKchungTakaStandardChemistryModel,
    rhoReactionThermo,
    chungTakaRealJsrkEThermoPhysics
);
...
```

In *chemistryModel/chemistryModel/basicChemistryModel/basicChemistryModelTemplates.C* file, change the *methodName* variable to be abe to recognize the name of new *SRKchungTakaStandardChemistryModel* class created so far as follows:

```
...
 76   const word& methodName
 77   (
 78     chemistryTypeDict.lookupOrDefault<word>
 79     (
 80         "method",
 81         chemistryTypeDict.lookupOrDefault<bool>("TDAC", false)
 82         ? "TDAC"
 83         : chemistryTypeDict.lookupOrDefault<bool>("SRKchungTakaStandard"
    , false)
 84         ? "SRKchungTakaStandard"
 85         : "standard"
 86     )
 87   );
 ...
```

Create new macros files to build up chemistry solver models to be valid to real fluid models as follows.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/chemistryModel/
   chemistrySolver/chemistrySolver/
cp -rf makeChemistrySolverTypes.H makeRealFluidChemistrySolverTypes.H
cp -rf makeChemistrySolvers.C makeRealFluidChemistrySolvers.C
```

In *makeRealFluidChemistrySolverTypes.H* file, replace the old definition of macros by the new one as follows:

```
//*---------------------------------------------------------------------*//
#ifndef makeRealFluidChemistrySolverTypes_H
#define makeRealFluidChemistrySolverTypes_H

#include "chemistrySolver.H"
#include "SRKchungTakaStandardChemistryModel.H"
//#include "TDACChemistryModel.H"
#include "noChemistrySolver.H"
#include "EulerImplicit.H"
#include "ode.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#define makeSRKchungTakaChemistrySolverType(SS, Comp, Thermo)      \
                                                                   \
  typedef SS<SRKchungTakaStandardChemistryModel<Comp, Thermo>>     \
  SS##Comp##Thermo;                                                \
                                                                   \
  defineTemplateTypeNameAndDebugWithName                           \
  (                                                                \
    SS##Comp##Thermo,                                              \
    (#SS"<" + word(SRKchungTakaStandardChemistryModel<Comp,        \
    Thermo>::typeName_()) + "<" + word(Comp::typeName_())          \
    + "," + Thermo::typeName() + ">>").c_str(),                    \
    0                                                              \
  );                                                               \
                                                                   \
  BasicChemistryModel<Comp>::                                      \
```

```
    add##thermo##ConstructorToTable<SS##Comp##Thermo>               \
    add##SS##Comp##Thermo##thermo##ConstructorTo                    \
    ##BasicChemistryModel##Comp##Table_;

#define makeSRKchungTakaChemistrySolverTypes(Comp, Thermo)          \
                                                                    \
  makeSRKchungTakaChemistrySolverType                               \
  (                                                                 \
    noChemistrySolver,                                              \
    Comp,                                                           \
    Thermo                                                          \
  );                                                                \
                                                                    \
  makeSRKchungTakaChemistrySolverType                               \
  (                                                                 \
    EulerImplicit,                                                  \
    Comp,                                                           \
    Thermo                                                          \
  );                                                                \
  makeSRKchungTakaChemistrySolverType                               \
  (                                                                 \
    ode,                                                            \
    Comp,                                                           \
    Thermo                                                          \
  );
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#endif
// ************************************************************************* //
```

In *makeRealFluidChemistrySolvers.C* file, replace the old macros by the new ones as follows:

```
//*---------------------------------------------------------------------------*//
#include "makeRealFluidChemistrySolverTypes.H"
```

```
#include "thermoPhysicsTypes.H"
#include "psiReactionThermo.H"
#include "rhoReactionThermo.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
  // Chemistry solvers based on sensibleEnthalpy
  makeSRKchungTakaChemistrySolverTypes(psiReactionThermo,
      chungTakaRealJsrkHThermoPhysics);
  makeSRKchungTakaChemistrySolverTypes(rhoReactionThermo,
      chungTakaRealJsrkHThermoPhysics);

  // Chemistry solvers based on sensibleInternalEnergy
  makeSRKchungTakaChemistrySolverTypes(psiReactionThermo,
      chungTakaRealJsrkEThermoPhysics);
  makeSRKchungTakaChemistrySolverTypes(rhoReactionThermo,
      chungTakaRealJsrkEThermoPhysics);
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

Go to *thermophysicalModels/chemistryModel/Make/* directory:

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/chemistryModel/Make/
```

