PRIMAT *(PRImordial MATter)*

PRIMAT is a *Mathematica* code which computes the abundances of elements at the end of the big-bang nucleosynthesis (BBN).
It can be downloaded by registering at http://www2.iap.fr/users/pitrou/primat.htm

Version 0.3.0 (01/10/2024)

Information

Short description

*The implementation follows the presentation of the companion paper Pitrou, Coc, Uzan, Vangioni, Physics Reports, 04, (2018) 005.
All equation numbers, when non specified, refer to this companion paper, in its arXiv version (arXiv:1801.08023).

*It is based on a previous Fortran code written by Alain Coc.

*The user can modify several parameters which are in the preamble of the code :

a) The nuclear reaction rates involved in the nuclear reaction network are tabulated in an external file which can be easily modified.
b) The corrections to the weak-interaction reactions (n + ν <-> p^+ + e^- and its related reactions), can be turned on and off with booleans.
The details of these corrections are given in the companion paper.
c) Cosmological parameters, that is essentially the number of baryons (Ω_b) and the number of neutrino generations N_ν can also be modified.

*The numerical integration is performed in two steps. First the cosmology and the thermodynamics of the plasma are solved,
and then the nuclear reactions are computed, the backreaction of the latter on the former being negligible (see companion paper).

*The results are given as a set of interpolating functions of time for the abundances of individual species.
If one is interested in final abundances only, these are also directly accessible by evaluating these functions at final time.

*A very basic Monte-Carlo exploration of uncertainties is provided at the end of the code. It is

used in associated example notebooks located in the 'Example' folder.
For each reaction, an uncertainty factor variance is provided and it is possible to run the code with these uncertainty parameters taking random values according to the variances (see [Coc et al. 2014]). A parallelization is possible for this Monte-Carlo exploration and the analysis of the results can be output in an external file and analyzed separately.

*Several examples and applications are gathered in the 'Examples' folder.

*Neutrino decoupling is handled using output files generated by the (non-public) Python code NEVO [Froustey et al. 2020].

*An option allows to use only a reduced network of equations (weak interactions and 12 nuclear reactions), resulting in much faster results, though slightly different for Li7 abundances.

*** BASIC USAGE ***

*In the Evaluation menu, select 'Evaluate notebook'. If asked the question 'evaluate initialization cells first ?', answer No.
Then Mathematica proceeds in evaluating all cells, in order, and it should reach the end of the notebook (with the plots and results) in less than one minute.

*If the user erases the precomputed weak rates which are precomputed and stored in the 'Interpolation' folder, or asks for these rates to be re-precomputed, this can take considerably longer, typically one hour.

*Furthermore, when PRIMAT-Main.nb (this notebook) is opened and saved in Mathematica, the cells which are' initialization cells' are saved into the file PRIMAT-Main.m.
This *.m file contains all the important definitions, functions and variables which can be loaded from another notebook to perform BBN computations and analysis of results.
The 'Examples' Folder contains several typical applications which work exactly like that. First it loads all necessary definitions stored in PRIMAT-Main.m and then it performs a few useful computations for each selected example.

Authors

This code is written and maintained by Cyril Pitrou[1] in collaboration with Alain Coc[2], J.-P. Uzan[3] and E. Vangioni[4].
Neutrino decoupling computations have been performed by Julien Froustey[5] with NEVO (non-public Python code).

[1,3,4,5] *Institut d'Astrophysique de Paris* (CNRS)
98 bis Boulevard Arago
75014 Paris, France

[2]former *CSNSM* (CNRS, IN2P3)
Orsay, France

emails and homepages :
-pitrou@iap.fr, http://www2.iap.fr/users/pitrou
-uzan@iap.fr
-vangioni@iap.fr
-alain.coc@csnsm.in2p3.fr
-froustey@iap.fr, http://www.iap.fr/useriap/froustey

Dates and versions

In V0.2.0:
-Reaction **Be7(d,p)** is now tabulated in this external file and not given analytically and is taken from [Rij19]
-Decoupling of neutrinos with oscillations (precomputed with NEVO, see [Froustey et al. 2020]).
-Constants reevaluated with Particle Data Group 2020 data. Main change is G_{Newt}.
-QED corrections (pair production) to some nuclear rates included (see [Pitrou&Pospelov 2020]).
-An option to use only reduced nuclear network has been added.
-QED at order (e^3) included in plasma thermodynamics (reduces Neff by 0.001, see [Bennett et al. 2019])
-Cosmological parameters are set to the last column of table 2 of [Planck 2018 VI].

In V0.2.1:
-New dpg rate (reaction **d + p <-> He3 + g**) in new reaction file BBNRatesAC2022.dat.
Rate taken from Moscoso et al. 2021

In V0.2.2:
-small modification to make it compatible with Mathematica 13.
-neutron lifetime update with PDG 2023.

In V0.2.3:
- Possibility to add distortions in an analytic form (with all the caveats due to the fact that neutrino self scatter and scatter of electrons/positrons and cannot tolerate spectral distortions for T>2MeV in principle).
- 'Nrelat' parameter added (number of relativistic decoupled species). So as to separate neutrinos from other thermalized but decoupled relativistic degrees of freedom.
- Switched the mass database from nubase2016 to nubase2020
- Values of physical constants from PDG 2024

In V0.3.0
- PRIMAT now looks more like a package even though it is not yet a Package.
When loading PRIMAT-Main.m from an external notebook, this now only loads definitions (and

takes 0.3 seconds), but no computation is performed. The user can then modify the various options about physics or numerics, and then call for the function which either perform one run (RunNumericalIntegrals) or a MonteCarlo estimation of errors (RunPRIMATMonteCarlo), and then read the results. Examples are provided in the Example folder. It has been tested under a variety of combination of options, but please reports situations in which this still generates bugs.
- It now takes less than a second for the small network of reactions (12 reactions) and 16 seconds for the full network (434 reactions), provided the weak rates are precomputed and stored in the CSV folder (which takes place the first time).
- There is a PythonInterface folder which explains how to call PRIMAT from python and how to read its results and modify its options. See readme file in that folder.
- The most crucial functions for performing numerical integration are compiled and stored on disk. Hence after the first run, the loss in compiling these functions is avoided by reading the compiled functions which are stored in the lib folder.
- A *$Parthenope* option allows to replace the rates **d+d <-> He3+n** (ddn) and **d+d <-> t+p** (ddp) with the ones used in Parthenope3.0 (Gariazzo et al. 2021).
- The names of the folders in PRIMAT have changed.
- Tested on Mac OS and with Mathematica 14 only.
- Rate for **d+a <-> Li6 + g** replaced by [Trezzi2017]. New reaction file BBNRatesAC2024.dat.

In[•]:=
```
Date[]
```
date

Out[•]=
```
{2024, 10, 5, 2, 9, 45.553047}
```

Mathematica version used :

In[•]:=
```
$Version
```
version actuelle

Out[•]=
```
14.1.0 for Mac OS X ARM (64-bit) (July 16, 2024)
```

GPL

In[ ]:=
```
(* Copyright (C) 2018- Cyril Pitrou, Alain Coc *)

(* This program is free software; you can redistribute it and/or
  modify it under the terms of the GNU General Public License as

  published by the Free Software Foundation; either version 2 of
  the License,or (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License

  along with this program; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place-Suite 330, Boston, MA 02111-1307,
  USA.
*)
```

Bibliography

Hereafter we use the following shorthands for the references cited.

BBN review with PRIMAT description

[Pitrou et al. 2018] C. Pitrou, A. Coc, J.-P. Uzan, E. Vangioni, Physics Reports, 04, (2018) 005, 1801.08023.
[Pitrou et al. 2021] C. Pitrou, A. Coc, J.-P. Uzan, E. Vangioni, MNRAS, ... , 2011.11320.

BBN references

[Bennett et al. 2019] J. J. Bennett, G. Buldgen, M. Drewes, Y. Y. Y. Wong, JCAP **03** (2020) 003, 1911.04504 .
[Bernstein 1989] J. Bernstein, L. S. Brown, G. Feinberg, Rev. Mod. Phys **61** (1989).
[Brown&Sawyer] L. S. Brown, R. F, Sawyer, Phys. Rev. **D63**, 083503 (2001) astro-ph/0006370.
[Cambier et al.] J. L. Cambier, J. R. Primack, and M. Sher, Nucl. Phys. **B209**, 372 (1982).
[Coc et al. 2012] A. Coc, S. Goriely, Y. Xu, M. Saimpert and E. Vangioni, Astrophys. J. **744** (2012) 1107.1117.
[Coc et al. 2014] A. Coc, J.-P. Uzan and E. Vangioni JCAP 1410 (2014) 050 1403.6694.
[Czarnecki et al. 2004] Czarnecki, Marciano and Sirlin, Phys Rev. D **70**, 093006 hep-ph/0406324.
[Dicus et al.] D. A. Dicus, E. W. Kolb, A. M. Gleeson, E. C. G. Sudarshan, V. L. Teplitz, and M. S. Turner, Phys. Rev. **D26**, 2694 (1982).

[Esposito et al. 1999] S. Esposito, G. Mangano, G. Miele and O. Pisanti, Nucl. Phys. **B540**, 3 (1999) astro-ph/9808196.
[Froustey&Pitrou 2019] J. Froustey, C. Pitrou, Phys. Rev. **D101** (2020) 043524, 1912.09378
[Grohs et al. 2012] E. Grohs, G. M. Fuller, C. T. Kishimoto, M. W. Paris, and A. Vlasenko, astro-ph/1512.02205.
[Horowitz] astro-ph/0109209 (for weak-magnetism definition and related computations).
[Horowitz&Li] C.J. Horowitz, G. Li, astro-ph/9908219
[Heckler 1994] A. F. Heckler, Phys. Rev. **D49** 611 (1994).
[Ivanov 2012] A. N. Ivanov, M. Pitschmann, N. I. Troitskaya Phys. Rev. **D88**, 073002 (2013) hep-ph/1212.0332.
[Kernan] P. J. Kernan, PhD thesis, *Two astroparticle physics problems; solar neutrinos, and primordial ^4He*. Ohio State University (1993).
[Lopez&Turner 1998] R. E. Lopez and M. S. Turner, Phys. Rev. **D59**, 103502 (1999) astro-ph/9807279.
[Lopez et al. 1997] R. E. Lopez, M. S. Turner and G. Gyuk, Phys. Rev. **D56**, 3191 (1997), astro-ph/9703065.
[Mangano et al. 2001] G. Mangano, G. Miele, S. Pastor, M. Peloso, Phys.Lett. **B534** (2002) 8-16, astro-ph/0111408.
[Mangano et al. 2005] G. Mangano, G. Miele, S. Pastor, T. Pinto, O. Pisanti, P. D. Serpico, Nucl.Phys. **B729** (2005) 221-234 hep-ph/0506164.
[PArthENoPE] O. Pisanti et al. Comp. Phys. Com **178** 956 (2008), 0705.0290. Update 2017: arXiv:1712.04378.
[Pitrou&Pospelov 2020] C. Pitrou and M. Pospelov, Phys Rev. C. **102** (2020) 015803, 1904.07795 .
[Seckel 1993] D. Seckel, hep-ph/9305311.
[Sirlin 1967] A. Sirlin Phys. Rev. 164, Vol **5**, p1767 (1967).

Other references. Incomplete neutrino decoupling :

[Froustey et. al. 2020] J. Froustey, C. Pitrou, C. Volpe, JCAP **12** (2020) 015, 2008.01074.
[Hannestad 1995], S. Hannestad, J. Madsen, Phys. Rev. **D52**, 1754 (1995), astro-ph/9506015.
[Gnedin 1997] N. Y. Gnedin, O. Y. Gnedin, AP. J. **509**, 11-15 (1998), astro-ph/9712199.

Reaction rates

They are listed in references of [Coc. et. al 2012] (see Table 4). We use the following acronyms :

NACRE (Angulo et al. 1999)
NACRE II (Xu et al. 2010, 2011)
DAACV04 (Descouvemont et al. 2004)
ILCCF10 (Iliadis et al. 2010)
CF88 (Caughlan& Fowler 1988)
MF89 (Malaney& Fowler 1989)
Boy93 (Boyd et al. 1993)

Bal95 (Balbes et al. 1995)
Hei98 (Heil et al. 1998)
Rau94 (Rauscher et al. 1994)
Des99 (Descouvemont 1999)
Bea01 (Beaumel et al. 2001)
Tan03 (Tang et al. 2003)
Wan91 (Wang et al. 1991)
Efr96 (Efros et al. 1996)
Wie87 (Wiescher et al. 1987 Wiescher,Harms,Goerres,Thielemann& Rybarcyk ApJ 316 (1997) 162 1001.2053)
Bar97 (Bardayan& Smith 1997)
Koe91 (Koehler& Graff 1991)
And06 (Ando et al. 2006)
Ser04 (Serpico et al. 2004)
Wag69 (Wagoner 1969)
Has09 (Hashimoto et al.2009)
Wie89 (Wiescher et al.1989)
FK90 (Fukugita& Kajino 1990)
Bru91 (Brune et al.1991)
Bec92 (Becchetti et al.1992)
Iga95 (Igashira et al.1995)
Cyb08 (Cyburt& Davids 2008)
Miz00 (Mizoi et al. 2000)
Nag06 (Nagai et al. PRC 74 (2006) 025804 AC2010)
Has09c (Hashimoto et al.PLB 674 (2009) 27)
FK90 (Fukugita& Kajino,PRD 42 (1990) 4251)
Rau94 (C. Rauscher et al.ApJ 429 (1994) 499)
Men12 (C Mendes et al.PRC 86 (2012) 064321)
Tang03 (Tang et al.PRC 67 (2003) 015804)
Bar97C (Bardayan& Smith PRC 56 (1997) 1647)
Kaw91 (Kawano,Fowler,Kavanagh,Malaney ApJ 372 (1991) 1-7)
Cam08 (Camargo et al.Phys.Rev.C 78,034605 (2008))
Ili16 (Iliadis et al. 2016)
Rij19 (Rijal et al. Phys. Rev. Lett. 122, 182701 (2019))
Gar21 (Gariazzo et al. 2021 Parthenope Revolution, Computer Physics Com. 271 (2022) 108205)
Yeh22 [Yeh, Shelton, Olive, Fields [JCAP 10 (2022) 046]
[Trezzi 2017] (Trezzi, D., Anders, M., Aliotta, M., et al. (Luna) 2017, Astroparticle Physics, 89, 57)
[Moscoso 2021] (J. Moscoso, R. S. Souza, A. Coc, C, Iliadis, Astrophysical Journal, 923:15 (2021)).

Numerical values

[Planck 2015 XIII] Ade et al. A.&A. 594, A13 (2016).
[Planck 2018 VI] A.&A. (2020). We use TT+TE+EE+lowE+Lensing+BAO parameters (last column of table 2).
[PDG17] Particle Data Group 2017.

[PDG20] Particle Data group 2020.
[PDG23] Particle Data group 2023
[PDG24] Particle Data Group 2024

Masses of nuclei previously from nubase2016.asc (Chinese Physics C 36 (2012) 1157-1286) which can be downloaded from Nuclear Data Services
Switched to the 2020 update of that database which is nubase_ 4.mas20.txt (Chinese Physics C 45 (2021) 030001) which is also found on from Nuclear Data Services.

Directory set up

We set the directory. This cell is not an initialization cell, and is not save in the .m file. However for some systems, when evaluating examples which load PRIMAT-Main.m, it is necessary to turn this cell into an initialization cell so that it is part of the PRIMAT-Main.m file.

In[ ]:=
```
SetDirectory[NotebookDirectory[]]
```
alloue répertoire répertoire de notebook

Out[ ]=
```
/Users/pitrou/Dropbox/iap/notebooks/PRIMAT2024
```

To check what is the directory of your notebook :

In[ ]:=
```
Print["The current Directory is ", Directory[]]
```
imprime répertoire répertoire

```
The current Directory is /Users/pitrou/Dropbox/iap/notebooks/PRIMAT2024
```

Constants of Physics

cgs system

We use cgs system with Kelvin. (For instance an erg is 1 g cm^2 s^{-2}).
By definition these are set to one in these units. Any change
of system of units can be made by modifying these variables only.

For instance if we want to use the m / kg / s system we need only put cm =
0.01 and gram = 0.001 below.

As a check, final results should not depend on
these conventions since abundance rates are dimensionless.

In[ ]:=
```
second = 1;
cm = 1;
gram = 1;
Kelvin = 1;
```

When taking values from the kg/m/s system, we use the factors

```
kg = 10^3 gram;
meter = 10^2 cm;
km = 10^3 meter;
Joule = kg meter^2 / second^2; (* This gives 10^7 ergs *)
DensityUnit = gram / cm^3;
Hz = 1 / second;
barn = 10^(-24) * cm^2;
```

```
Giga = 10^9;
Mega = 10^6;
Kilo = 10^3;
```

```
pc=3.0856777807×10^16 meter; (* The parsec *)
Mpc=Mega pc;
```

- Fundamental constants (exact values since the 2019 revision of the International System of Units)

```
kB = 1.380649×10^-23 Joule / Kelvin;(* Boltzmann constant in J/K *)
clight = 2.99792458 * 10^8 * meter / second; (* speed of light in cm/s *)
hbar = 6.62607015/(2 π) 10^-34 Joule second;
Avogadro = 6.02214076×10^23;
```

When using masses of particles, we use eV and MeV that we convert in the cgs system

```
eV = 1.602176634×10^-19 Joule;
keV = Kilo eV;
MeV = Mega eV;
GeV = Giga eV;
```

Interactions constants

```
GN = 6.67430×10^-11 meter^3 / kg / second^2; (* Gravitation constant PDG2024 *)
GF = 1.1663788 * 10^-5 / (GeV)^2; (* Fermi Constant PDG2024*)
```

constante

```
gA = 1.2754;
(* Axial current constant of structure of
 the nucleons. It was 1.2723(+-23) in PDG2016 *)
(* However post 2002 data suggest
 1.2755(11) as advised by William Marciano*)
(* PDG2020 value: 1.2756(13) *)
(* We now use PDG2024 value 1.2754(13) which
 is much more consistent with the neutron lifetime *)
```

In[ ]:=
```
fWM = 3.7058 / 2 (*1.853*); (* Weak magnetism see 1212.0332*)
radiusproton = 0.841 * 10^-15 meter (*(arXiv:1212.0332)*);
```

fWM is the weak magnetism constant. See Eq. [Horowitz&Li] for definition with the value given by its Table 1.
Note that all expressions in [Seckel 1993] seem to have a factor 2 difference. That is all interaction rates involving the weak magnetism in [Seckel 1993] are underestimated by a factor 2. Expressions in [Lopez et al. 1997] seem correct however.

In[ ]:=
```
αFS = 1 / 137.035999084;(* Fine structure constant =e^2/(4π) *)
```

Particle masses

Throughout, masses stand always for mc^2 so that they are in fact energies. This avoids putting unnecessary c^2 factors

In[ ]:=
```
me = 0.510998950 MeV; (*PDG2024*)
mn = 939.56542052 MeV;(*PDG2024*)
mp = 938.27208816 MeV; (*PDG2024*)
Q = mn - mp; (* Mass difference between neutrons and protons *)
m_Nucleon = mn;

m_W = 80.3692 GeV; (* Mass of the W Boson. *)
m_Z = 91.1880 GeV;
```

The energy difference between neutron and proton in MeV gives 1.2933324(5) [PDG 2024].

In[ ]:=
```
Q / MeV
```

Out[ ]=
```
1.2933324
```

The atomic mass, the Helium4 mass (in units of atomic mass) and Hydrogen mass (in units of atomic mass).

In[ ]:=
```
(* Values from AME 2012, Audi et al.;
   valeurs
ma = 931.494061 MeV;(* Audi2012 *)
He4Overma=4.0026032541; (* Audi2012 *)
H1Overma=1.00782503223;(* Audi2012 *)*)

(* Values from AME 2020, Wang et al 2021 Chinese Phys. C 45 030003 *)
   valeurs                                                    constante C
ma = 931.49410242 MeV;
He4Overma = 4.002603254130;
H1Overma = 1.007825031898;
```

Options

Physical parameters

Main parameters

Baryons density fraction. This is $\Omega_b h^2$.

```
In[ ]:= Meanh2Ωb0Planck = 0.022425;(*[Planck 2018 VI, Last column of
                                                    dernier
 table 2 (and checked from the Planck Legacy Markov chains)]*)
σh2Ωb0Planck = 0.000136;(* Standard deviation*)
(* Planck 2018 VI states 0.00014 but from the
 Legacy Markov chains we infer the value 0.000136 *)
```

```
In[ ]:= Meanh2Ωb0 = Meanh2Ωb0Planck; (*Can be redefined*)
σh2Ωb0 = σh2Ωb0Planck;
```

Neutron life time PDG 2024
https://pdg.lbl.gov/2024/listings/rpp2024-list-n.pdf

```
In[ ]:= Meanτneutron = 878.4;(* This can be redefined *)
στneutron = 0.5;
```

We can add extra relativistic degrees of freedom that would be
considered as being fully decoupled. Hence they behave as neutrino in the
Fully decoupling approximation with there energy density ~$1/a^4$ exactly.
By convention these relativistic degrees of freedom are a number of Fermi–Dirac species (with no chemical potentials) or sterile/decoupled neutrino species.

```
In[ ]:= Nrelat := 0.;
```

These will be updated whenever Meanτneutron and Meanh2Ωb0 are modified (because we use :=)

```
In[ ]:= τneutron := Meanτneutron;
h2Ωb0 := Meanh2Ωb0;
```

Secondary parameters

CMB temperature today

```
In[ ]:= TCMB0 := 2.7255 Kelvin;
σTCMB0 := 0.0006 Kelvin;(* [Planck 2015 XIII] *)
```

h is the Hubble rate in units of 100 km/s/Mpc.

In[ ]:=
```
h := 0.6766; (*+-0.0042 *)(*[Planck 2018 VI TT+TE+EE+lowE+Lensing+BAO]*)
```

Cold Dark Matter density fraction. This is $\Omega_c h^2$.

In[ ]:=
```
Meanh2Ωc0 = 0.11933;(* [Planck 2018 VI, last column of Fig. 2]*)
σh2Ωc0 = 0.00091;
h2Ωc0 := Meanh2Ωc0;
```

BBN options

■ Nuclear rates

Most nuclear reactions rates are tabulated in a separate file. The rest of the rates are given as analytic fits.
The external file loaded with all reactions definitions and rates is given by the name TabulatedReactionsFile.

The set of nuclear reactions is given by the external files plus all the analytic forms inside the code, unless the user asks for the reduced network (12 nuclear reaction), see below,
in which case only the first 12 nuclear reactions are selected and no extra analytic reactions

In[ ]:=
```
TabulatedReactionsFile = "BBNRatesAC2024.dat";
TabulatedReactionsFileSmallNetwork = "BBNRatesAC2024_small_network.dat";
(* For small network of 12 nuclear reactions *)
```
boucle for

In[ ]:=
```
TabulatedReactionsFileParthenope =
 "BBNRatesAC2024_ddn_ddp_Parthenope.dat" (*A file fhere the d+d↔
 n+He3 and d+d↔
  t+p rates are the ones used in Parthenope (Gariazzo 2021)*)
TabulatedReactionsFileSmallNetworkParthenope =
  "BBNRatesAC2024_small_network_ddn_ddp_Parthenope.dat";
(*A file where the d+d↔
 n+He3 and d+d↔t+p rates are the ones used in Parthenope (Gariazzo 2021),
and with just 12 nuclear reactions.*)
```

Out[ ]=
```
BBNRatesAC2024_ddn_ddp_Parthenope.dat
```

An option to use the Parthenope rates for d + d <-> t + p and d + d <-> n + He3. If it is true, then the codes uses TabulatedReactionsFileParthenope or TabulatedReactionsFileSmallNetwork-Parthenope if the small network option (see below) is True.

In[ ]:=
```
$Parthenope = False;
```
faux

We can decide to keep only a subset of the equations by specifying the maximum nuclear mass. The default value is Infinity, meaning that we do not cut the network.

```
In[ ]:= MaximumNuclearMass = Infinity;
                             infini
```

We can also reduce the network to its 12 first reactions. This is enough for the first light elements. It consists in the first 12 reactions of the tabulated reactions (those in the file TabulatedReactionsFile).

Furthermore, if $ReducedNetwork is selected, no extra analytic reactions are added so that we really have only a total of 12+1 reactions (because we also have weak interactions).

```
In[ ]:= $ReducedNetwork = False;
                          faux
```

```
In[ ]:= SmallNuclearNetworkSize = 12;
```

■ Monte-Carlo uncertainty estimation options

```
In[ ]:= $RandomNuclearRates = False;
                              faux
        $MaxVariationRate = 1000;
```

If $RandomNuclearRates is True, then each time we build the equations of the nuclear network, we generate rates with a log normal distribution according to the 'f' specified (see Eq. 4.5 of [Coc et al. 2014] for definition of f).
We limit f to the values 1/$MaxVariationRate<f<$MaxVariationRate .

■ Rescaling of some rates

It is possible to rescale some rates by choosing their rescaling factor below. First, the option $RescaleSomeRates must be set to True otherwise the rescaling factors are ignored.

```
In[ ]:= $RescaleSomeRates = False;
                            faux
```

For each reaction whose rate we want to rescale, we need to know the exact name of the reaction. The names of reactions are collected in tables below in the core of the notebook. For a reaction of the type a + b -> c + d, the name is "abTOcd". However one needs to know the order of the initial particles and of the final particles, and to put correctly the photons if they appear. Hence it is recommended not to guess the name of the rate, but to properly read it from the Tables below located in the section "Collecting all reaction rates".
For instance the reaction He3 + t > He4 + d (we use d for H2 and t for H3 and a for He4) has name He3tTOHe4d. The factor to modify it is therefore He3tTOHe4dFactor.

As an example, we show how to introduce a rescaling factor for the d + p -> He3 + γ, d + d -> He3 +

n, and d + d -> H3 + p reactions hereafter.
If we want to vary other rates, one needs only to set the numerical value of their associated factors
.

In[ ]:=
```
If[$RescaleSomeRates,
si
  dpTOHe3gFactor = 1;
  ddTOHe3nFactor = 1;
  ddTOtpFactor = 1;
 ];
```

Note that this procedure is only valid for variation of rates which are given as tables in the external file. If the user wants to vary a rate whose form is analytic, it is sufficient to add a factor “Reaction-Factor” in the definition of the analytic forward rate (forward[T9_]:= ....). It is then advised to write here (so that all modifications can be turned on and off without forgetting that some rates have been modified)

If[$RescaleSomeRates,
ReactionFactor=100;,
ReactionFactor=1;
]

We recall that the syntax for If/then/else is

If[Condition,
commandtrue;,
commandfalse;
]

- QED corrections (pair production) to nuclear rates

If this options is set to True, then we rescale some rates due to pair productions in the final state. See [Pitrou&Pospelov 2020].

In[ ]:=
```
$NuclearRatesQEDCorrections = True;(* recommended value is True *)
                              vrai                          vrai
```

Weak rates options

Since the weak interaction rates, that is the rates of weak reactions of the type n + ν <-> p + e and associated reactions, depend only on temperature,
these can be computed once and for all so as to explore the dependence in cosmological parameters or other parameters.

If $RecomputeWeakRates is set to True, the code recomputes the weak rates.
Otherwise it reads them from files previously stored. If the file does not exist, it recomputes the

rates.

If $ParallelWeakRates=True this is done using parallelization over the various CPUs that Mathematica can detect (be aware that when $NEVO=TRue , it slows things done because of badly sharing memory. I cannot find the reason.)

In[ ]:=
```
$RecomputeWeakRates = False;
$ParallelWeakRates = False;
```

There are several booleans corresponding to the different types of corrections which can be considered in these weak rates.
The name of the file used for storing the rates is built out of these booleans.

Since the corrections do not add linearly, there are in principle many choices of corrections, but the only meaningful
choices are those without any corrections and those with all corrections included.
Or maybe those with just one correction added, so as to check its amplitude.

In[ ]:=
```
$RadiativeCorrections = True;(*Recommended True*)

$ResummedLogsRadiativeCorrections = True;(*Recommended True*)
$RelativisticFermiFunction = True;(*Recommended True*)
```

1) If $RadiativeCorrections is set to True, we use Coulomb and radiative corrections (see section III.E of companion paper. The corrections are implemented in Eqs. 101 and 104.).
If $ResummedLogsRadiativeCorrections is set to True we use Eq. 15 of [Czarnecki et al 2004] which amounts to resumming some higher order radiative corrections, and this is also Eq. B35 of the companion paper.

If $ResummedLogsRadiativeCorrections is False we use simply Eq. 7 of the same reference, which corresponds to Eq. 103 and B30 of the companion paper.
If $RelativisticFermiFunction is True we use the relativistic Fermi function (Eq. 5 in [Ivanov 2012]), which corresponds to Eq. 100 of the companion paper. Otherwise we use the standard non-relativistic Fermi function, given by Eq. 99 of the companion paper.

In[ ]:=
```
$RadiativeThermal = True;
(*Recommended True but it is much slower. Since the modification brought
 by these corrections is very small, setting False is acceptable.*)
$CorrectionBremsstrahlung = True;(*Recommended True*)
```

2) If $RadiativeThermal set to True, the thermal radiative corrections are taken into account.
The first expressions date back from [Dicus et al.]. However other expressions were derived subse-

quently in [Lopez&Turner 1998], [Cambier et al.] and [Esposito et al. 1999] among other references. [Kernan] pointed typos and compared the differing results of [Cambier et al.] and [Dicus et al.]. These were correctly computed in [Brown&Sawyer].
In the companion paper, the thermal radiative corrections are given by Eq. 108 with the various contributions given by Eqs. 109-113. However, it is easier to compute the first contribution of 108 using Eqs. 109 (with definition B41), but to compute the second and third contributions of 108 using Eqs. B50-B51.

We showed in companion paper that Bremsstrahlung corrections need also to be added to be fully consistent with [Brown&Sawyer]. This is controlled by the boolean $CorrectionBremsstrahlung. If $RadiativeThermal is set to True, then these bremsstrahlung corrections (corresponding to Eqs. 107a and 107b in companion paper with the definitions B48-B49) are also incorporated if $CorrectionBremsstrahlung is True.

In[•]:=
```
$FiniteNucleonMass = True; (*Recommended True*)
                     vrai                   vrai
```

3) If $FiniteNucleonMass is set to True, we take into account the finite mass of nucleons by keeping corrections in $1/M_n$ in the collision integrals of the weak rates.
Our method is described in the companion paper and differs from earlier literature.
There is a suboption for these finite mass corrections which is

In[•]:=
```
$CoupledFMandRC = True; (*Recommended True*)
                  vrai                  vrai
```

If $CoupledFMandRC is False, finite mass corrections are computed from the Born results with no radiative null temperature corrections. If this is set to True, the finite mass corrections are applied to the rates on which the Coulomb and radiative corrections are taken into account. True is the advised value.

The expressions implemented are Eqs. 114 when $CoupledFMandRC is False, with the definition B23. If $CoupledFMandRC is True, then we use Eqs. 115 with the Fermi and radiative corrections corresponding to the above choices for radiative corrections.

4) Mass shifts due to QED plasma effects are **ALREADY** taken into account in thermal radiative corrections.
However it is possible to turn the option $QEDMassShift to true to check how this affects the rate when taken individually.
Apart to satisfy this curiosity, **this option should be set to False** in all cases.

In[•]:=
```
$QEDMassShift = False;
                faux
```

Plasma physics options

```
In[•]:= $QEDPlasmaCorrections = True;(*Recommended True*)
                                 vrai                vrai
        $CompleteQEDPressure = True;(*Recommended True*)
                               vrai                vrai
        $QEDOe3 = True; (*Recommended True*)
                  vrai                vrai
```

If $QEDPlasmaCorrections is set to True, the QED corrections to the pressure and the energy density are taken into account. This affects for instance the expansion rate via the Friedmann equation.
Expressions can be found in [Lopez&Turner 1998], [Mangano et al. 2001] or in [Heckler 1994]. See also the companion paper where Eqs. 48-52 are used inside Eq. 55 to modify the entropy and Eq. 58 to modify the energy density.
If $CompleteQEDPressure is False, the subdominant term is ignored. Otherwise it is included. We checked it is so subdominant that it does not change the results.
If $QEDOe3 is True, then order e^3 corrections to the Plasma (to P and ρ) are included, Following [*Bennett et. al. 2019].*

```
In[•]:= $RecomputePlasmaCorrections = False;
                                      faux
```

Since the QED effects depend on temperature, they need to be computed only once for all and they are stored on a file. But we can force the recomputation of these corrections by setting $RecomputePlasmaCorrections to True;

Neutrinos options

Neutrinos generations. Usually this is 3.

```
In[•]:= NeutrinosGenerations := 3.;
```

Neutrino incomplete decoupling options

```
In[•]:= $IncompleteNeutrinoDecoupling = True;(*Recommended True*)
                                        vrai                vrai
```

If $IncompleteNeutrinoDecoupling is set to True, the entropy transfer from $e^{\pm}$ annihilations to neutrinos is taken into account. Indeed if we consider the details of the decoupling of neutrinos, it is found that decoupling is incomplete by the time electrons and positrons annihilate into photons. This results in a slight overheating of neutrinos and cooling of the electrons/photons plasma. The advised value for $IncompleteNeutrinoDecoupling is True.

In[ ]:=
```
$NEVO = True; (* Recommended True *)
        vrai                    vrai
$TrueTνe = True;(* Recommended True *)
           vrai                  vrai
$FileSuffix = "";
(*Suffix of the file name in which NEVO results are stored. *)
```

The original PRIMAT approach consisted in using a fit for the heating function of the neutrinos due to incomplete decoupling. The fitting functions which is used is the one given in [PArthENoPE]. It is used in Eq. 63 of the companion paper. In this approach, all three flavors of neutrinos have equilibrium distributions at a common temperature derived in Eq. 64 of the companion paper. This original method is chosen if $NEVO = False.

However, if $NEVO is set to True, we directly import the neutrino effective temperatures (and the heating function) from the results obtained with NEVO, a Python code designed to implement neutrino evolution through the MeV age (developed by Julien Froustey and Cyril Pitrou, and described in [Froustey et al. 2020]). This enables to take into account two additional effects: the different decoupling of electron and muon/tau neutrinos, resulting in different effective temperatures (implemented if $TrueT*νe* is set to True) and non-thermal distortions of the neutrinos distribution functions (implemented if $SpectralDistortions is set to True, see below).

Important: it would be inconsistent to set $SpectralDistortions = True but $IncompleteNeutrinoDecoupling = False, except if the are analytic distortions included (see below).
That is why when computing the weak rates, spectral distortions are included only if both these options are set to True, or if $AnalyticDistortions is set to True.

Important : Each time you change one of the 3 above weak rates options ($NEVO or $TrueT*νe* or $SpectralDistortions), $RecomputeWeakRates should be set to True to take into account these new options and store the corresponding rates on disk. The rates which are provided with the distribution were computed with these three options set to True (which we recommend to use).

The common neutrino average temperature can therefore be computed in two different (but, in principle, equivalent) ways:
1/ through the heating function
2/ directly from the effective temperatures.
If $Taverage$\mathcal{N}$ is set to True, the first method is used but we prefer the second method, hence we set it to False.

In[ ]:=
```
$Taverage𝒩 = False;(*Recommended False *)
             faux                   faux
```

Files in which NEVO results are stored

In[ ]:=
```
(* When providing different files for neutrinos and antineutrinos *)
(*FileNEVOneutrinos = "CSV/NEVOneutrinos"<>$FileSuffix<>".csv";
FileNEVOantineutrinos = "CSV/NEVOantineutrinos"<>$FileSuffix<>".csv";*)

$NeedNEVODistortions :=
  If[$NEVO && $SpectralDistortions && Not[$AnalyticDistortions],
   If[$RecomputeWeakRates, True, If[Not[FileExistsQ[NamePENFilenp]] ||
       Not[FileExistsQ[NamePENFilepn]], True, False]], False];

FileNEVOneutrinos := "CSV/" <>
   If[$QEDPlasmaCorrections, If[$QEDOe3, "NEVOPRIMAT" , "NEVOPRIMAT_QED2"],
    "NEVOPRIMAT_NoQED"] <> $LocalSuffix <> ".csv";
FileNEVOantineutrinos := FileNEVOneutrinos;
```

Degenerate Neutrinos options

Possible initial chemical potential in the neutrino initial conditions (the same for all generations). Incompatible with $NEVO which solves for the neutrino oscillations and allows different potentials in the various flavours but this is controlled by what has been solved and output with NEVO. The basic files provided are for vanishing initial chemical potentials.

In[ ]:=
```
$DegenerateNeutrinos = False;

(*Recommended False if using the default runs of NEVO.*)

μOverTν = 0.;

ξν := If[$DegenerateNeutrinos, μOverTν, 0];
```

If neutrinos have a chemical potential, then $DegenerateNeutrinos = True. Standard BBN is with non-degenerate neutrinos and when $DegenerateNeutrinos = False the value of μOverTν is ignored. Note that in the case of degenerate neutrinos, one must run the code with RecomputeWeakRates = True because this modifies the weak rates. And for every value of μOverTν one must recompute the weak rates (which can take a lot of time). Note that the incomplete decoupling effects, which use NEVO's results would then be inconsistent because they were computed without chemical potential. μOverTν is also noted ξ_ν in the companion paper. See section VI.C.

Spectral distortions

In[ ]:=
```
$SpectralDistortions = True;(* Recommended True *)
                       vrai                  vrai
```

If $SpectralDistortions = True then spectral distortions are added to electronic neutrinos (and also all other flavours but this could be changed)

1) $NEVO distortions
If $IncompleteNeutrinoDecoupling = True and $NEVO=$TrueT*ν*e=True, then it is advised to also ask for $SpectralDistortions since they were computed in the decoupling code NEVO. They are then used if $AnalyticDistortions=False below and superseded otherwise. In the case NEVO distortions are used, they induce no change of neutrino energy density since in that case distortions are defined with respect to a reference spectrum in such a way that there is no energy density contained in the distortions (of both neutrino and antineutrino).

2) The user can define **custom distortions** (which are the same in neutrinos and antineutrinos) with analytic forms by setting $AnalyticDistortions=True.
It is slightly inconsistent because these would impact the neutrino decoupling. Hence the most consistent set of parameters consists in setting $IncompleteNeutrinoDecoupling=False,
so they can be viewed as distortion which are added after decoupling but before neutron/proton freeze out.
If $AnalyticDistortions = True, then an analytic form of the distortions is used. The user can modify the code to define custom distortions inside the code itself.
For the moment the same distortion shape is put in all neutrino flavors. Also only the electronic neutrino distortion affects the weak-interaction, the fact we put the distortion in all flavors has an impact on the energy density contained in the distortion.

Two basic analytic forms are however readily provided :

a) A *μ*-type distortion, corresponding to the difference between a Fermi-Dirac spectrum with a given chemical potential with and a Fermi-Dirac spectrum with a reference chemical potential.
If $DegenerateNeutrinos=True above, then the chemical potential of the reference neutrino spectrum is with a non-vanishing chemical potential $\xi\nu=\mu$OverTν (and otherwise $\xi\nu=0$), that is $\frac{1}{e^{y-\xi\nu}+1}$ for neutrinos and $\frac{1}{e^{y-\xi\nu}+1}$ for antineutrinos (with $y = E/T_\nu$). Then the *μ*-type distortion, corresponds to a Fermi-Dirac spectrum with $\xi\nu + \Delta\xi\nu$. Hence the distortion for neutrinos is defined by $\frac{1}{e^{y-(\xi\nu+\Delta\xi\nu)}+1} - \frac{1}{e^{y-\xi\nu}+1}$ and for antineutrinos it is $\frac{1}{e^{y+(\xi\nu+\Delta\xi\nu)}+1} - \frac{1}{e^{y+\xi\nu}+1}$. It is ill-considered to use this kind of distortion, because whatever the chemical potential, it can be put in the reference chemical potential. Hence this usage is only meant to test the code, for instance checking that having μOverTν=Somevalue and $\Delta\xi\nu$=0 is equivalent to μOverTν=0 and $\Delta\xi\nu$ = somevalue.

b) A Y-type distortion corresponding to the equivalent of the SZ (Sunyaev-Zeldovich) distortion, that is corresponding to the spectrum shaped by the Kompaneets equation (Kompaneets 1957, Soviet Phys JETP, Vol. 4 N. 5 p 730) but with an up-scattered Fermi-Dirac distribution (instead of

an up-scattered Planck spectrum as for the SZ effect on photons of the CMB).

The collision term of the Kompaneets equation is of the form $\frac{1}{E^2}\partial_E\left(E^4\,\partial_E f\right)$ where f is the reference spectrum (Fermi-Dirac with or without chemical potential depending on $DegenerateNeutrinos option and μOverTν value).

Hence the distortion is of the form $Y_{SZ} * \frac{1}{E^2}\partial_E\left(E^4\,\partial_E\,\frac{1}{e^{y-\xi\nu}+1}\right)$ for neutrinos and $Y_{SZ} * \frac{1}{E^2}\partial_E\left(E^4\,\partial_E\,\frac{1}{e^{y+\xi\nu}+1}\right)$. Y_{SZ} is the amount of up-scattering and is proportional to the optical depth multiplied by (Ts/ms) where Ts and ms are temperature and mass of the heavy species which is responsible for the up-scattering. The total energy density in neutrinos is then modified to be $\rho\nu(1 + 4\,Y_{SZ})$ where $\rho\nu$ is the energy density without the SZ distortion.

Note that the true Kompaneets equation must be a stationary point if the species responsible for the up-scattering is at the same temperature T_s as the scattered species. Hence the total rate per optical depth is

$\frac{df}{d\tau} = \frac{1}{m_s E^2}\partial_E\left[E^4\,(T_s\partial_E f + f(1-f))\right]$ (note the change of sign with respect to a Kompaneets equation for bosons=photons where we have f(1+f) since for fermions we have Pauli-blocking instead of stimulated emission for bosons),

and if f is initially a Fermi-Dirac distribution at temperature T_ν (with whatever chemical potential), then in the low scattering approximation (Born approximation)

f(1-f) = - $T_\nu\,\partial_E f$ hence the Kompaneets equation is in the form

$\frac{df}{d\tau} \sim \frac{(T_s-T_\nu)}{m_s}\,\frac{1}{E^2}\partial_E\left(E^4\,\partial_E f\right)$ which implies that the distortion is approximately $f - f_{FD} = Y_{SZ}\,\frac{1}{E^2}\partial_E\left(E^4\,\partial_E f_{FD}\right)$ with $Y_{SZ} = \tau * \frac{(T_s-T_\nu)}{m_s}$.

In[•]:=
```
$AnalyticDistortions = False;(* Recommended False. Only
                        |faux                  |faux
 for experimental investigation of neutrino distortions. *)
Δξν = 0.;
YSZ = 0.;
```

Monte - Carlo estimation of errors

There are 3 booleans which control what variables are varied randomly.
-Nuclear rates ($RandomNuclearRates)
-Neutron lifetime ($Random$\tau$neutron)
-And possibly baryons abundance according to [Planck 2015] results when this is interesting to vary it as well ($Randomh2Ωb).

In[•]:=
```
$Randomτneutron = False;
                  |faux
$Randomh2Ωb = False;
              |faux
```

This variable determines if parallelization is used when doing Monte Carlo

In[•]:=
```
$ParallelBool = False;
                |faux
```

Numerical options

Miscellaneous options

```
In[•]:= $InterpolateAnalytics = True;
                                vrai
```

It is slightly faster to interpolate the analytic expressions of nuclear reaction rates. Setting $InterpolateAnalytics to True is recommended.

```
In[•]:= $HistoryLength = 10;
        longueur d'historique
```

This is to avoid *Mathematica* to store too many results in memory. Only the past $HistoryLength results are kept in computer memory (this is standard Mathematica option).

```
In[•]:= $PaperPlots = False;
                      faux
        $ResultsPlots = False;
                        faux
```

If $PaperPlots is set to True, the most important plots are constructed and they are output in pdf in the 'Plots' subfolder.
Unless interested in reproducing the plots of the companion paper, this should be set to False to avoid any loss of time in the code.

If $ResultsPlots is True the results for the abundances are plotted at the end and exported in pdf files. Similarly, to avoid loss of time this shouldbe set to False.

Numerical precision

```
In[•]:= $CompileNDSolve = True;
                          vrai
```

If $CompileNDSolve is set to True (advised), then the differential equation solver uses compiled functions. This is slightly faster.

```
In[•]:= $StoreCompilation = True;
                            vrai
        (*Stores compilation in lib folder if True. Otherwise it is
                                                    vrai
         compiled everytime the Kernel is loaded (a few seconds lost) *)
        $Recompile = False;
                     faux
        (*If this true it forces the compilation of previously compiled
          si
          and stored functions. May be necessary if you change
          something subtle. Or you erase all files in the lib
                              ou
          folder and this will force the Recompilation as well..*)
```

```
In[•]:= $BDFOrder = 2;
```

Order of numerical scheme for Backward Differentiation Scheme (BDS) integration.
-Order 1 works well but is slow.
-Order 2 (advised) is faster. Higher orders result in numerical crash.

```
In[•]:= PrecisionNDSolve = 2;(* Put 2 here is the best choice *)
                             mets en fichier
```

PrecisionNDSolve is a precision parameter used in the differential equation solver. It is tuned such that 0 gives reasonable results, 1 gets very good results, and 2 gets excellent results and 3 gets super dupper precise results (10^{-5} precision).

```
In[•]:= AccuracyNDSolve := 15 + PrecisionNDSolve;
```

We slave AccuracyNDSolve to PrecisionNDSolve. This is roughly the number of digits of precision behind the dot, so increasing it will lead to always better results but this can crash if the accuracy required is too strong.

```
In[•]:= NTemperaturePoints = 1000; (*1000 is enough*)
```

Number of points in discretization of temperature between the
highest temperature (10^{12} K) and the lowest ($\sim 10^{7.5}$ K). 1000 is enough.
Sampling is performed with NTemperaturePoints +
1 points. Spacing is performed logarithmically, with Log_{10}.

```
In[•]:= InterpOrder = 3;
```

Order of polynomials used in interpolations of reaction rates. Usually with Spline functions.

```
In[•]:= $FastPENRatesIntegrals = True;
                                 vrai
```

If $FastPENRatesIntegrals is set to True, it uses a Simpson method for numerical integrals. Otherwise it uses the Mathematica function NIntegrate which is more accurate (adaptative refinement of integral) but much slower.

```
In[•]:= $PENRatesIntegralsPoints = 200;(*200 is enough *)
```

In case $FastPENRatesIntegrals is set to True, $PENRatesIntegralsPoints is the number of points used to perform the numerical integrals. 200 is enough. 400 is ultra precise.

```
In[•]:= SetOptions[SelectedNotebook[], PrintPrecision → 8]
        alloue options  notebook sélectionné  précision d'impression
```

We increase the number of digits which are displayed

In[ ]:=
```
$Verbose = True;
           vrai
```

This variables allows to track what happens in the computations

Check Incompatible options

This section is incomplegte . We should be much more careful and systematic.

If Plasma QED corrections are not used, then the full expression for the Pressure QED correction cannot be used, nor the order e^3 correction.

If NEVO results are not used for neutrino decoupling,
then we cannot have the true ν_e temperature.
If we do not use NEVO and there are no AnalyticDistortions,
then we cannot have Spectral Distortions.

In[ ]:=
```
CheckOptions := (
    If[Not@$QEDPlasmaCorrections && $CompleteQEDPressure,
    si  négation
     Print["Inconsistent Options, I set $CompleteQEDPressure=False"];
     imprime                    options      unité imaginaire                    faux
     $CompleteQEDPressure = False;];
                            faux
    If[Not@$QEDPlasmaCorrections && $QEDOe3,
    si  négation
     Print["Inconsistent Options, I set $QEDOe3=False"];
     imprime                    options      unité imaginaire      faux
     $QEDOe3 = False;];
               faux
    If[Not@$NEVO && $TrueTνe,
    si  négation
     Print["Inconsistent Options, I set $TrueTνe=False"];
     imprime                    options      unité imaginaire      faux
     $TrueTνe = False;];
                faux
    If[Not@$NEVO && Not[$AnalyticDistortions] && $SpectralDistortions,
    si  négation          négation
     Print["Inconsistent Options, I set $SpectralDistortions=False"];
     imprime                    options      unité imaginaire                    faux
     $SpectralDistortions = False;];
                            faux
    If[$NEVO && Not[$IncompleteNeutrinoDecoupling],
    si             négation
     Print["Inconsistent Options, I set $IncompleteNeutrinoDecoupling=True"];
     imprime                    options      unité imaginaire                                  vrai
     $IncompleteNeutrinoDecoupling = True;]
                                          vrai
  );
```

In[ ]:=
```
CheckOptions
```

Initial definitions

Temperature eras

We choose to split the BBN numerical calculations in three eras.
1) First the high temperature era between Tstart = 10^{11} K and TMiddle. Only neutrons and protons abundances are tracked and this is ruled by weak interactions.
2) The intermediary era, between TMiddle and T18, where only a small network of reactions (17 nuclear reactions plus weak interactions or less if the reduced network is used) is used.
3) A low temperature region, between T18 and Tend, where Tend is usually slightly lower than 10^8 K, typically Tend = 6×10^7 K, and where all nuclides and reactions are considered.

So we have Tstart > TMiddle > T18 > Tf.

In[ ]:=
```
Tstart = 10^11 Kelvin;
TMiddle := 0.9999 * 10^10 Kelvin;
T18 := 1.25 * 10^9 Kelvin;
Tend = 6. * 10^7 Kelvin;
```

Temperature sampling

We choose to sample temperature starting from
10^{12} K. This is interesting to check high T behaviour of some effects.

In[ ]:=
```
Ti = 10^12 Kelvin;
Tf = 10^7 Kelvin;
LogTi = 1. Log10[Ti];
          logarithme en ba
LogTf = 1. Log10[Tf];
          logarithme en ba
(*zmin=me/(kB*Ti);
zmax=me/(kB*Tf);*)
```

We first build the list of LogT points (ListLogT) and then the list of T points (ListT).

In[ ]:=
```
ListLogT = Sort@DeleteDuplicates@Join[{10.},
            trie    supprime répétitions    joins
     Table[i, {i, LogTf, LogTi, (LogTi - LogTf) / NTemperaturePoints}]];
     table
ListT = 1. × 10^ListLogT;
```

```
In[•]:= ListTRange[T1_, T2_] := Module[{len = Length@ListT,
                                  |module          |longueur
    imindown, imaxup, Tmin = Min[T1, T2], Tmax = Max[T1, T2]},
                                  |minimum          |maximum
   imindown = Max[1, -1 + Position[ListT, SelectFirst[ListT, # > Tmin &]][[1, 1]]];
              |maximum    |position          |sélectionne premier
   imaxup = Min[len, Position[ListT, SelectFirst[ListT, # ≥ Tmax &]][[1, 1]]];
            |minimum  |position          |sélectionne premier
   ListT[[imindown ;; imaxup]]
  ]
```

ListTRange is a function to select a sublist in this list of temperature, according to a range of temperature which makes sure to have either the points on the boundary or at least one point beyond (to avoid problems with interpolating functions).
If T = 10^{10} is not in the list, we add it to avoid problems with interpolations of reactions rates which all start at 10^10 and below.

Neutrinos basics

Analytic expression for Fermi - Dirac distributions : number and energy densities. We hard code then below to save time.

```
In[•]:= (*ρFD[c_]=1/(2π^2) ∫_0^Infinity y^3/(e^(y-c)+1) dy*)
        (*nFD[c_]=1/(2π^2) ∫_0^Infinity y^2/(e^(y-c)+1) dy*)
        (*ρFD[0]*)
        (*Simplify@Normal@Series[ρFD[c]+ρFD[-c],{c,0,10}]*)
          |simplifie  |forme n···  |développement en série entière

In[•]:= (*ηv[c_]=(nFD[c]-nFD[-c])/(2Zeta[3]/π^2);*)
        (*Series[ηv[c],{c,0,3}];*)
          |développement en série entière

In[•]:= ρFDNonDegenerate = 7 π^2/240;

        TotalρFD[c_] = c^2/4 + c^4/(8 π^2) + 7 π^2/120;
```

Effective number of neutrinos generation due to chemical potential
(this is different from N_{eff} which takes
into account also QED and incomplete neutrino decoupling)

```
In[•]:= Nneu := NeutrinosGenerations * TotalρFD[ξv]/(2 ρFDNonDegenerate)
```

Temperature of CMB today in Kelvin. We consider the case where QED effects are ignored or taken into account. This is the implementation of (the inverse of) Eq. 56 in companion paper.

In[ ]:=
```
FourOverElevenQED := 1 / (11/4 - 25 αFS/(8 π) + If[$QEDOe3, 10 αFS^(3/2)/π^2 Sqrt[π/3], 0]);

FourOverElevenNoQED := 4/11;
FourOverEleven :=
  If[$QEDPlasmaCorrections, FourOverElevenQED, FourOverElevenNoQED];

Tν0 := (FourOverEleven)^(1/3) TCMB0;
```

In[ ]:=
```
(1/FourOverEleven)^(1/3)
```

Out[ ]=
```
1.3998958
```

Temperature of neutrinos today is lower than photons because they decoupled earlier and electron/positron annihilation has only reheated photons. This leads to the famous ratio of 4/11 between the T^3 of the neutrinos and photons. However, since decoupling is slightly incomplete, this in principle should be corrected like in [Mangano et al. 2005] or [Grohs et al. 2012].

Additionally, there is another source of modification to this 4/11 ratio which comes from the QED corrections to the plasma thermodynamic quantities (modification of pressure and energy density and thus of entropy). Taking the high temperature modification leads to the correction added above in the variable FourOverEleven. See e.g. Eq. 41 of [Lopez&Turner 1998] and/or the companion paper (Eq. 56). The effect of incomplete neutrino decoupling is considered further below.

Hubble rate today

Cosmological constant fraction Ω_Λ. Obtained by summing baryons and cold dark matter, given that radiation is negligible today.
This is just for information and it is not used since the cosmological constant is totally negligible for its influence in the expansion rate during BBN.

In[ ]:=
```
1 - (h2Ωb0 + h2Ωc0) / h^2
```

Out[ ]=
```
0.69034764
```

Hubble rate

In[ ]:=
```
H0 := 100 h km / second / Mpc; (* Hubble constant today *)
H100 = 100 km / second / Mpc;
(*Fake Hubble rate given by 100 km/s/Mpc so that h = H0/H100 *)
```

We define two critical densities. One for the actual Hubble rate, and one for the rate at 100km/s/Mpc.

In[•]:=
```
ρcrit := 3./(8 π GN) (H0)^2 (* in g cm^-3 by construction *)
ρcrit100 := 3./(8 π GN) (H100)^2 (* in g cm^-3 by construction *)
```

Density of photons and neutrinos

The Black Body constant is defined as

In[•]:=
```
aBB = π^2/(15 hbar^3 (clight)^5)
```

Out[•]=
```
2.3167357 × 10^28
```

Energy density and number density of CMB today (See appendix A1 in companion paper)

In[•]:=
```
ρCMB0 := aBB (kB TCMB0)^4 ;(* in g cm^-3*)

nCMB0 := 2 Zeta[3]/(π^2 hbar^3 (clight)^3) (kB TCMB0)^3
```

We recover the number of photons per cubic centimeter (410) :

In[•]:=
```
nCMB0
```

Out[•]=
```
410.72685
```

The fraction of energy content due to photons is simply

In[•]:=
```
Ωγ0 := ρCMB0 / ρcrit;
```

For neutrinos, we must take into account the temperature of neutrinos today, the number of neutrinos, and the fact that they are fermions.
See companion paper for details.

In[•]:=
```
Ων0 := Nneu * 7/8 * (FourOverEleven)^(4/3) Ωγ0;
```

The contribution to the energy content is obtained by the ratio between energy densities and critical density. We check that today it is around 0.1%.

In[•]:=
```
Ωγ0 h^2/h2Ωb0
```

Out[•]=
```
0.0011027762
```

Density of baryons

```
In[•]:= He4 := 0.2471 ;(* Chemical composition at the end of BBN. In
                                                                    dans
  principle one should account for He4 produced by stars...*)
xH1 := 1 - He4 ;
mbaryon0 := (xH1 H1Overma + He4 He4Overma / 4) ma;
```

This is the (average) mass of baryons today (that is of nucleons), taking into account that part is in Hydrogen and part in Helium. We use the current chemical composition with 24.75 % of Helium but this is subject to controversy. Indeed the abundance of baryons is measured with CMB, and thus refers to an epoch (z ~1100) where the composition was the same as the one at the end of BBN. See Eqs. C5 C6 in companion paper.

```
In[•]:= mbaryon0 / ma
        ma / mbaryon0
Out[•]=
        1.0060523
Out[•]=
        0.99398413
```

$$\text{In[•]:=}\quad \frac{\left(\frac{\text{He4Overma}}{4} - \text{H1Overma}\right)}{\text{H1Overma}}$$

```
        % * He4
Out[•]=
        -0.0071185158
Out[•]=
        -0.0017589852
```

The number density of baryons, that is of nucleons is then given by the baryons mass density divided by the average mass of baryons.

```
In[•]:= ρB0 := h2Ωb0 * ρcrit100;
```

$$\text{In[•]:=}\quad \text{nbaryons0} := \frac{\rho\text{B0}}{\left(\text{mbaryon0} / (\text{clight})^2\right)}$$

The ratio between baryons number and photons number is by definition the η parameter and its value for the parameters chosen is

$$\text{In[•]:=}\quad \frac{\text{nbaryons0}}{n_{\text{CMB0}}}$$

```
        1 / %
Out[•]=
        6.1388106 × 10^-10
Out[•]=
        1.62898 × 10^9
```

It is convenient to define the ratio between $\Omega_b h^2$ and this η parameter.

```
In[•]:= Ωbh2Over$\eta$ := nCMB0 / ρcrit100 * mbaryon0 / (clight)^2
```

```
In[•]:= Ωbh2Over$\eta$
        % / Meanh2Ωb0Planck
Out[•]=
        3.6529877 × 10^7
Out[•]=
        1.62898 × 10^9
In[•]:= 1 / Ωbh2Over$\eta$
Out[•]=
        2.7374852 × 10^-8
```

The η parameter is then obtained from the baryon density fraction as

```
In[•]:= $\eta$factor := h2Ωb0 / Ωbh2Over$\eta$
```

Baryons density is obtained from its valued today scaled by dilution (no thermal effects, so it is only the energy density due to rest mass of baryons).

```
In[•]:= ρB[av_] := ρB0 / av^3;
        nB[av_] := nbaryons0 / av^3;
```

For nuclear reactions, the mass density of baryons is in fact a number density of species multiplied by the atomic mass (see appendix C1 of companion paper for a detailed discussion).
This differs slightly from the mass density of baryons and we take this into account.

If $CorrectBaryonsEnergyDensityinBBNRRates is set to False, then we use the baryons density naively in nuclear rates.
Otherwise we take into account that the baryons density is in fact the number density times the atomic mass as explained in App. C1 of the companion paper.

```
In[•]:= $CorrectBaryonsEnergyDensityinBBNRRates = True;
                                                  vrai
        ρBForBBN[av_] :=
          ρB[av] × If[$CorrectBaryonsEnergyDensityinBBNRRates, ma / mbaryon0, 1];
                    si
         (* This is Eq. C8 of the companion paper *)
```

Distribution functions

Basic Fermi - Dirac (FD) and Bose - Einstein (BE) functions. x here $1/(k_B T)$.

```
In[•]:= FD[EoverT_] = 1/(Exp[EoverT] + 1);(* Fermi Dirac Distribution *)
        FD[Energy_, x_] = 1/(Exp[x Energy] + 1);
        BE[EoverT_] = 1/(Exp[EoverT] - 1); (* Bose Einstein Distribution *)
        BE[Energy_, x_] = 1/(Exp[x Energy] - 1);


        (* For neutrinos with a chemical potential *)
          boucle for
        FDν[Energy_, ϕ_, x_] = 1/(Exp[x Energy - ϕ] + 1);
```

Derivatives of FD wrt to energy.

```
In[•]:= FDp[Energy_, x_] = D[1/(Exp[x Energy] + 1), Energy];
                             dériver
```

Customized Mathematica tools

This function NP displays a certain number of digits for a given real number

```
In[•]:= NP[number_] := NumberForm[number, 8]
                       forme de nombres
```

This function displays a table in grid form, that is with lines between the entries

```
In[•]:= MyGrid[Table_List] := Grid[Table, Frame → All]
          liste              grille  table  cadre  tout
```

This function performs interpolation on a list of points (x, f(x)) to the required order.

In[ ]:=
```
MyInterpolation[Tab_List] :=
                  |liste
  Interpolation[Tab, InterpolationOrder → InterpOrder];
  |interpolation              |ordre d'interpolation


(* Does not work to interpolate the log
 of rates because it fails when rates vanish !!!*)
MyInterpolationLog[Tab_List] :=
                      |liste
  Function[{x}, Exp[Interpolation[{#[[1]], Log[#[[2]]]} & /@ Tab,
  |fonction      |ex··· |interpolation                  |logarithme
       InterpolationOrder → InterpOrder][x]]];
       |ordre d'interpolation


$InterpolateLogRate = False;
                      |faux
MyInterpolationRate[Tab_List] :=
                      |liste
 If[$InterpolateLogRate, MyInterpolationLog[Tab], MyInterpolation[Tab]]
 |si
```

Tools to avoid too small numbers in numerics.
MyChop chops small numbers and replaces them by 0.

In[ ]:=
```
MyChop[el_?NumericQ] := Chop[el, $MinMachineNumber];
            |expression num··· |remplace ··· |nombre minimal à virgule flotta
SetAttributes[MyChop, Listable];
|alloue attributs             |listable
```

Redefinition of Set to allow to set values to quantities already set

In[ ]:=
```
MySet[Hold[expr_], value_] := (expr = value);
      |maintiens
MySetDelayed[Hold[expr_], value_] := (expr := value);
              |maintiens
```

A function to load a precompile function, or to store it iw it was not computed and stored previously .
If $Recompile=True, then the compilation of the function is performed even if there is a version stored and it tries to delete the previous library.

In[ ]:=
```
UniqueLibraryname[funcLoc_, fnamestring_] :=
  "lib/" <> fnamestring <> "." <> StringSplit[funcLoc, "."][[-1]];
                                  |fractionne chaînes de caractères
(*UniqueLibraryname[funcLoc_]:=
  "lib/compiled"<>ToString[RandomInteger[{1000,9999}]]<>
                    |en chaîne ··· |nombre entier aléatoire
   "."<>StringSplit[funcLoc,"."][[-1]];*)
        |fractionne chaînes de caractères
```

In[ ]:=
```
Clear[LoadCompiledFunction]
|efface
```

```
LoadCompiledFunction[Hold[fname_], Hold[functiondef_],
                     |maintiens      |maintiens
    fnamestring_String] := Module[{funcLoc, ShouldCompile},
    |chaîne de caractères  |module
    If[$StoreCompilation,
    |si
      (*fnamestring=ToString[HoldForm[fname]];*)
                    |en chaîne ⋯ |maintiens forme
      (*Print["fnamestring is ",fnamestring];*)
        |imprime
      mxname = "lib/" <> fnamestring <> ".mx";


      ShouldCompile = False;
                      |faux
      If[Not[FileExistsQ[mxname]], ShouldCompile = True;,
     |si  |né⋯ |fichier existe ?                          |vrai
       Get[mxname];
       |reçois
       If[Not[FileExistsQ[newfuncLoc]],
       |si  |né⋯ |fichier existe ?
         ShouldCompile = True;,
                         |vrai
         If[$Recompile,
         |si
           Print["I will erase the associate library : ", newfuncLoc];
           |imprime  |unité imaginaire
           DeleteFile[newfuncLoc];
           |supprime fichier
           ShouldCompile = True
                           |vrai
          ];
       ];
      ];



      If[ShouldCompile || $Recompile,
     |si


       (*Print[cpf];*)
         |imprime
       fname = functiondef;
       funcLoc = fname[[-1, 1]];
       (*newfuncLoc=UniqueLibraryname[funcLoc];*)
       newfuncLoc = UniqueLibraryname[funcLoc, fnamestring];
       Print["I store the following compiled function : ",
      |imprime  |unité imaginaire
        fnamestring, " in ", mxname, " with library ", newfuncLoc];
       (*newfuncLoc="lib/"<>FileNameTake[funcLoc,-1];*)
                          |prends nom de fichier
       CopyFile[funcLoc, newfuncLoc, OverwriteTarget → False];
      |copie fichier      |cible de réécriture      |faux
```

```
        DumpSave[mxname, {fname, newfuncLoc}];
        sauvegarde décharge
        fname = ReplacePart[fname, {-1, 1} → newfuncLoc];
                remplace partie
        (*Print[fname/. {CompiledFunction→List,LibraryFunction→List}];*),
          imprime       fonction compilée      liste   fonction de bibliothèque  liste
        (*If exists already we load it and
          si
         fo the appropriate replacement of library paths.*)
        Get[mxname];
        reçois
        If[$Verbose, Print["I load compiled function from ",
        si             imprime  unité imaginaire
            mxname, " and the library linked is ", newfuncLoc];];
        fname = ReplacePart[fname, {-1, 1} → newfuncLoc];
                remplace partie
        (*Print[fname/. {CompiledFunction→List,LibraryFunction→List}];*)
          imprime       fonction compilée      liste   fonction de bibliothèque  liste
      ];,
      fname = functiondef;];
  ];

LoadCompiledFunction[Hold[fname_], Hold[functiondef_]] :=
                     maintiens      maintiens
  LoadCompiledFunction[Hold[fname],
                       maintiens
    Hold[functiondef], ToString[HoldForm[fname]]];
    maintiens        en chaîne ⋯ maintiens forme
```

Personal simple integral with second order polynomial interpolation (Simpson method).

In[ ]:=

```
Clear[TableSimpsonCDef, TableSimpsonC]
efface
(*The function which builds the compile function*)
TableSimpsonCDef := Compile[
                    compile
   {{a, _Real}, {b, _Real}, {Np, _Integer}}, With[{h = 1. (b - a) / Np, n2 = Np / 2},
        nombre réel  nombre réel  nombre entier  avec
    With[{h3 = h / 3.}, Join[{{a, h3}}, Table[{a + 2. j h, 2 h3}, {j, 1, n2 - 1}],
     avec               joins          table
      Table[{a + (2. j - 1) h, 4 h3}, {j, 1, n2}], {{b, h3}}]]],
      table
   CompilationTarget → "C", "RuntimeOptions" → "Speed",
   cible de compilation      con⋯ options de durée d'exécution
   Parallelization → True];
   parallélisation   vrai
```

In[ ]:=
```
(* The function which attaches it to the one we
 want (and possibly read on disk if stored previously)*)
TableSimpsonC :=
  (LoadCompiledFunction[Hold[TableSimpsonC], Hold[TableSimpsonCDef]];
                        maintiens             maintiens
   TableSimpsonC);
```

In[ ]:=
```
TableSimpsonC[1, 2, 10^4]; // Timing
                              chronométrage
```

```
I load compiled function from lib/TableSimpsonC.mx
  and the library linked is lib/TableSimpsonC.dylib
```

Out[ ]=
```
{0.002511, Null}
```

In[ ]:=
```
(*TableSimpsonC:=TableSimpsonC=
   Compile[{{a,_Real},{b,_Real},{Np,_Integer}},With[{h=1.(b-a)/Np,n2=Np/2},
   compile          nombre réel   nombre réel   nombre entier   avec
     With[{h3=h/3.},Join[{{a,h3}},Table[{a+2. j h,2 h3},{j,1,n2-1}],
     avec             joins          table
       Table[{a+(2. j-1) h,4 h3},{j,1,n2}],{{b,h3}}]]],
       table
    CompilationTarget→"C","RuntimeOptions"→"Speed"];*)
    cible de compilation  co···  options de durée d'exécution
```

Generic compilation of a function and of its integration.

In[ ]:=
```
MyCompile[LV_List, Body_] := Compile[LV, Evaluate[Body],
                             liste       compile  évalue
  "RuntimeOptions" → "Speed", CompilationTarget → "C",
   options de durée d'exécution               cible de compilation   constante C
  CompilationOptions → {"InlineExternalDefinitions" → True},
  options de compilation                                  vrai
  RuntimeAttributes → {Listable}, Parallelization → True
  attributs de durée d'exécution  listable  parallélisation  vrai
  (*See if it improvies or not*)]
```

Compilation of a scalar product.

In[ ]:=
```
Clear[V1dotV2Def, V1dotV2]
efface
V1dotV2Def :=
  Compile[{{V1, _Real, 1}, {V2, _Real, 1}}, V1.V2, CompilationTarget → "C",
  compile          nombre réel   nombre réel   cible de compilation   const
   "RuntimeOptions" → "Speed"(*,Parallelization→True*)];
    options de durée d'exécution  parallélisation  vrai
V1dotV2 := (LoadCompiledFunction[Hold[V1dotV2], Hold[V1dotV2Def]];
                                 maintiens      maintiens
   V1dotV2);
```

In[ ]:=
```
(*V1dotV2[Table[i,{i,1,10000}],Table[i^2,{i,1,10000}]]//Timing*)
          table                  table                  chronométra
```

Compiled version of the Simpson integral

```
In[•]:= IntegrateFunction[fun_, pemin_, pemax_, Np_] :=
          With[{interv = (pemax - pemin) / (Np), tab = TableSimpsonC[pemin, pemax, Np]},
           V1dotV2[tab[[All, 2]], MyChop[fun[tab[[All, 1]]]]];
```

A function to import an external file and which returns an error and quits if the file does not exist.

```
In[•]:= SafeImport[args__] := Module[{out}, out = Catch[Check[Import[args],
                Print["File ", {args}[[1]], " not found. Quiting Kernel."];
                Throw[$Failed];, Import::nffil]];
          If[out === $Failed,
           Beep[];
           MessageDialog["Impossible to load a file. Kernel has been aborted."];
           Quit[]];
          out]
```

Tools for plots. Some useful grid of ticks

```
In[•]:= MyFrameTicksLog = {{Automatic, Automatic},
            {{{Log[10^8], "10^8"}, {Log[10^8.5], "10^8.5"}, {Log[10^9], "10^9"},
              {Log[10^9.5], "10^9.5"}, {Log[10^10], "10^10"}, {Log[10^10.5], "10^10.5"},
              {Log[10^11], "10^11"}, {Log[10^11.5], "10^11.5"}}, Automatic}};

        MyFrameTicks =
          {{Automatic, Automatic}, {{{10^8, "10^8"}, {10^8.5, "10^8.5"}, {10^9, "10^9"},
              {10^9.5, "10^9.5"}, {10^10, "10^10"}, {10^10.5, "10^10.5"},
              {10^11, "10^11"}, {10^11.5, "10^11.5"}}, Automatic}};
```

Names of precomputed weak rates files

We first build the name of the file to store the PEN rates. It includes as a postfix the values of the booleans for the various effects taken or not taken into account.

In[ ]:=
```
LetterFromBoolean[Bool_] := If[Bool, "T", "F"];
                            |si
StringFromBoolean[BoolList_List] :=
                  |liste
  StringJoin[LetterFromBoolean /@ BoolList];
  |joins chaînes de caractères
BooleanSuffixPEN := Module[{string1, string2, string3},
                    |module
   string1 = StringFromBoolean[{$RadiativeCorrections, $RadiativeThermal,
      $FiniteNucleonMass, $CoupledFMandRC, $NEVO, $SpectralDistortions}];
   string2 = If[$DegenerateNeutrinos, "_xi_" <> ToString[ξν], ""] ;
             |si                                |en chaîne de caractères
   string3 = If[$SpectralDistortions && $AnalyticDistortions,
             |si
     "_xidist_" <> ToString[Δξν] <> "_ysz_" <> ToString[YSZ], ""] ;
                  |en chaîne de caractères          |en chaîne de caractères
   string1 <> string2 <> string3
  ];
BooleanSuffixPlasma := StringFromBoolean[{$CompleteQEDPressure, $QEDOe3}]
```

In[ ]:=
```
NamePENFilenp := "CSV/PENRatenp_" <> BooleanSuffixPEN <> ".dat";
NamePENFilepn := "CSV/PENRatepn_" <> BooleanSuffixPEN <> ".dat";
```

Loading neutrino decoupling results from NEVO

We load the file which stores the output of NEVO, and perform needed interpolations.

In[ ]:=
```
LoadNEVOGrid := (If[$NEVO,
                |si
    If[Not[ValueQ[LinearGrid]] || $RecomputeWeakRates,
    |si  |né··· |valeur ?
      FileGrid = "CSV/NEVOGrid" <> $FileSuffix <> ".csv";
      TableGrid = Flatten@SafeImport[FileGrid, "CSV"];
                  |aplatis
      ymin = First@TableGrid;
             |premier
      ymax  = Last@TableGrid;
              |dernier
      GS = Length[TableGrid];
           |longueur
      Ny = GS;
      LinearGrid = DeleteDuplicates[
                   |supprime répétitions
        Join[Table[y, {y, ymin, ymax, (ymax - ymin) / (Ny - 1)}], {ymax}]];
        |joins  |table
     ];
   ];
 )
```

In[ ]:=
```
LoadNEVOGrid
```

If QED effects are taken into account, we import results from a NEVO simulation incorporating the same QED corrections to the pressure and the energy density (but not to the weak rates of the reactions relevant from neutrino decoupling, as it results in a higher order effect).

In NEVO, the "time" variable (proxy for the scale factor) is $x_{\text{NEVO}} \propto m_e\, a$, while in PRIMAT we have used the notation $x = 1/(k_B\, T)$.
To avoid possible confusion we explicitly use xNEVO below.
See e.g. [Froustey et al. 2020] for historical definitions of x, y, z in the context of neutrino decoupling (y_{NEVO} = a p and z_{NEVO}= a T_γ), but this notation is standard in neutrino decoupling literature.

We perform interpolations :

In[ ]:=
```
TreatNEVOFile[fn_] := (
  TableNEVO = TableNEVOLoaded[fn];
  (*First column of NEVO.cvs file is xNEVO*)
  xNEVO[fn] = TableNEVO[[All, 1]];
  (* Second column is zNEVO*)
  zNEVO[fn] = TableNEVO[[All, 2]];
  (* By definition the temperature of photons is related by : *)
  TγNEVO[fn] = me * zNEVO[fn] / xNEVO[fn] / kB;

  (* The heating function 𝒩, is the 6th column *)
  HeatingNNEVO[fn] = TableNEVO[[All, 6]];
  𝒩NEVO[fn] = Interpolation[Transpose[
     {me / (kB TγNEVO[fn]), HeatingNNEVO[fn]}], InterpolationOrder → 1];
  (*TcomNEVO[fn]=me/xNEVO[fn]/kB;*)

  Tγmax[fn] = First[TγNEVO[fn]];
  Tγmin[fn] = Last[TγNEVO[fn]];

  TγOFxNEVO[fn] = Interpolation[
    Transpose[{xNEVO[fn], TγNEVO[fn]}], InterpolationOrder → 1];

  (*Columns 3, 4,
  and 5 are the effective temperatures of nu_e, nu_mu and nu_tau *)
  Tνe[fn] = TableNEVO[[All, 3]] * me / xNEVO[fn] / kB;
  Tνμ[fn] = TableNEVO[[All, 4]] * me / xNEVO[fn] / kB;
  Tντ[fn] = TableNEVO[[All, 5]] * me / xNEVO[fn] / kB;
```

```
(* We build interpolation functions of these neutrino temperatures,
as function of photon/plasma temperature *)
(*TνeOFTγNEVO[fn]=
  Interpolation[Transpose[{TγNEVO[fn],Tνe[fn]}],InterpolationOrder→1];*)
(*TνμOFTγNEVO[fn]=
  Interpolation[Transpose[{TγNEVO[fn],Tνμ[fn]}],InterpolationOrder→1];*)
(*TντOFTγNEVO[fn]=
  Interpolation[Transpose[{TγNEVO[fn],Tντ[fn]}],InterpolationOrder→1];*)

(*TνeOFxNEVO[fn]=
 Interpolation[Transpose[{xNEVO[fn],Tνe[fn]}],InterpolationOrder→1];
TνμOFxNEVO[fn]=
 Interpolation[Transpose[{xNEVO[fn],Tνμ[fn]}],InterpolationOrder→1];
TντOFxNEVO[fn]=
 Interpolation[Transpose[{xNEVO[fn],Tντ[fn]}],InterpolationOrder→1];*)

(*We build the ratio of the nu_e temperature with the
 photon temperature, as a function of the photon temperature *)
TνeOverTγOFTγNEVO[fn] = Interpolation[
  Transpose[{TγNEVO[fn], Tνe[fn] / TγNEVO[fn]}], InterpolationOrder → 1];
(* We avoid issues when outside of the interpolation range *)
TνeOverTγOFTγNEVOS[fn][T_?NumericQ] :=
 With[{Tgmax = Tγmax[fn], Tgmin = Tγmin[fn]},
  If[T ≥ Tgmax, TνeOverTγOFTγNEVO[fn][Tgmax], If[T ≥ Tgmin,
    TνeOverTγOFTγNEVO[fn][T], TνeOverTγOFTγNEVO[fn][Tgmin]] ] ];

(* We also build the same type of ratio,
but with the average temperature of neutrinos *)
TνAverageOverTγOFTγNEVO[fn] = Interpolation[
  Transpose[{TγNEVO[fn], (1 / 3 * (Tνe[fn] ^ 4 + Tνμ[fn] ^ 4 + Tντ[fn] ^ 4)) ^ (1 / 4) /
      TγNEVO[fn]}], InterpolationOrder → 1];

TνAverageOverTγOFxNEVO[fn] = Interpolation[
  Transpose[{xNEVO[fn], (1 / 3 * (Tνe[fn] ^ 4 + Tνμ[fn] ^ 4 + Tντ[fn] ^ 4)) ^ (1 / 4) /
```

```
      TγNEVO[fn]}], InterpolationOrder → 1];
                    ordre d'interpolation

TνAverageOverTγOFTγNEVOS[fn][T_?NumericQ] :=
                                expression numérique ?
 With[{Tgmax = Tγmax[fn], Tgmin = Tγmin[fn]},
 avec
  If[T ≥ Tgmax, TνAverageOverTγOFTγNEVO[fn][Tgmax],
  si
   If[T ≥ Tgmin, TνAverageOverTγOFTγNEVO[fn][T],
   si
    TνAverageOverTγOFTγNEVO[fn][Tgmin]] ]];

(* xNEVO as a function of the inverse neutrino temperature. *)
(*xNEVOofβ[fn]=Interpolation[
                interpolation
   Transpose[{me/(kB Tνe[fn]),xNEVO[fn]}],InterpolationOrder→1];*)
   transpose                                  ordre d'interpolation

(*xNEVOofT=Interpolation@Transpose[{TγNEVO,xNEVO}];*)
            interpolation  transpose

(*zNEVOofT[fn]=
 Interpolation[Transpose[{TγNEVO[fn],zNEVO[fn]}],InterpolationOrder→1];
 interpolation  transpose                          ordre d'interpolation

zNEVORescale[fn]=TCMB0/zNEVOofT[fn][Tγmin[fn]];

zNEVOofTSafe[fn][T_?NumericQ]:=With[{Tgmax=Tγmax[fn],Tgmin=Tγmin[fn]},
                    expression n···  avec
  zNEVORescale[fn]* If[T≥Tgmax,zNEVOofT[fn][Tgmax],
                    si
    If[T≥Tgmin,zNEVOofT[fn][T],zNEVOofT[fn][Tgmin]] ]];*)
    si

xNEVOofT[fn] = Interpolation[
               interpolation
  Transpose[{TγNEVO[fn], xNEVO[fn]}], InterpolationOrder → 1];
  transpose                          ordre d'interpolation

xNEVOofTSafe[fn][T_?NumericQ] :=
                    expression numérique ?
 With[{Tgmax = Tγmax[fn], Tgmin = Tγmin[fn]},
 avec
  If[T ≥ Tgmax, xNEVOofT[fn][Tgmax] * (T / Tgmax),
  si
   If[T ≥ Tgmin, xNEVOofT[fn][T], xNEVOofT[fn][Tgmin] * T / Tgmin ]]] ;
   si

(*TcomNEVOofT[fn]=Interpolation[
                interpolation
```

```
    Transpose[{TγNEVO[fn],TcomNEVO[fn]}],InterpolationOrder→1];
  TcomNEVOofTSafe[fn][T_?NumericQ]:=With[{Tgmax=Tγmax[fn],Tgmin=Tγmin[fn]},
    If[T≥Tgmax,TcomNEVOofT[fn][Tgmax]*(T/Tgmax),
     If[T≥Tgmin,TcomNEVOofT[fn][T],TcomNEVOofT[fn][Tgmin]*T/Tgmin ]]] ;*)



  If[$NeedNEVODistortions,
   LogxNEVO[fn] = Log@xNEVO[fn];
   Logxmin[fn] = First@LogxNEVO[fn];
   Logxmax[fn] = Last@LogxNEVO[fn];
   Lengthxrange[fn] = Dimensions[TableNEVO][[1]];

   (* Here the 7 is because the
     interpolation data starts at the 8-th column *)
   (* Since the grid in x is logarithmic,
   and the grid in y is irregular (Gauss-Laguerre Quadrature possibly),
   Mathematica cannot perform a 2D interpolation. To bypass this problem,
   we first interpolate the spectra for each xNEVO*)
   LinearTable2DNue[fn] = Table[Interpolation[
       Transpose[{TableGrid, TableNEVO[[xi, Table[6 + i, {i, 1, GS}]]}]][
      LinearGrid], {xi, 1, Lengthxrange[fn]}];
   (* And we then use these interpolations to resample
     the spectrum at each xNEVO, on a linear grid of y.*)
(* The Log(xNEVO) grid is linearly spaced, and we now have a 2D linearly
    spaced grid that we can interpolate in two dimensions *)
   dfNuelogxy[fn] = Interpolation@Flatten[
      Table[{{Re@LogxNEVO[fn][[lxi]], LinearGrid[[yi]]}, LinearTable2DNue[fn][[
          lxi, yi]]}, {yi, 1, Ny}, {lxi, 1, Lengthxrange[fn]}], 1];
(* If points are outside interpolation we put 0 *)
   dfNuelogxySafey[fn][Lx_, y_] :=
    If[y ≥ ymin && y ≤ ymax, dfNuelogxy[fn][Lx, y], 0];
```

```
  (* If points are before or after the x values of NEVO,
     si
  we use the frozen (initial or final) spectra*)
  dfNuelogxySafexSafey[fn][Lx_, y_] :=
   With[{Lxmax = Logxmax[fn], Lxmin = Logxmin[fn]},
   avec
    If[Lx ≥ Lxmin, If[Lx ≤ Logxmax[fn], dfNuelogxySafey[fn][Lx, y],
    si                si
      dfNuelogxySafey[fn][Lxmax, y]], dfNuelogxySafey[fn][Lxmin, y]] ] ;

  (* Finally, after all this, we obtain a function which depends
   on xNEVO (the scale factor) and the comoving momentum y*)
  dfNuexy[fn][x_, y_] := dfNuelogxySafexSafey[fn][Log@x, y];
                                                    logarithme
 ];)
```

We also interpolate (in 2D) the spectrum (in xNEVO and in comoving momentum y = a p).
Only the spectrum of ν_e is given in the NEVO.csv file, and it starts from the 7 – th column.

The grid in comoving momentum (variable y = a p), is stored in the file NEVOGrid which has been loaded in FileGrid

Let us actually perform the loading of NEVO results:

In[ ]:=
```
LoadNEVOFile[fn_] := (
   TableNEVOLoaded[fn] = SafeImport[fn, "CSV"]);
```

In[ ]:=
```
TempAverageNNbar[T1_, T2_] := ((T1^4 + T2^4) / 2)^(1 / 4)

LoadNEVOTemperature := (If[$NEVO,
                           si
   $LocalSuffix = ((*Print["$NeedNEVODistortions is ",$NeedNEVODistortions];*)
                     imprime
     If[$NeedNEVODistortions, "", "_col_1_7"]);
     si
   If[Not[ValueQ[TνAverageOverTγOFTγNEVOSafe]] || $RecomputeWeakRates,
   si  né···  valeur ?
    LoadNEVOFile[FileNEVOneutrinos];
    LoadNEVOFile[FileNEVOantineutrinos];

    TreatNEVOFile[FileNEVOneutrinos];
    TreatNEVOFile[FileNEVOantineutrinos];

    TνAverageOverTγOFTγNEVOSafe[T_?NumericQ] := TempAverageNNbar[
                                                expression numérique ?
      TνAverageOverTγOFTγNEVOS[FileNEVOneutrinos][T],
      TνAverageOverTγOFTγNEVOS[FileNEVOantineutrinos][T]
     ];

    TνeOverTγOFTγNEVOSafe[T_?NumericQ] := TempAverageNNbar[
                                          expression numérique ?
      TνeOverTγOFTγNEVOS[FileNEVOneutrinos][T],
      TνeOverTγOFTγNEVOS[FileNEVOantineutrinos][T]
     ];

    (*Careful this is overwritten
     in LoadDistortions. Need to be more consistent.*)
    zminN = me / (kB * Tγmax[FileNEVOneutrinos]);
    zmaxN = me / (kB * Tγmin[FileNEVOneutrinos]);
   ];
  ])
```

In[ ]:=
```
LoadNEVOTemperature
```

In[ ]:=
```
LoadNEVO := (
  If[$Verbose, Print["Loading NEVO"]];
  si           imprime
  LoadNEVOGrid;
  LoadNEVOTemperature;
 )
```

In[ ]:=
```
LoadNEVO;
```

```
Loading NEVO
```

We check the final and initial spectrum. We recognize the Fig. 2 of [Mangano et al. 2005] (with oscillations, and only the ν_e)

```
If[$NEVO && $IncompleteNeutrinoDecoupling && $RecomputeWeakRates &&
 $SpectralDistortions, Plot[{100 * dfNuexy[FileNEVOneutrinos][0.03, y],
  100 * dfNuexy[FileNEVOneutrinos][20, y]}, {y, ymin, 12},
 PlotRange → {-1, 6}, Frame → True, FrameLabel → {"y = a p", "100 δf_e"}]]
```

Neutrino spectral distortions

What is stored by NEVO was the deviation with respect to a pure Fermi-Dirac distribution and the neutrino temperature. (Slightly inconsistent to put a chemical potential in neutrinos here, since decoupling was computed without. Similarly it was computed without extra relativistic degrees of freedom). In the following, remember that the energy is in units of the electron mass.

The shape chosen for the Y - type distortion contains 4 times more energy density than the Fermi-Dirac (it is obvious using integrations by parts and valid even if the FD has a chemical potential)

```
MuDistortionNeutrinos[y_, ξ_, dξ_] = (1/(Exp[y - (ξ + dξ)] + 1) - 1/(Exp[y - ξ] + 1));
MuDistortionAntineutrinos[y_, ξ_, dξ_] = (1/(Exp[y + (ξ + dξ)] + 1) - 1/(Exp[y + ξ] + 1));
```

```
(*Integrate[
  y^3 (MuDistortionNeutrinos[y,ξ,dξ]+MuDistortionAntineutrinos[y,ξ,dξ]),
  {y,0,Infinity},Assumptions→Element[ϕ,Reals]]
 FullSimplify[%,Assumptions→Element[ξ,Reals]&&Element[dξ,Reals]]*)
Inty3MuDistortion[ξ_, dξ_] = 1/4 dξ (dξ + 2 ξ) (dξ^2 + 2 dξ ξ + 2 (π^2 + ξ^2));
```

```
YDistortionNeutrinos[y_, ξ_] = Simplify[1 / y^2 D[y^4 D[1/(Exp[y - ξ] + 1), y], y]];

YDistortionAntineutrinos[y_, ξ_] =
  Simplify[1 / y^2 D[y^4 D[1/(Exp[y + ξ] + 1), y], y]];
```

In[ ]:=
```
(*Integrate[y^3 (YDistortionNeutrinos[y,ξ]+YDistortionAntineutrinos[y,ξ]),
  {y,0,Infinity},Assumptions→Element[ξ,Reals]]
 FullSimplify[%,Assumptions→Element[ξ,Reals]]*)
Inty3SZdistortion[ξ_] = 7 π^4/15 + 2 π^2 ξ^2 + ξ^4;
```

In[ ]:=
```
LoadDistortions := (

  If[$SpectralDistortions,

      If[$AnalyticDistortions,

       If[$Verbose, Print["Loading analytic distortions"];];
       (*analytic distortions*)
       (* This is where we put the
     analytic form of distortions in ad hoc analytic form *)

       dFDneuRawy[y_, ξ_, ysz_] =
     MuDistortionNeutrinos[y, ξ, Δξν] + ysz * YDistortionNeutrinos[y, ξ];
       dFDantineuRawy[y_, ξ_, ysz_] = MuDistortionAntineutrinos[
      y, ξ, Δξν] + ysz * YDistortionAntineutrinos[y, ξ];

       (* However this distortion is called with en=E/me,ϕ,
    x=me/(kB Tγ) and znu = me/(kB Tν), hence we build y = en*znu *)
       dFDneuRaw[en_, ϕ_, x_, znu_] := dFDneuRawy[en * znu, ϕ, YSZ];
    (*The function expects the temperature of photons (via x = me/kB Tγ)
     but it is not used in the distortion of neutrinos *)

    dFDantineuRaw[en_, ϕ_, x_, znu_] := dFDantineuRawy[en * znu, ϕ, YSZ];

       (*The energy density in the distortion
     must be added later for the Friedmann equation*)
       (* Replace NeutrinosGenerations by 1 if the
      distortions are only in electronic (anti)-neutrinos. *)

    ρνSD[Tnu_] := NeutrinosGenerations * (kB Tnu)^4/(2 π^2 hbar^3 clight^5)
      (Inty3MuDistortion[ξν, Δξν] + YSZ * Inty3SZdistortion[ξν]);
    zminD = me / (kB * Ti);
    zmaxD = me / (kB * Tf);,
```

```
(* *** *)
If[$NEVO,

  If[$NeedNEVODistortions,

    If[$Verbose, Print["Loading NEVO distortions"];];

    (* Need to be consistent with average chemical potential *)
    (* This is the difference between the distorted spectrum,
and the Fermi-Dirac spectrum at the νe temperature. *)
    (* We recall that the νe
 temperature is defined as the temperature of the
 fermion gas which has the same energy density. *)
    (* In NEVO we have defined the non-distorted spectrum as a
 FD without chemical potential hence we ignore ϕ here *)
    dFDneuRaw[en_, ϕ_, x_, znu_] := (Module[{Tγ, xNEV, y},

      Tγ = me / kB / x;
      xNEV = xNEVOofTSafe[FileNEVOneutrinos][Tγ];
      y = en * xNEV;
      If[y ≥ ymin && y ≤ ymax,
       (1 + dfNuexy[FileNEVOneutrinos][xNEV, y]) / (Exp[y] + 1) - 1 / (Exp[en * znu] + 1), 0]]
     );

    dFDantineuRaw[en_, ϕ_, x_, znu_] := (Module[{Tγ, xNEV, y},

      Tγ = me / kB / x;
      xNEV = xNEVOofTSafe[FileNEVOantineutrinos][Tγ];
      y = en * xNEV;
      If[y ≥ ymin && y ≤ ymax,
       (1 + dfNuexy[FileNEVOantineutrinos][xNEV, y]) / (Exp[y] + 1) -
        1 / (Exp[en * znu] + 1), 0]]
     );];

   (* When using NEVO, the energy density of distortions is 0
by construction (choice of the reference temperature)*)
   ρνSD[Tnu_] := 0;
   zminD = me / (kB * Tγmax[FileNEVOneutrinos]);
```

```
        zmaxD = me / (kB * Tγmin[FileNEVOneutrinos]);
       ];
     ];


Clear[dFDneu];
(* The negative energy corresponds to Pauli Blocking factors,
hence the extra minus sign*)
(* We also treat the cases which are outside the grid of
 interpolations using that spectra are frozen once decoupled *)
dFDneu[en_?NumericQ, ϕ_, x_, znu_, sgnq_] :=
 dFDneuRaw[en, ϕ, x, znu] /; en ≥ 0 && sgnq > 0 && x ≥ zminD && x ≤ zmaxD;
(*Neutrinos on the initial state in n→p reactions*)
dFDneu[en_?NumericQ, ϕ_, x_, znu_, sgnq_] :=
 dFDantineuRaw[en, ϕ, x, znu] /; en ≥ 0 && sgnq < 0 && x ≥ zminD && x ≤ zmaxD;
(*AntiNeutrinos on the initial state in p→n reactions*)
dFDneu[en_?NumericQ, ϕ_, x_, znu_, sgnq_] :=
 -dFDantineuRaw[-en, ϕ, x, znu] /; en < 0 && sgnq > 0 && x ≥ zminD && x ≤ zmaxD;
(*Corresponds to antineutrinos blocking factor.*)
dFDneu[en_?NumericQ, ϕ_, x_, znu_, sgnq_] :=
 -dFDneuRaw[-en, ϕ, x, znu] /; en < 0 && sgnq < 0 && x ≥ zminD && x ≤ zmaxD;
(*Corresponds to neutrinos blocking factor.*)
dFDneu[en_?NumericQ, ϕ_, x_, znu_, sgnq_] := 0 /; x < zminD || x > zmaxD;
SetAttributes[dFDneu, Listable];
];);
```

In[•]:= **LoadDistortions**

It is this function dFDneu that we use later to take into account the effect of distortions in n-p conversions.
It also take into account that the distortion is different in neutrinos and in antineutrinos.