In *Make/files* file, add *makeRealFluidChemistrySolvers.C* into the list of source files and change the position to save the binary file, *chemistryModel.so*, at user directory.

```
...
chemistrySolver/chemistrySolver/makeRealFluidChemistrySolvers.C //add new
   source file
...
LIB = $(FOAM_USER_LIBBIN)/libchemistryModel  // save at user directory
```

In *Make/options* file, change the path to include the header files of *specie.so*, *fluidThermophysicalModels.so*, and *reactionThermophysicalModels* libraries as follows:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
```

```
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/functions/Polynomial \
    -I$(LIB_SRC)/ODE/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude

LIB_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lreactionThermophysicalModels \
    -lspecie \
    -lODE \
    -lfiniteVolume \
    -lmeshTools
```

Compile to make *chemistryModel.so*.

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/chemistryModel
wclean
wmake libso
```

If errors occur, check carefully your steps again to make sure you do not miss any thing.

2.2.2. Making interface outside of the thermophysicalModels library

TurbulenceModels and *combustionModels* libraries must be updated since they utilize the *thermophysicalModels* library. To update these two libraries, we should copy them from original OpenFOAM into your directory, then modify the path to link to the new *thermophysicalModels* library that has been made as described above.

Go to */yourDirectory/src* directory.

```
cd ~/OpenFOAM/yourDirectory/src
cp -rf -p $WM_PROJECT_DIR/src/TurbulenceModels .
cp -rf -p $WM_PROJECT_DIR/src/combustionModels .
```

Go to */yourDirectory/src/TurbulenceModels/turbulenceModels* directory.

```
cd ~/OpenFOAM/yourDirectory/src/TurbulenceModels/turbulenceModels
```

In *turbulenceModels/Make/files* file, change the position to save the object file, *turbulenceModels.so*, at user directory.

```
...
LIB = $(FOAM_USER_LIBBIN)/libturbulenceModels
```

Compile to make *turbulenceModels.so*.

```
cd ~/OpenFOAM/yourDirectory/src/TurbulenceModels/turbulenceModels
wclean
wmake libso
```

Go to */yourDirectory/src/TurbulenceModels/compressible* directory.

```
cd ~/OpenFOAM/yourDirectory/src/TurbulenceModels/compressible
```

In *compressible/Make/files* file, change the position to save the object file, *compressibleTurbulenceModels.so*, at user directory.

```
...
LIB = $(FOAM_USER_LIBBIN)/libcompressibleTurbulenceModels
```

In *compressible/Make/options* file, change the path to include the header files of *specie.so* and *fluidThermophysicalModels.so* libraries as follows:

```
EXE_INC = \
    -I../turbulenceModels/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/solidThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/solidSpecie/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \

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

```
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lsolidThermo \
    -lsolidSpecie \
    -lturbulenceModels \
    -lspecie \
    -lfiniteVolume \
    -lmeshTools
```

Compile to make *compressibleTurbulenceModels.so*.

```
cd ~/OpenFOAM/yourDirectory/src/TurbulenceModels/compressible
wclean
wmake libso
```

Go to */yourDirectory/src/combustionModels* directory.

```
cd ~/OpenFOAM/yourDirectory/src/combustionModels
```

In *Make/files* file, change the position to save the object file, *combustionModels.so*, at user directory.

```
...
LIB = $(FOAM_USER_LIBBIN)/libcombustionModels
```

In *Make/options* file, change the path to include the header files of *specie.so*, *fluidThermophysicalModels.so*, *chemistryModel.so*, *turbulenceModels.so*, and *compressibleTurbulenceModels.so* libraries as follows:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_REALFLUID_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_REALFLUID_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude
```

```
LIB_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lcompressibleTransportModels \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -lchemistryModel \
    -lfiniteVolume \
    -lmeshTools
```

Compile to make *combustionModels.so*.

```
cd ~/OpenFOAM/yourDirectory/src/combustionModels
wclean
wmake libso
```

2.2.3. Test new created runtime thermo-packages

reactingFoam solver is selected to check the availability of real-fluid runtime *thermo-packages*, in which 2D laminar counterflow non-premixed flame is solved as a test case.

To use *reactingFoam*, we need to link it to the new libraries that we created so far. First, go to *reactingFoam* directory.

```
sol
cd combustion/reactingFoam
```

Modify *Make/options* file as follows:

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude \
    -I$(LIB_REALFLUID_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_REALFLUID_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/basic/lnInclude \
```

```
    -I$(LIB_REALFLUID_SRC)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_SRC)/ODE/lnInclude \
    -I$(LIB_REALFLUID_SRC)/combustionModels/lnInclude

EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -lreactionThermophysicalModels \
    -lspecie \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lchemistryModel \
    -lODE \
    -lcombustionModels
```

Then, recompile it.

```
wclean
wmake
```

Now, *reactingFoam* is ready to use. Go to 2D laminar counterflow flame test case in OpenFOAM.

```
tut
cd combustion/reactingFoam/laminar/counterFlowFlame2D
```

Generate mesh and check with ideal gas models first.

```
blockMesh
reactingFoam
```

It should be executed without error.

Specify the *thermoType* dictionary in *constant/thermophysicalProperties* dictionary file

to use new created runtime *thermo-packages* as follows:

```
thermoType  //This is a new thermo packages we have created so far
{
    type            hePsiThermo;
    mixture         SRKchungTakaReactingMixture;
    transport       chungTaka;
    thermo          rfJanaf;
    energy          sensibleEnthalpy;
    equationOfState soaveRedlichKwong;
    specie          rfSpecie;
}
```

Specify the *chemistryType* dictionary in *constant/chemistryProperties* dictionary file to use new created runtime *thermo-packages* as follows:

```
chemistryType
{
    solver  EulerImplicit;
    method  SRKchungTakaStandard; //new chemistry model for real fluid
}
```

Then, execute the *reactingFoam* again.

```
reactingFoam
```

It should be executed without error. So far, we have successfully created new runtime selectable *thermo-packages* for real-fluid THERMPHYS models. Note that the THERMPHYS models of these new *thermo-packages* here are ideal gas models since we have not implemented the actual real-fluid models yet.

3. Detail implementation of real-fluid models

The details of real-fluid models are described in [1–4]. Since the implemented source code of real-fluid models are too long to be described in details in this document, readers are recommended referring directly to our source code for more convenience. The following classes should be replaced by our source files: *rfSpecie*, *soaveRedlichKwong*, *rfJanafThermo*,

chungTakaTransport, *SRKchungTakaMixture*, and *SRKchungTakaReactingMixture*. It is of importance to note that *SRKchungTakaStandardChemistryModel* is also a new created class but it does not need to be changed so far.

4. Detail modification of some existing classes

There are several groups of classes have to be modified to be compatible with new real-fluid model classes due to the use of templates in OpenFOAM.

4.1. Modification of Transport *classes*

All classes inside *Transport* group involving *constTransport*, *logPolynomialTransport*, *polynomialTransport*, *sutherlandTransport* have to be modified as follows:

In **.H* file, create a new function as:

```
...
public:
...
 Species diffusivity
inline scalar Dimix(label speciei, const scalar p, const scalar T) const
{
    return 1.0;
}
...
```

Goto *thermophysicalModels/specie* and compile this library again after modifying the code:

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/specie
wmake libso
```

If errors occur, check carefully your steps again to make sure you do not miss any thing.

4.2. Modification of Mixture *classes*

In *thermophysicalModels/reactionThermo/mixtures* directory, modify *singleComponentMixture*, *egrMixture*, *homogeneousMixture*, *inhomogeneousMixture*, and *veryInhomogeneousMixture* classes as follows:

In **.H* file:

```
#include "PtrList.H"

private:
    PtrList<ThermoType> speciesData_;

protected:
    label numberOfSpecies_;
    List<word> ListSpeciesName_;
...

public:
...
    const PtrList<ThermoType>& speciesData()
    {
        return speciesData_;
    }

    const List<word>& ListSpeciesName()
    {
        return ListSpeciesName_;
    }

    inline const label& numberOfSpecies() const
    {
        return numberOfSpecies_;
    }
...
```

In **.C* file:

```
// In constructor:
:
...
ListSpeciesName_(1)
```

```
{
...
    speciesData_(1);
    speciesData_.set
       (
          0,
          new ThermoType("mixture", thermoDict.subDict("mixture"))
       );
       // note that subDict's name here differs from class by class.
       // Particularly, it is:
       // "mixture" in singleComponentMixture class;
       // "fuel" in egrMixture class;
       // "reactants" in homogeneousMixture class;
       // "fuel" in inhomogeneousMixture class;
       // "fuel" in veryInhomogeneousMixture class;
    numberOfSpecies_ = speciesData_.size();
    ListSpeciesName_[0] = "mixture";
...
}
```

In *thermophysicalModels/reactionThermo/mixtures* directory, modify the *multiComponentMixture* class as follows:

In *multiComponentMixture.H* file:

```
...
protected:
    label numberOfSpecies_;
    List<word> ListSpeciesName_;
...

public:
...
    const List<word>& ListSpeciesName()
    {
```

```
        return ListSpeciesName_;
    }

    inline const label& numberOfSpecies() const
    {
        return numberOfSpecies_;
    }
...
```

In *multiComponentMixture.C* file:

```
// In constructor:
template<class ThermoType>
Foam::multiComponentMixture<ThermoType>::multiComponentMixture
(
    const dictionary& thermoDict,
    const wordList& specieNames,
    const HashPtrTable<ThermoType>& thermoData,
    const fvMesh& mesh,
    const word& phaseName
)
:
    basicSpecieMixture(thermoDict, specieNames, mesh, phaseName),
    speciesData_(species_.size()),
    mixture_("mixture", *thermoData[specieNames[0]]),
    mixtureVol_("volMixture", *thermoData[specieNames[0]]),
    //
    numberOfSpecies_(species_.size()),
    ListSpeciesName_(species_.size())
    //
{
    forAll(species_, i)
    {
        speciesData_.set
        (
```

```
            i,
            new ThermoType(*thermoData[species_[i]])
        );
    }
    //
    forAll(ListSpeciesName_, i)
    {
        ListSpeciesName_[i] = speciesData_[i].name();
    }
    //
    correctMassFractions();
}
```

In *thermophysicalModels/basic/mixtures* directory, modify the *pureMixture* class as follows:

In *pureMixture.H* file:

```
#include "PtrList.H"
#include "volFields.H"

private:
PtrList<ThermoType> speciesData_;
PtrList<volScalarField> Y_;

protected:
    label numberOfSpecies_;
    List<word> ListSpeciesName_;

public:
...
const PtrList<ThermoType>& speciesData()
{
    return speciesData_;
}
```

```
inline PtrList<volScalarField>& Y()
{
    return Y_;
}

inline const PtrList<volScalarField>& Y() const
{
    return Y_;
}

const List<word>& ListSpeciesName()
{
    return ListSpeciesName_;
}

inline const label& numberOfSpecies() const
{
    return numberOfSpecies_;
}
...
```

In *pureMixture.C* file:

```
// In constructor
template<class ThermoType>
Foam::pureMixture<ThermoType>::pureMixture
(
    const dictionary& thermoDict,
    const fvMesh& mesh,
    const word& phaseName
)
:
 basicMixture(thermoDict, mesh, phaseName),
 mixture_(thermoDict.subDict("mixture")),
 ListSpeciesName_(1)
```

```
{
    speciesData_(1);
    speciesData_.set
    (
        0,
        new ThermoType(thermoDict.subDict("mixture"))
    );
    numberOfSpecies_ = speciesData_.size();
    ListSpeciesName_[0] = "mixture";

    Y_(1);
    tmp<volScalarField> tYdefault;

    forAll(Y_, i)
    {
        IOobject header
        (
            IOobject::groupName("Y", phaseName),
            mesh.time().timeName(),
            mesh,
            IOobject::NO_READ
        );

        // check if field exists and can be read
        if (header.typeHeaderOk<volScalarField>(true))
        {
            Y_.set
            (
                i,
                new volScalarField
                (
                    IOobject
                    (
                        IOobject::groupName("Y", phaseName),
```

```
                    mesh.time().timeName(),
                    mesh,
                    IOobject::MUST_READ,
                    IOobject::AUTO_WRITE
                ),
                mesh
            )
        );
    }
    else
    {
        // Read Ydefault if not already read
        if (!tYdefault.valid())
        {
            word YdefaultName(IOobject::groupName("Ydefault",
                phaseName));

            IOobject timeIO
            (
                YdefaultName,
                mesh.time().timeName(),
                mesh,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            );

            IOobject constantIO
            (
                YdefaultName,
                mesh.time().constant(),
                mesh,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            );
```

```
            IOobject time0IO
            (
                YdefaultName,
                Time::timeName(0),
                mesh,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            );

            if (timeIO.typeHeaderOk<volScalarField>(true))
            {
                tYdefault = new volScalarField(timeIO, mesh);
            }
            else if (constantIO.typeHeaderOk<volScalarField>(true))
            {
                tYdefault = new volScalarField(constantIO, mesh);
            }
            else
            {
                tYdefault = new volScalarField(time0IO, mesh);
            }
        }

        Y_.set
        (
            i,
            new volScalarField
            (
                IOobject
                (
                    IOobject::groupName("Y", phaseName),
                    mesh.time().timeName(),
                    mesh,
```

```
                        IOobject::NO_READ,
                        IOobject::AUTO_WRITE
                    ),
                    tYdefault()
                )
            );
        }
    }
}
```

Goto *thermophysicalModels/basic* and compile this library again after modifying the code:

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/basic
wmake libso
```

If errors occur, check carefully your steps again to make sure you do not miss any thing.

4.3. Modification of BasicThermo *classes*

In *psiThermo* and *rhoThermo* class, add these following public virtual functions into **.H* files. These functions will be overridden by functions in their child classes due to polymorphism.