We can visualize the spectral distortion if needed. We check visually that the energy density contained in the spectral distortion is 0, because the effective neutrino temperature accounts for it.

```
In[•]:= If[$RecomputeWeakRates,
        si
 dFDneuRawyT[y_, Tγ_] := With[{x = me / kB / Tγ}, dFDneuRaw[y, 0, x, 1]];
                         avec
 dFDantineuRawyT[y_, Tγ_] := With[{x = me / kB / Tγ}, dFDantineuRaw[y, 0, x, 1]];
                             avec
Plot[{y^3 dFDneuRawyT[y, 10^10.2], y^3 dFDantineuRawyT[y, 10^10.2]},
tracé
  {y, 0.1, 30}, Frame → True]]
                cadre  vrai
```

It is this function δFDν that we use later to take into account the effect of distortions in n-p conversions.

Plasma thermodynamics (a - T relation)

Thermodynamic integrals

Defined in appendix A1 of companion paper. These are the integrals needed to obtain the thermodynamic quantities of FD or BE distributions and correspond to Eqs. A5 in companion paper.

```
In[•]:= Clear[Imn]
        efface

Imn[sgn_][m_, n_][x_] := NIntegrate[ (pe^2 + x^2)^((m-1)/2) pe^(n+1) / (Exp[Sqrt[pe^2 + x^2]] + sgn),
                         intègre numériquement

  {pe, 0, Infinity}, Method → {Automatic, "SymbolicProcessing" → 0}]
       infini        méthode   automatique

ImnT[sgn_][m_, n_][T_] := Imn[sgn][m, n][me / kB T]

(* Interpolations *)
ImnI[sgn_][m_, n_] := ImnI[sgn][m, n] =
  Interpolation@Table[{me / kB Tv, Imn[sgn][m, n][me / kB Tv]}, {Tv, ListT}]
  interpolation      table

ImnIT[sgn_][m_, n_][T_] := ImnI[sgn][m, n][me / kB T]
```

QED corrections to plasma thermodynamics

QED mass corrections

From this, using Eq. 12 and 13 of [Mangano.et.al 2001] (or Eq .35 of [Lopez & Turner 1998] for the mass of the electron), we get the modification to the mass of the electron and of the photon. For

this we ignore the last term in Eq. 12 of [Mangano.et.al 2001] or Eq. 35 of [Lopez&Turner 1998]. See also companion paper (Eqs. 44 and 46).

The mass shift is expressed in units of the electron mass so as to be dimensionless. So what we define as dme2 is really $\delta(m_e)^2 \big/ (m_e)^2$ and similarly for dmγ2 it is $\delta(m_\gamma)^2 \big/ (m_e)^2$

In[ ]:=
```
dme2[T_] := (kB T / me)^2 (2 π αFS / 3 + 4 αFS / π ImnT[1][0, 1][T])
(* Only main part of mass shift *)
dmγ2[T_] := 8 αFS / π ImnT[1][0, 1][T] (kB T / me)^2
```

We perform interpolations of these mass shifts over the relevant range of temperatures. We store it on disk in the files dme2.dat and dmg2.dat if they have never been computed.
If they have already been computed we just load the results (unless the Boolean option $RecomputePlasmaCorrections is set to True). Once having the table of values for the mass shift as a function of temperature, we perform an interpolation

In[ ]:=
```
Loaddme := (
  dme2Tab = Check[Import["CSV/dme2.dat", "TSV"],
            (vérifie  (importe
    Print["Precomputed data not found. We recompute and store the data."];
    (imprime
    $Failed, Import::nffil];
    (échoué  (importe

  dmg2Tab = Check[Import["CSV/dmg2.dat", "TSV"],
            (vérifie  (importe
    Print["Precomputed data not found. We recompute and store the data."];
    (imprime
    $Failed, Import::nffil];
    (échoué  (importe

  Timing[
  (chronométrage
   If[dme2Tab == $Failed || dmg2Tab == $Failed || $RecomputePlasmaCorrections,
   (si  (échoué  (échoué
    If[$Verbose, Print["Recompute dme2 and dmg2."]];
    (si  (imprime

    dme2Tab = Table[{T, dme2[T]}, {T, ListT}];
              (table
    dmg2Tab = Table[{T, dmγ2[T]}, {T, ListT}];
              (table

    Export["CSV/dme2.dat", dme2Tab, "TSV"];
    (exporte
    Export["CSV/dmg2.dat", dmg2Tab, "TSV"];
    (exporte
   ];
   dme2I = MyInterpolation@ToExpression@dme2Tab;
                                (en expression
   dmγ2I = MyInterpolation@ToExpression@dmg2Tab;
                                (en expression
  ];)
```

In[ ]:=
```
Loaddme
```

We define a function which gives a value for all temperature and not just in the range of the interpolation so as to avoid any numerical problem.

In[ ]:=
```
dme2N[T_?NumericQ] := Which[T < Tf, 0, T ≤ Ti , dme2I[T], T > Ti, dme2I[Ti]];
                     (expression num···  (quel
dmγ2N[T_?NumericQ] := Which[T < Tf, 0, T ≤ Ti , dmγ2I[T], T > Ti, dme2I[Ti]];
                     (expression num···  (quel
```

We also define these interpolations in terms of the inverse temperature (in units of electron mass, that is the quantity $x = m_e/(k_B\,T)$)

```
In[•]:= dme2x[x_] := dme2N[me / (kB x)];
```

We plot the result for illustration purposes (only if option $PaperPlots is True).

```
In[•]:= If[$PaperPlots, LogLogPlot[Abs@dme2N[Tv] / Tv^2, {Tv, 10^8, 10^12}, Frame → True,
          FrameLabel → {"T (K)", "δm_e^2/T^2"}, PlotStyle → {Black, Thickness[0.0035]}]]

        If[$PaperPlots,
          Export["Plots/Plotdme2.pdf", Style[%, Magnification → 1], "PDF"];]
```

QED pressure corrections

Pressure corrections are obtained from Eq. 13 of [Heckler 1994] when including only electron mass shift, or Eq. 16 of [Mangano et al. 2001] for both electron mass and photon mass shifts. See also companion paper (around Eqs. 48 and 49) It is made of the dominant term dPa, and the subdominant terms dPb which are the two contributions of Eq. 48 in companion paper.

```
In[•]:= dPa[T_] := dPa[T] = αFS/π (kB T)^4 (-2/3 ImnT[1][0, 1][T] - 2/π^2 (ImnT[1][0, 1][T])^2);
```

For the subdominant contribution we use reduced variables. But contrary to the rest of the code where p stands for p/me here it stands for p/T.

```
In[•]:= Clear[Fdp1dp2Def, Fdp1dp2]

        Fdp1dp2Def := Compile[{{p1, _Real}, {p2, _Real}, {x, _Real}}, Evaluate[With[
            {e1 = √(p1^2 + x^2), e2 = √(p2^2 + x^2)},
            αFS/π^3 x^2 p1^2 p2^2/(p1 p2 e1 e2) Log[Abs[(p1 + p2)/(p1 - p2)]] 1/((Exp[e1] + 1) (Exp[e2] + 1))
          ]], "RuntimeOptions" → "Speed", CompilationTarget → "C"];

        Fdp1dp2 := (LoadCompiledFunction[Hold[Fdp1dp2], Hold[Fdp1dp2Def]];
          Fdp1dp2);
```

In[ ]:=
```
(*Fdp1dp2[p1_,p2_,x_]:=With[
    {e1=Sqrt[p1^2+x^2],e2=Sqrt[p2^2+x^2]},
    αFS/π^3 x^2 p1^2 p2^2/(p1 p2 e1 e2) Log[Abs[(p1+p2)/(p1-p2)]] 1/((Exp[e1]+1)(Exp[e2]+1))
  ];*)


Fdp1dp2N[p1_?NumericQ, p2_?NumericQ, x_] := Fdp1dp2[p1, p2, x];


Clear[dPb]
dPb[Tv_] := dPb[Tv] = (kB Tv)^4 With[{x = me / (kB Tv)},
      0.5 NIntegrate[
        Fdp1dp2N[(p1pp2 + p1mp2) / 2, (p1pp2 - p1mp2) / 2, x]
         + Fdp1dp2N[(p1pp2 - p1mp2) / 2, (p1pp2 + p1mp2) / 2, x],
        {p1mp2, 0.0001, Max[20, 20 * x]}, {p1pp2, 0.0001 + Abs[p1mp2],
         Max[20, 20 * x] + Abs[p1mp2]}, PrecisionGoal → 4]
    ];
```

In[ ]:=
```
If[$PaperPlots, PlotdPadPb = ListLogLogPlot[
    {Table[{Tv, Abs@dPa[Tv] / (kB Tv)^4}, {Tv, ListTRange[10^8.5, 10^11]}],
     Table[{Tv, Abs@dPb[Tv] / (kB Tv)^4}, {Tv, ListTRange[10^8.5, 10^11]}]},
    FrameLabel → {"T (K)", "δP/(k_B T)^4"}, LabelStyle → {FontSize → 12},
    FrameTicks → MyFrameTicksLog,
    PlotStyle → {{Red, Thickness[0.0035]}, {Blue, Dashed, Thickness[0.0035]}},
    Frame → True, FrameStyle → Thickness[0.004],
    Joined → True, PlotRange → {10^-10, 10^-2}]]

If[$PaperPlots,
 Export["Plots/PlotdPadPb.pdf", Style[PlotdPadPb, Magnification → 1], "PDF"];]
```

QED of order e^3. See appendix of [Froustey et al 2020]. Initial formulas from [Bennett et al. 2019].

```
In[•]:= dPe3[T_] := dPe3[T] = (αFS)^(3/2) * (4/3) Sqrt[2 Pi]
                                                racine ⋯ nombre pi
        (kB T)^4 ((ImnT[1][0, 1][T] + ImnT[1][2, -1][T]) / Pi^2)^(3/2);
                                                          nombre pi
```

The pressure is then obtained (restoring the correct dimensions)

```
In[•]:= dP[T_] :=
         dP[T] = dPa[T] + If[$CompleteQEDPressure, dPb[T], 0] + If[$QEDOe3, dPe3[T], 0]
                          si                                    si


        (*dPI:=dPI=Interpolation@Table[{Tv,dP[Tv]},{Tv,ListT}]*)
                    interpolation          table
```

```
In[•]:= LoaddP := (
          dPTab = Check[Import["CSV/dP_" <> BooleanSuffixPlasma <> ".dat", "TSV"],
                  vérifie  importe
            Print["Precomputed data not found. We recompute and store the data."];
            imprime
            $Failed, Import::nffil];
            échoué   importe
            Timing[If[dPTab == $Failed || $RecomputePlasmaCorrections,
            chrono⋯  si              échoué
            If[$Verbose, Print["Recompute dP."]];
            si           imprime
            dPTab = Table[{T, dP[T]}, {T, ListT}];
                    table


            Export["CSV/dP_" <> BooleanSuffixPlasma <> ".dat", dPTab, "TSV"];
            exporte
           ];
           dPI = MyInterpolation@ToExpression@dPTab;
                                 en expression
          ];)
```

We check the high temperature limit, which is given in Eq. 30 of [Lopez&Turner 1998] or Eq. 1 of [Heckler 1994]. See also companion paper.

```
In[•]:= LoaddP
```

In[•]:=
```
dPa[10^12] / (kB 10^12)^4
dPb[10^12] / (kB 10^12)^4
dP[10^12] / (kB 10^12)^4
-(5/288) 4 π αFS
```

Out[•]=
```
-0.0015919089
```

```
I load compiled function from lib/Fdp1dp2.mx
  and the library linked is lib/Fdp1dp2.dylib
```

Out[•]=
$$4.1712452 \times 10^{-9}$$

Out[•]=
```
-0.0014501471
```

Out[•]=
```
-0.0015920354
```

In[•]:=
```
dPe3[10^12] / (kB 10^12)^4
αFS^(3/2) 2/9 Sqrt[Pi/3]
```
racin··· nombre p

Out[•]=
```
0.00014175759
```

Out[•]=
```
0.00014175872
```

QED energy density corrections

Energy density corrections are obtained from the thermodynamic identity ρ = -P + T dP/dT. See Eq. 50 of companion paper.

In[•]:=
```
Clear[dρ]
```
efface
```
dρ[T_] := dρ[T] = -dP[T] + T dPI'[T]
```

```
In[•]:= If[$PaperPlots, Plotdrho = ListLogLogPlot[
        si                      tracé log-log de liste
    {Table[{Tv, Abs@dρ[Tv] / (kB Tv)^4}, {Tv, ListTRange[10^8.5, 10^11]}]},
     table          valeur absolue
    FrameLabel → {"T  (K)", "δρ/(k_B T)^4"},
    étiquette de cadre
    LabelStyle → {FontSize → 12}, FrameTicks → MyFrameTicksLog,
    style d'étiquette   taille de police de ca···  graduations de cadre
    PlotStyle → {{Red, Thickness[0.0035]}, {Blue, Dashed, Thickness[0.0035]}},
    style de tracé   rouge  épaisseur   bleu  en tirets  épaisseur
    Frame → True, FrameStyle → Thickness[0.004],
    cadre   vrai   style de cadre   épaisseur
    Joined → True, PlotRange → {10^-10, 10^-2}]]
    joint    vrai   zone de tracé

If[$PaperPlots,
 si
 Export["Plots/Plotdrho.pdf", Style[Plotdrho, Magnification → 1], "PDF"];]
 exporte                      style          agrandissement          fonction de dens
```

QED modified relativistic degrees of freedom

The modified relativistic degrees of freedom (see [Lopez & Turner 1998] for definition) are [see also Eq. 52 of companion paper]

```
In[•]:= dgP[T_] := dP[T] 90 / (π^2 (kB T)^4);
        dgρ[T_] := dρ[T] 30 / (π^2 (kB T)^4);
```

We check the high temperature limits (Eq. 54 of companion paper)

```
In[•]:= dgP[10^12]
        (*3dgρ[10^12]*)
        -25 αFS / (4 π) + If[$QEDOe3, 20 Sqrt[Pi / 3] αFS^(3/2) / π^2, 0]
        si                racin··  nombre pi
```

```
Out[•]=
-0.013223756
```

```
Out[•]=
-0.013224937
```

We interpolate these relativistic degrees of freedom and we store them in a file ‘dg.dat’. If this file is already present we do not recompute unless the option $RecomputePlasmaCorrections is set to True. We then perform an interpolation in time of the modified relativistic degrees of freedom

In[ ]:=
```
Loaddgs := (
  dgρdgP = Check[Import["CSV/dg_" <> BooleanSuffixPlasma <> ".dat", "TSV"],
           (vérifie  (importe
    Print["Precomputed data not found. We recompute and store the data."];
    (imprime
    $Failed, Import::nffil];
    (échoué   (importe

  Timing[If[dgρdgP == $Failed || $RecomputePlasmaCorrections,
  (chrono···  (si              (échoué

    If[$Verbose, Print["Recompute dgrho and dgP."]];
    (si             (imprime
    dgρTab = Table[{T, dgρ[T]}, {T, ListT}];
             (table
    dgPTab = Table[{T, dgP[T]}, {T, ListT}];
             (table

    dgρdgP = {dgρTab, dgPTab};
    Export["CSV/dg_" <> BooleanSuffixPlasma <> ".dat", dgρdgP, "TSV"];
    (exporte
   ];
   dgρI = MyInterpolation@ToExpression[dgρdgP⟦1⟧];
                           (en expression
   dgPI = MyInterpolation@ToExpression[dgρdgP⟦2⟧];
                           (en expression

  ];)
```

In[ ]:=
```
Loaddgs
```

We also define functions which are valid everywhere so as to avoid numerical problems (indeed at very low and very large temperature δg_ρ and δg_P are constants).

In[ ]:=
```
dgρN[T_?NumericQ] := Which[T < Tf, 0, T ≤ Ti , dgρI[T], T > Ti, dgρI[Ti]];
          (expression num···  (quel
dgPN[T_?NumericQ] := Which[T < Tf, 0, T ≤ Ti , dgPI[T], T > Ti, dgPI[Ti]];
          (expression num···  (quel
```

We define the relativistic degrees of freedom in function of inverse temperature (in units of electron mass), that is a functions of $x = m_e / (k_B\, T)$.

In[ ]:=
```
dgρx[x_] := dgρN[me / (kB x)];
dgPx[x_] := dgPN[me / (kB x)];
```

We reproduce Fig. 14 of [Lopez & Turner 1998]

In[ ]:=
```
If[$PaperPlots, PlotdPdρ = LogLinearPlot[
 {Abs@dgPN[Tv], Abs@dgρN[Tv], 25 αFS / (4 π)}, {Tv, 10^8.5, 10^11},
  Frame → True, FrameLabel → {"T  (K)", "-2δP/P        -2δρ/ρ"},
  LabelStyle → {FontSize → 12}, FrameTicks → MyFrameTicks,
  FrameStyle → Thickness[0.004], PlotStyle → {{Thickness[0.004], Red},
    {Blue, Thickness[0.004], Dashing[{0.018}]}, {Black, Thickness[0.003]}}]]

If[$PaperPlots,
 Export["Plots/PlotdPdrho.pdf", Style[PlotdPdρ, Magnification → 1], "PDF"];]
```

Entropy and energy density of the plasma

Here, we compute the thermodynamics using thermodynamical equilibrium.
Indeed if we assume total neutrino decoupling then the collision rates inside the electrons/protons/photons plasma are so high that it is always both at thermal and chemical equilibrium.

Furthermore there are so many more photons than baryons today that the chemical potential of electrons and positrons can be ignored. See companion paper for a discussion on the chemical potentials of electrons/positrons.

We have two functions to tabulate.

The first function, gives the extra amount of entropy at high temperature due to electrons and positrons, in units of the entropy of photons.
It is noted $\mathcal{S}$ in the companion paper (Eq. 30b). We distinguish the case with and without QED plasma corrections (See Eq. 50 for QED plasma corrections).

In[ ]:=
```
DSTNoQEDT[T_] :=
  With[{x = me / (kB T)}, 1 + 45/(2 π^4) (1/3 Imn[1][0, 3][x] + Imn[1][2, 1][x])];

(*DSTNoQED=MyInterpolation@Table[{T,DSTNoQEDT[T]},{T,ListT}];*)
```

```
In[ ]:= LoadDSTNoQED := (
  DSTNoQEDTab = Check[Import["CSV/DSTNoQED.dat", "TSV"],
                      vérifie  importe
    Print["Precomputed data not found. We recompute and store the data."];
    imprime
    $Failed, Import::nffil];
    échoué     importe
  Timing[If[DSTNoQEDTab == $Failed || $RecomputePlasmaCorrections,
  chrono···  si                       échoué
    If[$Verbose, Print["Recompute S function."]];
    si             imprime
    DSTNoQEDTab = Table[{T, DSTNoQEDT[T]}, {T, ListT}];
                  table


    Export["CSV/DSTNoQED.dat", DSTNoQEDTab, "TSV"];
    exporte
   ];
DSTNoQED = MyInterpolation@ToExpression@DSTNoQEDTab;];)
                                 en expression
```

```
In[ ]:= LoadDSTNoQED
```

```
In[ ]:= DSTQED[Tv_] := (3 dgρN[Tv] + dgPN[Tv]) / 8 + DSTNoQED[Tv];



DST[Tv_] := If[$QEDPlasmaCorrections, DSTQED[Tv], DSTNoQED[Tv]]
            si
DSTN[T_?NumericQ] = Which[T < Tf, 1, T ≤ Ti , DST[T], T > Ti, DST[Ti]];
          expression n···  quel
```

The second function, gives the extra amount of energy density at high temperature due to electrons and positrons, in units of the energy density of photons.
It is noted $\mathcal{E}$ in the companion paper in Eq. 41b. We distinguish the case with and without QED plasma corrections (see Eq. 58 QED plasma corrections).

```
In[ ]:= DρTNoQEDT[T_] := With[{x = me / (kB T)}, 30/π^4 (Imn[1][2, 1][x])];

(*DρTNoQED=MyInterpolation@Table[{T,DρTNoQEDT[T]},{T,ListT}];*)
                                 table
```

```
In[•]:= LoadDρTNoQED := (
          DρTNoQEDTab = Check[Import["CSV/DrhoTNoQED.dat", "TSV"],
                        vérifie  importe
            Print["Precomputed data not found. We recompute and store the data."];
            imprime
            $Failed, Import::nffil];
            échoué   importe
           Timing[If[DρTNoQEDTab == $Failed || $RecomputePlasmaCorrections,
           chrono···  si                    échoué

            If[$Verbose, Print["Recompute Drho."]];
            si           imprime
            DρTNoQEDTab = Table[{T, DρTNoQEDT[T]}, {T, ListT}];
                          table

            Export["CSV/DrhoTNoQED.dat", DρTNoQEDTab, "TSV"];
            exporte
           ];
           DρTNoQED = MyInterpolation@ToExpression@DρTNoQEDTab;];)
                                       en expression

In[•]:= LoadDρTNoQED

In[•]:= DρT[T_] := If[$QEDPlasmaCorrections, dgρN[T]/2, 0] + DρTNoQED[T];
         si
```

We check that the ratio of entropy long before and long after electron/psoitrons annihilation is the famous 4/11, possibly corrected by the QED corrections.

```
In[•]:= DST[10^8] / DST[10^12]
        FourOverEleven // N
                          valeur r

Out[•]=
        0.36451369

Out[•]=
        0.36451285
```

In[ ]:=
```
If[$PaperPlots,
 LogLinearPlot[{DSTNoQED[T] - 1, DρTNoQED[T]},
  {T, 10^8, 10^12}, Frame → True, FrameStyle → Thickness[0.004],
  FrameLabel → {"T(K)", "S-1      E-1"}, LabelStyle → {FontSize → 12},
  GridLines → {{{me / kB, {Darker@Gray, Thickness[0.005]}}}, {}}, PlotStyle →
   {{Red, Thickness[0.0035]}, {Blue, Thickness[0.0035], Dashing[0.01]}}]]
If[$PaperPlots,
 Export["Plots/PlotCalSCalE.pdf", Style[%, Magnification → 1], "PDF"];]
```

In[ ]:=
```
LoadPlasma := (
  If[$Verbose, Print["Loading plasma results."]];
  Loaddme;
  LoaddP;
  Loaddgs;
  LoadDSTNoQED;
  LoadDρTNoQED;
 )
```

Incomplete decoupling of neutrinos (method using heating functions)

We now refine to take into account the incomplete decoupling of neutrinos

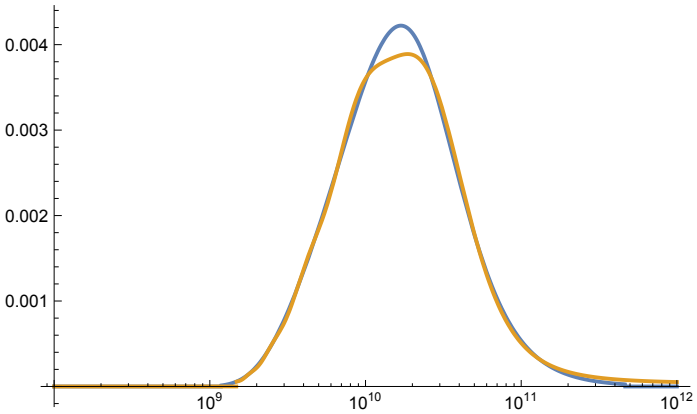
In the previous PRIMAT version, we were using a fit from [PArthENoPE] for the heating function which caracterizes how neutrinos are (slightly) rehetaing due to incomplete decoupling.
Now, if $NEVO option is set to True, we use the one computed by NEVO. The heating function $\mathcal{N}(z)$ of Parthenope is found from Eqs. A.24 – A.25 in [PArthENoPE].

In[ ]:=
```
Listnl = {-10.21703221236002, 61.24438067531452,
   -340.3323864212157, 1057.2707914654834, -2045.577491331372,
   2605.9087171012848, -2266.1521815470196, 1374.2623075963388,
   -586.0618273295763, 174.87532902234145, -35.715878215468045,
   4.7538967685808755, -0.3713438862054167, 0.012908416591272199};

𝒩Parthenope[z_] :=
 If[z ≥ 4, 0, Exp[Plus @@ Table[Listnl⟦i + 1⟧ z^i, {i, 0, 13}]]]
 |si              |ex···  |plus      |table

𝒩[zNEVO_?NumericQ] := If[$NEVO,
                |expression num··· |si
   If[zNEVO < zminN || zNEVO ≥ 5, 0, 𝒩NEVO[FileNEVOneutrinos][zNEVO]],
   |si
   If[zNEVO ≥ 5, 0, 𝒩Parthenope[zNEVO]]
   |si
  ];
```

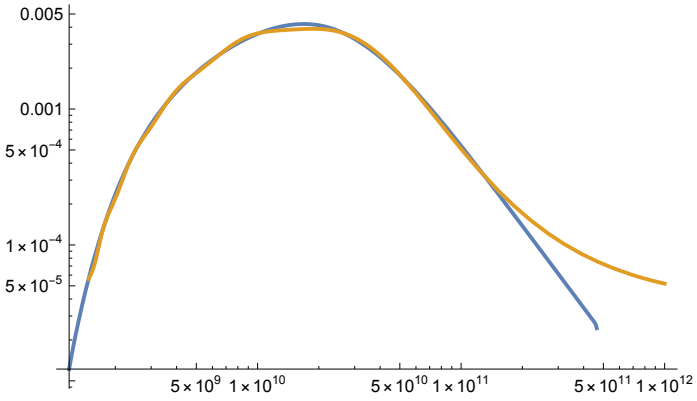
Checking the agreement between $\mathcal{N}_{\text{NEVO}}$ and $\mathcal{N}_{\text{PArthENoPE}}$
(for T > 10 MeV Parthenope fit is not valid, hence the large difference)
We see some disagreement around 1.5 – 2 MeV $\left(\sim 1.5 - 2 \times 10^{10}\ \text{K}\right)$.

In[ ]:=
```
LogLinearPlot[{𝒩[me / (kB * Tv)], 𝒩Parthenope[me / (kB * Tv)]},
 tracé log-linéaire
  {Tv, 10^8, 10^12}, PlotRange → All]
                      zone de tracé   tout
LogLogPlot[{𝒩[me / (kB * Tv)], 𝒩Parthenope[me / (kB * Tv)]},
 tracé log-log
  {Tv, 10^8, 10^12}, PlotRange → All]
                      zone de tracé   tout
```

Out[ ]=



Out[ ]=



We can also perform a fit in Hermite polynomials of the NEVO heating function. Just for information.

```
In[•]:= If[$NEVO,
        si
 Nlz[lz_] := 𝒩NEVO[FileNEVOneutrinos][Exp@lz];
                                        exponentielle

 (* Gram-Charlier expansion. Works better in that case *)
 MyHermiteN[n_, x_] := Expand@Simplify[HermiteH[n, x / Sqrt[2]] / Sqrt[2]^n];
                       développe simplifie  H de Hermite  racine carrée racine carrée

Fitfun = A / (Sqrt[2 Pi] σ) Exp[-(lz - μ)^2 / (2 σ^2)] (1
                racine ⋯ nombr·· exponentielle
      + (S3 / 3! MyHermiteN[3, (lz - μ) / σ])
      + (S4 / 4! MyHermiteN[4, (lz - μ) / σ])
      + (S5 / 5! MyHermiteN[5, (lz - μ) / σ])
      + (S6 / 6! MyHermiteN[6, (lz - μ) / σ])
      + (S7 / 7! MyHermiteN[7, (lz - μ) / σ])
      + (S8 / 8! MyHermiteN[8, (lz - μ) / σ])
      + (S9 / 9! MyHermiteN[9, (lz - μ) / σ])
      + (S10 / 10! MyHermiteN[10, (lz - μ) / σ]));
fitlzmax = 1.7(*-1+2.7*);
 fitlzmin = -4.3(*-1-3.*);
TableVals = Table[{lz, Nlz[lz]}, {lz, fitlzmin, fitlzmax, 0.01}];
             table
Fitvals = FindFit[TableVals, Fitfun, {{A, 0.01}, {μ, -1}, {σ, 1}, {S3, 0},
           trouve ajustement
    {S4, 0}, {S5, 0}, {S6, 0}, {S7, 0}, {S8, 0}, {S9, 0}, {S10, 0}}, lz];
NlzFit[lz_] = Fitfun /. Fitvals;
 NFit[z_] := NlzFit[Log[z]];
                    logarithme
LogLogPlot[{𝒩NEVO[FileNEVOneutrinos][me / (kB * Tv)],
tracé log-log
   𝒩Parthenope[me / (kB * Tv)], NFit[me / (kB * Tv)]},
  {Tv, 1.2 * 10^9, 4 * 10^11}, PlotRange → All, Frame → True];
                                zone de tracé  tout  cadre  vrai

]
```

By construction $\mathcal{N}$ is the heat rate transfered in unit of Hubble rate. Or more precisely, the volumic heat rate (the source on the r.h.s of $d\rho/dt$ equation) is $dq/dt = H\ (kB\ T)^4\ \mathcal{N}$ with plus sign for neutrinos and minus sign for electron / photons plasma.

See companion paper for more details in section II.F.
We transform it to a function of temperature or Log[T].

```
In[•]:= 𝒩T[Tv_] := 𝒩[me / (kB Tv)];
        𝒩lT[lTv_] := 𝒩[me / (kB Exp@lTv)];
                                exponentielle
```

Visualization of the heating period

```
In[•]:= If[$PaperPlots, LogLogPlot[𝒩T[Tv], {Tv, Ti, 10^9}, Frame → True,
        si                 tracé log–log                   cadre    vrai
          FrameStyle → Thickness[0.004], FrameLabel → {"T  (K)", "𝒩(T)"},
          style de cadre    épaisseur                  étiquette de cadre
          LabelStyle → {FontSize → 12}, PlotStyle → {Black, Thickness[0.003]}]]
          style d'étiquette  taille de police de ca···  style de tracé  noir  épaisseur
        If[$PaperPlots,
        si
         Export["Plots/PlotCalN.pdf", Style[%, Magnification → 1], "PDF"];]
         exporte                      style   agrandissement          fonction de dens
```

```
In[•]:= DS2lTQED[lTv_] := 2 * 2 π^2 / 45 DSTQED[Exp@lTv];
                                          exponentielle

        DST2QED[Tv_] := 2 * 2 π^2 / 45 DSTQED[Tv]

        DS2lTNoQED[lTv_] := 2 * 2 π^2 / 45 DSTNoQED[Exp@lTv];
                                             exponentielle

        DST2NoQED[Tv_] := 2 * 2 π^2 / 45 DSTNoQED[Tv]
```

We first solve d ln(aT) / d ln(T) so as to get a(T). This is computed using the fact that there is a cooling of electron/photon plasma due to interactions with neutrinos.
We distinguish the case with and without QED corrections.

The equation solved is Eq. 62 of companion paper. SolveaOFTwhenID calls the solver and can be recalled anytime we have varied parameters.

```
In[ ]:= SolveaOFTwhenID :=
  (laTCQED = NDSolveValue[{laTCN'[lTv] == (𝒩lT[lTv] - DS2lTQED'[lTv]) / (𝒩lT[lTv] + 3 * DS2lTQED[lTv]),
        laTCN[Log@Tf] == Log[TCMB0 / DSTQED[Tf]^(1/3)]}, {laTCN}, {lTv, Log@Ti, Log@Tf},
      PrecisionGoal → 30 (*40s*), AccuracyGoal → 7 (*9*)][[1]];
   laTCNoQED =
    NDSolveValue[{laTCNNoQED'[lTv] == (𝒩lT[lTv] - DS2lTNoQED'[lTv]) / (𝒩lT[lTv] + 3 * DS2lTNoQED[lTv]),
        laTCNNoQED[Log@Tf] == Log[TCMB0 / DSTNoQED[Tf]^(1/3)]},
      {laTCNNoQED}, {lTv, Log@Ti, Log@Tf}, PrecisionGoal → 30
      (*40*), AccuracyGoal → 7 (*9*)][[1]];
  );
```

We call SolveaOFTwhenID at first evaluation

```
In[ ]:= SolveaOFTwhenID
```

```
In[ ]:= aTCQED[Tv_] := Exp[laTCQED[Log@Tv]];
        aCQED[Tv_] := aTCQED[Tv] / Tv;
```

```
In[ ]:= aCQED[Tf] Tf / TCMB0
        aCQED[Ti] Ti / TCMB0
Out[ ]=
        1.
Out[ ]=
        0.71532269
```

```
In[ ]:= aTCNoQED[Tv_] := Exp[laTCNoQED[Log@Tv]];
        aCNoQED[Tv_] := aTCNoQED[Tv] / Tv;
```

```
In[ ]:= aCNoQED[Tf] Tf / TCMB0
Out[ ]=
        1.
```

We invert numerically a(T) to obtain T(a). We also do it for z=AT as a function of a. This is operation is performed when we call the wrapping function InvertaofTwhenID.

In[ ]:=
```
InvertaofTwhenID :=
  (TofaCQED = Interpolation@Table[{aCQED[T], T}, {T, ListT}];
              |interpolation          |table
   TofaCNoQED = Interpolation@Table[{aCNoQED[T], T}, {T, ListT}];
                |interpolation          |table

   aTofaCQED = Interpolation@Table[{aCQED[T], aTCQED[T]}, {T, ListT}];
               |interpolation          |table
   aTofaCNoQED = Interpolation@Table[{aCNoQED[T], aTCNoQED[T]}, {T, ListT}];
                 |interpolation          |table
  );
```

We call it at first evaluation

In[ ]:=
```
InvertaofTwhenID
```

In[ ]:=
```
aC[T_] := If[$QEDPlasmaCorrections, aCQED[T], aCNoQED[T]]
          |si
```

We now solve d (ρ_v) / d ln (a)
(we have to pay attention to units, hence the hbar and clight placed in the system).

In fact we solve for the evolution of a^4 ρ_v as a function of Log[a]. This is Eq. 63 of companion paper. The function SolveρvOFawhenID calls the solver.

In[ ]:=
```
SolveρνOFawhenID :=
 (Timing[a4ρνLogaQED = NDSolveValue[{barρaNQED'[lav] == 1/(hbar^3 clight^5)
                                    (kB aTofaCQED[Exp@lav])^4 𝒩T[TofaCQED[Exp@lav]],
                                    barρaNQED[Log@aCQED@Ti] == aBB (kB aTCQED[Ti])^4 7/8 Nneu},
                                   {barρaNQED}, {lav, Log[aCQED[Ti]], Log[aCQED[Tf]]},
                                   (*Method→"StiffnessSwitching",*)
                                   PrecisionGoal → 13, AccuracyGoal → 43][[1]];];

 Timing[a4ρνLogaNoQED = NDSolveValue[{barρaNNoQED'[lav] == 1/(hbar^3 clight^5)
                                      (kB aTofaCNoQED[Exp@lav])^4 𝒩T[TofaCNoQED[Exp@lav]],
                                      barρaNNoQED[Log@aCNoQED@Ti] == aBB (kB aTCNoQED[Ti])^4 7/8 Nneu},
                                     {barρaNNoQED}, {lav, Log[aCNoQED[Ti]], Log[aCNoQED[Tf]]},
                                     (*Method→"StiffnessSwitching",*)
                                     PrecisionGoal → 13, AccuracyGoal → 43][[1]];];
 );
```

We call SolveρνOFawhenID at first evaluation

In[ ]:=
```
SolveρνOFawhenID
```

In[ ]:=
```
a4ρνCQED[av_] := a4ρνLogaQED[Log@av];

ρνCQED[av_] := a4ρνCQED[av]/av^4;
```

In[ ]:=
```
a4ρνCNoQED[av_] := a4ρνLogaNoQED[Log@av];

ρνCNoQED[av_] := a4ρνCNoQED[av]/av^4;
```

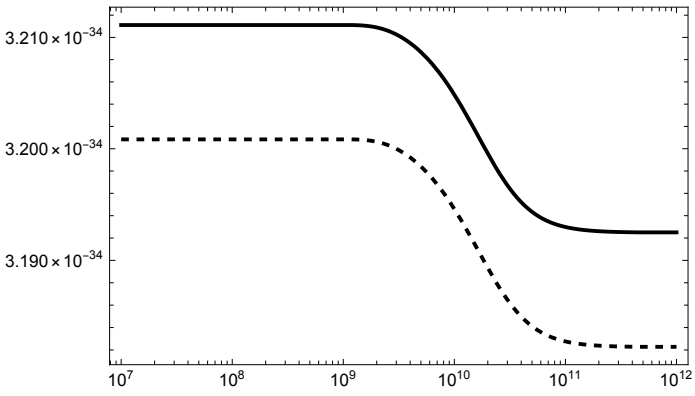
In[ ]:=
```
aTofaCNoQED[a[10^12]] / TCMB0
```

Out[ ]=
```
0.36690516

InterpolatingFunction[ Domain: {{1.95 ×10^-12, 2.73 ×10^-7}}
                       Output: scalar ][a[1 000 000 000 000]]
```

In[ ]:=
```
(*ρνC[av_]:=If[$QEDPlasmaCorrections,ρνCQED[av],ρνCNoQED[av]];*)
                 si

ρνIncompleteDecoupling[av_] :=
  If[$QEDPlasmaCorrections, ρνCQED[av], ρνCNoQED[av]];
  si
```

In[ ]:=
```
LogLogPlot[{a4ρνCQED[aC[Tv]], a4ρνCNoQED[aC[Tv]]},
 tracé log-log
 {Tv, Ti, Tf}, Frame → True, PlotStyle → {Black, {Dashed, Black}}]
                cadre      vrai  style de tracé  noir  en tirets  noir
```

Out[ ]=



We gather all operations needed to compute incomplete neutrino decoupling. Hence the function RecomputeIncompleteNeutrinoDecoupling can be called whenever we change some parameters so as to modify the incomplete decoupling of neutrinos.

In[ ]:=
```
RecomputeIncompleteNeutrinoDecoupling := (
  If[$Verbose, Print["Computing incomplete decoupling of neutrinos."]];
  si                 imprime
  SolveaOFTwhenID;
  InvertaofTwhenID;
  SolveρνOFawhenID;
 )
```

Neutrino Temperature

We extract the corresponding temperature of neutrinos

For full neutrino decoupling it is easy and deduced from entropy conservation.

We choose the correct temperature ratio depending on options chosen and we build the corre-

sponding energy density

```
In[•]:= TνoverTDecoupling[T_] := (FourOverEleven DST[T])^(1/3);
        ρνDecoupling[Tν_] := aBB (kB TνoverTDecoupling[Tν] Tν)^4 7/8 Nneu;
```

When considering incomplete decoupling of neutrinos, we find the temperature variation, which is defined as a brightness temperature. See Eq. 64 of companion paper.

```
In[•]:= TνoverTIncompleteDecouplingQED[Tν_] := (ρνCQED[aCQED[Tν]] / (aBB (kB Tν)^4 7/8 Nneu))^(1/4);

        TνoverTIncompleteDecouplingNoQED[Tν_] := (ρνCNoQED[aCNoQED[Tν]] / (aBB (kB Tν)^4 7/8 Nneu))^(1/4);

        TνoverTIncompleteDecoupling[T_] := If[$QEDPlasmaCorrections,
          TνoverTIncompleteDecouplingQED[T], TνoverTIncompleteDecouplingNoQED[T]]
```

So that now we can decide to use either the full decoupling or the incomplete decoupling neutrino temperature, depending on the options chosen

```
In[•]:= TνAverageoverT[T_] := If[$IncompleteNeutrinoDecoupling,
           If[$TaverageN, TνoverTIncompleteDecoupling[T], If[$NEVO,
             TνAverageOverTγOFTγNEVOSafe[T], TνoverTIncompleteDecoupling[T]]],
           TνoverTDecoupling[T]];


        TνoverT[Tν_] := If[$IncompleteNeutrinoDecoupling,
           If[$TrueTνe && $NEVO, TνeOverTγOFTγNEVOSafe[Tν], TνAverageoverT[Tν]],
           TνoverTDecoupling[Tν]
           ];
```

Comparison of the two implementations of Taverage.

```
In[•]:= (*If[$NEVO&&$IncompleteNeutrinoDecoupling,
          si
         LogLinearPlot[{TνoverTIncompleteDecoupling[Tν],
         tracé log-linéaire
           TνAverageOverTγOFTγNEVOSafe[Tν]},{Tν,10^8,4*10^11},Frame→True]]*)
                                                    cadre  vrai
        (*If[$NEVO&&$IncompleteNeutrinoDecoupling,LogLinearPlot[
          si                                      tracé log-linéaire
          {TνoverTIncompleteDecoupling[Tν]/TνAverageOverTγOFTγNEVOSafe[Tν]-1},
          {Tν,10^8,4*10^11},Frame→True]]*)
                              cadre  vrai
```

Effective Description of Neutrinos

We check the translation into equivalent number of neutrinos. See section II.G of companion paper for the definition of Neff.
This section is only informative.

```
In[•]:= TνoverTDecouplingNoQED[T_] := (FourOverElevenNoQED * DSTNoQED[T])^(1/3);
        TνoverTDecouplingQED[T_] := (FourOverElevenQED * DSTQED[T])^(1/3);

        EffectiveNeutrinosQED[Tν_] :=
          NeutrinosGenerations (TνoverTIncompleteDecouplingQED[Tν] / TνoverTDecouplingNoQED[Tν])^4;
        EffectiveNeutrinosNoQED[Tν_] :=
          NeutrinosGenerations (TνoverTIncompleteDecouplingNoQED[Tν] / TνoverTDecouplingNoQED[Tν])^4;
```

z (z is defined as z=aT with the convention z=1 deep before BBN).

```
In[•]:= zOFTDecouplingNoQED[T_] := (DSTNoQED[Ti] / DSTNoQED[T])^(1/3);
        zOFTDecouplingQED[T_] := (DSTQED[Ti] / DSTQED[T])^(1/3);

        zOFTIncompleteDecouplingNoQED[T_] := (aCNoQED[T] T) / (aCNoQED[Ti] Ti);
        zOFTIncompleteDecouplingQED[T_] := (aCQED[T] T) / (aCQED[Ti] Ti);
```

The various z at the end depending on the physics. See table I in companion paper.

In[•]:=
```
zendQED = zOFTIncompleteDecouplingQED[Tf]
zendNoQED = zOFTIncompleteDecouplingNoQED[Tf]

zendDecouplingQED = zOFTDecouplingQED[Tf]
zendDecouplingNoQED = zOFTDecouplingNoQED[Tf]
(11./4)^(1/3)
```

Out[•]=
```
1.3979705
```

Out[•]=
```
1.3990921
```

Out[•]=
```
1.3998948
```

Out[•]=
```
1.4010185
```

Out[•]=
```
1.4010197
```

Effective number of neutrinos

In[•]:=
```
EffectiveNeutrinosQED[10^8]
EffectiveNeutrinosNoQED[10^8]
3 ((11./4)^(1/3)/zendDecouplingQED)^4
```

Out[•]=
```
3.0438979
```

Out[•]=
```
3.0341584
```

Out[•]=
```
3.0096545
```

Neff from the neutrino energy density

In[•]:=
```
Neff := If[$IncompleteNeutrinoDecoupling, If[$QEDPlasmaCorrections,
   EffectiveNeutrinosQED[10^8], EffectiveNeutrinosNoQED[10^8]],
  If[$QEDPlasmaCorrections, 3 ((11./4)^(1/3)/zendDecouplingQED)^4, 3]]
```

In[•]:=
```
Neff
```

Out[•]=
```
3.0438979
```

The most accurate estimation of Neff, given the options chosen, is (because it uses directly the results of NEVO, and note that it requires NeutrinosGenerations=3 or whatever has been used in NEVO.):

```
In[•]:= Neff := NeutrinosGenerations *
          (TνAverageoverT[10^8] / TνoverTDecouplingNoQED[10^8])^4
```

```
In[•]:= Neff
Out[•]=
3.0439773
```

z_ν at the end of integration
(Only computed in the case of incomplete decoupling
because otherwise it remains unity)

```
In[•]:= zνOFTQED[T_] :=
          TνoverTIncompleteDecouplingQED[T] * zOFTIncompleteDecouplingQED[T];
        zνOFTNoQED[T_] :=
          TνoverTIncompleteDecouplingNoQED[T] * zOFTIncompleteDecouplingNoQED[T];
        zνOFT[T_] := If[$QEDPlasmaCorrections, zνOFTQED[T], zνOFTNoQED[T]]
                     si
```

```
In[•]:= zνendQED = zνOFTQED[10^8]
        zνendNoQED = zνOFTNoQED[10^8]
Out[•]=
1.0014539
Out[•]=
1.0014548
```

The total energy density increase in neutrinos is

```
In[•]:= zνendQED^4
        zνendNoQED^4
Out[•]=
1.0058284
Out[•]=
1.0058317
```

N_{eff} is by definition $(z_\nu\, z_{dec} / z)^4$. See notation in companion paper.
So we recheck that we find the
N_{eff} given in the companion paper in Table I.

In[ ]:=
```
3 * (zvendQED * zendDecouplingNoQED / zendQED)^4

3 * (zvendNoQED * zendDecouplingNoQED / zendNoQED)^4

3 * (zendDecouplingNoQED / zendDecouplingQED)^4
```

Out[ ]=
```
3.0438881
```

Out[ ]=
```
3.0341486
```

Out[ ]=
```
3.0096447
```

However, in the case we use NEVO, the expression given directly above for Neff is exactly the one computed inside NEVO itself.