```
...
//- Diffusion coefficient of specie ith in the mixture [m^2/s]
virtual tmp<volScalarField> Dimix(const label speciei) const = 0;

//- Diffusion coefficient of specie ith in the mixture for patch [m^2/s]
virtual tmp<scalarField> Dimix(const label speciei, const label patchi)
   const = 0;

//- New functions for realFluidRhoReactingFoam solver
//- Enthalpy/Internal energy of specie ith [J/kg]
virtual tmp<volScalarField> hei(label speciei) const = 0;
```

```
//- Enthalpy/Internal energy of specie ith for patch [J/kg]
virtual tmp<scalarField> hei(label speciei, const label patchi) const = 0;

//- Molecular weight of individual specie ith [kg/kmol]
virtual tmp<volScalarField> Wi(label speciei) const = 0;

//- Molecular weight of individual specie ith for patch [kg/kmol]
virtual tmp<scalarField> Wi(label speciei, const label patchi) const = 0;

//- return name of species from index
virtual const List<word>& ListOfSpeciesName() const  = 0;
...
```

4.4. Modification of Type *classes*

In *heThermo* class, add a these following code.

In *heThermo.H* file:

```
...
protected:
//- return list of species name
List<word> ListSpeciesName_;
//- Store List of energy field of indivisual species
PtrList<volScalarField> heList_;
//- Store list of molecular weight field of indivisual species
PtrList<volScalarField> WList_;

public:
...
// Access to thermodynamic state variables
  //- New functions for realFluidReactingFoam solver
    //- Return he[J/kg] of indivisual speciei
    virtual tmp<volScalarField> hei(label speciei) const;
    //- Return he[J/kg] of indivisual speciei
    virtual tmp<scalarField> hei(label speciei, const label patchi) const;
    //- Return molecular weight Wi[kg/kmol] of indivisual speciei
```

```
    virtual tmp<volScalarField> Wi(label speciei) const;
    //- Return molecular weight Wi[kg/kmol] of indivisual speciei
    virtual tmp<scalarField> Wi(label speciei, const label patchi) const;
...
//- return list of species name
virtual const List<word>& ListOfSpeciesName() const
{
   return ListSpeciesName_;
}
...
```

In *heThermo.C* file:

```
// In init() function
template<class BasicThermo, class MixtureType>
void Foam::heThermo<BasicThermo, MixtureType>::init()
{
...
    //- Calculate he_i, Wi internal fields for individual species
    const PtrList<typename MixtureType::thermoType>& speciesData_ = this->
       speciesData();

    forAll(heList_, i)
    {
        forAll(heList_[i].primitiveFieldRef(), celli)
        {
            heList_[i].primitiveFieldRef()[celli]
          = speciesData_[i].HE(pCells[celli], TCells[celli]);

            WList_[i].primitiveFieldRef()[celli]
          = speciesData_[i].W();
        }
    }
    //
```

```
    //- Calculate he_i, Wi boundary fiels for individual species
    forAll(WList_, i)
    {
        forAll(WList_[i].boundaryFieldRef(), patchi)
        {
           forAll(WList_[i].boundaryFieldRef()[patchi], facei)
           {
               WList_[i].boundaryFieldRef()[patchi][facei]
             = speciesData_[i].W();

               heList_[i].boundaryFieldRef()[patchi][facei]
             = speciesData_[i].HE
               (
                   this->p_.boundaryField()[patchi][facei],
                   this->T_.boundaryField()[patchi][facei]
               );
           }
        }
    }
    //
}


// In both constructor functions
...
:
...
ListSpeciesName_(MixtureType::numberOfSpecies()),
heList_(MixtureType::numberOfSpecies()),
WList_(MixtureType::numberOfSpecies())
{
    forAll(heList_, i)
    {
        heList_.set
```

```
        (
            i,
            new volScalarField
            (
                IOobject
                (
                    "hei",
                    mesh.time().timeName(),
                    mesh,
                    IOobject::NO_READ,
                    IOobject::NO_WRITE
                ),
                mesh,
                he_.dimensions()
            )
        );
    }

    forAll(WList_, i)
    {
        WList_.set
        (
            i,
            new volScalarField
            (
                IOobject
                (
                    "W",
                    mesh.time().timeName(),
                    mesh,
                    IOobject::NO_READ,
                    IOobject::NO_WRITE,
                    false
                ),
```

```
                mesh,
                dimMass/dimMoles
            )
        );
    }

    List<word> ListSpeciesName = MixtureType::ListSpeciesName();
    forAll(ListSpeciesName_, i)
    {
        ListSpeciesName_[i] = ListSpeciesName[i];
    }
...
}


//- New functions for realFluidReactingFoam solver
//- Return he_i of individual species ith
template<class BasicThermo, class MixtureType>
Foam::tmp<Foam::volScalarField>
Foam::heThermo<BasicThermo, MixtureType>::hei
(
    label speciei
) const
{
    return heList_[speciei];
}

template<class BasicThermo, class MixtureType>
Foam::tmp<Foam::scalarField>
Foam::heThermo<BasicThermo, MixtureType>::hei
(
    label speciei,
    const label patchi
) const
```

```
{
    return heList_[speciei].boundaryField()[patchi];
}

//- Return molarcular weight W_i of individual species ith
template<class BasicThermo, class MixtureType>
Foam::tmp<Foam::volScalarField>
Foam::heThermo<BasicThermo, MixtureType>::Wi
(
    label speciei
) const
{
    return WList_[speciei];
}

template<class BasicThermo, class MixtureType>
Foam::tmp<Foam::scalarField>
Foam::heThermo<BasicThermo, MixtureType>::Wi
(
    label speciei,
    const label patchi
) const
{
    return WList_[speciei].boundaryField()[patchi];
}
```

In *hePsiThermo* class, add a these following code. In *hePsiThermo.H* file:

```
private:
//- Store list of mass diffusion coefficients
PtrList<volScalarField> Dimix_;
...

public:
...
```

```
//- Diffusion coefficient of specie ith in the mixture [m^2/s]
virtual tmp<volScalarField> Dimix(const label speciei) const
{
    return Dimix_[speciei];
}

//- Diffusion coefficient of specie ith in the mixture for patch [m^2/s]
virtual tmp<scalarField> Dimix(const label speciei, const label patchi)
   const
{
    return Dimix_[speciei].boundaryField()[patchi];
}
...
```

In *hePsiThermo.C* file:

```
// In constructor function
:
...
Dimix_(MixtureType::numberOfSpecies())
{
    forAll(Dimix_, i)
    {
        Dimix_.set
        (
            i,
            new volScalarField
            (
                IOobject
                (
                    this->phasePropertyName("thermo:Dimix"),
                    mesh.time().timeName(),
                    mesh,
                    IOobject::NO_READ,
                    IOobject::NO_WRITE
```

```
                ),
                mesh,
                dimensionSet(0, 2, -1, 0, 0)
            )
        );
    }
...
}

//In calculate() function
...
const PtrList<typename MixtureType::thermoType>& speciesData_ = this->
   speciesData();
forAll(TCells, celli)
{
...
    forAll(Dimix_, i)
    {
        Dimix_[i].primitiveFieldRef()[celli]
      = mixture_.Dimix(i, pCells[celli], TCells[celli]);
    }
}
...
forAll(this->T_.boundaryField(), patchi)
{
    ...
    if (pT.fixesValue())
    {
        forAll(pT, facei)
        {
            ...
            forAll(Dimix_, i)
            {
                Dimix_[i].boundaryFieldRef()[patchi][facei]
```

```
                = mixture_.Dimix(i, pp[facei], pT[facei]);
            }
        }
    }
    else
    {
        forAll(pT, facei)
        {
        ...
            forAll(Dimix_, i)
            {
                Dimix_[i].boundaryFieldRef()[patchi][facei]
                = mixture_.Dimix(i, pp[facei], pT[facei]);
            }
        }
    }
}
```

In *heRhoThermo* class, add a these following code. In *heRhoThermo.H* file:

```
private:
//- Store list of mass diffusion coefficients
PtrList<volScalarField> Dimix_;
...

public:
...
//- Diffusion coefficient of specie ith in the mixture [m^2/s]
virtual tmp<volScalarField> Dimix(const label speciei) const
{
    return Dimix_[speciei];
}

//- Diffusion coefficient of specie ith in the mixture for patch [m^2/s]
virtual tmp<scalarField> Dimix(const label speciei, const label patchi)
```

```
    const
{
    return Dimix_[speciei].boundaryField()[patchi];
}
...
```

In *heRhoThermo.C* file:

```
// In constructor
:
...
Dimix_(MixtureType::numberOfSpecies())
{
    forAll(Dimix_, i)
    {
        Dimix_.set
        (
            i,
            new volScalarField
            (
                IOobject
                (
                    this->phasePropertyName("thermo:Dimix"),
                    mesh.time().timeName(),
                    mesh,
                    IOobject::NO_READ,
                    IOobject::NO_WRITE
                ),
                mesh,
                dimensionSet(0, 2, -1, 0, 0)
            )
        );
    }
...
}
```

```
//In calculate() function
...
const PtrList<typename MixtureType::thermoType>& speciesData_ = this->
    speciesData();
//Diffusion coefficient has not been implemented yet for rho-based systems
    .
forAll(TCells, celli)
{
...
    forAll(Dimix_, i)
    {
        Dimix_[i].primitiveFieldRef()[celli] = 1.0;
    }
...
}
...
forAll(this->T_.boundaryField(), patchi)
{
    ...
    if (pT.fixesValue())
    {
        forAll(pT, facei)
        {
            ...
            forAll(Dimix_, i)
            {
                Dimix_[i].boundaryFieldRef()[patchi][facei] = 1.0;
            }
        }
    }
    else
    {
        forAll(pT, facei)
```

```
        {
        ...
            forAll(Dimix_, i)
            {
                Dimix_[i].boundaryFieldRef()[patchi][facei] = 1.0;
            }
        }
    }
}
```

In *thermophysicalModels/reactionThermo/psiuReactionThermo* directory, modify *heheuPsiThermo* class as follows: In *heheuPsiThermo.H* file:

```
private:
//- Store list of mass diffusion coefficients
PtrList<volScalarField> Dimix_;
...
public:
...
//- Diffusion coefficient of specie ith in the mixture [m^2/s]
virtual tmp<volScalarField> Dimix(const label speciei) const
{
    return Dimix_[speciei];
}

//- Diffusion coefficient of specie ith in the mixture for patch [m^2/s]
virtual tmp<scalarField> Dimix(const label speciei, const label patchi)
   const
{
    return Dimix_[speciei].boundaryField()[patchi];
}
...
```

In *heheuPsiThermo.C* file:

```
// In constructor
```

```
:
...
Dimix_(MixtureType::numberOfSpecies())
{
...
    this->heuBoundaryCorrection(this->heu_);
    //
    forAll(Dimix_, i)
    {
        Dimix_.set
        (
            i,
            new volScalarField
            (
                IOobject
                (
                    this->phasePropertyName("thermo:Dimix"),
                    mesh.time().timeName(),
                    mesh,
                    IOobject::NO_READ,
                    IOobject::NO_WRITE
                ),
                mesh,
                dimensionSet(0, 2, -1, 0, 0)
            )
        );
    }
    //
    calculate();
...
}

//In calculate() function
...
```

```
//Diffusion coefficient has not been implemented yet for psiu-based
    systems.
const PtrList<typename MixtureType::thermoType>& speciesData_ = this->
    speciesData();
forAll(TCells, celli)
{
...
    forAll(Dimix_, i)
    {
        Dimix_[i].primitiveFieldRef()[celli] = 1.0;
    }
}
...
forAll(this->T_.boundaryField(), patchi)
{
    ...
    if (pT.fixesValue())
    {
        forAll(pT, facei)
        {
            ...
            forAll(Dimix_, i)
            {
                Dimix_[i].boundaryFieldRef()[patchi][facei] = 1.0;
            }
        }
    }
    else
    {
        forAll(pT, facei)
        {
        ...
            forAll(Dimix_, i)
            {
```

```
                Dimix_[i].boundaryFieldRef()[patchi][facei] = 1.0;
            }
        }
    }
}
```

Compile all libraries again after implementing real-fluid models and modifying some existing classes as mentioned above as follows (in order):

```
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/basic
wmake libso
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/reactionThermo
wmake libso
cd ~/OpenFOAM/yourDirectory/src/thermophysicalModels/chemistryModel
wmake libso
cd ~/OpenFOAM/yourDirectory/src/TurbulenceModels/turbulenceModels
wmake libso
cd ~/OpenFOAM/yourDirectory/src/TurbulenceModels/compressible
wmake libso
cd ~/OpenFOAM/yourDirectory/src/combustionModels
wmake libso
```

If there is no error, the real-fluid based *thermophysicalModels* now is ready to use. It is of importance to note that any solver that utilize *thermophysicalModels*, *compressibleTurbulenceModels*, and *combustionModels* should be recompiled after implementing real-fluid based *thermophysicalModels*.

5. Using the new library

The new library can be used for any reacting flow solver in OpenFOAM 6.0 that adopt a set of implemented real-fluid models by using either *psiReactionThermo* or *rhoReactionThermo* classes. To use these classes, their header files (*psiReactionThermo.H* and *rhoReactionThermo.H*) should be included in a source file of the solver. When an object of these classes is created, we can call their functions to return the corresponding thermophysical

properties. The following piece of code demonstrates how to utilize the new library using the *psiReactionThermo* class in *realFluidReactingFoam* developed from *reactingFoam*:

```
//Create an object named thermo of psiReactionThermo type.
autoPtr<psiReactionThermo> pThermo(psiReactionThermo::New(mesh));
psiReactionThermo& thermo = pThermo();
...
// call functions to return THERMPHYS properties
thermo.rho();     // return the density field
thermo.Dimix(i);  // return the mass diffusivity of specie ith
```

In the running case directory, the *thermoType* dictionary needs to be specified in the *constant/thermophysicalProperties* file as follows:

```
thermoType
{
    type            hePsiThermo;      // heRhoThermo for rho-based system.
    mixture         SRKchungTakaReactingMixture;
    transport       chungTaka;
    thermo          rfJanaf;
    energy          sensibleEnthalpy; //or sensibleInternalEnergy
    equationOfState soaveRedlichKwong;
    specie          rfSpecie;
}
```

In the running case directory, the *chemistryType* dictionary needs to be specified in the *constant/chemistryProperties* file as follows:

```
chemistryType
{
    solver   EulerImplicit; //or "ode" or "none"
    method   SRKchungTakaStandard;
}
```

Some input data for real-fluid calculations also need to be specified at *dataForRealFluid* entry in the *constant/thermo.compressibleGas* dictionary file for each species in the following format:

```
dataForRealFluid    // of Oxygen species
{
    Tc      154.58;  // the critical temperature, K
    Pc      5.043;   // the critical pressure, MPa
    Vc      73.529;  // the critical volume, cm^3/mol
    omega   0.025;   // the dimensionless acentric factor
    kappai  0.0;     // the dimensionless association factor
    miui    0.0;     // the dimensionless dipole moment
    sigmvi  16.6;    // the dimensionless diffusion volume
}
```

Readers are referred to [7] to find input data of other species, and referred to the source code of the *realFluidReactingFoam* and tutorials for test cases provided in the our repository for better understanding of using the new library in a solver.

The original *reactingFoam* solver can be used for testing real fluid implementation processes as mentioned in sec-2.2.3 but the result is not guaranteed since its governing equations have not been validated for real fluid models yet. We highly recommend readers using provided *realFluidReactingFoam* solver. It is a quasi direct numerical simulation (quasi-DNS) solver developed based on a work of Li et al. [8] in which its governing equations have been validated at low pressure. In the current work, we have validated for laminar reacting flows under both low pressure and supercritical conditions, up to 200 atm (see our paper for details of the validation). This solver will be updated and validated for reacting turbulence flows in the future.

References

[1] G. Soave, Equilibrium constants from a modified Redlich-Kwong equation of state, Chem. Eng. Sci. 27 (1972) 1197–1203.

[2] M. S. Graboski, T. E. Daubert, A modified Soave equation of state for phase equilibrium calculations. 1. Hydrocarbon systems, Ind. Eng. Chem. Process. Des. Dev. 17 (1978) 443–448.

[3] T. C. Horng, M. Ajlan, L. L. Lee, K. E. Starling, M. Ajlan, Generalized multiparameter correlation for nonpolar and polar fluid transport properties, Ind. Eng. Chem. Res. 27 (1988) 671–679.

[4] S. Takahashi, S. Takahashi, Preparation of a generalized chart for the diffusion coefficients of gases at high pressures, J. Chem. Eng. Japan 7 (1975) 417–420.

[5] D. Peng, D. Robinson, New two-constant equation of state, Ind. Eng. Chem. Fundam. 15 (1976) 59–64.

[6] R. J. Kee, F. M. Rupley, E. Meeks, J. A. Miller, CHEMKIN-III: a fortran chemical kinetics package for the analysis of gas-phase chemical and plasma kinetics, SAND96-8216 (1996).

[7] B. E. Poling, J. M. Prausnitz, J. P. O'Connell, The properties of gases and liquids, McGraw-Hill, 2001.

[8] T. Li, J. Pan, F. Kong, B. Xu, X. Wang, A quasi-direct numerical simulation solver for compressible reacting flows, Comput. Fluids 213 (2020) 104718.