In[ ]:=
```
If[$PaperPlots, ListLogLinearPlot[
si                tracé log linéaire de liste
  Table[{Tv, TvoverTIncompleteDecouplingQED[Tv] / TvoverTDecouplingQED[Tv]},
  table
    {Tv, ListTRange[10^8, 10^12]}], Joined → True,
                                    joint    vrai
  PlotStyle → Black, FrameStyle → Thickness[0.004],
  style de tracé    noir    style de cadre    épaisseur
  Frame → True, FrameLabel → {"T(K)", "T_ν^ID / T_ν^dec"}]]
  cadre    vrai    étiquette de cadre
If[$PaperPlots,
si
 Export["Plots/PlotTnuvariation.pdf", Style[%, Magnification → 1], "PDF"];]
 exporte                                  style    agrandissement    fonction de de


If[$PaperPlots,
si
 ListLogLinearPlot[Table[{Tv, TvoverT[Tv]}, {Tv, ListTRange[10^8, 10^12]}],
 tracé log linéaire de liste    table
  Joined → True, PlotStyle → Black, FrameStyle → Thickness[0.004],
  joint    vrai    style de tracé    noir    style de cadre    épaisseur
  Frame → True, FrameLabel → {"T(K)", "T_ν^ID / T_ν^dec"}]]
  cadre    vrai    étiquette de cadre
If[$PaperPlots,
si
 Export["Plots/PlotTnuOverT.pdf", Style[%, Magnification → 1], "PDF"];]
 exporte                                  style    agrandissement    fonction de densité
```

```
In[•]:= If[$PaperPlots, ListLogLinearPlot[
        si                tracé log linéaire de liste
   {Table[{Tv, EffectiveNeutrinosQED[Tv]}, {Tv, ListTRange[10^8, 10^12]}],
    table
    Table[{Tv, EffectiveNeutrinosNoQED[Tv]}, {Tv, ListTRange[10^8, 10^12]}],
    table
    Table[{Tv, 3 (zOFTDecouplingNoQED[Tv] / zOFTDecouplingQED[Tv])^4},
    table
     {Tv, ListTRange[10^8, 10^12]}](*,
    Table[{Tv,3(zvOFT[Tv])^4},{Tv,ListTRange[10^8,10^12]}]*)},
    table
   Joined → True, PlotStyle → {Black, {Red, Dashed}, {Blue, Dotted}},
   joint    vrai   style de tracé   noir   rouge en tirets   bleu  en pointillé
   Frame → True, FrameLabel → {"T(K)", "N_eff"}, LabelStyle → {FontSize → 12},
   cadre    vrai   étiquette de cadre                style d'étiquette   taille de police de caractères
   FrameStyle → Thickness[0.004], PlotRange → {2.99, 3.05}]]
   style de cadre   épaisseur              zone de tracé
 If[$PaperPlots,
 si
  Export["Plots/PlotNeff.pdf", Style[%, Magnification → 1], "PDF"];]
  exporte                      style   agrandissement   fonction de densité de prob
```

Scale factor - temperature relation

This is the function which gives the scale factor as a function of the temperature.

1) If neutrino decoupling is total, then the total entropy is conserved and so it is only a function of temperature (entropy density) and scale factor (volume), since S= s a^3.So this can be solved without a differential equation in time.
2) If Incomplete Neutrino decoupling is taken into account, we use the result aC previously obtained, that is we track the evolution of entropy, (see function SolveaOFTwhenID above).

```
In[•]:= a[T_] := If[$IncompleteNeutrinoDecoupling, aC[T], TCMB0 / (T DST[T]^(1/3))];
        si
```

```
In[•]:= (*LogLinearPlot[{a[Tv], TCMB0/Tv},{Tv,10^9,10^10},
          tracé log-linéaire
 Frame→True,FrameLabel→{"T (K)","a(T)/a_0      T_CMB0/T_CMB"}]*)
 cadre   vrai   étiquette de cadre
```

Just for simplicity we define again the z and z_ν variables which are z = aT and z = aT_ν respectively

```
In[•]:= zT[T_] := (a[T] T) / (a[Ti] Ti);

        znuT[T_] := (a[T] × TνoverT[T] T) / (a[Ti] × TνoverT[Ti] Ti);
```

And again we check how they varied

In[ ]:=
```
zT[Tf] // NP
znuT[Tf] // NP
```

Out[ ]//NumberForm=
```
1.3979705
```

Out[ ]//NumberForm=
```
1.001753
```

We build a table with (a, T) so as to obtain the inverse T(a) via an interpolation.
In the incomplete neutrino case we have already performed such inversion so there is a slight loss of time and computer energy.

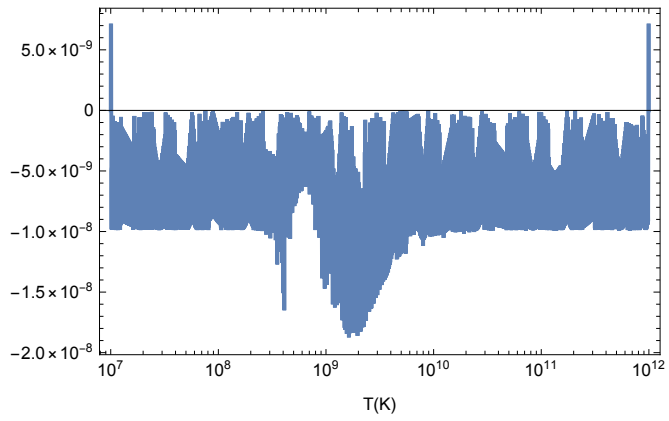
In[ ]:=
```
InvertaOFT := (Tofa = Interpolation@Table[{a[T], T}, {T, ListT}];);
                      interpolation      table
(*aI=Interpolation@Table[{T,a[T]},{T,ListT}];*)
     interpolation      table
```

In[ ]:=
```
InvertaOFT
```

We can check how accurate is this inversion by checking how a(T(a)) is the identity. The error is below 10^{-8} so our numerics are precise enough.

In[ ]:=
```
LogLinearPlot[{Tofa[a[T]] / T - 1},
tracé log-linéaire
 {T, Ti, Tf}, Frame → True, FrameLabel → {"T(K)", ""}]
              cadre    vrai   étiquette de cadre
```

Out[ ]=



η factor with temperature dependence (because photons density has evolved due to electron positron recombination)

In[ ]:=
```
ηfactorT[Tv_] := nB[a[Tv]] * π^2 / (2 Zeta[3]) (hbar clight / (kB Tv))^3;
```

Another way to obtain this quantity

In[ ]:=
```
ηfactorTBis[Tv_] := ηfactor * (zT[Tf] / zT[Tv])^3;
```

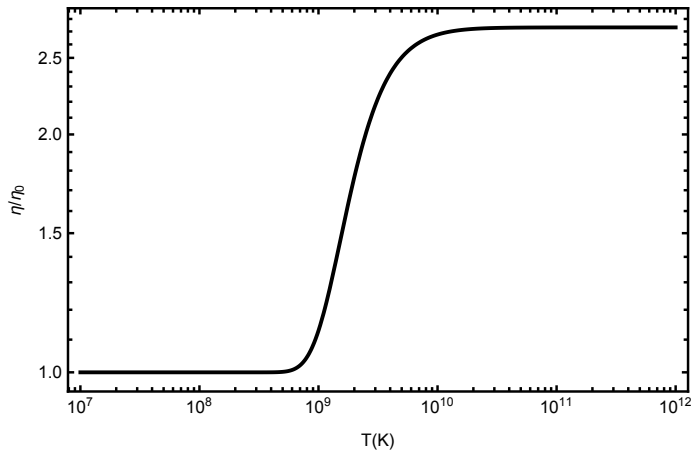
```
In[•]:= ηfactorTBis[Ti] // NP
        ηfactorT[Ti] // NP
```

```
Out[•]//NumberForm=
        1.6771744 × 10^-9
```

```
Out[•]//NumberForm=
        1.6771744 × 10^-9
```

```
In[•]:= LogLogPlot[ηfactorT[Tv] / ηfactor, {Tv, Ti, Tf}, Frame → True,
        |tracé log-log                                  |cadre  |vrai
         FrameStyle → Thickness[0.004], PlotStyle → Black, FrameLabel → {"T(K)", "η/η₀"}]
         |style de cadre  |épaisseur              |style de tracé  |noir  |étiquette de cadre
        If[$PaperPlots,
        |si
         Export["Plots/PlotEta.pdf", Style[%, Magnification → 1], "PDF"];]
         |exporte                    |style  |agrandissement  |fonction de densité de probabilité
```

```
Out[•]=
```



Check of normalization. We chose that today the scale factor is unity.

```
In[•]:= a[Tf] Tf / TCMB0 // NP
```

```
Out[•]//NumberForm=
        1.
```

Cosmology and Friedmann equation (scale factor - time relation)

Energy density

The Friedmann equation gives the Hubble expansion rate.
For completeness we put baryons and CDM even though it makes a difference of order 10^{-5} which is below what we can achieve anyway with homogeneous computations.

$H^2 = \frac{8\pi G\rho}{3} = \frac{\rho}{\rho_{crit}}$ so we build first the energy density.

We select the neutrino energy density depending on options for decoupling

```
In[•]:= ρν[T_] := If[$IncompleteNeutrinoDecoupling, ρνIncompleteDecoupling[a[T]],
                  si
     ρνDecoupling[T]] + If[$SpectralDistortions, ρνSD[TνoverT[T] * T], 0];
                        si
```

If relativistic species added, we must also add them

```
In[•]:= ρνRelat[Tν_] := aBB (kB TνoverTDecoupling[Tν] Tν)^4 7/8 * Nrelat;
(*Tν ~1/a exactly in the decoupling limit hence this corresponds to
 the temperature of an unkown relativistic but decoupled species *)
```

Two methods to get the total energy density. Either from energy density of neutrinos or from temperature. This should be (and it is...) totally equivalent.

```
In[•]:= ρtot1[T_] := (aBB (kB T)^4 (1 + DρT[T]) + ρν[T] + ρνRelat[T]
     + nbaryons0 (mbaryon0 * (1 + h2Ωc0 / h2Ωb0) + 3/2 (kB T)) / (clight)^2 / (a[T])^3);

ρtot2[T_] := (aBB (kB T)^4 (1 + 7/8 Nneu (TνAverageoverT[T])^4 + DρT[T])
     + If[$SpectralDistortions, ρνSD[TνoverT[T] * T], 0] + ρνRelat[T]
       si
     + nbaryons0 (mbaryon0 * (1 + h2Ωc0 / h2Ωb0) + 3/2 (kB T)) / (clight)^2 / (a[T])^3);
```

```
In[•]:= ρtot[T_] := ρtot2[T]; (* If $TaverageN=True,
                                 si          vrai
ρtot1 and ρtot2 are equivalent. In general, we prefer ρtot2
                                 dans
 which uses the effective temperatures (better consistency).*)
```

```
In[•]:= ρtot2[10^10] // Timing
                 chronomé
ρtot1[10^10]
```

```
Out[•]=
{0.000228, 442 826.38}
```

```
Out[•]=
442 829.43
```

We check that we have the correct asymptotic behaviours for the degrees of freedom (Beware that QED corrections alter a bit the results)

```
In[•]:= ρtot[T] / (aBB (kB T)^4) /. T → 10^12 // NP
        (2 + 4 * 7 / 8 + 3 * 2 * 7 / 8) / 2 // N // NP
                                              valeur numériqu
Out[•]//NumberForm=
        5.3683834
Out[•]//NumberForm=
        5.375
```

```
In[•]:= ρtot[T] / (aBB (kB T)^4) /. T → 10^7.5 // NP
        (2 + 3 * 2 * 7 / 8 * (FourOverEleven) ^ (4 / 3)) / 2 // N // NP
                                                    valeur nur
Out[•]//NumberForm=
        1.6918036
Out[•]//NumberForm=
        1.6835124
```

We plot the energy density as a function of temperature.

```
In[•]:= If[$PaperPlots, ListLogLogPlot[
        si                tracé log–log de liste
          Table[{T, 1 / T^4 ρtot[T]}, {T, ListT}], Frame → True, Joined → True,
          table                                    cadre  vrai   joint   vrai
          PlotStyle → Black, FrameLabel → {"T  (K)", "T^-4 ρ(T)     g cm^-3 K^-4"}]]
          style de tracé  noir  étiquette de cadre
        If[$PaperPlots, Export["Plots/PlotρT.pdf", Style[%, Magnification → 1], "PDF"];]
        si              exporte                    style  agrandissement  fonction de
```

Hubble rate and scale factor

Hubble function from Friedmann equation in flat FL space-time is then immediate

```
In[•]:= HofT[T_] := (8 π GN / 3 ρtot[T])^(1/2);

        H[a_] := HofT[Tofa[a]];(*(8π GN / 3 ρtot[Tofa[a]])^(1/2);*)
```

This is the solver for dt/da giving t(a). Direct integration of the Friedmann equation

```
In[•]:= Computetofa :=
          (tofa = (NDSolveValue[{tv'[av] == 1 / (av H[av]), tv[a[Ti]] == 1 / (2 H[a[Ti]])},
          valeur donnée par solution numérique  équation différentielle
            tv, {av, a[Ti], a[Tf]}, PrecisionGoal → 8, AccuracyGoal → 10]);)
                                    objectif de précision  objectif d'exactitude
```

We check that it is quick enough.

In[•]:=
```
Timing@Computetofa
```
chronométrage

Out[•]=
```
{0.028633, Null}
```

This is the solver for the inverse function, that is for da/dt, giving a(t)

In[•]:=
```
Computeaoft :=
 (aoft = NDSolveValue[{av'[tv] == 1/tofa'[av[tv]], av[tofa@a[Ti]] == a[Ti]}, av,
                      {tv, tofa@a[Ti], tofa@a[Tf]}, PrecisionGoal → 13, AccuracyGoal → 25];)
```
valeur donnée par solution numérique équation différentielle
objectif de précision
objectif d'exactitude

We check that it is quick enough.

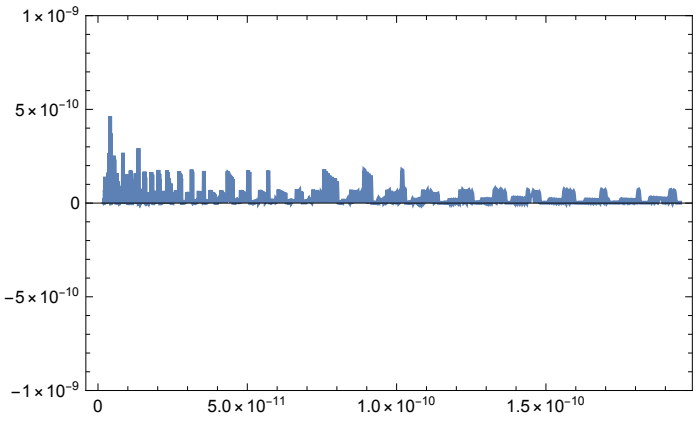
In[•]:=
```
Timing@Computeaoft
```
chronométrage

Out[•]=
```
{0.046055, Null}
```

We also check that computing a(t(a)) gives negligible error to identity. It is of order 10^{-9} so we are totally OK.

In[•]:=
```
Plot[1 / av * aoft@tofa@av - 1, {av, a[Ti], a[Ti] * 100},
 PlotRange → {-10^-9, 10^-9}, Frame → True]
```
tracé
zone de tracé
cadre
vrai

Out[•]=



We define the temperature as a function of time. Since we have T(a) (conserved entropy) and a(t) (Friedman equation), we just combine both.
In case of incomplete decoupling of neutrinos, we could not use conservation of entropy to get T(a) but we used the fit for the heating rate which allowed us to obtain numerically T(a) and a(T).

In[•]:=
```
Toft[tv_] := Tofa[aoft[tv]];
```

Weak reactions n+ν ↔ p+e

Distribution functions derivatives

These Fermi-Dirac function and its derivatives are needed further. en stands for energy. x for $1/(k_B\,T)$.

FDeipj means that it is the Fermi Dirac distribution multiplied by Energy^i and derived j times wrt to Energy. See Def B25 in companion paper.

In[ ]:=
```
(*FDe2p0[en_,x_]=Simplify[FD[en,x]en^2];
FDe3p0[en_,x_]=Simplify[FD[en,x]en^3];
FDe2p2[en_,x_]=Simplify@D[D[FD[en,x]en^2,en],en];

FDe3p2[en_,x_]=Simplify@D[D[FD[en,x]en^3,en],en];
FDe4p2[en_,x_]=Simplify@D[D[FD[en,x]en^4,en],en];
FDe2p1[en_,x_]=Simplify@D[FD[en,x]en^2,en];
FDe3p1[en_,x_]=Simplify@D[FD[en,x]en^3,en];
FDe4p1[en_,x_]=Simplify@D[FD[en,x]en^4,en];*)
```

In[ ]:=
```
FDve2p0[en_, ϕ_, x_] = Simplify[FDv[en, ϕ, x] en^2];
FDve3p0[en_, ϕ_, x_] = Simplify[FDv[en, ϕ, x] en^3];
FDve2p2[en_, ϕ_, x_] = Simplify[D[D[FDv[en, ϕ, x] en^2, en], en]];

FDve3p2[en_, ϕ_, x_] = Simplify[D[D[FDv[en, ϕ, x] en^3, en], en]];
FDve4p2[en_, ϕ_, x_] = Simplify[D[D[FDv[en, ϕ, x] en^4, en], en]];
FDve2p1[en_, ϕ_, x_] = Simplify[D[FDv[en, ϕ, x] en^2, en]];
FDve3p1[en_, ϕ_, x_] = Simplify[D[FDv[en, ϕ, x] en^3, en]];
FDve4p1[en_, ϕ_, x_] = Simplify[D[FDv[en, ϕ, x] en^4, en]];
```

λ_0

Born approximation λ_0

See e.g. Eq. 13 of [Lopez & Turner 1998]. Or Eq. 92 of companion paper.

The constant λ_0 is a proxy for $\tau_n^{-1} = \lambda_0 * \left[\cos^2(\theta_c)\, G_F^2 \left(C_V^2 + 3\, C_A^2\right)/\left(2\,\pi^3\right)\right]$

```
In[•]:= λBORN := λBORN = With[{q = Q / me}, NIntegrate[en (en - q)^2 Sqrt[en^2 - 1], {en, 1, q}]]
```

Corrections to λ_0

Radiative corrections to λ_0

1) If $ResummedLogsRadiativeCorrections=False, we use Eq. 7 of [Czarnecki et al. 2004] which is B30 of companion paper.
Combined with Eq 20b of [Sirlin 1967] for Sirlin's universal function, that is Eq. B32 in companion paper.

```
In[•]:= AgCzarnecki = -0.34;
        CCzarnecki = 0.891;
        mA = 1.2 GeV;
        ConstantSirlin = 4 Log[mZ / mp] + Log[mp / mA] + 2 CCzarnecki + AgCzarnecki;
```

For information, this quantity is evaluated to 40 in [Dicus et al.]

```
In[•]:= ConstantSirlin + 3 Log[mp / (me)] - 3 / 4

Out[•]=
        41.298801
```

```
In[•]:= Rd[x_] := ArcTanh[x] / x;
```

NB : ArcTanh[x] = 1/2 Log[$\frac{(1+x)}{(1-x)}$]

```
In[•]:= Lfun := Lfun = Function[{x},
          Evaluate[Integrate[Log[1 - t] / t, {t, 0, x}, Assumptions → x < 1 && x > 0]]]
        (* Lfun is called the Spence function *)
```

It can be computed from a Taylor expansion (this used to be done in the past), but a direct evaluation by Mathematica is much better

In[ ]:=
```
Clear[LfunSeries]
LfunSeries := LfunSeries = Function[{b}, Evaluate[
    Normal@Series[-1/4*(1+b)^6*4/b Lfun[2 b/(1+b)], {b, 0, 12(*22*)}]]]
```

Sirlin universal function (Eq 20b of [Sirlin 1967]) on which we add the constants of Eq. 7 of [Czarnecki et al. 2004]) is obtained as (this is B32 of companion paper)

In[ ]:=
```
$SeriesSpenceFunction = False;

SirlinGFunction[b_, y_, en_] := (3 Log[mp/(me)] - 3/4 +
    4 (Rd[b] - 1) (y/(3 en) - 3/2 + Log[2 y]) + Rd[b] (2 (1 + b^2) + y^2/(6 en^2) - 4 b Rd[b]) +
    If[$SeriesSpenceFunction, -4/(1+b)^6*LfunSeries[b], 4/b Lfun[2 b/(1+b)]]);

Cd[b_, y_, en_] := (ConstantSirlin + SirlinGFunction[b, y, en]);
```

NB : In [Dicus et al. 1982] Eq 2.14 (or in [Lopez&Turner 1998] Eq. 17) they use the approximation $\frac{4}{b}\,\text{Lfun}\left[\frac{2b}{1+b}\right] = -4\left(2 + 11\,b + \frac{224}{9}\,b^2 + \frac{89}{3}\,b^3 + \frac{1496}{75}\,b^4 + \frac{596}{75}\,b^5 + \frac{128}{49}\,b^6\right) \Big/ (1+b)^6$
However there is a typo and some incorrect coefficients. In [Kernan] there are approximate coefficients which are nearly correct.

2) if $ResummedLogsRadiativeCorrections = True, we use a resummed version of all the Log[m_Z/m_P]. It consists in using Eq. 15 of [Czarnecki 2004] or B35 in companion paper.
See also [Esposito et al. 1998])

We build the multiplicative factor for inner radiative corrections (Eq. 15 of [Czarnecki et al. 2004]), that is B35 of companion paper.

In[ ]:=
```
LFactor = 1.02094;
SFactor = 1.02248;
δfactor = -0.00043*2 Pi/αFS;
NLL = -0.0001;
```

```
RadiativeCorrectionsResummed[b_, y_, en_] :=
  (1 + αFS/(2 π) (SirlinGFunction[b, y, en] - 3 Log[mp/(2 Q)])) *
   (LFactor + αFS/π CCzarnecki + αFS/(2 π) δfactor) *
   (SFactor + 1/(134 * 2 * Pi) * (Log[mp/mA] + AgCzarnecki) + NLL);
```

Finally we define a function which selects either choice depending on option

```
RadiativeCorrections[b_, y_, en_] := If[$ResummedLogsRadiativeCorrections,
  RadiativeCorrectionsResummed[b, y, en], (1 + αFS/(2 π) Cd[b, y, en])]
```

- Fermi function for Coulomb interactions of electron and proton (if on the same side of the interaction).
  Either the relativistic or the non-relativistic depending on option $RelativisticFermiFunction. That is either Eq. 99 or Eq. 100 of companion paper depending on option chosen.

```
FermiRelat[b_] :=
  With[{γ = Sqrt[1 - αFS^2] - 1, λCompton = 1 / (me / (hbar clight))},
   (1 + γ / 2) * 4 (2 radiusproton b/λCompton)^(2 γ) *
    1/Gamma[3 + 2 γ]^2 Exp[π αFS/b] * 1/(1 - b^2)^γ Abs[Gamma[1 + γ + I αFS/b]]^2];

FermiNonRelat[b_] := (2 π αFS / b)/(1 - Exp[-2 π αFS / b]);
```

```
If[$RelativisticFermiFunction,

 Fermi[b_] := FermiRelat[b];
 bFermi[b_] := b Fermi[b];,

 Fermi[b_] := FermiNonRelat[b];
 bFermi[b_] := (2 π αFS)/(1 - Exp[-2 π αFS / b]);]
```

```
In[•]:= If[$PaperPlots,
        si
  DFermi = Plot[{100 * (FermiRelat[b] / FermiNonRelat[b] - 1)}, {b, 0, 1},
           tracé
    Frame → True, FrameStyle → Thickness[0.004], PlotStyle → {Black}, FrameLabel →
    cadre  vrai  style de cadre  épaisseur  style de tracé  noir  étiquette de cadre
     {"x", "100 x (F^rel(x)/F^non rel(x) -1)"}, LabelStyle → {FontSize → 12}]]
                                                  style d'étiquette  taille de police de caractères
  If[$PaperPlots,
  si
    Export["Plots/PlotDeltaFermi.pdf", Style[DFermi, Magnification → 1], "PDF"];];
    exporte                          style  agrandissement  fonction de de
```

λ_0 when taking only Fermi function and not radiative corrections

```
In[•]:= λFermiOnly := λFermiOnly = With[{q = Q / me , b = √(en^2 - 1) / en, y = Q / me - en},
                                   avec
    NIntegrate[en (en - q)^2 en * bFermi[b], {en, 1.0000001, q}]]
    intègre numériquement
```

This already a 3.44 % correction

```
In[•]:= λFermiOnly / (λBORN)
Out[•]=
1.034376
```

Adding the inner radiative corrections, the constant involved in the decay of the neutron is

```
In[•]:= λRad := λRad = With[{q = Q / me , b = √(en^2 - 1) / en, y = Q / me - en},
                       avec
    NIntegrate[en (en - q)^2 en (RadiativeCorrections[b, y, en]) * bFermi[b],
    intègre numériquement
     {en, 1.0000001, q}]]
```

Note that in companion paper, Eq. 106, the value 1.75767 is quoted when it should be the value found here 1.758384.

This adds another 3.90 % correction as found in Eq 16 of [Czarnecki et al. 2004].

```
In[•]:= λRad / λFermiOnly // Timing
                             chronométrage
Out[•]=
{0.500956, 1.0390252}
```

Finite mass corrections to λ_0

See companion papers for expression of finite nucleon mass corrections.
We take the general form of finite mass correction and remove all terms which involve temperature. So we consider Eq. 118 of companion paper.

```
In[•]:= IntegrateCorrectionNeutronDecay[fun_] :=
          NIntegrate[fun[pe], {pe, 0.0000001, Sqrt[(Q / me)^2 - 1]}, WorkingPrecision → 15];
        (* intègre numériquement ; précision de travail *)
```

```
In[•]:= χFMNeutronDecay[en_, pe_] :=
          With[{M = mn / me, enu = en - Q / me,
          (* avec *)
            f1 = ((1 + gA)^2 + 4 fWM gA) / (1 + 3 gA^2), f2 = ((1 - gA)^2 - 4 fWM gA) / (1 + 3 gA^2), f3 = (gA^2 - 1) / (1 + 3 gA^2)},
            f1 * enu^2 (pe^2 / (M * en))
            + f2 * enu^3 (-1 / M)
            + (f1 + f2 + f3) 1 / (2 M) * (4 enu^3 + 2 enu pe^2)
            + f3 * 1 / (3 M) 3 enu^2 pe^2 / (en)
          ];
```

Without coupling to radiative corrections we get

```
In[•]:= IλFMBasic[pe_] := With[{en = Sqrt[pe^2 + 1]}, With[{b = pe / en}, pe^2 *
          (* avec ; racine carrée ; avec *)
            (χFMNeutronDecay[en, pe])
          ]];
```

```
In[•]:= IntegrateCorrectionNeutronDecay[IλFMBasic] / λBORN
Out[•]=
-0.0020647828
```

However we couple to radiative corrections if $CoupledFMandRC = True

```
In[•]:= IλFM[pe_] := With[{en = Sqrt[pe^2 + 1]}, With[{b = pe / en}, pe^2 *
          (* avec ; avec *)
            (χFMNeutronDecay[en, pe] * If[$RadiativeCorrections && $CoupledFMandRC,
            (* si *)
              (RadiativeCorrections[b, Abs[en - Q / me], en]) Fermi[b], 1])
            (* valeur absolue *)
          ]];
```

```
In[•]:= λFM := λFM = If[$FiniteNucleonMass, IntegrateCorrectionNeutronDecay[IλFM], 0]
        (* si *)
```

```
In[•]:= λFM
Out[•]=
-0.00362834063730089
```

When taking only into account Fermi function and finite mass effect on should get 1.6887. See Eq. 6 of [Czarnecki et al. 2004]

```
In[•]:= λFermiOnly + λFM
        1.6887 / % // NP
Out[•]=
        1.6887121
Out[•]//NumberForm=
        0.99999283
```

We compare the correction to the Born result

```
In[•]:= CorrectionRate = λFM / λBORN
Out[•]=
        -0.0022176793
```

If Coulomb and Radiative corrections are set to False (0) this is exactly (-0.00206) what is found by [Lopez & Turner 1997] (see three lines after Eq. 23, in the text).
See also the value -0.00201 found in Eq. 20 of [Seckel 1993].

We also reproduce for comparison the exact finite mass correction to the neutron decay rate as computed by Eq. 19 of [Lopez & Turner 1998], which uses two-dimensional integrals. Again we find the -0.00206 correction so our finite mass method based one-dimensional integrals is reliable.

```
In[•]:= λexact := λexact = With[{pe = Sqrt[en^2 - 1]},
                               (* avec  racine carrée *)
          With[{enu = ((mn^2 - mp^2) / me^2 + 1 - 2 mn / me en) / (2 (mn / me - en + pe Cnu))},
          (* avec *)
           With[{Ep = mn / me - enu - en,
           (* avec *)
             f1 = ((1 + gA)^2 + 4 fWM gA) / (1 + 3 gA^2), f2 = ((1 - gA)^2 - 4 fWM gA) / (1 + 3 gA^2),
             f3 = (gA^2 - 1) / (1 + 3 gA^2)}, With[{J = 1 + (enu + pe Cnu) / (Ep),
                                                    (* avec *)
              M2 = f1 mn / me enu (Ep en - (-pe^2 - pe enu Cnu)) + f2 mn / me en
                   (Ep enu - (-pe enu Cnu - enu^2)) + f3 mn / me Ep (enu en - Cnu enu pe) },
             NIntegrate[1 / 2 * M2 pe enu / (mn / me Ep J), {en, 1,
             (* intègre numériquement *)
               (Q - (Q^2 - me^2) / (2 mn)) / me }, {Cnu, -1, 1}, PrecisionGoal → 10
                                                                 (* objectif de précision *)
              (*,Method→{Automatic,"SymbolicProcessing"→0}*)]
               (* méthode  automatique *)
           ]]]];
In[•]:= (λexact - λBORN) / λBORN // NP
Out[•]//NumberForm=
        -0.0020636787
```

Total correction

The total correction for the neutron decay constant λ_0 is the sum of the Radiative corrected one plus the finite mass effects

We recall the result from radiative corrections (again we stress that Eq. 106 in companion paper should report that value).

In[•]:=
```
λRad
```
Out[•]=
```
1.7583844
```

The result from mass corrections (see text before Eq. 120 in companion paper.)

In[•]:=
```
λFM
```
Out[•]=
```
-0.00362834063730089
```

And we sum them to get Eq. 120 of companion paper.

In[•]:=
```
λRadandFM := λRad + λFM
```

This gives a total correction which is

In[•]:=
```
λRadandFM / λBORN
```
Out[•]=
```
1.072525
```

If we compare with [Cooper et al 2010] (Phys. Rev. C 81, 035503 (2010)). They find that this parameter should be

In[•]:=
```
λCooper = 1.03887 * 1.6887
λCzarnecki = 1.0390 * 1.6887 (* =
     (1+RC)*f with f=1.6887 and RC = 0.0390(8) [Czarnecki et al. 2004]] *)
```
Out[•]=
```
1.7543398
```
Out[•]=
```
1.7545593
```

We compute the ratio to see how far we are

In[•]:=
```
λRadandFM / λCooper
λRadandFM / λCzarnecki
```
Out[•]=
```
1.0002373
```
Out[•]=
```
1.0001121
```

We check that it is in agreement with the theoretical formula. This expression below should reproduce the neutron decay time. It is amazingly close !

In[•]:=
```
MixingCosAngle = 0.97420;
(*  0.97420(+-16) from PDG2017 (The review onf Vud Vus of the PDG 2017).*)
(*The value from PDG2020 review on Vud Vus is 0.97370
 (+-20) but this brings tension on the unitarity of the CKM
 matrix. Hence we stick to the previous value. This is only used
 to check if one recovers from theory the neutrino lifetime.*)
MyK := MyK = MixingCosAngle^2 (GF)^2 (1 + 3 (gA)^2) me^5 / (2 π^3 hbar)
```

In[•]:=
```
1 / MyK / λRadandFM
1 / MyK / λCzarnecki
1 / MyK / λCooper
```

Out[•]=
```
879.38804
```

Out[•]=
```
879.48664
```

Out[•]=
```
879.59669
```

Indeed with the PDG2017 value of Vud, we recover τneutron ~ 879.4 or so hence we feel that the neutron lifetime value is not a much more solid ground than what it was in the pas.

Let us also compare with Eq. 17 of [Czarnecki et al. 2004] to check how close we are in including all corrections. See text after Eq. 120 in companion paper as well.

In[•]:=
```
ConstantVud = hbar * (2 π^3) / (me)^5 / (GF)^2 / λRadandFM
ConstantVud2 = hbar * (2 π^3) / (me )^5 / (GF)^2 / λCzarnecki
ConstantVud3 = hbar * (2 π^3) / (me )^5 / (GF)^2 / λCooper
```

Out[•]=
```
4907.3764
```

Out[•]=
```
4907.9266
```

Out[•]=
```
4908.5407
```

Born rates

We compute in this section the Born rates for n -> p and p -> n.

Tools to perform integration on electron momentum

```
pemin = 0.00001;
pemiddle[x_] :=
  Sqrt[Max[pemin^2, (Q / me)^2 - 1 - If[$QEDMassShift, dme2x[x], 0]]];
pemaxC[x_] := Max[12, 30 / x];
pemax[x_] := Max[12, 30 / x];
```

$TnuEqualT is some cheat to check detailed balance. Should be False. When it is True neutrinos have always the plasma temperature.

```
$TnuEqualT = False;
```

```
IntegratedpNpoints[fun_, sgnq_, Tv_, Npoints_] :=
 With[{x = me / (kB Tv), znu = me / (kB Tv TvoverT[Tv])},
  If[$FastPENRatesIntegrals,
   IntegrateFunction[
    fun[#, x, If[$TnuEqualT, x, znu], sgnq] &, pemin, pemaxC[x], Npoints],
   NIntegrate[fun[pe, x, If[$TnuEqualT, x, znu], sgnq],
    {pe, pemin, pemiddle[x], pemax[x]}]
  ]]

IntegrateRatedp[fun_, sgnq_, Tv_] :=
  IntegratedpNpoints[fun, sgnq, Tv, $PENRatesIntegralsPoints];
```

Energy as a function of momentum, taking into account QED mass shifts.
This is only used if the option is set to True but it is useless because it is taken into account in finite temperature corrections

```
enOFpe[pe_, x_] := Sqrt[pe^2 + 1 + If[$QEDMassShift, dme2x[x], 0]];
```

Functions to build integrands in electron momentum. Without and with CCR corrections. This is then passed to the integrating routine IntegrateRatedp.

```
IPENdpFromχNoCCR[en_, pe_, x_, znu_, sgnq_, χfunction_] :=
  With[{q = Q / me}, With[{b = pe / en},
    pe^2 * (χfunction[en, pe, x, znu, sgnq] + χfunction[-en, pe, x, znu, sgnq])
   ]];
```

```
In[•]:= Fermi[sgnq_, signE_, b_?NumericQ] := If[sgnq signE > 0, Fermi[b], 1];
                                              expression num··· si
        SetAttributes[Fermi, Listable];
        alloue attributs                  listable
```

```
In[•]:= IPENdpFromχCCR[en_, pe_, x_, znu_, sgnq_, χfunction_] :=
          With[{q = Q / me , b = pe / en},
          avec
           pe^2 * (χfunction[en, pe, x, znu, sgnq]
                (RadiativeCorrections[b, Abs[sgnq Q / me - en], en])
                                                 valeur absolue
                Fermi[sgnq, 1, b] + χfunction[-en, pe, x, znu, sgnq]
                (RadiativeCorrections[b, Abs[sgnq Q / me + en], en]) Fermi[sgnq, -1, b])
                                                 valeur absolue
          ];
```

Eq 2.29 in [Brown & Sawyer] for the Born approximation. It is Eq. 79 of companion paper

```
In[•]:= χ[en_, pe_, x_, znu_, sgnq_] :=
          With[{q = Q / me }, FDν[en - sgnq q, sgnq ξν, znu] × FD[-en, x] (en - sgnq q)^2];
          avec
```

Eq 2.30 in [Brown & Sawyer], that is the integration of Eq. 78 in companion paper

```
In[•]:= IPENdp[pe_, x_, znu_, sgnq_] :=
         IPENdpFromχNoCCR[enOFpe[pe, x], pe, x, If[$TnuEqualT, x, znu], sgnq, χ]
                                                 si
```

We also define a function which is incorrect in which we force the neutrinos to have the temperature of photons. This is to check detailed balance. Indeed, detailed balance is satisfied only if all species have the same temperature.

```
In[•]:= IPENdpCheatNeutrinoTemperature[pe_, x_, znu_, sgnq_] :=
         IPENdpFromχNoCCR[Sqrt[pe^2 + 1], pe, x, x, sgnq, χ]
                          racine carrée
```

Born rates are given by Eq 2.30 in [Brown & Sawyer].

```
In[•]:= λnTOpBORN[Tv_] := IntegrateRatedp[IPENdp, 1, Tv];
        λpTOnBORN[Tv_] := IntegrateRatedp[IPENdp, -1, Tv];
```

Born rates where the neutrino temperature is cheated to be equal to photons temperature are then given by

```
In[•]:= λnTOpBORNCheatNeutrino[Tv_] :=
          IntegrateRatedp[IPENdpCheatNeutrinoTemperature, 1, Tv];
        λpTOnBORNCheatNeutrino[Tv_] :=
          IntegrateRatedp[IPENdpCheatNeutrinoTemperature, -1, Tv];
```

Spectral distortions of neutrino spectrum

Shift in the weak rates due to spectral distortions of neutrino distribution functions.

In[•]:=
```
δχ[en_, pe_, x_, znu_, sgnq_] := With[{q = Q / me},
                                      avec
   dFDneu[en - sgnq q, sgnq ξν, x, znu, sgnq] × FD[-en, x] (en - sgnq q)^2];
```

We add this spectral distortions effect at the Born approximation level.

In[•]:=
```
(* Spectral distortions with Born approximation*)
IPENdpSD[pe_, x_, znu_, sgnq_] :=
 IPENdpFromχNoCCR[enOFpe[pe, x], pe, x, If[$TnuEqualT, x, znu], sgnq, δχ]
                                                 si
IPENdpSDCCR[pe_, x_, znu_, sgnq_] :=
 IPENdpFromχCCR[enOFpe[pe, x], pe, x, If[$TnuEqualT, x, znu], sgnq, δχ]
                                                 si


λnTOpSD[Tv_] := IntegrateRatedp[IPENdpSD, 1, Tv];
λpTOnSD[Tv_] := IntegrateRatedp[IPENdpSD, -1, Tv];
λnTOpSDCCR[Tv_] := IntegrateRatedp[IPENdpSDCCR, 1, Tv];
λpTOnSDCCR[Tv_] := IntegrateRatedp[IPENdpSDCCR, -1, Tv];
```

Finite mass effects

For finite mass effects, we use a (Fokker-Planck) expansion which leads to one-dimensional integrals.
We include also the weak-magnetism which is important indeed. This is B23 of companion paper.

In[ ]:=
```
χFM[en_, pe_, x_, znu_, sgnq_] :=
  With[{ϕ = sgnq ξν, q = Q / me , M = (mp + mn  - sgnq Q) / (2 me ),
    Mp = mp / me , Mn = mn  / me , enu = en - sgnq Q / me,
    f1 = ((1 + sgnq gA)^2 + 4 fWM sgnq gA) / (1 + 3 gA^2),
    f2 = ((1 - sgnq gA)^2 - 4 fWM sgnq gA) / (1 + 3 gA^2), f3 = (gA^2 - 1) / (1 + 3 gA^2)},
   f1 * FDνe2p0[enu, ϕ, znu] × FD[-en, x] (pe^2 / (M * en))
   + f2 * FDνe3p0[enu, ϕ, znu] × FD[-en, x] (-1 / M)
   + (f1 + f2 + f3) 1 / (2 x M) *
    (FDνe4p2[enu, ϕ, znu] × FD[-en, x] + FDνe2p2[enu, ϕ, znu] × FD[-en, x] pe^2)
   + (f1 + f2 + f3) 1 / (2 M) *
    (FDνe4p1[enu, ϕ, znu] × FD[-en, x] + FDνe2p1[enu, ϕ, znu] × FD[-en, x] pe^2)
   - (f1 + f2) 1 / (x M) * (FDνe3p1[enu, ϕ, znu] × FD[-en, x] +
      FDνe2p1[enu, ϕ, znu] × FD[-en, x] pe^2 / (-en))
   - f3 * 3 / (x M) FDνe2p0[enu, ϕ, znu] × FD[-en, x]
   (* This term seems to give very small corrections *)
   + f3 * 1 / (3 M) FDνe3p1[enu, ϕ, znu] × FD[-en, x] pe^2 / (en)
   + f3 * 2 / (2 x * 3 M) FDνe3p2[enu, ϕ, znu] × FD[-en, x] pe^2 / (en)
   - (f1 + f2 + f3) * 3 / (2 x) * (1 - (Mn / Mp)^sgnq) * (FDνe2p1[enu, ϕ, znu] × FD[-en, x])
   (* Note that this last line is different from the published
   version of the Physics Report where there is a typo.*)
  ];
```

Integrand for finite mass corrections. We couple it to radiative corrections.

In[ ]:=
```
IPENdpFMNoCCR[pe_, x_, znu_, sgnq_] :=
 IPENdpFromχNoCCR[enOFpe[pe, x], pe, x, If[$TnuEqualT, x, znu], sgnq, χFM]

IPENdpFMCCR[pe_, x_, znu_, sgnq_] :=
 IPENdpFromχCCR[enOFpe[pe, x], pe, x, If[$TnuEqualT, x, znu], sgnq, χFM]
```

Integrand for finite mass corrections when neutrinos are forced to have the same temperature as

photons.

```
In[•]:= IPENdpFMCheatNeutrinoTemperature[pe_, x_, znu_, sgnq_] :=
         IPENdpFromχNoCCR[enOFpe[pe, x], pe, x, x, sgnq, χFM]
```

```
In[•]:= Clear[λnTOpFMCCR, λpTOnFMCCR, λnTOpFMNoCCR,
        efface
         λpTOnFMNoCCR, λnTOpCheatNeutrinoFM, λpTOnCheatNeutrinoFM]
```

Finite mass corrections using the λ_0 which is corrected with radiative corrections. The computation below implements Eqs. 114 of companion paper.

```
In[•]:= λnTOpFMCCR[Tv_] := IntegrateRatedp[IPENdpFMCCR, 1, Tv];
        λpTOnFMCCR[Tv_] := IntegrateRatedp[IPENdpFMCCR, -1, Tv];
```

Finite mass corrections using the λ_0 which is computed with the Born infinite mass expression (we do not advise to use it).

```
In[•]:= λnTOpFMNoCCR[Tv_] := IntegrateRatedp[IPENdpFMNoCCR, 1, Tv];
        λpTOnFMNoCCR[Tv_] := IntegrateRatedp[IPENdpFMNoCCR, -1, Tv];
```

Finite mass corrections cheating with the neutrino temperature and setting it equal to photons temperature

```
In[•]:= λnTOpCheatNeutrinoFM[Tv_] :=
          IntegrateRatedp[IPENdpFMCheatNeutrinoTemperature, 1, Tv];
        λpTOnCheatNeutrinoFM[Tv_] :=
          IntegrateRatedp[IPENdpFMCheatNeutrinoTemperature, -1, Tv];
```

Examples of neutron decay corrections for some low temperatures:

In[•]:= λnTOpFMCCR[10^8] / λnTOpBORN[10^8] // Timing
chronométrage

λnTOpFMCCR[10^7.5] / λnTOpBORN[10^7.5] // Timing
chronomé

```
I load compiled function from lib/V1dotV2.mx
  and the library linked is lib/V1dotV2.dylib
```

General : 75.233517 1.874807739985 $\times 10^{-312}$ is too small to represent as a normalized machine number; precision may be lost.

General : 77.321277 9.490314974612 $\times 10^{-317}$ is too small to represent as a normalized machine number; precision may be lost.

General : 79.437786 4.800075890319 $\times 10^{-321}$ is too small to represent as a normalized machine number; precision may be lost.

General : Further output of General::munfl will be suppressed during this calculation.

Out[•]=
{0.024362, -0.0022393145}

General : 7.4257525 2.002398212180 $\times 10^{-310}$ is too small to represent as a normalized machine number; precision may be lost.

General : 8.081966 8.075703949412 $\times 10^{-324}$ is too small to represent as a normalized machine number; precision may be lost.

General : 8.7665096 3.178774641385 $\times 10^{-337}$ is too small to represent as a normalized machine number; precision may be lost.

General : Further output of General::munfl will be suppressed during this calculation.

Out[•]=
{0.027585, -0.0022257891}

In[•]:= λnTOpFMCCR[.8 × 10^10] / λnTOpBORN[.8 × 10^10] // Timing
chronomé

λnTOpFMCCR[10^10] / λnTOpBORN[10^10] // Timing
chronométrage

λnTOpFMCCR[10^10.5] / λnTOpBORN[10^10.5] // Timing
chronométrage

Out[•]=
{0.004708, -0.0091466758}

Out[•]=
{0.004807, -0.010908969}

Out[•]=
{0.004036, -0.030540512}

In[•]:=
```
λnTOpFMCCR[10^11]
λpTOnFMCCR[10^11]

λnTOpFMCCR[10^10]
λpTOnFMCCR[10^10]

λnTOpFMCCR[6.061*10^9]
λpTOnFMCCR[6.061*10^9]
```

Out[•]=
$$-6.2185433\times 10^{6}$$

Out[•]=
$$-5.2220794\times 10^{6}$$

Out[•]=
$$-12.742449$$

Out[•]=
$$-2.0676676$$

Out[•]=
$$-0.95778352$$

Out[•]=
$$-0.039047307$$

In[•]:=
```
λnTOpFMCCR[4.024*10^7]
λpTOnFMCCR[4.02456*10^7]
```

General : 12.615135 6.064029504912 $\times 10^{-318}$ is too small to represent as a normalized machine number; precision may be lost.

General : 13.470205 1.615566951417 $\times 10^{-328}$ is too small to represent as a normalized machine number; precision may be lost.

General : 14.353768 4.251022329977 $\times 10^{-339}$ is too small to represent as a normalized machine number; precision may be lost.

General : Further output of General::munfl will be suppressed during this calculation.

Out[•]=
$$-0.0036449714$$

General : 12.467878 4.825268063337 $\times 10^{-316}$ is too small to represent as a normalized machine number; precision may be lost.

General : 12.518603 1.1034580530894 $\times 10^{-316}$ is too small to represent as a normalized machine number; precision may be lost.

General : 12.669239 1.404391746448 $\times 10^{-318}$ is too small to represent as a normalized machine number; precision may be lost.

General : Further output of General::munfl will be suppressed during this calculation.

Out[•]=
$$1.270711\times 10^{-169}$$

Plots of the finite mass corrections

We plot the amplitude of the finite mass corrections. This is essentially similar to Fig 8.1 of [Kernan] but the result is a little different since here we have put all the finite nucleon mass effects,

and [Kernan] did not.

In[•]:=
```
If[$PaperPlots,
 si

  TabδλFMnTOp = Table[{T, (λnTOpFMCCR[T] * (τneutron λRadandFM)^-1) / (λnTOpBORN[T] * (τneutron λBORN)^-1)},
                table

    {T, ListTRange[10^9, 10^11]}];
  TabδλFMpTOn = Table[{T, (λpTOnFMCCR[T] * (τneutron λRadandFM)^-1) / (λpTOnBORN[T] * (τneutron λBORN)^-1)},
                table

    {T, ListTRange[10^9, 10^11]}];

TFreeze = 0.8 MeV / kB;

PlotdeltaGammaFM =
  Show[ListLogLinearPlot[{TabδλFMnTOp, TabδλFMpTOn}, Frame → True,
  mon··· tracé log linéaire de liste                  cadre  vrai

    FrameStyle → Thickness[0.004], Joined → True, PlotRange → {-10^-1, 10^-2},
    style de cadre  épaisseur      joint  vrai  zone de tracé

    FrameLabel → {"T (K)", "δΓ/Γ"}, LabelStyle → {FontSize → 12}, PlotStyle →
    étiquette de cadre              style d'étiquette  taille de police de ca··· style de tracé

     {{Black, Thickness[0.003]}, {Black, Dashing[{0.01}], Thickness[0.003]}},
      noir  épaisseur             noir  style de tirets      épaisseur

    GridLines → {{{TFreeze, {Gray, Thickness[0.005]}}}, {}},
    lignes de grille            gris  épaisseur

    FrameTicks → MyFrameTicksLog],
    graduations de cadre

   Graphics[{Rotate[Text[Style["0.8 MeV", FontSize → 10, Black],
   graphique  fais pivo··  texte  style                 taille de police de ···  noir

        {Log@TFreeze - .1, -0.06}], 90 Degree]}]]

]
If[$PaperPlots, Export["Plots/PlotdeltaGammaFM.pdf",
si                 exporte

    Style[PlotdeltaGammaFM, Magnification → 1], "PDF"];];
    style                   agrandissement      fonction de densité de probabilité
```

```
In[•]:= If[$PaperPlots,
  si
 TabδλFMnTOp2 = Table[{T, Identity[(λnTOpFMCCR[T] * (τneutron λRadandFM)^-1) /
                table  identité
      (λnTOpBORN[T] * (τneutron λBORN)^-1)]},
   {T, ListTRange[5 × 10^8, 10^10.5]}];
 TabδλFMpTOn2 =
  Table[{T, Identity[(λpTOnFMCCR[T] * (τneutron λRadandFM)^-1) / (λpTOnBORN[T] *
  table  identité
        (τneutron λBORN)^-1)]}, {T, ListTRange[5 × 10^8, 10^10.5]}];

 PlotdeltaGammaFM2 = Show[ListPlot[{TabδλFMnTOp2, TabδλFMpTOn2}, Frame → True,
                     mon···  tracé de liste                      cadre  vrai
    FrameStyle → Thickness[0.004], Joined → True, PlotRange → {-3 * 10^-2, 10^-2},
    style de cadre  épaisseur  joint  vrai  zone de tracé
    FrameLabel → {"T (10^10 K)", "δΓ/Γ"}, LabelStyle → {FontSize → 12}, PlotStyle →
    étiquette de cadre  style d'étiquette  taille de police de ca···  style de tracé
     {{Red, Thickness[0.003]}, {Blue, Dashing[{0.01}], Thickness[0.003]}},
      rouge  épaisseur  bleu  style de tirets  épaisseur
    FrameTicks → {{Automatic, Automatic}, {{{0.5 * 10^10, ".5"}, {10^10, "1"},
    graduations de cadre  automatique  automatique
        {1.5 * 10^10, "1.5"}, {2 × 10^10, "2"}, {2.5 × 10^10, "2.5"}}, Automatic}},
                                                                    automatique
    GridLines → {{{TFreeze, {Gray, Thickness[0.005]}}}, {}}],
    lignes de grille  gris  épaisseur
   Graphics[{Rotate[Text[Style["0.8 MeV", FontSize → 10, Black],
   graphique  fais pivo··  texte  style  taille de police de ···  noir
       {TFreeze 0.93, -0.02}], 90 Degree]}]]
                                  degré
]

If[$PaperPlots, Export["Plots/PlotdeltaGammaFM2.pdf",
si  exporte
    Style[PlotdeltaGammaFM2, Magnification → 1], "PDF"];];
    style  agrandissement  fonction de densité de probabilité
```

Checking detailed balance

Born approximation detailed balance checks.

```
In[•]:= DetailedBalanceRatio0[T_] := Exp[-Q/(kB T) - ξν];
         exponentielle
```

At Born order, detailed balance is of course satisfied by construction.

```
DetailedBalance0[T_] :=
  (λnTOpBORN[T]) / (λpTOnBORN[T]) * DetailedBalanceRatio0[T];
DetailedBalanceCheatNeutrino0[T_] :=
  (λnTOpBORNCheatNeutrino[T]) / (λpTOnBORNCheatNeutrino[T]) *
   DetailedBalanceRatio0[T];
```

If we enforce $T_\nu = T$ the precision is insane. It is numerical precision because detailed balance is rooted in our method.

```
If[$PaperPlots,
 ListLogLinearPlot[{Table[
    {T, DetailedBalanceCheatNeutrino0[T] - 1}, {T, ListTRange[10^9, 10^12]}]},
  Joined → True, PlotStyle → {Black, {Black, Dashed}}, Frame → True]
]
```

Including corrections in Q/mp,
detailed balance is given by (where α should be set to 0).

```
DetailedBalanceRatio[T_] := Exp[-Q/(kB T) - ξν] (1 + (1 + α) Q/mp)^(3/2);
```

[Lopez&Turner 1997] define some quantity to parameterize how good detailed balance is obtained, this is the parameter α.
α should be 0 so we use it to estimate the failure of detailed balance

We first solve for α correctly, and then we do it when forcing the temperature of neutrinos to be the one of photons, in which case detailed balance must be true. Indeed if we do not force the temperature of neutrinos to be the one of photons, then it is only true before electrons/positrons annihilate, and at low temperature detailed balance is not satisfied.

```
Clear[αLopez, αLopezCheatNeutrino]

αLopez[T_] := αLopez[T] =
  α /. Solve[Normal@Series[(λnTOpBORN[T] + λnTOpFMNoCCR[T]) / (λpTOnBORN[T] +
          λpTOnFMNoCCR[T]) * DetailedBalanceRatio[T], {α, 0, 1}] == 1, α][[1]]

αLopezCheatNeutrino[T_] := αLopezCheatNeutrino[T] = α /.
   Solve[Normal@Series[(λnTOpBORNCheatNeutrino[T] + λnTOpCheatNeutrinoFM[T]) /
           (λpTOnBORNCheatNeutrino[T] + λpTOnCheatNeutrinoFM[T]) *
          DetailedBalanceRatio[T], {α, 0, 1}] == 1, α][[1]]
```

We plot something similar to Fig. 4 of [Lopez&Turner 1997]. The remaining error should come from second order corrections. For instance at low temperature it remains an effect of order $(Q/mn)^2$

so α is expected to differ from a null value by something of order Q/mn =0.002. We also see that below 5x10^{10} K, detailed balance does not work because neutrinos no longer have the temperature of photons but it works if we force neutrinos to be at the temperature of photons.

```
In[•]:= If[$PaperPlots, PlotDetailedBalance = Show[
        si                                    montre
    ListLogLinearPlot[{Table[{T, αLopez[T]}, {T, ListTRange[10^8.5, 10^11.5]}],
    tracé log linéaire de liste      table
      Table[{T, αLopezCheatNeutrino[T]}, {T, ListTRange[10^8.5, 10^11.5]}]},
      table
     Frame → True, FrameTicks → MyFrameTicksLog,
     cadre   vrai   graduations de cadre
     GridLines → {{{TFreeze, {Gray, Thickness[0.005]}}}, {}},
     lignes de grille                  gris   épaisseur
     Joined → True, FrameLabel → {"T (K)", "α"},
     joint        vrai   étiquette de cadre
     FrameStyle → Thickness[0.003], LabelStyle → {FontSize → 12},
     style de cadre   épaisseur          style d'étiquette   taille de police de caractères
     PlotRange → {-1, 1}, PlotStyle → {{Red}, {Black, Dashed, Blue}}],
     zone de tracé          style de tracé   rouge   noir   en tirets   bleu
    Graphics[{Rotate[Text[Style["0.8 MeV", FontSize → 10, Black],
    graphique   fais pivo··  texte  style                 taille de police de ···  noir
        {Log@TFreeze - 0.2, 0.5}], 90 Degree]}]]
          logarithme                degré
]

If[$PaperPlots, Export["Plots/PlotDetailedBalance.pdf",
si                exporte
     Style[PlotDetailedBalance, Magnification → 1], "PDF"];];
     style                      agrandissement          fonction de densité de probabilité
```

Radiative Corrections (T=0)

The CCR corrected rates are described in details in the companion paper.

When electron and protons are on the same side we use the Fermi function to apply the Coulomb corrections.

For all processes the radiative correction are a factor (1+$\frac{\alpha}{2\pi}$ C), (similarly to Eq. 18 of [Lopez&Turner 1998] or Eq. 2.13 of [Dicus .et. al]) as described in companion paper.

```
In[•]:= IPENdpCCR[pe_, x_, znu_, sgnq_] :=
         IPENdpFromχCCR[enOFpe[pe, x], pe, x, If[$TnuEqualT, x, znu], sgnq, χ]
                                                 si
```

```
In[•]:= λnTOpCCR[Tv_] := IntegrateRatedp[IPENdpCCR, 1, Tv];
        λpTOnCCR[Tv_] := IntegrateRatedp[IPENdpCCR, -1, Tv];
```

Plots of the radiative corrections

```
In[•]:= If[$PaperPlots, TabδλnTOp = Table[{T, (λnTOpCCR[T] * (τneutron λRadandFM)^-1)/(λnTOpBORN[T] (τneutron λBORN)^-1) - 1},
  si                      table

    {T, ListTRange[10^8.5, 10^10.5]}];

  TabδλpTOn = Table[{T, (λpTOnCCR[T] * (τneutron λRadandFM)^-1)/(λpTOnBORN[T] (τneutron λBORN)^-1) - 1},
                    table

    {T, ListTRange[10^8.5, 10^10.5]}];

  PlotdeltaGammaCCR =
   Show[ListLogLinearPlot[{TabδλnTOp, TabδλpTOn}, Frame → True,
   mon···  tracé log linéaire de liste                cadre  vrai

     FrameStyle → Thickness[0.004], Joined → True, FrameLabel → {"T (K)", "δΓ/Γ"},
     style de cadre  épaisseur  joint  vrai  étiquette de cadre

     LabelStyle → {FontSize → 12}, PlotStyle → {{Red}, {Blue, Dashing[{0.01}]}},
     style d'étiquette  taille de police de ca···  style de tracé  rouge  bleu  style de tirets

     GridLines → {{{TFreeze, {Gray, Thickness[0.004]}}}, {}},
     lignes de grille  gris  épaisseur

     FrameTicks → {{Automatic, Automatic}, {{{Log[10^8.5], "10^8.5"},
     graduations de cadre  automatique  automatique  logarithme

       {Log[10^9], "10^9"}, {Log[10^9.5], "10^9.5"}, {Log[10^10], "10^10"},
       logarithme  logarithme  logarithme

       {Log[10^10.5], "10^10.5"}, {Log[10^11], "10^11"}}, Automatic}},
       logarithme  logarithme  automatique

    Graphics[{Rotate[Text[Style["0.8 MeV", FontSize → 10, Black],
    graphique  fais pivo··  texte  style  taille de police de ···  noir

      {Log@TFreeze - 0.1, -0.01}], 90 Degree]}]]
       logarithme  degré

]

If[$PaperPlots, Export["Plots/PlotdeltaGammaCCR.pdf",
si  exporte

    Style[PlotdeltaGammaCCR, Magnification → 1], "PDF"];];
    style  agrandissement  fonction de densité de probabilité

(*ListPlot[{TabδλnTOp,TabδλpTOn},Frame→True,Joined→True,
  tracé de liste  cadre  vrai  joint  vrai

   FrameLabel→{"T (K)","δΓ/Γ"},PlotStyle→{Black,{Black,Dashing[{0.01}]}}]
   étiquette de cadre  style de tracé  noir  noir  style de tirets

  Export["Plots/NoLogPlotdeltaGammaCCR.pdf",
  exporte

   Style[%,Magnification→1],"PDF"];*)
   style  agrandissement  fonction de densité de probabilité
```

Finite-temperature Radiative Corrections

Method

We use exclusively [Brown&Sawyer] for the finite temperature radiative corrections,
but the equation are gathered and summarized in companion paper essentially in section III.F and related appendix. The various terms are in Eq. 108.
Note also that on top of that, we also need to add what we called Brehmstrahlung corrections (Eqs. 107 in companion paper)

Real photons processes integrand

Bose Einstein distribution function (and if sq = -1 it is stimulated emission)

In[•]:=
```
BEQ[en_, sq_] := sq BE[sq en];
```

$\tilde{\chi}$ is defined in companion paper in Eq. B45.

In[•]:=
```
χtilde[en_, znu_, sgnq_] :=
 With[{q = Q / me }, FDν[en - sgnq q, sgnq ξν, znu] (en - sgnq q) ^ 2]
 avec
```

We first compute real photons processes (absorption or stimulated emission)
We put Fermi function everywhere to be consistent.
The integrand is (Eq. 109 of companion paper)

In[ ]:=
```
IPENCCRT[en_, k_, x_, znu_, sgnq_] := With[{p = Sqrt[en^2 - 1]},
  With[{b = p / en, A = (2 en^2 + k^2) Log[(en + p)/(en - p)] - 4 p en, B = 2 en Log[(en + p)/(en - p)] - 4 p},
   (αFS/(2 π)) * (BE[x k]/k) *
    (A (FD[-en, x] × Fermi[sgnq, 1, b] (χtilde[en - k, znu, sgnq] +
          χtilde[en + k, znu, sgnq] - 2 χtilde[en, znu, sgnq]) +
        FD[en, x] × Fermi[sgnq, -1, b] (χtilde[-en + k, znu, sgnq] +
          χtilde[-en - k, znu, sgnq] - 2 χtilde[-en, znu, sgnq]))
      - k B * (FD[-en, x] × Fermi[sgnq, 1, b] (χtilde[en - k, znu, sgnq] -
          χtilde[en + k, znu, sgnq]) + FD[en, x] × Fermi[sgnq, -1, b]
        (χtilde[-en + k, znu, sgnq] - χtilde[-en - k, znu, sgnq]))
    )
  ]];

(* Compiled version to compute the integrals slightly faster *)
IPENCCRTC :=
 IPENCCRTC = MyCompile[{{en, _Real}, {k, _Real}, {x, _Real}, {znu, _Real},
    {sgnq, _Integer}}, Evaluate[IPENCCRT[en, k, x, znu, sgnq]]];
IPENCCRTCN[en_?NumericQ, k_, x_, znu_, sgnq_] := IPENCCRTC[en, k, x, znu, sgnq]
```

Bremsstrahlung corrections (needed to obtain all real photons processes and eventually detailed balance).
The integrand is (Eqs. B48 and B49 using definitions B41)

In[ ]:=

```
Clear[IPENCCRDiffBremsstrahlungCN,
 IPENCCRDiffBremsstrahlungC, IPENCCRDiffBremsstrahlung]
IPENCCRDiffBremsstrahlung[en_, k_, x_, znu_, sgnq_] :=
  With[{p = Sqrt[en^2 - 1], q = Q / me},
   With[{b = p / en, A = (2 en^2 + k^2) Log[(en + p)/(en - p)] - 4 p en, B = 2 en Log[(en + p)/(en - p)] - 4 p},
    With[{Fp = A + k B, Fm = A - k B},
     (αFS)/(2 π k)
      ((FD[-en, x] × Fermi[sgnq, 1, b] (Fp χtilde[en + k, znu, sgnq] - If[k < Abs[
                en - sgnq q], Fp FD[en - sgnq q, znu] (Abs[en - sgnq q] - k)^2, 0]))
        + (FD[en, x] × Fermi[sgnq, -1, b]
          (Fm χtilde[-en + k, znu, sgnq] - If[k < Abs[en + sgnq q],
            Fp FD[-en - sgnq q, znu] (Abs[en + sgnq q] - k)^2, 0]))
      )
    ]]];

(* We compile for the integration *)
IPENCCRDiffBremsstrahlungC := IPENCCRDiffBremsstrahlungC = MyCompile[
   {{en, _Real}, {k, _Real}, {x, _Real}, {znu, _Real}, {sgnq, _Integer}},
   Evaluate[IPENCCRDiffBremsstrahlung[en, k, x, znu, sgnq]]];
IPENCCRDiffBremsstrahlungCN[en_?NumericQ, k_, x_, znu_, sgnq_] :=
 IPENCCRDiffBremsstrahlungC[en, k, x, znu, sgnq]
```

The expression IPENFiveBodyT0 implements the integrand of 5.21 of [Brown&Sawyer] (and not 5.20. Careful).
We showed in companion paper that this is only a part of the bremsstrahlung corrections needed.

In[ ]:=
```
IPENFiveBodyT0[en_, k_, x_, znu_, sgnq_] := With[{p = Sqrt[en^2 - 1]},
  With[{A = (2 en^2 + k^2) Log[(en + p)/(en - p)] - 4 p en, B = 2 en Log[(en + p)/(en - p)] - 4 p},
   (αFS/(2 π k)) (FD[en, x]) χtilde[-en + k, znu, sgnq] (A - k B)]]


(* Compiled version *)
IPENFiveBodyT0C :=
  IPENFiveBodyT0C = Compile[{{en, _Real}, {k, _Real}, {x, _Real},
      {znu, _Real}, {sgnq, _Integer}}, Evaluate[With[{p = Sqrt[en^2 - 1]},
      With[{A = (2 en^2 + k^2) Log[(en + p)/(en - p)] - 4 p en, B = 2 en Log[(en + p)/(en - p)] - 4 p},
       (αFS/(2 π k)) (-BE[-k x]) * (FD[en, x]) χtilde[-en + k, znu, sgnq] (A - k B)]]],
    "RuntimeOptions" → "Speed", CompilationTarget → "C"];
IPENFiveBodyT0CN[en_?NumericQ, k_, x_, znu_, sgnq_] :=
 IPENFiveBodyT0C[en, k, x, znu, sgnq]
```

Mass shift and pe+ee integrand

We finally consider the mass shift and ep + ee corrections

We split the contributions of the last two terms of 5.14 in [Brown and Sawyer] in two parts.
Indeed these are given by 5.15 [Brown and Sawyer]. The first part has only one integral, while the second part has two integrals.
The integrand of first part with only one integral is C1dE, and the integrand of the second part with two integrals is C2dE1dE2 (see 5.16 [Brown and Sawyer]).

This is the simple integration in Eq. B50a of companion paper (more precisely the integrand and the integration itself is performed further below)

In[ ]:=
```
C1dE[en_, x_, znu_, sgnq_] := With[{pe = Sqrt[en^2 - 1], q = Q/me},
  -(αFS en)/(2 π pe) * (2 π^2)/(3 x^2) (χ[en, pe, x, znu, sgnq] + χ[-en, pe, x, znu, sgnq])];
```

NB : L is given by 5.18 [Brown and Sawyer]

And this is the double integration integrand of Eq.B50a of companion paper

In[•]:=

```
C2dE1dE2[e1_, e2_, x_, znu_, sgnq_] := With[
    {p1 = Sqrt[e1^2 - 1], p2 = Sqrt[e2^2 - 1], q = Q / me},
    With[{L = Log[(e1 e2 + p1 p2 + 1)/(e1 e2 - p1 p2 + 1)]},
     (αFS/(2 π)) (χ[e1, p1, x, znu, sgnq] + χ[-e1, p1, x, znu, sgnq])
      * (-(1/4) Log[((p1 + p2)/(p1 - p2))^2]^2 *
          (FDp[e2, x] (p2/p1) (e1^2/e2) (e1 + e2) + FD[e2, x] (e1^2/(p1 p2)) (e2 + e1/e2^2))
        + Log[((p1 + p2)/(p1 - p2))^2] (FDp[e2, x] (p2^2 (e1/e2) (1/p1^2 + 2) - e1^2 (p2/p1) L) +
            FD[e2, x] ((e1/(p1^2 e2^2)) (e2^2 + 2 p1^2 + 1) - (e1^2 + e2^2)/(e1 + e2) - (e1^2 e2/(p1 p2)) L))
        - FD[e2, x] (4 e1 (p2/p1) + 2 e2 L))
    ]];


(* Compiled version *)
C2dE1dE2C :=
  C2dE1dE2C = Compile[{{e1, _Real}, {e2, _Real}, {x, _Real}, {znu, _Real},
      {sgnq, _Integer}}, Evaluate[With[{p1 = Sqrt[e1^2 - 1], p2 = Sqrt[e2^2 - 1], q = Q / me},
      With[{L = Log[(e1 e2 + p1 p2 + 1)/(e1 e2 - p1 p2 + 1)]},
       (αFS/(2 π)) (χ[e1, p1, x, znu, sgnq] + χ[-e1, p1, x, znu, sgnq])
        * (-(1/4) Log[((p1 + p2)/(p1 - p2))^2]^2 *
            (FDp[e2, x] (p2/p1) (e1^2/e2) (e1 + e2) + FD[e2, x] (e1^2/(p1 p2)) (e2 + e1/e2^2))
          + Log[((p1 + p2)/(p1 - p2))^2] (FDp[e2, x] (p2^2 (e1/e2) (1/p1^2 + 2) - e1^2 (p2/p1) L) +
```

```
                FD[e2, x] (e1/(p1^2 e2^2) (e2^2 + 2 p1^2 + 1) - (e1^2 + e2^2)/(e1 + e2) - e1^2 e2/(p1 p2) L)
            - FD[e2, x] (4 e1 p2/p1 + 2 e2 L)
        ]]], "RuntimeOptions" → "Speed", CompilationTarget → "C"];
             |options de durée d'exécution  |cible de compilation  |constante C

C2dE1dE2CN[e1_?NumericQ, e2_?NumericQ, x_, znu_, sgnq_] :=
           |expression numéri···  |expression numérique ?
  C2dE1dE2C[e1, e2, x, znu, sgnq];
```

Integrations on momenta

We perform all integrations now that we have expressed their integrands.

TruePhoton -> real photon processes
Diffbremsstrahlung -> bremsstrahlung corrections
Thermal -> mass shift and pe+ee corrections

Let us start with n -> p processes

In[•]:=
```
Clear[λnTOpThermal, λnTOpThermalTruePhoton,
|efface
 λpTOnThermalTruePhoton, λnTOp5bodies, λpTOnThermal,
 λnTOpThermalDiffBremsstrahlung, λpTOnThermalDiffBremsstrahlung]
```

In[•]:=
```
λnTOpThermalTruePhoton[Tv_] := (*λnTOpThermalTruePhoton[Tv]=*)
  With[{x = me/(kB Tv), znu = me/(kB Tv TνoverT[Tv]), q = Q/me},
  |avec
   NIntegrate[IPENCCRTCN[en, k, x, If[$TnuEqualT, x, znu], 1],
   |intègre numériquement                          |si
    {k, 0.001, Max[10, 20/x]}, {en, 1.001, Max[10, 20/x]}, PrecisionGoal → 4]
               |maximum                  |maximum                |objectif de précision
  ];

λnTOpThermalDiffBremsstrahlung[Tv_] :=
  (*λnTOpThermalDiffBremsstrahlung[Tv]=*)
  With[{x = me/(kB Tv), znu = me/(kB Tv TνoverT[Tv]), q = Q/me},
  |avec
   NIntegrate[IPENCCRDiffBremsstrahlungCN[en, k,
   |intègre numériquement
     x, If[$TnuEqualT, x, znu], 1], {en, 1.001, Max[10, 20/x]},
     |si                                      |maximum
    {k, 0.001, Abs[en - q], Abs[en + q], Max[10, 20/x]}, PrecisionGoal → 4]
               |valeur absolue |valeur absolue |maximum              |objectif de précision
  ];
```

The global 1/2 factor is Jacobian of the change of variables (we integrate in the sum and difference of energies).

```
In[•]:= λnTOpThermal[Tv_] := (*λnTOpThermal[Tv]=*)
  With[{x = me/(kB Tv), znu = me/(kB Tv TνoverT[Tv]), q = Q/me},
   NIntegrate[C1dE[en, x, If[$TnuEqualT, x, znu], 1], {en, 1, Max[25, 150/x]}]
    + NIntegrate[1/2 C2dE1dE2CN[(e1pe2 + e1me2)/2, (e1pe2 - e1me2)/2, x,
       If[$TnuEqualT, x, znu], 1], {e1me2, -Max[10, 15/x], -0.001},
      {e1pe2, 2.001 + Abs[e1me2], 2 + Abs[e1me2] + Max[10, 15/x]},
      PrecisionGoal → 3, Exclusions → {0}]
    + NIntegrate[1/2 C2dE1dE2CN[(e1pe2 + e1me2)/2, (e1pe2 - e1me2)/2, x,
       If[$TnuEqualT, x, znu], 1], {e1me2, 0.001, Max[10, 15/x]}, {e1pe2,
      2.001 + Abs[e1me2], 2 + Abs[e1me2] + Max[10, 15/x]}, PrecisionGoal → 3]
  ];
```

The 5 body is just for comparison with [Brown & Sawyer]

```
In[•]:= λnTOp5bodies[Tv_] := (*λnTOp5bodies[Tv]=*)
  With[{x = me/(kB Tv), znu = me/(kB Tv TνoverT[Tv]), q = Q/me},
   NIntegrate[IPENFiveBodyT0CN[en, k, x, If[$TnuEqualT, x, znu], 1], {en, 1,
      Max[20, 20/x]}, {k, en + q, en + q + Max[20, 20/x]}, PrecisionGoal → 4]
  ];
```

Let us finish with p -> n processes

When we have computed the corrections to n -> p rates, the converse are obtained from detailed balance arguments. That is we perform the replacement Q -> -Q as argued in [Brown&Sawyer]. This is also explained in details in the companion paper.

```
In[•]:= λpTOnThermalTruePhoton[Tv_] :=
  (*λpTOnThermalTruePhoton[Tv]=*)If[Tv < 10^8.2
   (* When the temperature is too low it is better to put 0 *), 0,
```

```
    With[{x = me / (kB Tv), znu = me / (kB Tv TνoverT[Tv]), q = Q / me},
     NIntegrate[IPENCCRTCN[en, k, x, If[$TnuEqualT, x, znu], -1],
      {k, 0.001, Max[10, 20 / x]}, {en, 1.001, Max[10, 20 / x]}, PrecisionGoal → 4]
    ];

λpTOnThermalDiffBremsstrahlung[Tv_] :=
  (*λpTOnThermalDiffBremsstrahlung[Tv]=*)If[Tv < 10^8.2
    (* When the temperature is too low it is better to put 0 *), 0,
    With[{x = me / (kB Tv), znu = me / (kB Tv TνoverT[Tv]), q = Q / me},
     NIntegrate[IPENCCRDiffBremsstrahlungCN[en, k,
       x, If[$TnuEqualT, x, znu], -1], {en, 1.001, Max[10, 20 / x]},
      {k, 0.001, Abs[en - q], Abs[en + q], Max[10, 20 / x]}, PrecisionGoal → 4]
    ];


λpTOnThermal[Tv_] := (*λpTOnThermal[Tv]=*)If[Tv < 10^8.2
    (* When the temperature is too low it is better to put 0 *), 0,
    With[{x = me / (kB Tv), znu = me / (kB Tv TνoverT[Tv]), q = Q / me},
     NIntegrate[
       C1dE[en, x, If[$TnuEqualT, x, znu], -1], {en, 1, Max[25, 150 / x]}]
      + NIntegrate[1 / 2 C2dE1dE2CN[(e1pe2 + e1me2) / 2, (e1pe2 - e1me2) / 2, x,
        If[$TnuEqualT, x, znu], -1], {e1me2, -Max[10, 15 / x], -0.001}, {e1pe2,
       2.001 + Abs[e1me2], 2 + Abs[e1me2] + Max[10, 15 / x]}, PrecisionGoal → 3]
      + NIntegrate[1 / 2 C2dE1dE2CN[(e1pe2 + e1me2) / 2, (e1pe2 - e1me2) / 2, x,
        If[$TnuEqualT, x, znu], -1], {e1me2, 0.001, Max[10, 15 / x]}, {e1pe2,
       2.001 + Abs[e1me2], 2 + Abs[e1me2] + Max[10, 15 / x]}, PrecisionGoal → 3]
    ];
```

Collecting all finite-temperature corrections

We collect the various contributions of Eq. 108. The TruePhoton contribution refers to the first term on the rhs of Eq. 108, and the Thermal contribution to the second and third. The Bremsstrahlung corrections are Eqs. 107.

In[•]:=
```
λnTOpCCRTh[Tv_] := (λnTOpThermal[Tv] +
    λnTOpThermalTruePhoton[Tv] + If[$CorrectionBremsstrahlung,
                                   si
     λnTOpThermalDiffBremsstrahlung[Tv], λnTOp5bodies[Tv]]);

λpTOnCCRTh[Tv_] := (λpTOnThermal[Tv] + λpTOnThermalTruePhoton[Tv] +
    If[$CorrectionBremsstrahlung, λpTOnThermalDiffBremsstrahlung[Tv], 0]);
    si
```

We gather all corrections according to the Booleans chosen at the beginning

In[•]:=
```
λ0 := If[$RadiativeCorrections, λRad, λBORN] + If[$FiniteNucleonMass, λFM, 0]
      si                                        si
```

Gathering all corrections

For info the normalized rates are :

In[•]:=
```
λnTOpNormalizedSDCCR[Tv_] := (λ0)^-1 λnTOpSDCCR[Tv];
λpTOnNormalizedSDCCR[Tv_] := (λ0)^-1 λpTOnSDCCR[Tv];
λnTOpNormalizedSD[Tv_] := (λ0)^-1 λnTOpSD[Tv];
λpTOnNormalizedSD[Tv_] := (λ0)^-1 λpTOnSD[Tv];
λnTOpNormalizedCCR[Tv_] := (λ0)^-1 λnTOpCCR[Tv];
λpTOnNormalizedCCR[Tv_] := (λ0)^-1 λpTOnCCR[Tv];
λnTOpNormalizedBORN[Tv_] := (λ0)^-1 λnTOpBORN[Tv];
λpTOnNormalizedBORN[Tv_] := (λ0)^-1 λpTOnBORN[Tv];
```

We add them depending on options chosen

In[ ]:=
```
Clear[λnTOp, λpTOn, λnTOpNormalized, λpTOnNormalized];
efface
λnTOpNormalized[Tv_] := (λ0)^-1 (
    If[$RadiativeCorrections, λnTOpCCR[Tv], λnTOpBORN[Tv]]
    si
     + If[$RadiativeThermal, λnTOpCCRTh[Tv], 0]
       si
     + If[$SpectralDistortions,
       si
       If[$RadiativeCorrections, λnTOpSDCCR[Tv], λnTOpSD[Tv]], 0]
       si
     + If[$FiniteNucleonMass,
       si
       If[$CoupledFMandRC, λnTOpFMCCR[Tv], λnTOpFMNoCCR[Tv]], 0]
       si
   );

λpTOnNormalized[Tv_] := (λ0)^-1 (
    If[$RadiativeCorrections, λpTOnCCR[Tv], λpTOnBORN[Tv]]
    si
     + If[$RadiativeThermal, λpTOnCCRTh[Tv], 0]
       si
     + If[$SpectralDistortions,
       si
       If[$RadiativeCorrections, λpTOnSDCCR[Tv], λpTOnSD[Tv]], 0]
       si
     + If[$FiniteNucleonMass,
       si
       If[$CoupledFMandRC, λpTOnFMCCR[Tv], λpTOnFMNoCCR[Tv]], 0]);
       si
```

In[ ]:=
```
Clear[λnTOp]
efface
λnTOp[Tv_] := 1 / τneutron λnTOpNormalized[Tv];
λpTOn[Tv_] := 1 / τneutron λpTOnNormalized[Tv];
```

Detailed balance check

```
In[•]:= Tt = 10^10.8;
        (λnTOpThermalTruePhoton[Tt] + 1 λnTOpThermalDiffBremsstrahlung[Tt]) /
         (λpTOnThermalTruePhoton[Tt] + 1 λpTOnThermalDiffBremsstrahlung[Tt])
        (*LnTOp[Tt]/LpTOn[Tt]*)
        λnTOpCCR[Tt] / λpTOnCCR[Tt]
        λnTOpBORN[Tt] / λpTOnBORN[Tt]
        λnTOpThermal[Tt] / λpTOnThermal[Tt]
        Exp[Q / kB / Tt]
        exponentielle
Out[•]=
        1.2685145
Out[•]=
        1.2685433
Out[•]=
        1.2685438
Out[•]=
        1.2685535
Out[•]=
        1.2685426
```

Effect of spectral distortions on the rates : relative variations wrt to the rate without them. This plot depends on which spectral distortions have been chosen above. If none, it is identically zero.

```
In[•]:= (*listT=ListTRange[10^9,10^11.5][[1;; ;;10]];
        (*We take only 10% of points to get the
         plot faster. Otherwise it takes a bit longer.*)
        ListLogLinearPlot[{Table[{T,λnTOpSDCCR[T]/λnTOpNormalized[T]},{T,listT}],
        tracé log linéaire de liste   table
          Table[{T,λpTOnSDCCR[T]/λpTOnNormalized[T]},{T,listT}]},
          table
         Joined→True,Frame→True,FrameStyle→Thickness[0.003],
         joint  vrai  cadre  vrai  style de cadre  épaisseur
         LabelStyle→{FontSize→10},FrameLabel→{"T (K)","δΓ/Γ"},
         style d'étiquette  taille de police de···  étiquette de cadre
         PlotStyle→{{Red,Thickness[0.003]},{Blue,Thickness[0.003]}}]*)
         style de tracé  ro···  épaisseur  bleu  épaisseur
```

Precomputation and storage of rates

We store the rate but without the division by $\tau_{neutron}$. So that
we can use the same fit for the reaction rates, and still vary $\tau_{neutron}$.

The rates in the files, once loaded, are interpolated once the division by $\tau_{neutron}$ has been added.
If spectral distortions are taken into account, we could avoid ParallelComputation because it seems it loses time in sharing the big interpolation.
TODO maybe there is a solution to that.

```
MyTableWeakRate := If[$ParallelWeakRates(*&&Not[$SpectralDistortions]*),
                   |si                   |négation
  (*Something does not work when we use NEVO. Some
   definitions seem not to be distributed correctly.*)
  DistributeDefinitions[λnTOpNormalized, λpTOnNormalized,
  |distribue définitions
   LoadNEVO, LoadDistortions, TableNEVOLoaded[FileNEVOneutrinos],
   TableNEVOLoaded[FileNEVOantineutrinos], TreatNEVOFile];
  ParallelEvaluate[Off[NIntegrate::slwcon];];
  |évalue en parallèle  |dé··· |intègre numériquement
  ParallelTable,
  |table en parallèle
  Table]
  |table
```

```
LoadWeakRates := (
    If[Not@FileExistsQ[NamePENFilenp] ||
    |si  |nég··· |fichier existe ?
      Not@FileExistsQ[NamePENFilepn] || $RecomputeWeakRates,
      |nég··· |fichier existe ?
     Off[NIntegrate::slwcon]; Off[General::munfl ];
     |dé··· |intègre numériquement        |dé··· |général
     If[$RecomputeWeakRates,
     |si
      Print["The $RecomputeWeakRates option forces a recomputation
      |imprime
          of weak rates."]];
     Print["Computing weak rates for n<->p conversions."];
     |imprime
     time = Timing[
            |chronométrage
        TabRatenp = MyTableWeakRate[{T, λnTOpNormalized[T]}, {T, ListT}];
        TabRatepn =
         MyTableWeakRate[{T, λpTOnNormalized[T]}, {T, ListT}];][[1]];
     If[$Verbose,
     |si
      Print["Time spent in (re)computing weak rates was ", time, " s"];];
      |imprime


     Export[NamePENFilenp, TabRatenp, "TSV"];
     |exporte
     Export[NamePENFilepn, TabRatepn, "TSV"];
     |exporte
     On[General::munfl ]; On[NIntegrate::slwcon];,
     |ac·· |général                  |ac·· |intègre numériquement


     If[$Verbose, Print["Loading weak rates for n<->p conversions."];];
     |si              |imprime
     TabRatenp = Import[NamePENFilenp, "TSV"];
                 |importe
     TabRatepn = Import[NamePENFilepn, "TSV"];
                 |importe
    ];
    λnTOpI = MyInterpolationRate[ToExpression[TabRatenp]];
                                  |en expression
    λpTOnI = MyInterpolationRate[ToExpression[TabRatepn]];
                                  |en expression
  );
```

In[ ]:= `LoadWeakRates`

We give standard names that are used in the network of reactions later. The factor $1/\tau_n$ is the one appearing in the constant defined in Eq. 91 of companion paper.

In[ ]:=
```
LnTOp[Tv_] := 1 / τneutron * λnTOpI[Tv];
LpTOn[Tv_] := 1 / τneutron * λpTOnI[Tv];
LbarnTOp[Tv_] := LpTOn[Tv];
```

We check that the rate for neutron decay is what we expect at low temperature, that is it is τneutron only. At 10^8 K it should be the case.

In[ ]:=
```
1 / LnTOp[Tf]
τneutron
```

Out[ ]=
```
878.38231
```

Out[ ]=
```
878.4
```

Freeze - out estimations

In[ ]:=
```
Tfreeze1 = FindRoot[{LnTOp[T] == HofT[T]}, {T, 10^10}][[1, 2]]
           trouve racine
Tfreeze2 = FindRoot[
           trouve racine
   {LnTOp[T] + LpTOn[T] == (Exp[Q / (kB T)] / (1 + Exp[Q / (kB T)])) Q / (kB T) HofT[T]},
                           exponentielle          exponentielle
   {T, 10^10}][[1, 2]]
```

Out[ ]=
```
8.2873901 × 10^9
```

Out[ ]=
```
8.9281629 × 10^9
```

Plots of finite-temperature corrections

This is very long so I comment this section but to get the plot of finite temperature corrections of the companion paper, this should be uncommented.

In[ ]:=
```
(*

Off[NIntegrate::slwcon];
dé··· intègre numériquement
TabdlambdanTOp=
 MyTableWeakRate[{T,(λRadandFM*(λnTOpThermal[T]+λnTOpThermalTruePhoton[T]+
        λnTOpThermalDiffBremsstrahlung[T]))/
    (λBORN*λnTOpBORN[T])},{T,ListTRange[1 10^9,10^11]}];
Print[TabdlambdanTOp];
imprime

TabdlambdapTOn=
 MyTableWeakRate[{T,(λRadandFM*(λpTOnThermal[T]+λpTOnThermalTruePhoton[T]+
```

```
        λpTOnThermalDiffBremsstrahlung[T]))/
    (λBORN*λpTOnBORN[T])},{T,ListTRange[1 10^9,10^11]}];

Export["CSV/TabdlambdanTOp.dat",TabdlambdanTOp,"TSV"];
exporte
Export["CSV/TabdlambdapTOn.dat",TabdlambdapTOn,"TSV"];
exporte

TabdlambdanTOpBrown=MyTableWeakRate[
  {T,(λRadandFM*(λnTOpThermal[T]+λnTOpThermalTruePhoton[T]))/(λBORN*λnTOpBORN[T])},{T,ListTRange[1 10^9,10^11]}];
TabdlambdanTOpBrown5Bodies=MyTableWeakRate[{T,
   (λRadandFM*(λnTOpThermal[T]+λnTOpThermalTruePhoton[T]+λnTOp5bodies[T]))/
    (λBORN*λnTOpBORN[T])},{T,ListTRange[1 10^9,10^11]}];
TabdlambdapTOnBrown=MyTableWeakRate[
  {T,(λRadandFM*(λpTOnThermal[T]+λpTOnThermalTruePhoton[T]))/(λBORN*λpTOnBORN[T])},{T,ListTRange[1 10^9,10^11]}];

Export["CSV/TabdlambdanTOpBrown.dat",TabdlambdanTOpBrown,"TSV"];
exporte
Export["CSV/TabdlambdanTOpBrown5Bodies.dat",
exporte
 TabdlambdanTOpBrown5Bodies,"TSV"];
Export["CSV/TabdlambdapTOnBrown.dat",TabdlambdapTOnBrown,"TSV"];
exporte

TabdlambdanTOpBrehm=MyTableWeakRate[
  {T,(λRadandFM*(λnTOpThermalDiffBremsstrahlung[T]))/(λBORN*λnTOpBORN[T])},{T,ListTRange[1 10^9,10^11]}];
TabdlambdapTOnBrehm=MyTableWeakRate[
  {T,(λRadandFM*(λpTOnThermalDiffBremsstrahlung[T]))/(λBORN*λpTOnBORN[T])},{T,ListTRange[1 10^9,10^11]}];
On[NIntegrate::slwcon];
ac·· intègre numériquement



Export["CSV/TabdlambdanTOpBrehm.dat",TabdlambdanTOpBrehm,"TSV"];
exporte
Export["CSV/TabdlambdapTOnBrehm.dat",TabdlambdapTOnBrehm,"TSV"];*)
exporte
```

```
In[ ]:= If[$PaperPlots,
        si
  TabδλnTOp = Import["CSV/TabdlambdanTOp.dat"];
              importe
  TabδλpTOn = Import["CSV/TabdlambdapTOn.dat"];
              importe

  TabδλnTOpBrown = Import["CSV/TabdlambdanTOpBrown.dat"];
                   importe
  TabδλnTOpBrown5Bodies = Import["CSV/TabdlambdanTOpBrown5Bodies.dat"];
                          importe
  TabδλpTOnBrown = Import["CSV/TabdlambdapTOnBrown.dat"];
                   importe

  TabδλnTOpBrehm = Import["CSV/TabdlambdanTOpBrehm.dat"];
                   importe
  TabδλpTOnBrehm = Import["CSV/TabdlambdapTOnBrehm.dat"];
                   importe
 ]

In[ ]:= If[$PaperPlots,
        si
  TFreeze = 0.8 MeV / kB;
  RCT = ListLogLinearPlot[{TabδλnTOp, TabδλpTOn, TabδλnTOpBrown, TabδλpTOnBrown,
          tracé log linéaire de liste
      TabδλnTOpBrehm, TabδλpTOnBrehm, TabδλnTOpBrown5Bodies}, Frame → True,
                                                               cadre  vrai
    FrameStyle → Thickness[0.004], Joined → True, FrameLabel → {"T (K)", "δΓ/Γ"},
    style de cadre  épaisseur  joint  vrai  étiquette de cadre
    LabelStyle → {FontSize → 12}, GridLines → {{{TFreeze, {Gray, Thickness[
    style d'étiquette  taille de police de ca···  lignes de grille  gris  épaisseur
          0.005]}}}, {}}, PlotStyle → {{Darker@Darker@Green, Thickness[0.005]},
                                       style de tracé  plus foncé  plus foncé  vert  épaisseur
      {Darker@Darker@Green, Thickness[0.005], Dashing[{0.02}]},
       plus foncé  plus foncé  vert  épaisseur  style de tirets
      {Red, Thickness[0.006]}, {Red, Thickness[0.006], Dashing[{0.02}]},
       rouge  épaisseur  rouge  épaisseur  style de tirets
      {Blue, Thickness[0.004]}, {Blue, Thickness[0.004], Dashing[{0.02}]},
       bleu  épaisseur  bleu  épaisseur  style de tirets
      {Red, Thickness[0.002], Thickness[0.003]}}, FrameTicks → MyFrameTicksLog]]

 If[$PaperPlots, Export["Plots/LogPlotdeltaGammaCCRT.pdf",
 si  exporte
    Style[RCT, Magnification → 1], "PDF"]];
    style  agrandissement  fonction de densité de probabilité
```

Nuclear reactions network

Nuclear Species

Names and (N,Z)

The is the list of short names with their (neutron, proton) weights. So by definition a neutron has (1, 0) and a proton has (0, 1) and He4 has (2, 2) and so on.

In[ ]:=
```
NamesWithWeightsAll =
  {{"n", {1, 0}}, {"p", {0, 1}}, {"d", {1, 1}}, {"t", {2, 1}},
   {"He3", {1, 2}}, {"a", {2, 2}},
   {"Li7", {4, 3}}, {"Be7", {3, 4}}, {"He5", {3, 2}},
   {"He6", {4, 2}}, {"Li6", {3, 3}}, {"Li8", {5, 3}}, {"Li9", {6, 3}},
   {"Be8", {4, 4}}, {"Be9", {5, 4}},
   {"Be10", {6, 4}}, {"Be11", {7, 4}}, {"Be12", {8, 4}},
   {"B8", {3, 5}}, {"B9", {4, 5}}, {"B10", {5, 5}}, {"B11", {6, 5}},
   {"B12", {7, 5}}, {"B13", {8, 5}}, {"B14", {9, 5}}, {"B15", {10, 5}},
   {"C9", {3, 6}}, {"C10", {4, 6}}, {"C11", {5, 6}}, {"C12", {6, 6}},
   {"C13", {7, 6}}, {"C14", {8, 6}}, {"C15", {9, 6}}, {"C16", {10, 6}},
   {"N12", {5, 7}}, {"N13", {6, 7}}, {"N14", {7, 7}},
   {"N15", {8, 7}}, {"N16", {9, 7}}, {"N17", {10, 7}},
   {"O13", {5, 8}}, {"O14", {6, 8}}, {"O15", {7, 8}}, {"O16", {8, 8}},
   {"O17", {9, 8}}, {"O18", {10, 8}}, {"O19", {11, 8}}, {"O20", {12, 8}},
   {"F17", {8, 9}}, {"F18", {9, 9}}, {"F19", {10, 9}}, {"F20", {11, 9}},
   {"Ne18", {8, 10}}, {"Ne19", {9, 10}}, {"Ne20", {10, 10}},
   {"Ne21", {11, 10}}, {"Ne22", {12, 10}}, {"Ne23", {13, 10}},
   {"Na20", {9, 11}}, {"Na21", {10, 11}},
   {"Na22", {11, 11}}, {"Na23", {12, 11}}};
```

Let us vizuallize the nuclides used in as a table in (Z,N).

In[ ]:=
```
TableNZNucleons = Table[" ", {i, 0, 13}, {j, 0, 11}];
                  table

Map[(TableNZNucleons[[Sequence @@ (#[[2]] + {1, 1})]] = #[[1]]) &,
applique à travers          séquence

  NamesWithWeightsAll];
```

In[ ]:=
```
Grid[Transpose@TableNZNucleons, Frame → All]
```
grille transpose cadre tout

```
(* This is table III in companion paper*)
```

Out[ ]=

	n												
p	d	t											
	He3	a	He5	He6									
			Li6	Li7	Li8	Li9							
			Be7	Be8	Be9	Be10	Be11	Be12					
			B8	B9	B10	B11	B12	B13	B14	B15			
			C9	C10	C11	C12	C13	C14	C15	C16			
					N12	N13	N14	N15	N16	N17			
					O13	O14	O15	O16	O17	O18	O19	O20	
								F17	F18	F19	F20		
								Ne18	Ne19	Ne20	Ne21	Ne22	Ne23
									Na20	Na21	Na22	Na23	

The list of the species names only

In[ ]:=
```
ShortNamesAll = NamesWithWeightsAll[[All, 1]];
```
tout

The list of {n, p} pairs only.

In[ ]:=
```
ListNPPairs = NamesWithWeightsAll[[All, 2]];
```
tout

Functions to check if a name exists

In[ ]:=
```
ExistName[name_] := MemberQ[ShortNamesAll, name];
```
appartient ?

```
ExistPair[pair_List] := MemberQ[ListNPPairs, pair];
```
liste appartient ?

This function selects the names of the species which all have the same mass number A

In[ ]:=
```
NamesMassNumberAll[A_] :=
  Select[NamesWithWeightsAll, ((Plus @@ (#[[2]])) == A) &][[All, 1]]
```
sélectionne plus tout

Dictionaries between names and numbers

We define dictionaries to handle species. We associate a number to each species
It is a simple correspondance between names and position of the species in the list, or between the names and the pair (neutron,proton).

In[•]:=
```
KeySpecies = Association@(Rule @@@ NamesWithWeightsAll)
             |association      |règle
```

Out[•]=
```
⟨|n → {1, 0}, p → {0, 1}, d → {1, 1}, t → {2, 1}, He3 → {1, 2}, a → {2, 2}, Li7 → {4, 3},
 Be7 → {3, 4}, He5 → {3, 2}, He6 → {4, 2}, Li6 → {3, 3}, Li8 → {5, 3}, Li9 → {6, 3},
 Be8 → {4, 4}, Be9 → {5, 4}, Be10 → {6, 4}, Be11 → {7, 4}, Be12 → {8, 4},
 B8 → {3, 5}, B9 → {4, 5}, B10 → {5, 5}, B11 → {6, 5}, B12 → {7, 5}, B13 → {8, 5},
 B14 → {9, 5}, B15 → {10, 5}, C9 → {3, 6}, C10 → {4, 6}, C11 → {5, 6},
 C12 → {6, 6}, C13 → {7, 6}, C14 → {8, 6}, C15 → {9, 6}, C16 → {10, 6},
 N12 → {5, 7}, N13 → {6, 7}, N14 → {7, 7}, N15 → {8, 7}, N16 → {9, 7},
 N17 → {10, 7}, O13 → {5, 8}, O14 → {6, 8}, O15 → {7, 8}, O16 → {8, 8},
 O17 → {9, 8}, O18 → {10, 8}, O19 → {11, 8}, O20 → {12, 8}, F17 → {8, 9},
 F18 → {9, 9}, F19 → {10, 9}, F20 → {11, 9}, Ne18 → {8, 10}, Ne19 → {9, 10},
 Ne20 → {10, 10}, Ne21 → {11, 10}, Ne22 → {12, 10}, Ne23 → {13, 10},
 Na20 → {9, 11}, Na21 → {10, 11}, Na22 → {11, 11}, Na23 → {12, 11}|⟩
```

We have a reverse dictionary if we want

In[•]:=
```
KeyNucleons = Association@(Rule @@@ (Reverse /@ NamesWithWeightsAll))
              |association      |règle      |inverse
```

Out[•]=
```
⟨|{1, 0} → n, {0, 1} → p, {1, 1} → d, {2, 1} → t, {1, 2} → He3, {2, 2} → a, {4, 3} → Li7,
 {3, 4} → Be7, {3, 2} → He5, {4, 2} → He6, {3, 3} → Li6, {5, 3} → Li8, {6, 3} → Li9,
 {4, 4} → Be8, {5, 4} → Be9, {6, 4} → Be10, {7, 4} → Be11, {8, 4} → Be12,
 {3, 5} → B8, {4, 5} → B9, {5, 5} → B10, {6, 5} → B11, {7, 5} → B12, {8, 5} → B13,
 {9, 5} → B14, {10, 5} → B15, {3, 6} → C9, {4, 6} → C10, {5, 6} → C11,
 {6, 6} → C12, {7, 6} → C13, {8, 6} → C14, {9, 6} → C15, {10, 6} → C16,
 {5, 7} → N12, {6, 7} → N13, {7, 7} → N14, {8, 7} → N15, {9, 7} → N16,
 {10, 7} → N17, {5, 8} → O13, {6, 8} → O14, {7, 8} → O15, {8, 8} → O16,
 {9, 8} → O17, {10, 8} → O18, {11, 8} → O19, {12, 8} → O20, {8, 9} → F17,
 {9, 9} → F18, {10, 9} → F19, {11, 9} → F20, {8, 10} → Ne18, {9, 10} → Ne19,
 {10, 10} → Ne20, {11, 10} → Ne21, {12, 10} → Ne22, {13, 10} → Ne23,
 {9, 11} → Na20, {10, 11} → Na21, {11, 11} → Na22, {12, 11} → Na23|⟩
```

N, Z, A from name

In[•]:=
```
Ni["Bm"] := 1;
Ni["Bp"] := -1;
Ni["g"] := 0;

Zi["Bm"] := -1;
Zi["Bp"] := 1;
Zi["g"] := 0;

Ai["Bm"] := 0;
Ai["Bp"] := 0;
Ai["g"] := 0;
```

```
In[•]:= Ni[key_] := KeySpecies[key][[1]]
        Zi[key_] := KeySpecies[key][[2]]
        Ai[key_] := Zi[key] + Ni[key]
```

```
In[•]:= Ai["Li7"]
        Ni["Li7"]
        Zi["Li7"]
        Ni["Bm"]
Out[•]=
        7
Out[•]=
        4
Out[•]=
        3
Out[•]=
        1
```

Binding energies and spins of nuclei

Tools to reshape the file "nubase2016.asc"

```
In[•]:= SpinFromCharList[charlist_List] :=
                          liste
          StringReplace[StringJoin @@ charlist, {"(" → "", ")" → "",
         remplace dans cha··· joins chaînes de caractères
            "," → "", "+" → "", "-" → "", " " → "", "#" → "", "*" → ""}]
        MassFromCharList[charlist_List] :=
                          liste
          StringReplace[StringJoin @@ charlist, {" " → "", "#" → ""}]
         remplace dans cha··· joins chaînes de caractères
```

OLD method : Nubase 2016. New methods with NUbase 2020 below

We load the file "nubtab06.asc" OLD Method. Now we prefere to load nubase 2020 from the file nubase_4.mas20.txt, see below.

NUBASE 2016 Syntax :
The first three characters of each line are the A .
Characters from 5 to 7 are Z (but the 8th character is a number for a possibel excited state. Hence we use character 5 to 8 for 10*Z, and then divide by 10. If it is an integer it is not an excited state. Otherwise it is discarded. This is a dirty hack)
From 20 to 29 it is mass excess .
From 80 to 93 it is some information on spin and parity .

```
In[•]:= (*StringListParticles=#[[1]]&/@Import["nubase2016.asc","TSV"];
                                        importe
        NubTabChar=Select[Characters/@StringListParticles,Length[#]≥93&];*)
                  sélectio··· caractères                    longueur
```

In[ ]:=
```
(*Alist=ToExpression/@StringJoin/@(Take[#,{1,3}]&/@NubTabChar);
Zlist=#/10&/@ToExpression/@StringJoin/@(Take[#,{5,8}]&/@NubTabChar);
MassExcessesString=MassFromCharList/@(Take[#,{20,29}]&/@NubTabChar);
Spins=SpinFromCharList/@(Take[#,{80,93}]&/@NubTabChar);
Nlist=Alist-Zlist;*)
```

In[ ]:=
```
(*MyGrid[ListNPBindingSpinName=
  Flatten[{KeyNucleons[{#[[1]],#[[2]]}],#}]&/@Select[Transpose[
    {Nlist,Zlist,MassExcessesString,Spins}],ExistPair[{#[[1]],#[[2]]}]&]]*)
```

We prefer using the most recent nubase (2020) (see comments about mass database at the top of this notebook) from the file nubase_4.mas20.txt (Chinese Physics C 45 (2021) 030001).

In[ ]:=
```
StringListParticles = Drop[#[[1]] & /@ Import["nubase_4.mas20.txt", "TSV"], 25];
NubTabChar = Select[Characters /@ StringListParticles, Length[#] ≥ 93 &];
```

NUBASE 2020 Syntax :
The first three characters of each line are the A.
Characters from 5 to 7 are Z (but the 8th character is a number for a possible excited state. Hence we use character 5 to 8 for 10*Z, and then divide by 10. If it is an integer it is not an excited state. Otherwise it is discarded. This is a dirty hack)
From 19 to 31 it is mass excess.
From 89 to 102 it is some information on spin and parity.

In[ ]:=
```
Alist = ToExpression /@ StringJoin /@ (Take[#, {1, 3}] & /@ NubTabChar);
Zlist = # / 10 & /@ ToExpression /@ StringJoin /@ (Take[#, {5, 8}] & /@ NubTabChar);
MassExcessesString = MassFromCharList /@ (Take[#, {19, 31}] & /@ NubTabChar);
Spins = SpinFromCharList /@ (Take[#, {89, 93}] & /@ NubTabChar);
Nlist = Alist - Zlist;
```

We gather all this information in a list (ListNPBindingSpinName) that we print as a Table here for readbility.

In[ ]:=
```
MyGrid[ListNPBindingSpinName =
  Flatten[{KeyNucleons[{#[[1]], #[[2]]}], #}] & /@ Select[Transpose[
    {Nlist, Zlist, MassExcessesString, Spins}], ExistPair[{#[[1]], #[[2]]}] &]]
```

Out[ ]=

n	1	0	8071.3181	1/2
p	0	1	7288.971064	1/2

d	1	1	13135.722895	1
t	2	1	14949.81090	1/2
He3	1	2	14931.21888	1/2
a	2	2	2424.91587	0
He5	3	2	11231	3/2
He6	4	2	17592.10	0
Li6	3	3	14086.8804	1
Li7	4	3	14907.105	3/2
Be7	3	4	15769.00	3/2
Li8	5	3	20945.80	2
Be8	4	4	4941.67	0
B8	3	5	22921.6	2
Li9	6	3	24954.91	3/2
Be9	5	4	11348.45	3/2
B9	4	5	12416.5	3/2
C9	3	6	28911.0	3/2
Be10	6	4	12607.49	0
B10	5	5	12050.611	3
C10	4	6	15698.67	0
Be11	7	4	20177.17	1/2
B11	6	5	8667.708	3/2
C11	5	6	10649.40	3/2
Be12	8	4	25077.8	0
B12	7	5	13369.4	1
C12	6	6	0.0	0
N12	5	7	17338.1	1
B13	8	5	16561.9	3/2
C13	7	6	3125.0093	1/2
N13	6	7	5345.48	1/2
O13	5	8	23115	3/2
B14	9	5	23664	2
C14	8	6	3019.893	0
N14	7	7	2863.4168	1
O14	6	8	8007.781	0
B15	10	5	28957	3/2
C15	9	6	9873.1	1/2
N15	8	7	101.4381	1/2
O15	7	8	2855.6	1/2
C16	10	6	13694	0
N16	9	7	5683.9	2
O16	8	8	-4737.0021	0
N17	10	7	7870	1/2
O17	9	8	-808.7642	5/2
F17	8	9	1951.70	5/2
O18	10	8	-782.8163	0
F18	9	9	873.1	1
Ne18	8	10	5317.6	0
O19	11	8	3332.9	5/2
F19	10	9	-1487.4451	1/2
Ne19	9	10	1752.05	1/2
O20	12	8	3796.2	0
F20	11	9	-17.463	2
Ne20	10	10	-7041.9322	0

Na20	9	11	6850.5	2
Ne21	11	10	-5731.78	3/2
Na21	10	11	-2184.86	3/2
Ne22	12	10	-8024.716	0
Na22	11	11	-5181.39	3
Ne23	13	10	-5154.05	5/2
Na23	12	11	-9529.8535	3/2

We define a dictionary for excess masses (in keV)

In[•]:=
```
ExcessMassKeys =
 Association[{#[[1]] → ToExpression[#[[4]]]} & /@ ListNPBindingSpinName]
```
association, en expression

Out[•]=
```
⟨| n → 8071.3181, p → 7288.9711, d → 13 135.723, t → 14 949.811, He3 → 14 931.219,
 a → 2424.9159, He5 → 11 231, He6 → 17 592.1, Li6 → 14 086.88, Li7 → 14 907.105,
 Be7 → 15 769., Li8 → 20 945.8, Be8 → 4941.67, B8 → 22 921.6, Li9 → 24 954.91,
 Be9 → 11 348.45, B9 → 12 416.5, C9 → 28 911., Be10 → 12 607.49, B10 → 12 050.611,
 C10 → 15 698.67, Be11 → 20 177.17, B11 → 8667.708, C11 → 10 649.4,
 Be12 → 25 077.8, B12 → 13 369.4, C12 → 0., N12 → 17 338.1, B13 → 16 561.9,
 C13 → 3125.0093, N13 → 5345.48, O13 → 23 115, B14 → 23 664, C14 → 3019.893,
 N14 → 2863.4168, O14 → 8007.781, B15 → 28 957, C15 → 9873.1, N15 → 101.4381,
 O15 → 2855.6, C16 → 13 694, N16 → 5683.9, O16 → -4737.0021, N17 → 7870,
 O17 → -808.7642, F17 → 1951.7, O18 → -782.8163, F18 → 873.1, Ne18 → 5317.6,
 O19 → 3332.9, F19 → -1487.4451, Ne19 → 1752.05, O20 → 3796.2, F20 → -17.463,
 Ne20 → -7041.9322, Na20 → 6850.5, Ne21 → -5731.78, Na21 → -2184.86,
 Ne22 → -8024.716, Na22 → -5181.39, Ne23 → -5154.05, Na23 → -9529.8535 |⟩
```

And a dictionary for spins

In[•]:=
```
SpinKeys = Association[{#[[1]] → ToExpression[#[[5]]]} & /@ ListNPBindingSpinName]
```
association, en expression

Out[•]=
$$\left\langle\left| n \to \frac{1}{2},\ p \to \frac{1}{2},\ d \to 1,\ t \to \frac{1}{2},\ He3 \to \frac{1}{2},\ a \to 0,\ He5 \to \frac{3}{2},\ He6 \to 0,\ Li6 \to 1,\ Li7 \to \frac{3}{2},\right.\right.$$
$$Be7 \to \frac{3}{2},\ Li8 \to 2,\ Be8 \to 0,\ B8 \to 2,\ Li9 \to \frac{3}{2},\ Be9 \to \frac{3}{2},\ B9 \to \frac{3}{2},\ C9 \to \frac{3}{2},\ Be10 \to 0,$$
$$B10 \to 3,\ C10 \to 0,\ Be11 \to \frac{1}{2},\ B11 \to \frac{3}{2},\ C11 \to \frac{3}{2},\ Be12 \to 0,\ B12 \to 1,\ C12 \to 0,\ N12 \to 1,$$
$$B13 \to \frac{3}{2},\ C13 \to \frac{1}{2},\ N13 \to \frac{1}{2},\ O13 \to \frac{3}{2},\ B14 \to 2,\ C14 \to 0,\ N14 \to 1,\ O14 \to 0,\ B15 \to \frac{3}{2},$$
$$C15 \to \frac{1}{2},\ N15 \to \frac{1}{2},\ O15 \to \frac{1}{2},\ C16 \to 0,\ N16 \to 2,\ O16 \to 0,\ N17 \to \frac{1}{2},\ O17 \to \frac{5}{2},\ F17 \to \frac{5}{2},$$
$$O18 \to 0,\ F18 \to 1,\ Ne18 \to 0,\ O19 \to \frac{5}{2},\ F19 \to \frac{1}{2},\ Ne19 \to \frac{1}{2},\ O20 \to 0,\ F20 \to 2,$$
$$\left.\left. Ne20 \to 0,\ Na20 \to 2,\ Ne21 \to \frac{3}{2},\ Na21 \to \frac{3}{2},\ Ne22 \to 0,\ Na22 \to 3,\ Ne23 \to \frac{5}{2},\ Na23 \to \frac{3}{2} \right|\right\rangle$$

From excess masses we can find binding energies (in keV). WE only need the excess mass of proton

and neutron and the (Z,A,N) of the nuclide.

In[•]:=
```
Eneutron := ExcessMassKeys["n"];
Eproton := ExcessMassKeys["p"];
```

In[•]:=
```
BindingEnergy[name_] := Module[{Pair, A, Z, N},
                        module                  valeur
  Pair = KeySpecies[name];
  Z = Pair[[2]];
  N = Pair[[1]];
  valeur numérique
  A = Z + N;
      valeur numérique
  N Eneutron + Z Eproton - ExcessMassKeys[name]]
  valeur numérique

Mass[name_] := Module[{Pair, A, Z, N},
               module                  valeur numérique
  Pair = KeySpecies[name];
  Z = Pair[[2]];
  N = Pair[[1]];
  valeur numérique
  A = Z + N;
      valeur numérique
  A ma + keV ExcessMassKeys[name] - Z me]
```

We check a few binding energies (in keV)

In[•]:=
```
BindingEnergy["n"]
BindingEnergy["p"]
BindingEnergy["d"]
BindingEnergy["a"]
```

Out[•]=
```
0.
```

Out[•]=
```
0.
```

Out[•]=
```
2224.5663
```

Out[•]=
```
28 295.662
```

In[•]:=
```
Mass["n"] / MeV
mn / MeV
```

Out[•]=
```
939.56542
```

Out[•]=
```
939.56542
```

In[ ]:=
```
Mass["p"] / MeV
mp / MeV
```

Out[ ]=
```
938.27207
```

Out[ ]=
```
938.27209
```

In[ ]:=
```
Mass["d"] / MeV
```

Out[ ]=
```
1875.6129
```

Nuclear Statistical Equilibrium

This is Eq. A24 of companion paper.

In[ ]:=
```
YNSE[name_, Yn_, Yp_, Tv_] := Module[{Pair, N, A, Z, mN, A32Overmn},
  mN = (mn + mp) / 2;
  Pair = KeySpecies[name];
  Z = Pair[[2]];
  N = Pair[[1]];
  A = Z + N;
  A32Overmn = (Mass[name] / (mn^(A-Z) * mp^Z))^(3/2);
  (2 * SpinKeys[name] + 1) Zeta[3]^(A-1) π^((1-A)/2) 2^((3 A-5)/2) A32Overmn
   (kB Tv)^(3 (A-1)/2) (ηfactorT[Tv])^(A-1) Yp^Z Yn^(A-Z) Exp[BindingEnergy[name] * keV / (kB Tv)]
 ]
```

Reverse reaction information

The reverse reaction depends on three constants (α, β, γ) defined in companion paper in Eq. 142. From the Spin, mass and binding energy we can find these constants.

In[ ]:=
```
Qreaction[ListIn_, ListOut_] := Module[
   {Ni = Length@ListIn, Nf = Length@ListOut, factorin, factorout, Units},
   factorin = Plus @@ ((BindingEnergy[#]) & /@ ListIn);
   factorout = Plus @@ ((BindingEnergy[#]) & /@ ListOut);
   Units = keV;
   -Units (factorin - factorout)
  ];
```

In[ ]:=
```
PowerT9[ListIn_, ListOut_] :=
  Module[{Ni = Length@ListIn, Nf = Length@ListOut},
   3 / 2. * (Ni - Nf)
  ];
```

In[ ]:=
```
FactorInverseReaction[ListIn_, ListOut_] := Module[
   {Ni = Length@ListIn, Nf = Length@ListOut, factorin, factorout, Units},
   factorin =
    Times @@ ((((2 SpinKeys[#[[1]]] + 1) (2 Pi / Mass[#[[1]]] / (kB 10^9))^(-3 / 2))^
              (#[[2]]) / (#[[2]]!)) & /@ (Tally@ListIn));
   factorout =
    Times @@ ((((2 SpinKeys[#[[1]]] + 1) (2 Pi / Mass[#[[1]]] / (kB 10^9))^(-3 / 2))^
              (#[[2]]) / (#[[2]]!)) & /@ (Tally@ListOut));
   Units = ((ma / clight^2) / (hbar clight)^3)^(Ni - Nf);
   factorin / factorout Units
  ];
```

In[ ]:=
```
GatherInfoReac[ListIn_, ListOut_] :=
  {Qreaction[ListIn, ListOut] / MeV, FactorInverseReaction[ListIn, ListOut],
   PowerT9[ListIn, ListOut], -Qreaction[ListIn, ListOut] / kB / 10^9};

RemoveNonNuclear[Species_List] :=
  Select[Species, # =!= "g" && # =!= "Bm" && # =!= "Bp" &];
InfoReaction[{ListIn_, ListOut_}] :=
  GatherInfoReac[RemoveNonNuclear@ListIn, RemoveNonNuclear@ListOut];
InfoReaction[ListIn_, ListOut_] := InfoReaction[{ListIn, ListOut}]
```

For a given reaction, defined by the list of initial particles and final particles, we get these constants with the function InfoReaction.
For example

In[ ]:=
```
InfoReaction[{"n", "p"}, {"d", "g"}]
```

Out[ ]=
$$\{2.2245663,\ 4.7161418\times 10^{9},\ 1.5,\ -25.81502\}$$

Check reaction coherence (formal conservation of N and Z)

In[•]:=
```
CheckReaction[{ListIn_, ListOut_}] := Module[{Znet, Nnet, Anet},
                                      |module
  Znet = -Plus @@ (Zi /@ ListIn) + Plus @@ (Zi /@ ListOut);
          |plus                    |plus
  Nnet = -Plus @@ (Ni /@ ListIn) + Plus @@ (Ni /@ ListOut);
          |plus                    |plus
  Anet = -Plus @@ (Ai /@ ListIn) + Plus @@ (Ai /@ ListOut);
          |plus                    |plus
  (*Print[ListIn," ",ListOut," ",Znet,Nnet,Anet];*)
    |imprime
  If[Znet =!= 0 || Nnet =!= 0 || Anet =!= 0,
  |si
   Print["ERROR! This reaction ", ListIn, " -> ", ListOut,
   |imprime
    " is not possible.\nThe net result for Z, N and A are ",
                                          |valeur numérique
    Znet, " ", Nnet, " ", Anet];
   MessageDialog["A reaction is not possible. Kernel has been aborted."];
   |dialogue avec message
   Print["We abort the evaluation !"];
   |imprime
   Beep[];
   |bip
   Quit[];
   |quitte noyau de système
   (*TODO Maybe a better
     handling of errors than just a violent Quit[]... *)
                                          |quitte noyau de système
  ];
 ]

CheckReaction[ListIn_, ListOut_] := CheckReaction[{ListIn, ListOut}]
```

In[•]:=
```
(*CheckReaction[{"n"},{"p","Bm"}]*)
```

Nuclear Reaction rates

Radiative Corrections from pair productions

For photon producing processes we need to rescale the rates because of pair producing reactions. This has been computed in [Pitrou&Pospelov 2020]

In[ ]:=
```
RescaleElectricDipoleKroll[Ea_] = 1 - 10 αFS/(9 π) + 3 αFS/(4 Ea^4 π) +
    3 αFS Log[16]/(4 Ea^4 π) + 2 αFS Log[64]/(9 π) - αFS Log[4/Ea^2]/(3 π) - 3 αFS Log[4/Ea^2]/(4 Ea^4 π);
 (* Ea is the available energy in units of electron mass.*)

EML[{Z1_, A1_}, {Z2_, A2_}][T9_] =
  0.1220 * Z1^(2/3) * Z2^(2/3) * ((A1 * A2)/(A1 + A2))^(1/3) T9^(2/3) MeV;
(* This is the most likely Kinetic energy in MeV*)


Clear[RescaleElectricDipoleZAZAQ]
efface
RescaleElectricDipoleZAZAQ[{Z1_, A1_}, {Z2_, A2_}, Qv_][T9_] =
  (RescaleElectricDipoleKroll[(EML[{Z1, A1}, {Z2, A2}][T9] + Qv)/me]);

(*We use a rescaling computed at the most likely kinetic energy,
On top of which we add the mass gap to the total available energy *)
active messages

Clear[RescaleElectricDipole]
efface
RescaleElectricDipole[Nucl1_, Nucl2_, Nuclfinal_] :=
 RescaleElectricDipole[Nucl1, Nucl2, Nuclfinal] = Function[{T9},
                                                  fonction
   Evaluate@RescaleElectricDipoleZAZAQ[{Zi[Nucl1], Ai[Nucl1]}, {Zi[Nucl2],
   évalue
     Ai[Nucl2]}, (Mass[Nucl1] + Mass[Nucl2] - Mass[Nuclfinal])][T9]]
```

The some rescaling factors due to possible pair production for reactions which produce a photon in the final state. These are functions of T9 (=Temperature in GigaK).

In[ ]:=
```
RescaleElectricDipole["d", "p", "He3"][0.8]
RescaleElectricDipole["t", "a", "Li7"][0.8]
RescaleElectricDipole["He3", "a", "Be7"][0.8]
RescaleElectricDipole["t", "p", "a"][0.8]
```

Out[ ]=
```
1.0021967
```

Out[ ]=
```
1.0010648
```

Out[ ]=
```
1.0005748
```

Out[ ]=
```
1.0041641
```

In[•]:=
```
RescaleElectricDipole["d", "p", "He3"][0.008]
RescaleElectricDipole["t", "a", "Li7"][0.008]
RescaleElectricDipole["He3", "a", "Be7"][0.008]
RescaleElectricDipole["t", "p", "a"][0.008]
```

Out[•]=
```
1.0021723
```

Out[•]=
```
1.0009542
```

Out[•]=
```
1.000346
```

Out[•]=
```
1.004157
```

Random Number Generation for nuclear rates uncertainties

Generator of random number according to Normal distribution. But we make sure to use always the same sequence to avoid noise in Monte-Carlo.
This is crucial because this reduces Monte-Carlo noise when evaluating uncertainty in rates.

So for a given seed we precompute a list of 1000 random numbers.
Then we call several times MyNormalRandom[seed] which gives successively the random numbers which were generated with the seed.

In[•]:=
```
Clear[TableRandom, MyNormalRandom]
(* efface *)
$NRandomPoints = 1000;
(* We put something larger than the max number of reactions *)
TableRandom[seed_] := TableRandom[seed] = (SeedRandom[seed];
                                          (* amorçage aléatoire *)
   Table[RandomVariate[NormalDistribution[]], {i, 1, $NRandomPoints}])
   (* table  variable aléatoire  loi normale *)
```

In[•]:=
```
InitializeRandom[seed_] := (IndexRandom[seed] = 1);
RandomFromTable[seed_] := With[{r = TableRandom[seed][[IndexRandom[seed]]]},
                          (* avec *)
  IndexRandom[seed] = IndexRandom[seed] + 1;
  r]
MyNormalRandom[seed_] := RandomFromTable[seed]
```

In[•]:=
```
$Seed := 0;
Initialize[$Seed];
(* initialise *)
NormalRealisation := If[$RandomNuclearRates, MyNormalRandom[$Seed], 0];
                     (* si *)
```

Importation of reactions from external files (336 reactions)

We collect tools to read the reactions from the external file. This is low level code... because we

need to deal with syntax.

This function constructs the reverse reaction. Its arguments are the name of the reverse reaction, the front factor, the power on T9 and the Q of the reaction.

In[ ]:=
```
ReverseReaction[Name_, FrontFactor_, PoweronT9_, Qoverkb_] :=
  With[{Reversname = ToExpression["Hold@Lbar" <> Name],
    name = Evaluate[Symbol["L" <> Name]]},
   If[FrontFactor > 0,
    MySet[Reversname, Function[{Tvr}, With[{T9 = Tvr / Giga},
        FrontFactor (T9)^PoweronT9 * Exp[Qoverkb / T9] * name[Tvr]]]];,
    MySet[Reversname, Function[{Tvr}, 0]];
   ]];
```

We need a tool which replaces "2a", by "a,a" in the description of the reaction. Hence a reaction with identical particles in the initial or final states X (e.g. something which gives two neutrons) can be written X + X or 2 X.

In[ ]:=
```
ListSimplificationNotation = {"He4" → "a", "H3" → "t", "H2" → "d", "H1" → "p"};
```

In[ ]:=
```
ReplaceMultipleElement[str_String] := Module[{first, rest, rule},
  If[StringLength[str] == 1, str,
   first = StringTake[str, 1] // ToExpression;
   If[NumericQ[first],
    rest = StringDrop[str, 1];
    rule = str :> Sequence @@ Table[rest, first];
    str /. rule,
    str]
  ]
 ]
ReplaceMultipleElementList[list_List] := ReplaceMultipleElement /@ list;
```

In[ ]:=
```
ReplaceMultipleElementList[{"He4", "2H1"}]
```

Out[ ]=
```
{He4, H1, H1}
```

For a line (the list of elements of this line more precisely) describing a reaction we build the rates and inverse rates, and we output a formal description of the reaction in terms of initial and final particles

In[ ]:=
```
TreatData[Data_] :=
  Module[{reac, constants, ireaction, MaxNumberReaction, ReferencePaper,
  |module
    dat, rest, list, resultat, reacreshaped, replacements, time},
   resultat = {};
   list = Data;
   ireaction = 1;
   MaxNumberReaction :=
    If[$ReducedNetwork, SmallNuclearNetworkSize, Infinity];
    |si                                          |infini
   While[(Length@list > 0) && ireaction ≤ MaxNumberReaction,
  |pendant··  |longueur
    reac = list[[1]];
    ReferencePaper = StringDrop[list[[2, 1]], 2];
                     |laisse tomber chaîne de caractères
    (*Print[ReferencePaper];*)
      |imprime
    (*Print[reac];*)
      |imprime
    constants = list[[3]];
    rest = Drop[list, 3];
           |laisse tomber
    dat = {};
    While[rest =!= {} && NumericQ[rest[[1, 1]]],
   |pendant que          |expression numérique ?
     dat = Append[dat, rest[[1]]];
           |ajoute en fin
     rest = Rest@rest;
            |reste
    ];
    list = rest;
    reacreshaped = Append[
                   |ajoute en fin
      {ReplaceMultipleElementList@Select[reac, (# =!= "+" && # =!= "*-") &],
                                 |sélectionne
       constants, ReferencePaper}, dat];
    resultat = Append[resultat, reacreshaped /. ListSimplificationNotation];
               |ajoute en fin
    ireaction++;
   ];
   resultat
  ];
```

In[ ]:=
```
TruncateRateVariation[rate_] := Min[$MaxVariationRate, rate]
                                 |minimum
```

In[ ]:=
```
TreatReactionLine[line_] :=
  Module[{rescalefactor, reac, constants, interpfunction, data, len,
  |module
    table, Tmin, rmin, wedgeposition, colonposition, InitialParticles,
    FinalParticles, BoolenFileData, Q, FrontFactor, PoweronT9, Qoverkb,
    Name, Lname, rv, ReferencePaper, InfoFromAudi2017, timeref},
   (*timeref=AbsoluteTime[];*)
              |temps absolu
   reac = line[[1]];
   (*Print["Treating reaction : ",reac];*)
     |imprime
   constants = line[[2]];
   ReferencePaper = line[[3]];
   data = line[[4]];

   len = Length@line;
         |longueur
   wedgeposition = Position[reac, ">"][[1, 1]];
                   |position
   colonposition = Position[reac, ";"][[1, 1]];
                   |position
   InitialParticles = Take[reac, {1, wedgeposition - 1}];
                      |prends
   FinalParticles = Take[reac, {wedgeposition + 1, colonposition - 1}];
                    |prends

   (* We quit if the reaction does not conserve formally Z or N,
                                                          |valeur numérique
   that is if it cannot exist *)
   CheckReaction[InitialParticles, FinalParticles];


   Name = StringJoin @@ ToString /@ InitialParticles <>
          |joins chaînes de ⋯ |en chaîne de caractères
     "TO" <> StringJoin @@ ToString /@ FinalParticles;
             |joins chaînes de ⋯ |en chaîne de caractères

   Q = constants[[1]];
   FrontFactor = constants[[2]];
   PoweronT9 = constants[[3]];
   Qoverkb = constants[[4]];


   (* We check the constants used in reverse rates *)
   InfoFromAudi2017 = InfoReaction[InitialParticles, FinalParticles];
   (*Print[InitialParticles," ",FinalParticles," ",InfoFromAudi2017];*)
     |imprime
```

```
If[Abs[FrontFactor / InfoFromAudi2017[[2]] - 1] > 0.001,
 Print[Name, " WARNING. We use α=", FrontFactor,
  " but we should use ", InfoFromAudi2017[[2]]
];

If[Abs[Qoverkb / InfoFromAudi2017[[4]] - 1] > 0.001,
 Print[Name, " WARNING. We use Q/k_B=",
  Qoverkb, " but we should use ", InfoFromAudi2017[[4]]
];

If[PoweronT9 =!= InfoFromAudi2017[[3]],
 Print[Name, " WARNING. We use power on T9 =",
  PoweronT9, " but we should use ", InfoFromAudi2017[[3]]
];
(* ****************)



(* *** *)
(*For exploration of parameters we can redefine some front
 factors to rescale reactions. For instance the DPG reaction*)
(* Added on request of Antony Lewis *)

rescalefactor = 1;
(*Print[Name,FullForm[Name]];*)

If[$RescaleSomeRates,
 If[NumericQ[Symbol[Name <> "Factor"]],
  rescalefactor = Symbol[Name <> "Factor"];
  Print[Name, "  reaction is rescaled by ", rescalefactor];
 ]
];




table = Map[{Giga #[[1]], #[[2]] Hz, #[[3]]} &, data];
```

```
Tmin = Last[table][[1]];
       dernier
rmin = Last[table][[2]];
       dernier
Lname = ToExpression["Hold@L" <> Name];
        en expression      maintiens
rv = NormalRealisation;

(* We also rescale some nuclear according to QED corrections
 (pair productions in the final state if it produces a γ *)
(* This is based on [Pitrou&Posepelov 2020] *)
RescaleFunction[T9_] := 1;
(* Possible RescaleFunction for some selected rates,
in function of T9*)

(* This is the np→dg correction from pair particle production,
found from our electric/dipole modelization of base rate,
etc... using Landau-Lifshitz.
  See [Pitrou&Pospelov 2020]*)

If[$NuclearRatesQEDCorrections,
si

 If[Name === "npTOdg", RescaleFunction[T9_] :=
 si
    Min[1.0009003934476768`, (1.0003328617393168` +
    minimum
       0.00010013475534938917` T9 + 0.00004089993260910648` T9^2 -
       0.00001182467353722953 5` T9^3 + 1.0522377796855455`*^-6 T9^4)];];

 If[Name === "dpTOHe3g",
 si
  RescaleFunction[T9_] := RescaleElectricDipole["d", "p", "He3"][T9];];

 If[Name === "tpTOag",
 si
  RescaleFunction[T9_] := RescaleElectricDipole["t", "p", "a"][T9];];

 If[Name === "taTOLi7g",
 si
  RescaleFunction[T9_] := RescaleElectricDipole["t", "a", "Li7"][T9];];

 If[Name === "He3aTOBe7g",
 si
  RescaleFunction[T9_] := RescaleElectricDipole["He3", "a", "Be7"][T9];];
];

MySet[Lname, MyInterpolationRate[
```

```
      {#[[1]], Identity[rescalefactor * RescaleFunction[#[[1]] / Giga] * #[[2]] *
                identité
            If[$RandomNuclearRates, TruncateRateVariation[#[[3]]^rv], 1]]} & /@
            si
       table]];
    (* We do not rescale the reverse because
     it is computed FROM the forward rate. So rescaling the
     forward rate by rescalefactor rescales them both *)
    ReverseReaction[Name, FrontFactor, PoweronT9, Qoverkb];
    (*Print["Duration for reaction ",
      imprime   durée
      Name," is ",AbsoluteTime[]-timeref];*)
                  temps absolu

    {Name, InitialParticles, FinalParticles, rv, ReferencePaper}

   ];

SetAttributes[TreatReactionLine, SequenceHold]
alloue attributs                  maintiens séquence
```

Lists of analytic reactions (86 reactions)

We have a list of 86 reactions for which we use analytic fits from the literature
In principle these reactions could be tabulated and incorporated into the external file but we prefer to keep their analytic forms.

- We need a few tools (this is painful low level code)

The Wagoner list not enough if we want to interpolate the analytic expressions . But here it is :

In[•]:=
```
ListTWagoner =
  {0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.011,
    0.012, 0.013, 0.014, 0.015, 0.016, 0.018, 0.02, 0.025, 0.03, 0.04,
    0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16,
    0.18, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1.,
    1.25, 1.5, 1.75, 2., 2.5, 3., 3.5, 4., 5., 6., 7., 8., 9., 10.} * 10^9;
```

In[ ]:=
```
ListTWagonerRange[T1_, T2_] := Module[{len = Length@ListTWagoner,
                                       |module          |longueur
   imindown, imaxup, Tmin = Min[T1, T2], Tmax = Max[T1, T2]},
                            |minimum             |maximum
  imindown = Max[1,
             |maximum
    -1 + Position[ListTWagoner, SelectFirst[ListTWagoner, # > Tmin &]][[1, 1]]];
         |position              |sélectionne premier
  imaxup = Min[len,
           |minimum
    Position[ListTWagoner, SelectFirst[ListTWagoner, # ≥ Tmax &]][[1, 1]]];
   |position              |sélectionne premier
  ListTWagoner[[imindown ;; imaxup]]
 ]
```

In[ ]:=
```
$ListTWagoner = False;
                |faux
TableInterpolationTemperature =
  If[$ListTWagoner, ListTWagoner, ListTRange[0.9 Tf, 10^10]];
  |si
```

In[ ]:=
```
SimplifyReactionStringRules =
  {"+" → " + ", ">" → " > ", ";" → " ; ", "2n" → " n + n ", "2p" → " p + p ",
   "2g" → " g + g ", "2d" → " d + d ", "2a" → " a + a ", "He4" → " a "};

ReshapheReactionString[string_String] := Select[
                                 |chaîne de caractères  |sélectionne
   StringSplit[StringReplace[string, SimplifyReactionStringRules], " "],
  |fractionne chaî···  |remplace dans chaîne de caractères
   (# =!= "+" && # =!= "*-" && # =!= "") &];

TreatReactionString[reac_String, source_String, f_] :=
                          |chaîne de caract···  |chaîne de caractères
 Module[{reacshaped, wedgeposition, colonposition,
 |module
   InitialParticles, FinalParticles, Name},
  reacshaped = ReshapheReactionString[reac];
  wedgeposition = Position[reacshaped, ">"][[1, 1]];
                  |position
  colonposition = Position[reacshaped, ";"][[1, 1]];
                  |position
  InitialParticles = Take[reacshaped, {1, wedgeposition - 1}];
                     |prends
  FinalParticles = Take[reacshaped, {wedgeposition + 1, colonposition - 1}];
                   |prends

  (* We check that the reaction is possible,
  that is it should conserve N and Z*)
                                  |valeur numérique
  (* If not the case,
     |si
  the code will violently quit after spitting out warning messages.*)
  CheckReaction[InitialParticles, FinalParticles];

  Name = StringJoin @@ ToString /@ InitialParticles <>
         |joins chaînes de ···  |en chaîne de caractères
    "TO" <> StringJoin @@ ToString /@ FinalParticles;
            |joins chaînes de ···  |en chaîne de caractères
  {Name, InitialParticles, FinalParticles, NormalRealisation, source, f}
 ]
```

In[ ]:=
```
PostTreatT9[var_, funT9_] := If[$InterpolateAnalytics,
                             |si
   MyInterpolationRate[Table[
                      |table
     {i, var MyChop[funT9[i / 10^9]]}, {i, TableInterpolationTemperature}]],
   (var MyChop[funT9[# / 10^9]]) &];

GenRateT9[var_, funT9_] := PostTreatT9[var, funT9];
```

- This is the actual function where all analytic rates are defined. And it outputs the list of reactions.

In[•]:=
```
DefineAnalyticRates :=
  Module[{f, Var, Name, λReac, λbarReac, treatedreac, source, reac,
   analyticforward, AddReaction, initialparticles, finalparticles,
   InfoFromAudi2017, FrontFactor, Qoverkb, PoweronT9, forward, reslist},

  (* Most recent implementation
   with automatic computation of reverse rate *)
  AddReaction[reac_String, source_String, f_, ForwardT9_, BoolBackward_] := (
    treatedreac = TreatReactionString[reac, source, f];
    Name = treatedreac[[1]];

    (* Building the backward ratio *)
    initialparticles = treatedreac[[2]];
    finalparticles = treatedreac[[3]];
    InfoFromAudi2017 = InfoReaction[initialparticles, finalparticles];
    (*Print[InitialParticles," ",FinalParticles," ",InfoFromAudi2017];*)
    FrontFactor = InfoFromAudi2017[[2]];
    Qoverkb = InfoFromAudi2017[[4]];
    PoweronT9 = InfoFromAudi2017[[3]];
    (* End of building backward ratio *)

    λReac = ToExpression["Hold@L" <> Name];
    λbarReac = ToExpression["Hold@Lbar" <> Name];
    Sow[treatedreac];
    Var = f^treatedreac[[4]];

    MySet[λReac, GenRateT9[Var, ForwardT9]];
    (*MySet[λbarReac,GenRateT9[Var,BackwardT9 ]];*)
    If[BoolBackward,
     MySet[λbarReac, GenRateT9[Var,
        (FrontFactor * #^PoweronT9 * Exp[Qoverkb / #] * ForwardT9[#]) & ]];,
     MySet[λbarReac, GenRateT9[0, 0 & ]];(* No backward reaction *)
    ];

    treatedreac);
```

```
reslist = Reap[
          |récolte

  (* This is where all extra analytic reactions must be listed.*)
  (* For each reactions added analytically we need to specify
     |boucle for
   a String source which is the paper in which it is found *)
     |chaîne de caractères
  (* Then we give a string reac which is the reaction considered.*)
  (* The factor of incertainty for Monte-Carlo is then given*)
  (* The analytic function forward[T9_],
  which is a function of T9 (that is the temperature in GK)*)
  (* With all these definitions we call AddReaction. The
     |avec
   last argument is a boolean. If True it computes also
                                  |si  |vrai
   the reverse rate from detailed balance arguments,
  and if False it does not do so. This is essentially for pure decay
         |faux
   reactions that there is no need to compute the reverse rates.*)

  (**=========================================================================
   *4He,3He,D,7Li (Extra reactions)
             |dérivée d
    ============================================================
      ==========*)
  source = "Nag06";
  reac = " d + n  > t + g ; dng";
  f = 2.;
  forward[T9_] := With[{T923 = T9^(2 / 3)}, (214. T9^0.075 + 7.42 T9)];
                  |avec
  AddReaction[reac, source, f, forward, True];
                                        |vrai
  (* End of first reaction added analytically *)
     |termine

  source = "Nag06";
  reac = "t+t>a+n+n;ttn";
  f = 3.;
  forward[T9_] := With[{T923 = T9^(2 / 3), T913 = T9^(1 / 3),
                  |avec
     T943 = T9^(4 / 3), T953 = T9^(5 / 3)}, (1 / T923 1.67*^9 E^(-4.872/T913
       (1. - 0.272 T9 + 0.086 T913 - 0.455 T923 + 0.148 T943 + 0.225 T953))];
  AddReaction[reac, source, f, forward, True];
                                        |vrai



  source = "Wag69";
```

```
reac = "He3 +n > He4 + g ; hng";
f = 3.;
forward[T9_] := 6.62 * (1 + 905 * T9);
AddReaction[reac, source, f, forward, True];
                                          vrai


source = "CF88";
reac = "He3 + t > He4 + d ; htd";
f = 10.;
forward[T9_] := With[{T9A = T9 / (1. + 0.128 * T9), T932 = T9 ^ (3 / 2)},
                avec
  With[{T9A13 = T9A ^ (1. / 3.), T9A56 = T9A ^ (5. / 6.)},
  avec
   5.46*^9 * T9A56 / T932 * Exp[-7.733 / T9A13]
                            exponentielle
  ]];
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88";
reac = "He3 + t > He4 + n + p ; htp";
f = 10.;
forward[T9_] := With[{T9A = T9 / (1. + 0.115 * T9), T932 = T9 ^ (3 / 2)},
                avec
  With[{T9A13 = T9A ^ (1. / 3.), T9A56 = T9A ^ (5. / 6.)},
  avec
   7.71*^9 * T9A56 / T932 * Exp[-7.733 / T9A13]
                            exponentielle
  ]];
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "NACRE";
reac = "a + a + n > Be9 + g ; aang";
f = 1.25;
forward[T9_] :=
 With[{T932 = T9 ^ (3 / 2), T923 = T9 ^ (2 / 3), T913 = T9 ^ (1 / 3)},
 avec
  With[{he4abe8 = 2.43*^9 * (1. + 74.5 * T9) / T923 * Exp[
  avec                                                exponentielle
         -13.49 / T913 - (T9 / 0.15) ^ 2] + 6.09*^5 / T932 * Exp[-1.054 / T9]},
                                                     exponentielle
   If[T9 < 0.03,
   si
    (he4abe8) * 6.69*^-12 * (1. - 192 * T9 + 2.48*^4 * T9 ^ 2 -
       1.50*^6 * T9 ^ 3 + 4.13*^7 * T9 ^ 4 - 3.90*^8 * T9 ^ 5),
```

```
      (he4abe8) * 2.42*^-12 * (1. - 1.52 * Log10[T9] +
                                          logarithme en base 10
        0.448 * (Log10[T9]) ^ 2 + 0.435 * (Log10[T9]) ^ 3)]];
                 logarithme en base 10      logarithme en base 10
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88&MF89";
reac = "Li7 + t > a + a + n + n; li7ta";
f = 3.;
forward[T9_] := With[{T923 = T9 ^ (2 / 3), T913 = T9 ^ (1 / 3)},
                  avec
  8.81*^+11 / T923 * Exp[-11.333 / T913]];
                     exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88&MF89";
reac = "Li7 + He3 > a + a + n + p; li7haa";
f = 3.;
forward[T9_] := With[{T923 = T9 ^ (2 / 3), T913 = T9 ^ (1 / 3)},
                  avec
  1.11*^+13 / T923 * Exp[-17.989 / T913]];
                     exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai
(* Idem problem T93 not divided in Coc *)



source = "Bal95";
reac = " Li8 + d > Li9 + p ; li8dp";
f = 3.;
forward[T9_] := With[{T923 = T9 ^ (2 / 3), T913 = T9 ^ (1 / 3)},
                  avec
  9.63*^6 / T923 * Exp[-10.324 / T913] * (1. + 0.404 * T913) * 74.];
                   exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "Has09c";
reac = " Li8 + d > Li7 + t ; li8dt";
f = 3.;
forward[T9_] := With[{T923 = T9 ^ (2 / 3), T913 = T9 ^ (1 / 3)},
                  avec
  (3.02*^8 / T9 ^ 0.624 * Exp[-3.51 / T9] +
                           exponentielle
    5.82*^11 / T923 * Exp[-19.72 / T913] * (1.0 + 0.280 * T913))];
                      exponentielle
```

```
AddReaction[reac, source, f, forward, True];
                                          vrai



(*Now given in tabulated file,
  maintenant
hence we comment out this old analytic form *)
(*source="CF88";
reac="Be7 + d > a + a + p ; be7dp";
f=3.;
forward[T9_]:=
 With[{T923=T9^(2/3),T913=T9^(1/3)},1.07*^+12/T923*Exp[-12.428/T913]];
 avec                                                  exponentielle
AddReaction[reac,source,f,forward,True];*)
                                          vrai



source = "CF88&MF89";
reac = "Be7 + t > a + a + n + p ; be7t";
f = 3.;
forward[T9_] := With[{T923 = T9^(2 / 3), T913 = T9^(1 / 3)},
                avec
  2.91*^+12 / T923 * Exp[-13.729 / T913]];
                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88&MF89";
reac = "Be7 + He3 > 2a + p + p  ; be7h";
f = 3.;
forward[T9_] := With[{T923 = T9^(2 / 3), T913 = T9^(1 / 3)},
                avec
  6.11*^+13 / T923 * Exp[-21.793 / T913]];
                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "Wie89";
reac = "C9 + a > N12 + p ; c9an";
f = 3.;
forward[T9_] := With[{T923 = T9^(2 / 3), T932 = T9^(3 / 2),
                avec
   T913 = T9^(1 / 3), T943 = T9^(4 / 3), T953 = T9^(5 / 3)},
  (1.668*^+15 / T923 * Exp[-31.272 / T913 - (T9 / .307)^2] *
                    exponentielle
     (1. + 1.33*^-2 * T913 - 6.42 * T923 - .599 * T9 + 14.4 * T943 + 3.42 * T953) +
```

```
    56.8 / T932 * Exp[-5.292 / T9] + 1.7*^+5 / T932 * Exp[-14.08 / T9] +
                  exponentielle                       exponentielle
    6.52*^7 / T932 * Exp[-23.09 / T9])];
                     exponentielle
AddReaction[reac, source, f, forward, True];
                                      vrai




(* *=================================================================
   =======
 *6Li (Extra reactions)
*============================================================
    =============*)

source = "CF88";
f = 3.;
(*(*This is an endothermic reaction. So we comment it and
 replace it with the exothermic reverse reaction below *)
reac="t+a>Li6+n;tan";
forward[T9_]:=
 With[{T9A=T9/(1.+49.18*T9)},With[{ T9A32=T9A^(3./2.),T932=T9^(3/2)},
 avec                               avec
   (1.80*^8*Exp[-55.494/T9]*(1.-.261*T9A32/T932)+
              exponentielle
     2.72*^9/T932*Exp[-57.884/T9])
                    exponentielle
  ]];*)


(* Here is the same reaction but presented backward,
   ici
such that it is exothermic in the forward direction *)
reac = "Li6+n>t+a;tan";
forward[T9_] := With[{T9A = T9 / (1. + 49.18 * T9)},
                avec
  With[{ T9A32 = T9A^(3. / 2.), T932 = T9^(3 / 2)},
  avec
    (1.80*^+8 * (1. - .261 * T9A32 / T932) * .935 +
      2.72*^9 / T932 * Exp[(55.494 - 57.884) / T9] * .935)
                     exponentielle
  ]];
AddReaction[reac, source, f, forward, True];
                                      vrai



source = "FK90";
reac = "He3 + t > Li6 + g ; htg";
f = 3.;
forward[T9_] :=
```

```
 With[{T92 = T9^2, T923 = T9^(2/3), T932 = T9^(3/2), T913 = T9^(1/3),
 avec
   T943 = T9^(4/3), T953 = T9^(5/3)}, 2.21*^5/T923*Exp[-7.720/T913]*
                                                    exponentielle
   (1. + 2.68*T923 + 0.868*T9 + 0.192*T943 + 0.174*T953 + 0.044*T92)];
AddReaction[reac, source, f, forward, True];
                                    vrai



 source = "CF88";
reac = "a + n + p > Li6 + g ; anpg";
f = 3.;
forward[T9_] :=
 If[T9 > 1, 4.62*^-6/T9^2*(1. + 0.075*T9)*Exp[-19.353/T9], 0];
 si                                      exponentielle
AddReaction[reac, source, f, forward, True];
                                    vrai



 source = "MF89";
reac = "Li6 + n > Li7 + g ; li6ng";
f = 3.;
forward[T9_] := 5.10*^3;
AddReaction[reac, source, f, forward, True];
                                    vrai


 source = "MF89";
reac = "Li6 + d > Li7 + p ; li6dp";
f = 3.;
forward[T9_] := With[{T923 = T9^(2/3), T913 = T9^(1/3)},
                avec
  1.48*^12/T923*Exp[-10.135/T913]];
                   exponentielle
AddReaction[reac, source, f, forward, True];
                                    vrai


 source = "MF89";
reac = "Li6 + d > Be7 + n ; li6dn";
f = 3.;
forward[T9_] := With[{T923 = T9^(2/3), T913 = T9^(1/3)},
                avec
  1.48*^12/T923*Exp[-10.135/T913]];
                   exponentielle
AddReaction[reac, source, f, forward, True];
                                    vrai



(**=================================================================:
  *Berylium& Boron (Main reactions)
 *=============================================================
```

```
      =============*)
source = "CF88";
reac = "Li6 + a > B10 + g ; li6ag";
f = 3.;
forward[T9_] := With[{T923 = T9^(2 / 3), T932 = T9^(3 / 2),
                avec
    T913 = T9^(1 / 3), T943 = T9^(4 / 3), T953 = T9^(5 / 3)},
   (4.06*^06 / T923 * Exp[-18.79 / T913 - (T9 / 1.326)^2] *
                    exponentielle
      (1. + 0.022 * T913 + 1.54 * T923 + 0.239 * T9 + 2.2 * T943 + 0.869 * T953)
     + 1.91*^3 / T932 * Exp[-3.484 / T9] + 1.01*^4 / T9 * Exp[-7.269 / T9])];
                          exponentielle                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "NACRE";
reac = " Li7 + a > B10 + n ; li7an / b10na";
f = 1.08;
forward[T9_] :=
 1.325 * 1.66*^7 * (1. + 1.064 * T9) * 1 / 1.3242 * Exp[-32.3755 / T9];
                                                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai


(* MF89 remplace Wiescher et al.ApJ 464 (1989) 464.C Voir Blackmon
                                                        constante C
   et al.PRC 54 (1996) 383& Heil et al.ApJ 507 (1998) 997.*)
 (*%MF89Hei98   *)
source = "MF89&Hei98";
reac = "Li7+n>Li8+g;li7ng";
f = 3.;
forward[T9_] :=
 With[{T932 = T9^(3 / 2)}, (6.015*^3 + 1.141*^4 / T932 * Exp[-2.576 / T9])];
 avec                                                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



(*Should we replace by exothermic reaction ? *)
source = "MF89";
reac = "Li7 + d > Li8 + p ; li7dp ! Q<0 !";
f = 3.;
forward[T9_] :=
 With[{T932 = T9^(3 / 2)}, 8.31*^8 / T932 * Exp[-6.998 / T9]];
 avec                                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai
```

```
source = "Rau94";
reac = "Li8 + n > Li9 + g ; li8ng";
f = 3.;
forward[T9_] :=
 With[{T932 = T9^(3 / 2)}, (3.260*^3 + 6.328*^4 / T932 * Exp[-2.866 / T9])];
 avec                                                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai


source = "Men12";
reac = "Li8 + p > a + a + n ; li8pn";
forward[T9_] := With[{T932 = T9^(3 / 2), T913 = T9^(1 / 3),
                 avec
   T923 = T9^(2 / 3), T92 = T9^2, T93 = T9^3, T94 = T9^4, T95 = T9^5},
  If[T9 < 5, (
  si
    5.36*^8 / T932 * Exp[-4.41 / T9] + 1.99*^8 / T932 * Exp[-7.08 / T9] +
                 exponentielle                 exponentielle
     5.85*^10 / T923 * Exp[-8.50 / T913] * (1. - 1.70 * T9 +
                 exponentielle
        0.849 * T92 - 0.175 * T93 + 1.62*^-2 * T94 - 5.60*^-4 * T95)),
   7.777*^7]];
AddReaction[reac, source, f, forward, True];
                                          vrai


source = "Bal95";
reac = "Li8 + d > Be9 + n ; li8dn";
f = 3.;
forward[T9_] := With[{T913 = T9^(1 / 3), T923 = T9^(2 / 3)},
                 avec
  9.63*^6 / T923 * Exp[-10.324 / T913] * (1. + 0.404 * T913) * 188.];
                  exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai


(**========================================================================
 *Berylium& Boron (Extra reactions)
 *=========================================================
    ============= *)
source = "Rau94";
reac = "Be9 + n > Be10 + g ; be9ng";
f = 3.;
forward[T9_] :=
 With[{T913 = T9^(1 / 3), T923 = T9^(2 / 3), T932 = T9^(3 / 2)}, (1.01*^3 +
 avec
    1.01*^4 / T932 * Exp[-6.487 / T9] + 5.41*^4 / T932 * Exp[-8.471 / T9])];
                 exponentielle                 exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai
```

```
source = "NACRE";
reac = "Be9 + p > a + a + p + n ; be9pn";
f = 1.05;
forward[T9_] :=
 5.06*^7 * Exp[-21.479 / T9] * (1. + 1.26 * T9 - 0.0302 * T9 ^ 2);
AddReaction[reac, source, f, forward, True];
```

```
source = "NACRE";
reac = "B11 + p > C11 + n ; b11pn ! Q < 0 !";
f = 1.1;
forward[T9_] := 1.36*^8 * Exp[-32.085 / T9] *
  (1. + 0.963 * T9 - 0.285 * T9 ^ 2 + 3.36*^-2 * T9 ^ 3 - 1.37*^-3 * T9 ^ 4);
AddReaction[reac, source, f, forward, True];
```

```
source = "Rau94";
reac = " Be10 + n > Be11 + g ; be10ng";
f = 3.;
forward[T9_] :=
 With[{T932 = T9 ^ (3 / 2)}, (5.96*^2 + 6.67*^5 / T932 * Exp[-14.85 / T9]) ];
AddReaction[reac, source, f, forward, True];
```

```
source = "Rau94";
reac = "Be11 + n > Be12 + g ; be11ng";
f = 3.;
forward[T9_] := With[{T932 = T9 ^ (3 / 2)}, 3.56*^2 ];
AddReaction[reac, source, f, forward, True];
```

```
source = "Des99Bea01";
reac = "B8 + p > C9 + g ; b8pg";
f = 3.;
forward[T9_] := With[{T932 = T9 ^ (3 / 2), T913 = T9 ^ (1 / 3), T92 = T9 ^ 2},
  6.253*^5 * Exp[-11.971 / T913] * (1. - 7.03*^-2 * T9 + 6.25*^-3 * T92)];
AddReaction[reac, source, f, forward, True];
```

```
(* ====================================================================
    *Leaks to CNO
 !*=================================================
   ================*)

source = "NACRE";
reac = "a + a + a > C12 + 2g ; aaag";
f = 1.15;
forward[T9_] :=
 With[{T932 = T9^(3/2), T923 = T9^(2/3), T913 = T9^(1/3)},
 avec
  With[{he4abe8 = 2.43*^9 * (1. + 74.5 * T9) / T923 * Exp[
  avec                                                exponentielle
        -13.49 / T913 - (T9 / 0.15)^2] + 6.09*^5 / T932 * Exp[-1.054 / T9],
                                                          exponentielle
    be8agc12 = 2.76*^7 *
       (1. + 5.47 * T9 + 326 * T9^2) / T923 * Exp[-23.570 / T913 - (T9 / 0.4)^2] +
                                              exponentielle
      130.7 / T932 * Exp[-3.338 / T9] + 2.51*^4 / T932 * Exp[-20.307 / T9]},
                  exponentielle                  exponentielle
   If[T9 < 0.03,
   si
    he4abe8 * be8agc12 * 3.07*^-16 * (1. - 29.1 * T9 + 1308 * T9^2),
    he4abe8 * be8agc12 * 3.44*^-16 * (1. + 0.0158 / T9^0.65)]]];
AddReaction[reac, source, f, forward, True];
                                    vrai


source = "Tang03";
reac = "C11+p>N12+g;c11pg";
f = 3.;
forward[T9_] :=
 With[{T923 = T9^(2/3), T932 = T9^(3/2), T913 = T9^(1/3),
 avec
   T943 = T9^(4/3), T953 = T9^(5/3)}, (1.670*^2 * Exp[-4.166 / T9] / T932 +
                                       exponentielle
    2.148*^5 * Exp[-13.281 / T913] / T923 * (1. + 4.639 * T913 -
                exponentielle
       2.641 * T923 - 1.543 * T9 + 2.030 * T943 + 4.657 * T953))];
AddReaction[reac, source, f, forward, True];
                                    vrai



source = "CF88";
reac = "B10 + a > N13 + n ; b10an";
f = 3.;
forward[T9_] := With[{T923 = T9^(2/3), T913 = T9^(1/3)},
                avec
  1.2*^13 / T923 * Exp[-27.989 / T913 - (T9 / 9.589)^2] ];
                  exponentielle
AddReaction[reac, source, f, forward, True];
                                    vrai
```

```
     source = "Wan91";
reac = "B11+a>C14+p;b11ap";
f = 3.;
forward[T9_] := With[{T923 = T9^(2 / 3), T932 = T9^(3 / 2),
                avec
   T913 = T9^(1 / 3), T943 = T9^(4 / 3), T953 = T9^(5 / 3)},
  (8.403*^15 * Exp[-31.914 / T913 - (T9 / 0.3432)^2] * (1. + 0.022 * T913 +
              exponentielle
       5.712 * T923 + 0.642 * T9 + 15.982 * T943 + 4.062 * T953)
    + 5.44*^-3 / T932 * Exp[-2.868 / T9] +
                    exponentielle
    2.419*^2 / T932 * Exp[-5.147 / T9] + 4.899*^2 / T932 * Exp[-5.157 / T9] +
                    exponentielle                          exponentielle
    4.944*^6 / T9^(3 / 5) * Exp[-11.26 / T9])];
                              exponentielle
AddReaction[reac, source, f, forward, True];
                                    vrai


source = "Rau94";
reac = "C11+n>C12+g;c11ng";
f = 3.;
forward[T9_] := With[{T932 = T9^(3 / 2)}, (3.18*^4 +
                avec
    3.30*^3 / T932 * Exp[-0.917 / T9] + 1.05*^6 / T932 * Exp[-5.57 / T9])];
                    exponentielle                  exponentielle
AddReaction[reac, source, f, forward, True];
                                    vrai


(*=====================================================================
   ====*)
(*       Decay Rates                                                   *)
(*=====================================================================
   ====*)
(* %Aud03  *)
(* All decay rates from %Aud03  *)
   tout

source = "Aud03";
reac = "He6>Li6+Bm;";
forward[T9_] := Log[2] / 8.0670*^-1 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                    faux
(* The 1 is because we do not put uncertainty on decays,
and the False because we do not put reverse reactions on decays *)
        faux
```

```
reac = "Li8>2a+Bm;";
forward[T9_] := Log[2] / 8.4030*^-1 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Li9>Be9+Bm;";
forward[T9_] := Log[2] / 1.7830*^-1 * 0.492 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Li9>a+a+n+Bm;";
forward[T9_] := Log[2] / 1.7830*^-1 * 0.508 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Be11>B11+Bm;";
forward[T9_] := Log[2] / (1.3810*^1) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Be12>B12+Bm;";
forward[T9_] := Log[2] / (2.15*^-2) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "B8>a+a+Bp;";
forward[T9_] := Log[2] / (7.70*^-1) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "B12>C12+Bm;";
forward[T9_] := Log[2] / (2.02*^-2) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "B13>C13+Bm;";
(* !04/11/2010 *)
forward[T9_] := Log[2] / (1.733*^-2) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "B14>C14+Bm;";
(* !04/11/2010 *)
```

```
forward[T9_] := Log[2] / (1.25*^-2) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "B15>C15+Bm;";
(* !04/11/2010 *)
forward[T9_] := Log[2] / (9.87*^-3) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "C9>a+a+p+Bp;";
forward[T9_] := Log[2] / (1.26*^-1) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "C10>B10+Bp;";
forward[T9_] := Log[2] / (19.29) ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "C11>B11+Bp;";
forward[T9_] := Log[2] / 1.2234*^3 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "C15>N15+Bm;";
(*28/10/2010*)
forward[T9_] := Log[2] / 2.449 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "C16>N16+Bm;";
(*14/01/2011*)
forward[T9_] := Log[2] / 7.4700*^-1 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "N12>C12+Bp;";
forward[T9_] := Log[2] / 1.100*^-2 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                      faux


reac = "N13>C13+Bp;";
```

```
(*14/01/2011*)
forward[T9_] := Log[2] / 5.979*^2 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "N16>O16+Bm;";
(*14/01/2011*)
forward[T9_] := Log[2] / 7.13 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "N17>O16+n+Bm;";
(*14/01/2011*)
forward[T9_] :=  Log[2] / 4.1730 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "O13>N13+Bp;";
(*14/01/2011*)
forward[T9_] :=  Log[2] / 8.58*^-3 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "O14>N14+Bp;";
(*14/01/2011*)
forward[T9_] := Log[2] / 70.598 ;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "O15>N15+Bp;";
(*14/01/2011*)
forward[T9_] := Log[2] / 122.24;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "O19>F19+Bm;";
(*14/01/2011*)
forward[T9_] := Log[2] / 26.464;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


 reac = "O20>F20+Bm;";
(*14/01/2011*)
```

```
forward[T9_] := Log[2] / 13.51;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "F17>O17+Bp;";
(*04/11/2010*)
forward[T9_] := Log[2] / 64.49;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "F18>O18+Bp;";
(*04/11/2010*)
forward[T9_] := Log[2] / 6.5863*^3;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "F20>Ne20+Bm;";
(*04/11/2010*)
forward[T9_] := Log[2] / 11.1630;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Ne18>F18+Bp;";
(*04/11/2010*)
forward[T9_] := Log[2] / 1.6720;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Ne19>F19+Bp;";
(*04/11/2010*)
forward[T9_] := Log[2] / 17.296;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Ne23>Na23+Bm;";
(*04/11/2010*)
forward[T9_] := Log[2] / 37.240;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                        faux


reac = "Na20>Ne20+Bp;";
(*14/01/2011*)
forward[T9_] := Log[2] / 4.4790*^-1;
                logarithme
```

```
AddReaction[reac, source, 1, forward, False];
                                          faux


reac = "Na21>Ne21+Bp;";
(*04/11/2010*)
forward[T9_] := Log[2] / 22.49;
                logarithme
AddReaction[reac, source, 1, forward, False];
                                          faux


(* *================================================================
   =======
   *New reactions following Thomas,Schramm et al.1993;
1994
 *================================================================
   ====*)

source = "Efr96";
reac = "He4 + 2n  > He6 + g ;";
f = 3.;
forward[T9_] := If[T9 < 2,
                si
  (2.65*^-3 * T9 ^ 2.555 * Exp[0.181 / Max[T9, .1]]),
                          exponentielle  maximum
  (2.93*^-1 * T9 ^ (-3.51*^-1) * Exp[-5.24 / T9])];
                                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "Iga95";
reac = "O16 + n  > O17 + g ;";
f = 3.;
forward[T9_] := (2.7*^1 + 1.38*^4 * T9 );
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88";
reac = "N14 + n  > C14 + p ;";
f = 3.;
forward[T9_] :=
 With[{T912 = T9 ^ (1 / 2)}, ( 7.19*^5 * (1. + .361 * T912 + .502 * T9) +
 avec
     3.34*^8 / T912 * Exp[-4.983 / T9]) * .333];
                          exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai
```

```
source = "CF88";
reac = "O14 + n  > N14 + p ;";
f = 3.;
forward[T9_] := With[{T912 = T9^(1 / 2)},
                avec
  (6.74*^7 * (1. + 0.658 * T912 + 0.379 * T9) * 2.99 )];
AddReaction[reac, source, f, forward, True];
                                        vrai



source = "Wie87";
reac = "O14 + a  > Ne18 + g ;";
f = 3.;
forward[T9_] :=
 With[{T932 = T9^(3 / 2)}, ( 1.16*^-1 / T932 * Exp[-11.73 / T9] +
 avec                                    exponentielle
    3.40*^1 / T932 * Exp[-22.61 / 79] + 9.10*^-3 * T9^5 * Exp[-12.159])];
                    exponentielle                  exponentielle
AddReaction[reac, source, f, forward, True];
                                        vrai



source = "NACRE";
reac = "C11 + a  > N14 + p ;";
f = 2.;
forward[T9_] := With[{T913 = T9^(1 / 3), T92 = T9^2},
                avec
  (0.2719 * 3.01*^16 * Exp[-31.884 / T913] *
                      exponentielle
    Exp[-1.379 * T9 + .215 * T92 - 2.13*^-2 * T92 * T9 + 8*^-4 * T92 * T92] *
    exponentielle
    (1. + 0.14 * Exp[-.275 / T9 - .210 * T9]) )];
                exponentielle
AddReaction[reac, source, f, forward, True];
                                        vrai



source = "Bar97C";
reac = "O14 + a  > F17 + p ;";
f = 3.;
forward[T9_] := With[{T932 = T9^(3 / 2), T923 = T9^(2 / 3),
                avec
   T913 = T9^(1 / 3), T943 = T9^(4 / 3), T953 = T9^(5 / 3)},
  With[{offset = 1.330*^5 / T932 * Exp[-11.86 / T9] +
  avec                                  exponentielle
      8.42*^-47 * T932 * Exp[-0.453 / T9] + 6.74*^4 / T932 * Exp[-13.60 / T9] +
                         exponentielle                  exponentielle
      1.21*^7 / T932 * Exp[-22.51 / T9] + 1.26*^8 / T932 * Exp[-26.00 / T9]},
                       exponentielle                  exponentielle
   (offset + If[T9 < 1,
             si
```

```
      7.906*^15 / T923 * Exp[-40.33 / T913] * (1. - 1.884*^1 * T913 + 2.446*^2 *
                                  exponentielle
          T923 - 7.735*^2 * T9 + 9.485*^2 * T943 - 3.961*^2 * T953), 0])]];
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "Koe91";
reac = " O17 + n > C14 + a ;";
f = 3.;
forward[T9_] := With[{T932 = T9^(3 / 2)}, ( 3.11*^4 +
                  avec
    9.18*^5 / T932 * Exp[-1.961 / T9] + 7.02*^7 / T932 * Exp[-2.759 / T9])];
                    exponentielle                        exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "NACRE";
reac = "F17 + n > N14 + a ;";
f = 1.05;
forward[T9_] := (1.38*^8 * T9^0.053 * Exp[-(55.0 - 54.943) / T9] *
                                        exponentielle
   (1. + .039 * Exp[-.012 / T9 + .217 * T9]) / 1.478 );
              exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88";
reac = "F18 + n > N15 + a ;";
f = 3.;
forward[T9_] :=
 With[{T912 = T9^(1 / 2)}, ( 3.14*^8 * (1. - 0.641 * T912 + 0.108 * T9) * 2.)];
 avec
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "Kaw91";
reac = "C14 + d  > N15 + n ;";
f = 3.;
forward[T9_] :=
 With[{T923 = T9^(2 / 3)}, (4.27*^13 / T923 * Exp[-16.939])];
 avec                                    exponentielle
AddReaction[reac, source, f, forward, True];
                                          vrai



source = "CF88";
```

```
reac = "p + p + n > d + p ;";
f = 3.;
forward[T9_] :=
 With[{T923 = T9^(2 / 3), T913 = T9^(1 / 3)}, (1.35*^7 * Exp[-3.720 / T913] *
 (1. + 0.784 * T913 + 0.346 * T923 + 0.690 * T9) / 2.3590*^9)];
AddReaction[reac, source, f, forward, True];

source = "Kaw91";
reac = "C14 + n > C15 + g ;";
f = 3.;
forward[T9_] := (3240. * T9 );
AddReaction[reac, source, f, forward, True];

source = "CF88";
reac = " O16 + p  > N13 + a ;";
f = 3.;
forward[T9_] := With[{T953 = T9^(5 / 3), T932 = T9^(3 / 2)},
  With[{T9A = T9 /
       (1. + 7.76*^-2 * T9 + 2.64*^-2 * T953 / (1. + 7.76*^-2 * T9)^(2. / 3.))},
    With[{T9A13 = T9A^(1. / 3.), T9A56 = T9A^(5. / 6.)},
     With[
      {SVRev = 1.88*^18 * T9A56 / T932 * Exp[-35.829 / T9A13] * 1.7232*^-1},
      With[{SVDir = SVRev / 0.172255 * Exp[-60.5573 / T9]},
       SVDir]]]];
AddReaction[reac, source, f, forward, True];

(* %TUNL&Cam08  !Camargo et al.Phys.Rev.C 78,034605 (2008) pour DC
 !Tilley (TUNL) Table 9.5 pour la resonance a 87 keV (dominante)  *)
source = "TUNL&Cam08";
reac = "Li8 + p > Be9 + g ;";
f = 3.;
forward[T9_] :=
 With[{T923 = T9^(2 / 3), T913 = T9^(1 / 3), T932 = T9^(3 / 2)},
```

```
      (3.516*^6 / T923 * Exp[-8.5155 / T913] +
                                exponentielle
        2.669*^4 / T932 * Exp[-1.010 / T9] )];
                                exponentielle
    AddReaction[reac, source, f, forward, True];
                                                vrai



    source = "Wan91";
    reac = "B11 + a  > N15 + g ;";
    f = 3.;
    forward[T9_] :=
     With[{T932 = T9^(3 / 2)}, (643. / T932 * Exp[-5.1526 / T9] )];
     avec                                    exponentielle
    AddReaction[reac, source, f, forward, True];
                                                vrai


   ];

  If[reslist[[2]] === {}, {}, reslist[[2, 1]]]
  si

  (* The output is the list of reactions in standard
   format (Name,List initial,List final,f factor) which is
                  liste             liste
   then used by the differential equation constructor *)

 ];
```

Collecting all reaction rates

We collect the description of all rates in a single table (ListReactions). This include the weak rate (1 reaction), all the reactions from the external file (generated by the function LoadRates), and the reactions which were given analytically by the function DefineAnalyticRates.

In[ ]:=
```
ReactionPEN = {"nTOp", {"n"}, {"p"}, 0, "Phys.Rept. 754 (2018) 1-66"};
(* Format is name, List of initial particles,
   format                liste
List of final particles, f factor for uncertainty*)
liste

ChosenReactionFile := If[Not[$Parthenope],
                         si   négation
   If[$ReducedNetwork,
   si
    TabulatedReactionsFileSmallNetwork, TabulatedReactionsFile],
   If[$ReducedNetwork, TabulatedReactionsFileSmallNetworkParthenope,
   si
    TabulatedReactionsFileParthenope]
  ];

TabulatedReactions :=
  (Select[SafeImport["Rates/" <> ChosenReactionFile],
   sélectionne
    (NumericQ[#[[1]]] || "*-" == #[[1]] || StringMatchQ[#[[1]], "\\*%" ~~ __]) &]);
     expression numérique ?                corresponds à chaîne de caractères ?

ReshapedTabulatedReactionsAll := TreatData[TabulatedReactions]

ReshapedTabulatedReactions := If[$ReducedNetwork,
                                 si
   Take[ReshapedTabulatedReactionsAll, SmallNuclearNetworkSize],
   prends
   ReshapedTabulatedReactionsAll];
```

We build tools to restrict the nuclides up to a maximum mass. Useless if MaximumNuclearMass has been set to Infinity.

In[ ]:=
```
SpeciesUpToMaximumMass[A_Integer] :=
                       nombre entier
  SpeciesUpToMaximumMass[A] = Union @@ Table[NamesMassNumberAll[i], {i, A}];
                              union     table
NonBaryonicParticle[key_] := key == "g" || key == "Bm" || key == "Bm";
ReactionUpToMaximumMass[A_Integer][Reaction_List] :=
                        nombre entier   liste
  And @@ ((MemberQ[SpeciesUpToMaximumMass[A], #] || NonBaryonicParticle[#]) & /@
  et          appartient ?
     Flatten@Reaction[[2 ;; 3]]);
     aplatis
ListReactionsUpToMass[A_Integer, ListReactions_List] :=
                      nombre entier   liste
  Select[ListReactions, ReactionUpToMaximumMass[A][#] &];
  sélectionne
ListReactionsUpToMass[Infinity, ListReactions_List] := ListReactions;
                      infini          liste
```

In[ ]:=
```
LoadRates := (
   Off[General::munfl];
   (* The list fo reactions which are tabulated in external .dat file*)
   ListReactionsFile = TreatReactionLine /@ ReshapedTabulatedReactions;

   (* The list of reactions defined analytically *)
   If[Not[$ReducedNetwork],
    ExtraAnalyticReactions = DefineAnalyticRates;,
    ExtraAnalyticReactions = {};
   ];

   (* We concatenate the weak reactions,
   the tabulated rates and the analytic rates*)
   ListReactions =
    Join[{ReactionPEN}, ListReactionsFile, ExtraAnalyticReactions];
   ListReactionsUpToChosenMass =
    ListReactionsUpToMass[MaximumNuclearMass, ListReactions];

   (*DumpSave["lib/ListReactions.mx",ListReactions];*)
   On[General::munfl];
  );
```

LoadRates is the function which does all that. Let us call it. It will stop and quit if one of the reactions is inconsistent (not conservation of N nor Z).

In[ ]:=
```
LoadRates;
```

For information let us print the list of reactions which are taken into account.

In[ ]:=
```
ReactionWithArrow[name_String] := StringReplace[name, "TO" → " -> "]

NiceDisplayReaction[reaction_List] :=
 Join[{ReactionWithArrow[First[reaction]]}, Rest[reaction]]

PadLeftNumberRow[tab_List] :=
 MapThread[Prepend[#2, #1] &, {Range@Length@tab, tab}]

PadLeftNumberRowFromZero[tab_List] :=
 MapThread[Prepend[#2, #1] &, {Range@Length@tab - 1, tab}]
```

```
In[•]:= MyGrid[Join[{{"Reaction Number", "Reaction Name", "Initial species",
          joins                          nombre
    "Final Species", "Reference"}}, PadLeftNumberRowFromZero[
    Drop[#, {4}] & /@ (NiceDisplayReaction /@ ListReactionsUpToChosenMass)]]]
          laisse tomber
```

Out[•]=

Reaction Number	Reaction Name	Initial species	Final Species	Reference	
0	n -> p	{n}	{p}	Phys.Rept. 754 (2018) 1-66	
1	np -> dg	{n, p}	{d, g}	And06	
2	dp -> He3g	{d, p}	{He3, g}	Moscoso2021	
3	dd -> He3n	{d, d}	{He3, n}	Gom17	
4	dd -> tp	{d, d}	{t, p}	Gom17	
5	tp -> ag	{t, p}	{a, g}	Ser04	
6	td -> an	{t, d}	{a, n}	deSouza19a	
7	ta -> Li7g	{t, a}	{Li7, g}	DAACV04	
8	He3n -> tp	{He3, n}	{t, p}	DAACV04	
9	He3d -> ap	{He3, d}	{a, p}	deSouza19b	
10	He3a -> Be7g	{He3, a}	{Be7, g}	Ili16	
11	Be7n -> Li7p	{Be7, n}	{Li7, p}	deSouza2020	
12	Li7p -> aa	{Li7, p}	{a, a}	DAACV04	
13	Li7p -> aag	{Li7, p}	{a, a, g}	NACRE	
14	Be7n -> aa	{Be7, n}	{a, a}	Bar16	
15	Be7d -> aap	{Be7, d}	{a, a, p}	Rij19	
16	da -> Li6g	{d, a}	{Li6, g}	Trezzi2017	
17	Li6p -> Be7g	{Li6, p}	{Be7, g}	NACRE	
18	Li6p -> He3a	{Li6, p}	{He3, a}	NACRE	
19	Be9t -> B11n	{Be9, t}	{B11, n}	TALYS2	
20	O18n -> O19g	{O18, n}	{O19, g}	TALYS2	
21	Li9p -> He6a	{Li9, p}	{He6, a}	=li7pa	
22	Li9d -> Be10n	{Li9, d}	{Be10, n}	TALYS2	
23	Be10a -> C14g	{Be10, a}	{C14, g}	TALYS2	
24	N12n -> C12p	{N12, n}	{C12, p}	TALYS2	
25	Li9p -> Be9n	{Li9, p}	{Be9, n}	TALYS2	
26	Li9a -> B12n	{Li9, a}	{B12, n}	CGXSV12	
27	Li9p -> Be10g	{Li9, p}	{Be10, g}	TALYS2	
28	N13n -> N14g	{N13, n}	{N14, g}	TALYS2	
29	B10a -> N14g	{B10, a}	{N14, g}	TALYS2	
30	B8a -> N12g	{B8, a}	{N12, g}	TALYS2	
31	B12p -> Be9a	{B12, p}	{Be9, a}	TALYS2	
32	Be10p -> B11g	{Be10, p}	{B11, g}	TALYS2	
33	Be10p -> Li7a	{Be10, p}	{Li7, a}	TALYS2	
34	Be11p -> Li8a	{Be11, p}	{Li8, a}	TALYS2	
35	Be11p -> B11n	{Be11, p}	{B11, n}	TALYS2	
36	B8n -> aap	{B8, n}	{a, a, p}	TALYS2	
37	B10n -> B11g	{B10, n}	{B11, g}	TALYS2	
38	B10a -> C13p	{B10, a}	{C13, p}	TALYS2	
39	O17n -> O18g	{O17, n}	{O18, g}	TALYS2	
40	F17n -> O17p	{F17, n}	{O17, p}	TALYS2	
41	F18n -> O18p	{F18, n}	{O18, p}	TALYS2	

42	Be10a -> C13n	{Be10, a}	{C13, n}	TALYS2	
43	Be11a -> C14n	{Be11, a}	{C14, n}	TALYS2	
44	N14a -> F18g	{N14, a}	{F18, g}	ILCCF10	
45	N15a -> F19g	{N15, a}	{F19, g}	ILCCF10	
46	O15a -> Ne19g	{O15, a}	{Ne19, g}	ILCCF10	
47	O16p -> F17g	{O16, p}	{F17, g}	ILCCF10	
48	O16a -> Ne20g	{O16, a}	{Ne20, g}	ILCCF10	
49	O17p -> F18g	{O17, p}	{F18, g}	ILCCF10	
50	O18p -> F19g	{O18, p}	{F19, g}	ILCCF10	
51	O18a -> Ne22g	{O18, a}	{Ne22, g}	ILCCF10	
52	F17p -> Ne18g	{F17, p}	{Ne18, g}	ILCCF10	
53	F18p -> Ne19g	{F18, p}	{Ne19, g}	ILCCF10	
54	Ne19p -> Na20g	{Ne19, p}	{Na20, g}	ILCCF10	
55	O17p -> N14a	{O17, p}	{N14, a}	ILCCF10	
56	O18p -> N15a	{O18, p}	{N15, a}	ILCCF10	
57	F18p -> O15a	{F18, p}	{O15, a}	ILCCF10	
58	C14a -> O18g	{C14, a}	{O18, g}	ILCCF10	
59	C14p -> N15g	{C14, p}	{N15, g}	ILCCF10	
60	Be12p -> Li9a	{Be12, p}	{Li9, a}	TALYS2	
61	Li6He3 -> aap	{Li6, He3}	{a, a, p}	TALYS2	
62	Li6t -> Be9g	{Li6, t}	{Be9, g}	TALYS2	
63	Li6t -> aan	{Li6, t}	{a, a, n}	TALYS2	
64	Li6t -> Li8p	{Li6, t}	{Li8, p}	TALYS2	
65	Li7d -> Be9g	{Li7, d}	{Be9, g}	CGXSV12	
66	Li7He3 -> B10g	{Li7, He3}	{B10, g}	TALYS2	
67	Li7He3 -> Li6a	{Li7, He3}	{Li6, a}	TALYS2	
68	Li7t -> Be10g	{Li7, t}	{Be10, g}	TALYS2	
69	Li8a -> B12g	{Li8, a}	{B12, g}	TALYS2	
70	Li8a -> B11n	{Li8, a}	{B11, n}	CGXSV12	
71	Li8d -> Be10g	{Li8, d}	{Be10, g}	TALYS2	
72	Li8He3 -> B11g	{Li8, He3}	{B11, g}	TALYS2	
73	Li8He3 -> B10n	{Li8, He3}	{B10, n}	TALYS2	
74	Li8He3 -> Be10p	{Li8, He3}	{Be10, p}	TALYS2	
75	Li8He3 -> Li7a	{Li8, He3}	{Li7, a}	TALYS2	
76	Li8t -> Be11g	{Li8, t}	{Be11, g}	TALYS2	
77	Li8t -> Be10n	{Li8, t}	{Be10, n}	TALYS2	
78	Li9a -> B13g	{Li9, a}	{B13, g}	TALYS2	
79	Li9d -> Be11g	{Li9, d}	{Be11, g}	TALYS2	
80	Li9He3 -> B12g	{Li9, He3}	{B12, g}	TALYS2	
81	Li9He3 -> B11n	{Li9, He3}	{B11, n}	TALYS2	
82	Li9He3 -> Be11p	{Li9, He3}	{Be11, p}	TALYS2	
83	Li9He3 -> Li8a	{Li9, He3}	{Li8, a}	TALYS2	
84	Li9t -> Be12g	{Li9, t}	{Be12, g}	TALYS2	
85	Li9t -> Be11n	{Li9, t}	{Be11, n}	TALYS2	
86	Be7He3 -> C10g	{Be7, He3}	{C10, g}	TALYS2	
87	Be7t -> B10g	{Be7, t}	{B10, g}	TALYS2	
88	Be7t -> Be9p	{Be7, t}	{Be9, p}	CGXSV12	
89	Be7t -> Li6a	{Be7, t}	{Li6, a}	TALYS2	
90	Be9a -> C13g	{Be9, a}	{C13, g}	TALYS2	
91	Be9d -> B11g	{Be9, d}	{B11, g}	TALYS2	

92	Be9d -> B10n	{Be9, d}	{B10, n}	TALYS2	
93	Be9d -> Be10p	{Be9, d}	{Be10, p}	TALYS2	
94	Be9d -> Li7a	{Be9, d}	{Li7, a}	TALYS2	
95	Be9He3 -> C12g	{Be9, He3}	{C12, g}	TALYS2	
96	Be9He3 -> C11n	{Be9, He3}	{C11, n}	TALYS2	
97	Be9He3 -> B11p	{Be9, He3}	{B11, p}	TALYS2	
98	Be9He3 -> aaa	{Be9, He3}	{a, a, a}	TALYS2	
99	Be9t -> B12g	{Be9, t}	{B12, g}	TALYS2	
100	Be9t -> Li8a	{Be9, t}	{Li8, a}	TALYS2	
101	Be10d -> B12g	{Be10, d}	{B12, g}	TALYS2	
102	Be10d -> B11n	{Be10, d}	{B11, n}	TALYS2	
103	Be10d -> Li8a	{Be10, d}	{Li8, a}	TALYS2	
104	Be10He3 -> C13g	{Be10, He3}	{C13, g}	TALYS2	
105	Be10He3 -> C12n	{Be10, He3}	{C12, n}	TALYS2	
106	Be10He3 -> B12p	{Be10, He3}	{B12, p}	TALYS2	
107	Be10He3 -> Be9a	{Be10, He3}	{Be9, a}	TALYS2	
108	Be10t -> B13g	{Be10, t}	{B13, g}	TALYS2	
109	Be10t -> B12n	{Be10, t}	{B12, n}	TALYS2	
110	Be10t -> Li9a	{Be10, t}	{Li9, a}	TALYS2	
111	Be11a -> C15g	{Be11, a}	{C15, g}	TALYS2	
112	Be11d -> B13g	{Be11, d}	{B13, g}	TALYS2	
113	Be11d -> B12n	{Be11, d}	{B12, n}	TALYS2	
114	Be11d -> Be12p	{Be11, d}	{Be12, p}	TALYS2	
115	Be11d -> Li9a	{Be11, d}	{Li9, a}	TALYS2	
116	Be11He3 -> C14g	{Be11, He3}	{C14, g}	TALYS2	
117	Be11He3 -> C13n	{Be11, He3}	{C13, n}	TALYS2	
118	Be11He3 -> B13p	{Be11, He3}	{B13, p}	TALYS2	
119	Be11He3 -> Be10a	{Be11, He3}	{Be10, a}	TALYS2	
120	Be11p -> B12g	{Be11, p}	{B12, g}	TALYS2	
121	Be11t -> B14g	{Be11, t}	{B14, g}	TALYS2	
122	Be11t -> B13n	{Be11, t}	{B13, n}	TALYS2	
123	Be12a -> C16g	{Be12, a}	{C16, g}	TALYS2	
124	Be12a -> C15n	{Be12, a}	{C15, n}	TALYS2	
125	Be12d -> B14g	{Be12, d}	{B14, g}	TALYS2	
126	Be12d -> B13n	{Be12, d}	{B13, n}	TALYS2	
127	Be12He3 -> C15g	{Be12, He3}	{C15, g}	TALYS2	
128	Be12He3 -> C14n	{Be12, He3}	{C14, n}	TALYS2	
129	Be12He3 -> B14p	{Be12, He3}	{B14, p}	TALYS2	

130	Be12He3 -> Be11a	{Be12, He3}	{Be11, a}	TALYS2	
131	Be12p -> B13g	{Be12, p}	{B13, g}	TALYS2	
132	Be12p -> B12n	{Be12, p}	{B12, n}	TALYS2	
133	Be12t -> B15g	{Be12, t}	{B15, g}	TALYS2	
134	Be12t -> B14n	{Be12, t}	{B14, n}	TALYS2	
135	B8a -> C11p	{B8, a}	{C11, p}	TALYS2	
136	B8d -> C10g	{B8, d}	{C10, g}	TALYS2	
137	B8He3 -> C10p	{B8, He3}	{C10, p}	TALYS2	
138	B8t -> C11g	{B8, t}	{C11, g}	TALYS2	
139	B8t -> C10n	{B8, t}	{C10, n}	TALYS2	
140	B8t -> B10p	{B8, t}	{B10, p}	TALYS2	
141	B8t -> Be7a	{B8, t}	{Be7, a}	TALYS2	
142	B10d -> C12g	{B10, d}	{C12, g}	TALYS2	
143	B10d -> C11n	{B10, d}	{C11, n}	TALYS2	
144	B10d -> B11p	{B10, d}	{B11, p}	TALYS2	
145	B10d -> aaa	{B10, d}	{a, a, a}	TALYS2	
146	B10He3 -> N13g	{B10, He3}	{N13, g}	TALYS2	
147	B10He3 -> N12n	{B10, He3}	{N12, n}	TALYS2	
148	B10He3 -> C12p	{B10, He3}	{C12, p}	TALYS2	
149	B10n -> Be10p	{B10, n}	{Be10, p}	TALYS2	
150	B10t -> C13g	{B10, t}	{C13, g}	TALYS2	
151	B10t -> C12n	{B10, t}	{C12, n}	TALYS2	
152	B10t -> B12p	{B10, t}	{B12, p}	TALYS2	
153	B10t -> Be9a	{B10, t}	{Be9, a}	TALYS2	
154	B11d -> C13g	{B11, d}	{C13, g}	TALYS2	
155	B11d -> C12n	{B11, d}	{C12, n}	CGXSV12	
156	B11d -> B12p	{B11, d}	{B12, p}	CGXSV12	
157	B11d -> Be9a	{B11, d}	{Be9, a}	TALYS2	
158	B11He3 -> N14g	{B11, He3}	{N14, g}	TALYS2	
159	B11He3 -> N13n	{B11, He3}	{N13, n}	TALYS2	
160	B11He3 -> C13p	{B11, He3}	{C13, p}	TALYS2	
161	B11He3 -> B10a	{B11, He3}	{B10, a}	TALYS2	
162	B11t -> C14g	{B11, t}	{C14, g}	TALYS2	
163	B11t -> C13n	{B11, t}	{C13, n}	TALYS2	
164	B11t -> Be10a	{B11, t}	{Be10, a}	TALYS2	
165	B12a -> N16g	{B12, a}	{N16, g}	TALYS2	
166	B12p -> C12n	{B12, p}	{C12, n}	TALYS2	
167	B12a -> N15n	{B12, a}	{N15, n}	TALYS2	
168	B12d -> C14g	{B12, d}	{C14, g}	TALYS2	
169	B12d -> C13n	{B12, d}	{C13, n}	TALYS2	
170	B12d -> B13p	{B12, d}	{B13, p}	TALYS2	
171	B12d -> Be10a	{B12, d}	{Be10, a}	TALYS2	
172	B12He3 -> N15g	{B12, He3}	{N15, g}	TALYS2	
173	B12He3 -> N14n	{B12, He3}	{N14, n}	TALYS2	
174	B12He3 -> C14p	{B12, He3}	{C14, p}	TALYS2	
175	B12He3 -> B11a	{B12, He3}	{B11, a}	TALYS2	
176	B12n -> B13g	{B12, n}	{B13, g}	TALYS2	
177	B12p -> C13g	{B12, p}	{C13, g}	TALYS2	
178	B12t -> C15g	{B12, t}	{C15, g}	TALYS2	
179	B12t -> C14n	{B12, t}	{C14, n}	TALYS2	
180	B12t -> Be11a	{B12, t}	{Be11, a}	TALYS2	

181	C9a -> O13g	{C9, a}	{O13, g}	TALYS2	
182	C9d -> C10p	{C9, d}	{C10, p}	TALYS2	
183	C9n -> C10g	{C9, n}	{C10, g}	TALYS2	
184	C9t -> N12g	{C9, t}	{N12, g}	TALYS2	
185	C9t -> C11p	{C9, t}	{C11, p}	TALYS2	
186	C9t -> B8a	{C9, t}	{B8, a}	TALYS2	
187	C11d -> N13g	{C11, d}	{N13, g}	TALYS2	
188	C11d -> C12p	{C11, d}	{C12, p}	CGXSV12	
189	C11He3 -> O14g	{C11, He3}	{O14, g}	TALYS2	
190	C11He3 -> N13p	{C11, He3}	{N13, p}	TALYS2	
191	C11He3 -> C10a	{C11, He3}	{C10, a}	TALYS2	
192	C11t -> N14g	{C11, t}	{N14, g}	TALYS2	
193	C11t -> N13n	{C11, t}	{N13, n}	TALYS2	
194	C11t -> C13p	{C11, t}	{C13, p}	TALYS2	
195	C11t -> B10a	{C11, t}	{B10, a}	TALYS2	
196	C12a -> O16g	{C12, a}	{O16, g}	NACRE	
197	C12d -> N14g	{C12, d}	{N14, g}	TALYS2	
198	C12d -> C13p	{C12, d}	{C13, p}	TALYS2	
199	C12He3 -> O15g	{C12, He3}	{O15, g}	TALYS2	
200	C12He3 -> N14p	{C12, He3}	{N14, p}	TALYS2	
201	C11a -> O15g	{C11, a}	{O15, g}	TALYS2	
202	C12He3 -> C11a	{C12, He3}	{C11, a}	TALYS2	
203	C12n -> C13g	{C12, n}	{C13, g}	TALYS2	
204	C12p -> N13g	{C12, p}	{N13, g}	NACRE	
205	C12t -> N15g	{C12, t}	{N15, g}	TALYS2	
206	C12t -> N14n	{C12, t}	{N14, n}	TALYS2	
207	C12t -> C14p	{C12, t}	{C14, p}	TALYS2	
208	C12t -> B11a	{C12, t}	{B11, a}	TALYS2	
209	C13a -> O17g	{C13, a}	{O17, g}	TALYS2	
210	C13d -> N15g	{C13, d}	{N15, g}	TALYS2	
211	C13d -> N14n	{C13, d}	{N14, n}	TALYS2	
212	C13d -> C14p	{C13, d}	{C14, p}	TALYS2	
213	C13d -> B11a	{C13, d}	{B11, a}	TALYS2	
214	C13He3 -> O16g	{C13, He3}	{O16, g}	TALYS2	
215	C13He3 -> O15n	{C13, He3}	{O15, n}	TALYS2	
216	C13He3 -> N15p	{C13, He3}	{N15, p}	TALYS2	
217	C13He3 -> C12a	{C13, He3}	{C12, a}	TALYS2	
218	C13n -> C14g	{C13, n}	{C14, g}	TALYS2	
219	C13p -> N14g	{C13, p}	{N14, g}	NACRE	
220	C13t -> N16g	{C13, t}	{N16, g}	TALYS2	
221	C13t -> N15n	{C13, t}	{N15, n}	TALYS2	
222	C13t -> C15p	{C13, t}	{C15, p}	TALYS2	
223	C13t -> B12a	{C13, t}	{B12, a}	TALYS2	
224	C14d -> N16g	{C14, d}	{N16, g}	TALYS2	
225	C14d -> B12a	{C14, d}	{B12, a}	TALYS2	
226	C14He3 -> O17g	{C14, He3}	{O17, g}	TALYS2	
227	C14He3 -> O16n	{C14, He3}	{O16, n}	TALYS2	
228	C14He3 -> N16p	{C14, He3}	{N16, p}	TALYS2	
229	C14He3 -> C13a	{C14, He3}	{C13, a}	TALYS2	
230	C14t -> N17g	{C14, t}	{N17, g}	TALYS2	
231	C14t -> N16n	{C14, t}	{N16, n}	TALYS2	
232	C15a -> O19g	{C15, a}	{O19, g}	TALYS2	
233	C15a -> O18n	{C15, a}	{O18, n}	TALYS2	

234	C15n -> C16g	{C15, n}	{C16, g}	TALYS2	
235	C15p -> N16g	{C15, p}	{N16, g}	TALYS2	
236	C15p -> N15n	{C15, p}	{N15, n}	TALYS2	
237	C15p -> B12a	{C15, p}	{B12, a}	TALYS2	
238	N12a -> O15p	{N12, a}	{O15, p}	TALYS2	
239	N12n -> N13g	{N12, n}	{N13, g}	TALYS2	
240	N12p -> O13g	{N12, p}	{O13, g}	TALYS2	
241	N13a -> F17g	{N13, a}	{F17, g}	TALYS2	
242	N13n -> C13p	{N13, n}	{C13, p}	TALYS2	
243	N13p -> O14g	{N13, p}	{O14, g}	NACRE	
244	N14n -> N15g	{N14, n}	{N15, g}	TALYS2	
245	N14p -> O15g	{N14, p}	{O15, g}	NACRE	
246	N15n -> N16g	{N15, n}	{N16, g}	TALYS2	
247	N15p -> O16g	{N15, p}	{O16, g}	NACRE	
248	O14n -> O15g	{O14, n}	{O15, g}	TALYS2	
249	O14n -> C11a	{O14, n}	{C11, a}	TALYS2	
250	O15n -> O16g	{O15, n}	{O16, g}	TALYS2	
251	O15n -> N15p	{O15, n}	{N15, p}	TALYS2	
252	O15n -> C12a	{O15, n}	{C12, a}	TALYS2	
253	O17a -> Ne21g	{O17, a}	{Ne21, g}	CF88	
254	O17a -> Ne20n	{O17, a}	{Ne20, n}	NACRE	
255	O19a -> Ne23g	{O19, a}	{Ne23, g}	TALYS2	
256	O19a -> Ne22n	{O19, a}	{Ne22, n}	TALYS2	
257	O19n -> O20g	{O19, n}	{O20, g}	TALYS2	
258	O19p -> F20g	{O19, p}	{F20, g}	TALYS2	
259	O19p -> F19n	{O19, p}	{F19, n}	TALYS2	
260	O19p -> N16a	{O19, p}	{N16, a}	TALYS2	
261	F17a -> Na21g	{F17, a}	{Na21, g}	TALYS2	
262	F17a -> Ne20p	{F17, a}	{Ne20, p}	TALYS2	
263	F17n -> F18g	{F17, n}	{F18, g}	TALYS2	
264	F18a -> Na22g	{F18, a}	{Na22, g}	TALYS2	
265	F18a -> Ne21p	{F18, a}	{Ne21, p}	TALYS2	
266	F18n -> F19g	{F18, n}	{F19, g}	TALYS2	
267	F19a -> Na23g	{F19, a}	{Na23, g}	TALYS2	
268	F19a -> Ne22p	{F19, a}	{Ne22, p}	TALYS2	
269	F19n -> F20g	{F19, n}	{F20, g}	TALYS2	
270	F19p -> Ne20g	{F19, p}	{Ne20, g}	NACRE	
271	F19p -> O16a	{F19, p}	{O16, a}	NACRE	
272	B8n -> Li6He3	{B8, n}	{Li6, He3}	TALYS2	
273	Li9p -> Li7t	{Li9, p}	{Li7, t}	TALYS2	
274	B8n -> Be7d	{B8, n}	{Be7, d}	TALYS2	
275	C9n -> Be7He3	{C9, n}	{Be7, He3}	TALYS2	
276	B10n -> aat	{B10, n}	{a, a, t}	TALYS2	
277	Be10p -> aat	{Be10, p}	{a, a, t}	TALYS2	
278	Be11p -> Be9t	{Be11, p}	{Be9, t}	TALYS2	
279	Be11p -> Be10d	{Be11, p}	{Be10, d}	TALYS2	
280	Be12p -> Be10t	{Be12, p}	{Be10, t}	TALYS2	
281	C9n -> B8d	{C9, n}	{B8, d}	TALYS2	
282	N13n -> C12d	{N13, n}	{C12, d}	TALYS2	
283	B10a -> C12d	{B10, a}	{C12, d}	TALYS2	
284	O14n -> C12He3	{O14, n}	{C12, He3}	TALYS2	
285	C15p -> C14d	{C15, p}	{C14, d}	TALYS2	
286	Ne18n -> O15a	{Ne18, n}	{O15, a}	TALYS2	

287	Ne19n -> O16a	{Ne19, n}	{O16, a}	TALYS2	
288	Na20n -> F17a	{Na20, n}	{F17, a}	TALYS2	
289	Ne18n -> F18p	{Ne18, n}	{F18, p}	TALYS2	
290	Ne19n -> F19p	{Ne19, n}	{F19, p}	TALYS2	
291	Li7He3 -> Be9p	{Li7, He3}	{Be9, p}	TALYS2	
292	Li6t -> Li7d	{Li6, t}	{Li7, d}	TALYS2	
293	Li6He3 -> Be7d	{Li6, He3}	{Be7, d}	TALYS2	
294	Li7He3 -> aad	{Li7, He3}	{a, a, d}	TALYS2	
295	Li8He3 -> Be9d	{Li8, He3}	{Be9, d}	TALYS2	
296	Li8He3 -> aat	{Li8, He3}	{a, a, t}	TALYS2	
297	Li9d -> Li8t	{Li9, d}	{Li8, t}	TALYS2	
298	Li9He3 -> Be10d	{Li9, He3}	{Be10, d}	TALYS2	
299	Li9He3 -> Be9t	{Li9, He3}	{Be9, t}	TALYS2	
300	Be7t -> aad	{Be7, t}	{a, a, d}	TALYS2	
301	Be7t -> Li7He3	{Be7, t}	{Li7, He3}	TALYS2	
302	Be9d -> aat	{Be9, d}	{a, a, t}	TALYS2	
303	Be9t -> Be10d	{Be9, t}	{Be10, d}	TALYS2	
304	Be9He3 -> B10d	{Be9, He3}	{B10, d}	TALYS2	
305	Be10He3 -> B11d	{Be10, He3}	{B11, d}	TALYS2	
306	Be10He3 -> B10t	{Be10, He3}	{B10, t}	TALYS2	
307	B8d -> Be7He3	{B8, d}	{Be7, He3}	TALYS2	
308	B8t -> aaHe3	{B8, t}	{a, a, He3}	TALYS2	
309	B10p -> aaHe3	{B10, p}	{a, a, He3}	TALYS2	
310	B10t -> B11d	{B10, t}	{B11, d}	TALYS2	
311	B10He3 -> C11d	{B10, He3}	{C11, d}	TALYS2	
312	B11t -> B13p	{B11, t}	{B13, p}	TALYS2	
313	B11He3 -> C12d	{B11, He3}	{C12, d}	TALYS2	
314	N12n -> C11d	{N12, n}	{C11, d}	TALYS2	
315	C11t -> C12d	{C11, t}	{C12, d}	TALYS2	
316	C11t -> B11He3	{C11, t}	{B11, He3}	TALYS2	
317	Be7He3 -> ppaa	{Be7, He3}	{p, p, a, a}	TALYS2	
318	dd -> ag	{d, d}	{a, g}	NACRE	
319	He3He3 -> app	{He3, He3}	{a, p, p}	NACRE	
320	Li7a -> B11g	{Li7, a}	{B11, g}	NACRE	
321	Be7p -> B8g	{Be7, p}	{B8, g}	NACRE	
322	Be7a -> C11g	{Be7, a}	{C11, g}	NACRE	
323	Be9p -> B10g	{Be9, p}	{B10, g}	NACRE	
324	Be9p -> aad	{Be9, p}	{a, a, d}	NACRE	
325	Be9p -> Li6a	{Be9, p}	{Li6, a}	NACRE	
326	Be9a -> C12n	{Be9, a}	{C12, n}	NACRE	
327	B10p -> C11g	{B10, p}	{C11, g}	NACRE	
328	B10p -> Be7a	{B10, p}	{Be7, a}	NACRE	
329	B11p -> C12g	{B11, p}	{C12, g}	NACRE	
330	B11p -> aaa	{B11, p}	{a, a, a}	NACRE	
331	B11a -> N14n	{B11, a}	{N14, n}	NACRE	
332	C13a -> O16n	{C13, a}	{O16, n}	NACRE	
333	N15p -> C12a	{N15, p}	{C12, a}	NACRE	
334	Li7t -> Be9n	{Li7, t}	{Be9, n}	CGXSV12	
335	B11n -> B12g	{B11, n}	{B12, g}	CGXSV12	

336	C11n -> aaa	{C11, n}	{a, a, a}	CGXSV12	
337	Li7d -> aan	{Li7, d}	{a, a, n}	CGXSV12	
338	dn -> tg	{d, n}	{t, g}	Nag06	2.
339	tt -> ann	{t, t}	{a, n, n}	Nag06	3.
340	He3n -> ag	{He3, n}	{a, g}	Wag69	3.
341	He3t -> ad	{He3, t}	{a, d}	CF88	10.
342	He3t -> anp	{He3, t}	{a, n, p}	CF88	10.
343	aan -> Be9g	{a, a, n}	{Be9, g}	NACRE	1.25
344	Li7t -> aann	{Li7, t}	{a, a, n, n}	CF88&MF89	3.
345	Li7He3 -> aanp	{Li7, He3}	{a, a, n, p}	CF88&MF89	3.
346	Li8d -> Li9p	{Li8, d}	{Li9, p}	Bal95	3.
347	Li8d -> Li7t	{Li8, d}	{Li7, t}	Has09c	3.
348	Be7t -> aanp	{Be7, t}	{a, a, n, p}	CF88&MF89	3.
349	Be7He3 -> aapp	{Be7, He3}	{a, a, p, p}	CF88&MF89	3.
350	C9a -> N12p	{C9, a}	{N12, p}	Wie89	3.
351	Li6n -> ta	{Li6, n}	{t, a}	CF88	3.
352	He3t -> Li6g	{He3, t}	{Li6, g}	FK90	3.
353	anp -> Li6g	{a, n, p}	{Li6, g}	CF88	3.
354	Li6n -> Li7g	{Li6, n}	{Li7, g}	MF89	3.
355	Li6d -> Li7p	{Li6, d}	{Li7, p}	MF89	3.
356	Li6d -> Be7n	{Li6, d}	{Be7, n}	MF89	3.
357	Li6a -> B10g	{Li6, a}	{B10, g}	CF88	3.
358	Li7a -> B10n	{Li7, a}	{B10, n}	NACRE	1.08
359	Li7n -> Li8g	{Li7, n}	{Li8, g}	MF89&Hei98	3.
360	Li7d -> Li8p	{Li7, d}	{Li8, p}	MF89	3.
361	Li8n -> Li9g	{Li8, n}	{Li9, g}	Rau94	3.
362	Li8p -> aan	{Li8, p}	{a, a, n}	Men12	3.
363	Li8d -> Be9n	{Li8, d}	{Be9, n}	Bal95	3.
364	Be9n -> Be10g	{Be9, n}	{Be10, g}	Rau94	3.
365	Be9p -> aapn	{Be9, p}	{a, a, p, n}	NACRE	1.05
366	B11p -> C11n	{B11, p}	{C11, n}	NACRE	1.1
367	Be10n -> Be11g	{Be10, n}	{Be11, g}	Rau94	3.
368	Be11n -> Be12g	{Be11, n}	{Be12, g}	Rau94	3.
369	B8p -> C9g	{B8, p}	{C9, g}	Des99Bea01	3.
370	aaa -> C12gg	{a, a, a}	{C12, g, g}	NACRE	1.15
371	C11p -> N12g	{C11, p}	{N12, g}	Tang03	3.
372	B10a -> N13n	{B10, a}	{N13, n}	CF88	3.
373	B11a -> C14p	{B11, a}	{C14, p}	Wan91	3.
374	C11n -> C12g	{C11, n}	{C12, g}	Rau94	3.
375	He6 -> Li6Bm	{He6}	{Li6, Bm}	Aud03	1
376	Li8 -> aaBm	{Li8}	{a, a, Bm}	Aud03	1
377	Li9 -> Be9Bm	{Li9}	{Be9, Bm}	Aud03	1
378	Li9 -> aanBm	{Li9}	{a, a, n, Bm}	Aud03	1
379	Be11 -> B11Bm	{Be11}	{B11, Bm}	Aud03	1
380	Be12 -> B12Bm	{Be12}	{B12, Bm}	Aud03	1
381	B8 -> aaBp	{B8}	{a, a, Bp}	Aud03	1
382	B12 -> C12Bm	{B12}	{C12, Bm}	Aud03	1
383	B13 -> C13Bm	{B13}	{C13, Bm}	Aud03	1
384	B14 -> C14Bm	{B14}	{C14, Bm}	Aud03	1
385	B15 -> C15Bm	{B15}	{C15, Bm}	Aud03	1
386	C9 -> aapBp	{C9}	{a, a, p, Bp}	Aud03	1
387	C10 -> B10Bp	{C10}	{B10, Bp}	Aud03	1
388	C11 -> B11Bp	{C11}	{B11, Bp}	Aud03	1

389	C15 -> N15Bm	{C15}	{N15, Bm}	Aud03	1
390	C16 -> N16Bm	{C16}	{N16, Bm}	Aud03	1
391	N12 -> C12Bp	{N12}	{C12, Bp}	Aud03	1
392	N13 -> C13Bp	{N13}	{C13, Bp}	Aud03	1
393	N16 -> O16Bm	{N16}	{O16, Bm}	Aud03	1
394	N17 -> O16nBm	{N17}	{O16, n, Bm}	Aud03	1
395	O13 -> N13Bp	{O13}	{N13, Bp}	Aud03	1
396	O14 -> N14Bp	{O14}	{N14, Bp}	Aud03	1
397	O15 -> N15Bp	{O15}	{N15, Bp}	Aud03	1
398	O19 -> F19Bm	{O19}	{F19, Bm}	Aud03	1
399	O20 -> F20Bm	{O20}	{F20, Bm}	Aud03	1
400	F17 -> O17Bp	{F17}	{O17, Bp}	Aud03	1
401	F18 -> O18Bp	{F18}	{O18, Bp}	Aud03	1
402	F20 -> Ne20Bm	{F20}	{Ne20, Bm}	Aud03	1
403	Ne18 -> F18Bp	{Ne18}	{F18, Bp}	Aud03	1
404	Ne19 -> F19Bp	{Ne19}	{F19, Bp}	Aud03	1
405	Ne23 -> Na23Bm	{Ne23}	{Na23, Bm}	Aud03	1
406	Na20 -> Ne20Bp	{Na20}	{Ne20, Bp}	Aud03	1
407	Na21 -> Ne21Bp	{Na21}	{Ne21, Bp}	Aud03	1
408	ann -> He6g	{a, n, n}	{He6, g}	Efr96	3.
409	O16n -> O17g	{O16, n}	{O17, g}	Iga95	3.
410	N14n -> C14p	{N14, n}	{C14, p}	CF88	3.
411	O14n -> N14p	{O14, n}	{N14, p}	CF88	3.
412	O14a -> Ne18g	{O14, a}	{Ne18, g}	Wie87	3.
413	C11a -> N14p	{C11, a}	{N14, p}	NACRE	2.
414	O14a -> F17p	{O14, a}	{F17, p}	Bar97C	3.
415	O17n -> C14a	{O17, n}	{C14, a}	Koe91	3.
416	F17n -> N14a	{F17, n}	{N14, a}	NACRE	1.05
417	F18n -> N15a	{F18, n}	{N15, a}	CF88	3.
418	C14d -> N15n	{C14, d}	{N15, n}	Kaw91	3.
419	ppn -> dp	{p, p, n}	{d, p}	CF88	3.
420	C14n -> C15g	{C14, n}	{C15, g}	Kaw91	3.
421	O16p -> N13a	{O16, p}	{N13, a}	CF88	3.
422	Li8p -> Be9g	{Li8, p}	{Be9, g}	TUNL&Cam08	3.
423	B11a -> N15g	{B11, a}	{N15, g}	Wan91	3.

Constants for reverse reactions

```
In[•]:= MyGrid[Join[{{"Reaction Number", "Reaction Name",
                      joins                nombre
          "Q (MeV)", "Front Factor", "T9 power", "Q/kB/10^9"}},
                     face a···  factorise
        PadLeftNumberRow[
         Join[{ReactionWithArrow[#[[1]]]}, InfoReaction[#[[2]], #[[3]]] & /@
         joins
          Rest@ListReactionsUpToChosenMass]]]
          reste
```

Out[•]=

Reaction Number	Reaction Name	Q (MeV)	Front Factor	T9 power	Q/kB/10^9
1	np -> dg	2.2245663	4.7161418×10^{9}	1.5	-25.81502
2	dp -> He3g	5.4934751	1.6335108×10^{10}	1.5	-63.749131
3	dd -> He3n	3.2689088	1.7318296	0.	-37.934112

4	dd -> tp	4.0326638	1.7349209	0.	-46.79712
5	tp -> ag	19.813866	2.6105759×10^{10}	1.5	-229.93037
6	td -> an	17.5893	5.5354059	0.	-204.11535
7	ta -> Li7g	2.4676218	1.1132992×10^{10}	1.5	-28.635562
8	He3n -> tp	0.76375502	1.001785	0.	-8.8630089
9	He3d -> ap	18.353055	5.5452865	0.	-212.97836
10	He3a -> Be7g	1.5871347	1.1128947×10^{10}	1.5	-18.417934
11	Be7n -> Li7p	1.644242	1.0021491	0.	-19.080637
12	Li7p -> aa	17.346244	4.6898011	0.	-201.29481
13	Li7p -> aag	17.346244	4.6898011	0.	-201.29481
14	Be7n -> aa	18.990486	4.6998798	0.	-220.37544
15	Be7d -> aap	16.76592	9.9655186×10^{-10}	-1.5	-194.56042
16	da -> Li6g	1.4737584	1.5305262×10^{10}	1.5	-17.102256
17	Li6p -> Be7g	5.6068515	1.1877781×10^{10}	1.5	-65.064809
18	Li6p -> He3a	4.0197167	1.067287	0.	-46.646875
19	Be9t -> B11n	9.5592348	3.8284908	0.	-110.93031
20	O18n -> O19g	3.9556018	3.0716051×10^{9}	1.5	-45.902853
21	Li9p -> He6a	12.226865	1.8556823	0.	-141.88688
22	Li9d -> Be10n	17.411825	14.484998	0.	-202.05584
23	Be10a -> C14g	12.012513	4.7767575×10^{10}	1.5	-139.39942
24	N12n -> C12p	18.120447	3.0129978	0.	-210.27906
25	Li9p -> Be9n	12.824113	1.0004549	0.	-148.81765
26	Li9a -> B12n	5.9391078	3.4308601	0.	-68.920484
27	Li9p -> Be10g	19.636391	6.8313304×10^{10}	1.5	-227.87086
28	N13n -> N14g	10.553381	1.1930599×10^{10}	1.5	-122.4669
29	B10a -> N14g	11.61211	1.1144709×10^{11}	1.5	-134.75294
30	B8a -> N12g	8.0084159	7.1824559×10^{10}	1.5	-92.933807
31	B12p -> Be9a	6.8850052	0.29160469	0.	-79.897168
32	Be10p -> B11g	11.228753	4.3271625×10^{9}	1.5	-130.30427
33	Be10p -> Li7a	2.5644402	0.10767485	0.	-29.759093
34	Be11p -> Li8a	4.0954252	0.16270048	0.	-47.525436
35	Be11p -> B11n	10.727115	0.49984572	0.	-124.483
36	B8n -> aap	18.854115	3.6007295×10^{-10}	-1.5	-218.79292
37	B10n -> B11g	11.454221	3.0347575×10^{10}	1.5	-132.92072
38	B10a -> C13p	4.0615465	9.3625853	0.	-47.13229
39	O17n -> O18g	8.0453702	1.1010408×10^{11}	1.5	-93.362644
40	F17n -> O17p	3.5428112	1.002282	0.	-41.112617
41	F18n -> O18p	2.4382633	3.0065131	0.	-28.294871
42	Be10a -> C13n	3.8360785	1.3349807	0.	-44.515842
43	Be11a -> C14n	11.510875	5.5178002	0.	-133.57815
44	N14a -> F18g	4.4152327	5.4197223×10^{10}	1.5	-51.236648
45	N15a -> F19g	4.0137991	5.5418287×10^{10}	1.5	-46.578204
46	O15a -> Ne19g	3.5284659	5.5418816×10^{10}	1.5	-40.946146
47	O16p -> F17g	0.60026896	3.0344557×10^{9}	1.5	-6.9658321
48	O16a -> Ne20g	4.729846	5.6527372×10^{10}	1.5	-54.887583
49	O17p -> F18g	5.6071069	3.6621854×10^{10}	1.5	-65.067773
50	O18p -> F19g	7.9935999	9.1999519×10^{9}	1.5	-92.761874
51	O18a -> Ne22g	9.6668156	5.8490824×10^{10}	1.5	-112.17874

52	F17p -> Ne18g	3.9230711	1.0985031×10^{11}	1.5	-45.525349
53	F18p -> Ne19g	6.4100211	2.7596299×10^{10}	1.5	-74.385206
54	Ne19p -> Na20g	2.1905211	7.3872228×10^{9}	1.5	-25.419941
55	O17p -> N14a	1.1918742	0.67571458	0.	-13.831126
56	O18p -> N15a	3.9798008	0.16600931	0.	-46.18367
57	F18p -> O15a	2.8815552	0.49795902	0.	-33.439059
58	C14a -> O18g	6.2276252	5.4206937×10^{10}	1.5	-72.268589
59	C14p -> N15g	10.207426	8.9988565×10^{9}	1.5	-118.45226
60	Be12p -> Li9a	4.9869452	0.097115699	0.	-57.871096
61	Li6He3 -> aap	16.879296	7.2462483×10^{-10}	-1.5	-195.8761
62	Li6t -> Be9g	17.688241	4.2265102×10^{10}	1.5	-205.26352
63	Li6t -> aan	16.115541	7.2333369×10^{-10}	-1.5	-187.01309
64	Li6t -> Li8p	0.80192024	2.0175581	0.	-9.3058979
65	Li7d -> Be9g	16.694378	5.8104637×10^{10}	1.5	-193.73021
66	Li7He3 -> B10g	17.787713	3.4631856×10^{10}	1.5	-206.41784
67	Li7He3 -> Li6a	13.326528	2.1970664	0.	-154.64793
68	Li7t -> Be10g	17.249426	2.4244992×10^{11}	1.5	-200.17128
69	Li8a -> B12g	10.001316	7.1839178×10^{10}	1.5	-116.06045
70	Li8a -> B11n	6.6316898	3.0721835	0.	-76.957564
71	Li8d -> Be10g	21.474033	3.0330305×10^{11}	1.5	-249.1958
72	Li8He3 -> B11g	27.209311	8.0344839×10^{10}	1.5	-315.75094
73	Li8He3 -> B10n	15.75509	2.6474879	0.	-182.83022
74	Li8He3 -> Be10p	15.980558	18.567558	0.	-185.44667
75	Li8He3 -> Li7a	18.544998	1.999259	0.	-215.20577
76	Li8t -> Be11g	15.718441	1.6045287×10^{11}	1.5	-182.40493
77	Li8t -> Be10n	15.216803	18.534474	0.	-176.58366
78	Li9a -> B13g	10.817926	4.5613956×10^{10}	1.5	-125.53682
79	Li9d -> Be11g	17.913463	1.2539657×10^{11}	1.5	-207.8771
80	Li9He3 -> B12g	26.516729	8.972508×10^{10}	1.5	-307.71386
81	Li9He3 -> B11n	23.147103	3.8370693	0.	-268.61097
82	Li9He3 -> Be11p	12.419988	7.6765073	0.	-144.12797
83	Li9He3 -> Li8a	16.515413	1.2489714	0.	-191.65341
84	Li9t -> Be12g	14.826921	2.688109×10^{11}	1.5	-172.05927
85	Li9t -> Be11n	11.656233	7.6628292	0.	-135.26496
86	Be7He3 -> C10g	15.001549	2.4232023×10^{11}	1.5	-174.08575
87	Be7t -> B10g	18.6682	3.4644443×10^{10}	1.5	-216.63546
88	Be7t -> Be9p	12.08139	3.5583331	0.	-140.19871
89	Be7t -> Li6a	14.207015	2.197865	0.	-164.86556
90	Be9a -> C13g	10.648357	9.1155478×10^{10}	1.5	-123.56905
91	Be9d -> B11g	15.816465	6.2650441×10^{10}	1.5	-183.54245
92	Be9d -> B10n	4.3622438	2.0644299	0.	-50.621737
93	Be9d -> Be10p	4.5877118	14.478412	0.	-53.238185
94	Be9d -> Li7a	7.152152	1.5589608	0.	-82.997278
95	Be9He3 -> C12g	26.279669	2.6899791×10^{11}	1.5	-304.96289
96	Be9He3 -> C11n	7.5589508	3.8265847	0.	-87.717981

97	Be9He3 -> B11p	10.32299	3.8353246	0.	-119.79332
98	Be9He3 -> aaa	19.004921	1.3427306×10^{-9}	-1.5	-220.54295
99	Be9t -> B12g	12.928861	8.9524482×10^{10}	1.5	-150.0332
100	Be9t -> Li8a	2.927545	1.2461791	0.	-33.972749
101	Be10d -> B12g	12.373813	2.1455101×10^{10}	1.5	-143.59214
102	Be10d -> B11n	9.0041868	0.91752172	0.	-104.48925
103	Be10d -> Li8a	2.372497	0.2986546	0.	-27.531685
104	Be10He3 -> C13g	24.4137	3.4912893×10^{10}	1.5	-283.30922
105	Be10He3 -> C12n	19.467391	3.9395002	0.	-225.90969
106	Be10He3 -> B12p	6.8803378	1.3134349	0.	-79.843005
107	Be10He3 -> Be9a	13.765343	0.38300378	0.	-159.74017
108	Be10t -> B13g	10.995401	1.7431259×10^{10}	1.5	-127.59633
109	Be10t -> B12n	6.1165828	1.3110946	0.	-70.979996
110	Be10t -> Li9a	0.17747503	0.38214751	0.	-2.0595122
111	Be11a -> C15g	12.728986	4.9701706×10^{10}	1.5	-147.71375
112	Be11d -> B13g	16.750993	3.29502×10^{10}	1.5	-194.3872
113	Be11d -> B12n	11.872175	2.478354	0.	-137.77087
114	Be11d -> Be12p	0.94612183	7.4382519	0.	-10.979288
115	Be11d -> Li9a	5.933067	0.72237103	0.	-68.850384
116	Be11He3 -> C14g	32.088496	1.4430348×10^{11}	1.5	-372.37153
117	Be11He3 -> C13n	23.912061	4.0329107	0.	-277.48795
118	Be11He3 -> B13p	11.257518	2.0171401	0.	-130.63807
119	Be11He3 -> Be10a	20.075983	3.0209505	0.	-232.97211
120	Be11p -> B12g	14.096741	1.1688269×10^{10}	1.5	-163.58589
121	Be11t -> B14g	11.462981	2.8798755×10^{10}	1.5	-133.02237
122	Be11t -> B13n	10.493763	2.0135459	0.	-121.77506
123	Be12a -> C16g	13.808716	5.1410655×10^{10}	1.5	-160.24349
124	Be12a -> C15n	9.5582978	1.4168163	0.	-110.91944
125	Be12d -> B14g	14.549523	1.3434221×10^{10}	1.5	-168.8402
126	Be12d -> B13n	13.580305	0.93929136	0.	-157.59289
127	Be12He3 -> C15g	30.135919	3.7053087×10^{10}	1.5	-349.71282
128	Be12He3 -> C14n	28.917808	4.1135717	0.	-335.57722
129	Be12He3 -> B14p	9.0560478	0.82241401	0.	-105.09107
130	Be12He3 -> Be11a	17.406933	0.74550937	0.	-201.99907
131	Be12p -> B13g	15.804871	4.4298312×10^{9}	1.5	-183.40791

132	Be12p -> B12n	10.926053	0.33319038	0.	-126.79158
133	Be12t -> B15g	11.070611	1.8492888×10^{10}	1.5	-128.4691
134	Be12t -> B14n	8.2922928	0.82094863	0.	-96.228062
135	B8a -> C11p	7.4081448	3.078465	0.	-85.967951
136	B8d -> C10g	20.358653	3.032602×10^{11}	1.5	-236.25236
137	B8He3 -> C10p	14.865178	18.564934	0.	-172.50323
138	B8t -> C11g	27.222011	8.0365664×10^{10}	1.5	-315.89832
139	B8t -> C10n	14.101423	18.531855	0.	-163.64022
140	B8t -> B10p	18.531829	2.6542225	0.	-215.05294
141	B8t -> Be7a	19.677495	2.0000464	0.	-228.34785
142	B10d -> C12g	25.186334	4.5131931×10^{11}	1.5	-292.27527
143	B10d -> C11n	6.4656158	6.4201674	0.	-75.030356
144	B10d -> B11p	9.2296548	6.434831	0.	-107.1057
145	B10d -> aaa	17.911586	2.2528065×10^{-9}	-1.5	-207.85533
146	B10He3 -> N13g	21.63635	2.4429654×10^{11}	1.5	-251.07941
147	B10He3 -> N12n	1.5724118	9.1698684	0.	-18.247081
148	B10He3 -> C12p	19.692859	27.628793	0.	-228.52614
149	B10n -> Be10p	0.22546804	7.0132737	0.	-2.6164479
150	B10t -> C13g	23.875413	2.4441739×10^{11}	1.5	-277.06266
151	B10t -> C12n	18.929104	27.579564	0.	-219.66313
152	B10t -> B12p	6.3420508	9.1950656	0.	-73.596444
153	B10t -> Be9a	13.227056	2.6813243	0.	-153.49361
154	B11d -> C13g	18.678422	1.3179673×10^{11}	1.5	-216.75408
155	B11d -> C12n	13.732113	14.871676	0.	-159.35455
156	B11d -> B12p	1.1450598	4.9582379	0.	-13.287868
157	B11d -> Be9a	8.030065	1.4458454	0.	-93.185035
158	B11He3 -> N14g	20.73551	9.6040755×10^{10}	1.5	-240.6256
159	B11He3 -> N13n	10.182129	8.0499526	0.	-118.1587
160	B11He3 -> C13p	13.184947	8.068311	0.	-153.00495
161	B11He3 -> B10a	9.1234	0.86176102	0.	-105.87266
162	B11t -> C14g	20.597626	2.8818164×10^{11}	1.5	-239.02552
163	B11t -> C13n	12.421192	8.0539349	0.	-144.14194
164	B11t -> Be10a	8.585113	6.0329971	0.	-99.6261
165	B12a -> N16g	10.110416	3.0822341×10^{10}	1.5	-117.3265
166	B12p -> C12n	12.587053	2.9993874	0.	-146.06668
167	B12a -> N15n	7.6215597	4.2481819	0.	-88.444527
168	B12d -> C14g	23.48523	2.016734×10^{11}	1.5	-272.53478
169	B12d -> C13n	15.308795	5.6362523	0.	-177.65119
170	B12d -> B13p	2.6542518	2.8190831	0.	-30.801313
171	B12d -> Be10a	11.472717	4.2219728	0.	-133.13535
172	B12He3 -> N15g	28.199181	1.1109997×10^{11}	1.5	-327.2379
173	B12He3 -> N14n	17.365884	4.1071575	0.	-201.52272
174	B12He3 -> C14p	17.991755	12.34601	0.	-208.78564
175	B12He3 -> B11a	17.207995	1.1183986	0.	-199.69049
176	B12n -> B13g	4.8788181	1.3295195×10^{10}	1.5	-56.616333
177	B12p -> C13g	17.533362	2.6581365×10^{10}	1.5	-203.46621
178	B12t -> C15g	18.446111	1.110088×10^{11}	1.5	-214.05823
179	B12t -> C14n	17.228	12.324012	0.	-199.92264
180	B12t -> Be11a	5.717125	2.2335009	0.	-66.344481
181	C9a -> O13g	8.2209159	4.5605348×10^{10}	1.5	-95.399767
182	C9d -> C10p	19.059082	14.516402	0.	-221.17146

183	C9n -> C10g	21.283648	6.8461412×10^{10}	1.5	-246.98648
184	C9t -> N12g	26.522711	8.9753808×10^{10}	1.5	-307.78328
185	C9t -> C11p	25.92244	3.8469286	0.	-300.81742
186	C9t -> B8a	18.514295	1.2496256	0.	-214.84947
187	C11d -> N13g	18.439643	1.3179718×10^{11}	1.5	-213.98317
188	C11d -> C12p	16.496152	14.905643	0.	-191.42989
189	C11He3 -> O14g	17.572838	2.8803074×10^{11}	1.5	-203.92432
190	C11He3 -> N13p	12.946168	8.0683386	0.	-150.23404
191	C11He3 -> C10a	7.457033	6.0305815	0.	-86.535275
192	C11t -> N14g	22.735794	9.6088594×10^{10}	1.5	-263.83793
193	C11t -> N13n	12.182413	8.0539624	0.	-141.37103
194	C11t -> C13p	15.185231	8.0723299	0.	-176.21728
195	C11t -> B10a	11.123684	0.86219027	0.	-129.08499
196	C12a -> O16g	7.161918	5.1331463×10^{10}	1.5	-83.110607
197	C12d -> N14g	10.272306	2.2368188×10^{10}	1.5	-119.20516
198	C12d -> C13p	2.7217425	1.8791345	0.	-31.584511
199	C12He3 -> O15g	12.075619	3.695553×10^{10}	1.5	-140.13174
200	C12He3 -> N14p	4.778831	1.3693321	0.	-55.456031
201	C11a -> O15g	10.218716	9.9335883×10^{10}	1.5	-118.58327
202	C12He3 -> C11a	1.856903	0.37202599	0.	-21.548465
203	C12n -> C13g	4.9463088	8.8622646×10^{9}	1.5	-57.39953
204	C12p -> N13g	1.9434911	8.8420996×10^{9}	1.5	-22.553277
205	C12t -> N15g	14.848373	3.6974888×10^{10}	1.5	-172.30821
206	C12t -> N14n	4.015076	1.3668922	0.	-46.593022
207	C12t -> C14p	4.6409468	4.1088429	0.	-53.855952
208	C12t -> B11a	3.857187	0.3722113	0.	-44.760797
209	C13a -> O17g	6.3586894	1.7616095×10^{10}	1.5	-73.789526
210	C13d -> N15g	16.159294	6.8274523×10^{10}	1.5	-187.52082
211	C13d -> N14n	5.3259973	2.523981	0.	-61.805632
212	C13d -> C14p	5.9518681	7.5870221	0.	-69.068562
213	C13d -> B11a	5.1681083	0.68729211	0.	-59.973407
214	C13He3 -> O16g	22.79323	1.5147807×10^{11}	1.5	-264.50445
215	C13He3 -> O15n	7.1293101	4.1699872	0.	-82.732208
216	C13He3 -> N15p	10.665819	4.1796188	0.	-123.77169
217	C13He3 -> C12a	15.631312	2.9509791	0.	-181.39385
218	C13n -> C14g	8.1764344	3.5781472×10^{10}	1.5	-94.883581
219	C13p -> N14g	7.5505636	1.1903452×10^{10}	1.5	-87.620652
220	C13t -> N16g	12.39092	3.0270853×10^{10}	1.5	-143.79066
221	C13t -> N15n	9.902064	4.1721715	0.	-114.90868
222	C13t -> C15p	0.91274914	4.176189	0.	-10.592014
223	C13t -> B12a	2.2805043	0.98210754	0.	-26.464154
224	C14d -> N16g	10.471716	1.3844044×10^{10}	1.5	-121.51922
225	C14d -> B12a	0.36130003	0.44915617	0.	-4.1927127
226	C14He3 -> O17g	18.759876	1.2875446×10^{10}	1.5	-217.69932
227	C14He3 -> O16n	14.616796	4.2334221	0.	-169.62087
228	C14He3 -> N16p	4.9782408	0.84750249	0.	-57.770086
229	C14He3 -> C13a	12.401187	0.73089104	0.	-143.9098
230	C14t -> N17g	10.099704	3.8603645×10^{10}	1.5	-117.2022
231	C14t -> N16n	4.2144858	0.84599241	0.	-48.907077
232	C15a -> O19g	8.9651159	1.8484807×10^{10}	1.5	-104.03585

233	C15a -> O18n	5.0095141	6.0179635	0.	-58.132997
234	C15n -> C16g	4.2504181	3.628604×10^{10}	1.5	-49.324054
235	C15p -> N16g	11.478171	7.2484394×10^{9}	1.5	-133.19864
236	C15p -> N15n	8.9893149	0.99903799	0.	-104.31667
237	C15p -> B12a	1.3677552	0.23516836	0.	-15.87214
238	N12a -> O15p	9.6184448	4.2576249	0.	-111.61742
239	N12n -> N13g	20.063938	2.6641226×10^{10}	1.5	-232.83233
240	N12p -> O13g	1.5120711	1.3264755×10^{10}	1.5	-17.546856
241	N13a -> F17g	5.8186959	1.7616071×10^{10}	1.5	-67.523162
242	N13n -> C13p	3.0028177	1.0022806	0.	-34.846253
243	N13p -> O14g	4.6266701	3.5698891×10^{10}	1.5	-53.690277
244	N14n -> N15g	10.833297	2.7050331×10^{10}	1.5	-125.71519
245	N14p -> O15g	7.2967879	2.6987996×10^{10}	1.5	-84.675707
246	N15n -> N16g	2.4888562	7.2554192×10^{9}	1.5	-28.881977
247	N15p -> O16g	12.127411	3.6242079×10^{10}	1.5	-140.73276
248	O14n -> O15g	13.223499	9.0194106×10^{9}	1.5	-153.45233
249	O14n -> C11a	3.0047832	0.090797105	0.	-34.869061
250	O15n -> O16g	15.66392	3.6325788×10^{10}	1.5	-181.77225
251	O15n -> N15p	3.5365089	1.0023097	0.	-41.039482
252	O15n -> C12a	8.5020022	0.70767101	0.	-98.661639
253	O17a -> Ne21g	7.3479317	8.6333627×10^{10}	1.5	-85.269206
254	O17a -> Ne20n	0.58676577	18.586092	0.	-6.809134
255	O19a -> Ne23g	10.911866	5.9348211×10^{10}	1.5	-126.62695
256	O19a -> Ne22n	5.7112138	19.04243	0.	-66.275884
257	O19n -> O20g	7.6080181	1.1107564×10^{11}	1.5	-88.287384
258	O19p -> F20g	10.639334	2.2176995×10^{10}	1.5	-123.46434
259	O19p -> F19n	4.0379981	2.995161	0.	-46.859022
260	O19p -> N16a	2.5130552	0.39212957	0.	-29.162795
261	F17a -> Na21g	6.5614759	8.6331919×10^{10}	1.5	-76.142766
262	F17a -> Ne20p	4.129577	18.628505	0.	-47.921751
263	F17n -> F18g	9.1499181	3.6705423×10^{10}	1.5	-106.18039
264	F18a -> Na22g	8.4794059	2.5065782×10^{10}	1.5	-98.399419
265	F18a -> Ne21p	1.7408248	2.3574347	0.	-20.201433
266	F18n -> F19g	10.431863	2.7659776×10^{10}	1.5	-121.05675
267	F19a -> Na23g	10.467324	2.9670853×10^{10}	1.5	-121.46825
268	F19a -> Ne22p	1.6732157	6.3577315	0.	-19.416862
269	F19n -> F20g	6.601336	7.4042748×10^{9}	1.5	-76.605323
270	F19p -> Ne20g	12.843458	3.6967391×10^{10}	1.5	-149.04214
271	F19p -> O16a	8.1136122	0.65397327	0.	-94.15456
272	B8n -> Li6He3	1.9748188	0.49690949	0.	-22.916821
273	Li9p -> Li7t	2.3869652	0.28176254	0.	-27.699581
274	B8n -> Be7d	2.0881952	0.36131883	0.	-24.232499
275	C9n -> Be7He3	6.2820992	0.28252455	0.	-72.900734
276	B10n -> aat	0.32228646	1.3566042×10^{-10}	-1.5	-3.7399791
277	Be10p -> aat	0.096818424	1.9343381×10^{-11}	-1.5	-1.1235312
278	Be11p -> Be9t	1.1678802	0.13055947	0.	-13.552687
279	Be11p -> Be10d	1.7229282	0.54477808	0.	-19.993751
280	Be12p -> Be10t	4.8094702	0.25413145	0.	-55.811584

281	C9n -> B8d	0.92499521	0.22575139	0.	-10.734124
282	N13n -> C12d	0.28107521	0.53337351	0.	-3.2617423
283	B10a -> C12d	1.339804	4.9823924	0.	-15.54778
284	O14n -> C12He3	1.1478802	0.24406119	0.	-13.320597
285	C15p -> C14d	1.0064552	0.52357817	0.	-11.679427
286	Ne18n -> O15a	8.1084022	0.16638821	0.	-94.094101
287	Ne19n -> O16a	12.135454	0.65547753	0.	-140.8261
288	Na20n -> F17a	10.545202	0.26925106	0.	-122.37199
289	Ne18n -> F18p	5.226847	0.33414036	0.	-60.655041
290	Ne19n -> F19p	4.0218421	1.0023002	0.	-46.67154
291	Li7He3 -> Be9p	11.200903	3.5570403	0.	-129.98108
292	Li6t -> Li7d	0.9938634	0.72739637	0.	-11.533306
293	Li6He3 -> Be7d	0.11337639	0.72713209	0.	-1.3156783
294	Li7He3 -> aad	11.852769	$2.8709949 \times 10^{-10}$	-1.5	-137.54568
295	Li8He3 -> Be9d	11.392846	1.2824306	0.	-132.20849
296	Li8He3 -> aat	16.077376	$3.5915933 \times 10^{-10}$	-1.5	-186.5702
297	Li9d -> Li8t	2.195022	0.78151655	0.	-25.472173
298	Li9He3 -> Be10d	14.142916	4.1819929	0.	-164.12172
299	Li9He3 -> Be9t	13.587868	1.0022407	0.	-157.68066
300	Be7t -> aad	12.733256	$2.8720383 \times 10^{-10}$	-1.5	-147.7633
301	Be7t -> Li7He3	0.88048702	1.0003635	0.	-10.217628
302	Be9d -> aat	4.6845303	$2.8006143 \times 10^{-10}$	-1.5	-54.361716
303	Be9t -> Be10d	0.555048	4.1726432	0.	-6.4410646
304	Be9He3 -> B10d	1.093335	0.59602569	0.	-12.687626
305	Be10He3 -> B11d	5.735278	0.26489954	0.	-66.555137
306	Be10He3 -> B10t	0.53828698	0.14284128	0.	-6.246561
307	B8d -> Be7He3	5.357104	1.2514853	0.	-62.166611
308	B8t -> aaHe3	18.09036	$3.5943137 \times 10^{-10}$	-1.5	-209.92991
309	B10p -> aaHe3	-0.44146856	1.354187×10^{-10}	-1.5	5.1230299
310	B10t -> B11d	5.196991	1.8545027	0.	-60.308576
311	B10He3 -> C11d	3.196707	1.8535794	0.	-37.096244
312	B11t -> B13p	-0.23335216	4.0283348	0.	2.7079394
313	B11He3 -> C12d	10.463204	4.2936315	0.	-121.42044
314	N12n -> C11d	1.6242952	0.20213806	0.	-18.849163
315	C11t -> C12d	12.463488	4.2957702	0.	-144.63277
316	C11t -> B11He3	2.000284	1.0004981	0.	-23.212332
317	Be7He3 -> ppaa	11.272445	1.220135×10^{-19}	-3.	-130.81129
318	dd -> ag	23.84653	4.5291426×10^{10}	1.5	-276.72749
319	He3He3 -> app	12.85958	$3.3947046 \times 10^{-10}$	-1.5	-149.22923
320	Li7a -> B11g	8.6643129	4.0187309×10^{10}	1.5	-100.54518

321	Be7p -> B8g	0.13637106	1.3052577×10^{10}	1.5	-1.5825205
322	Be7a -> C11g	7.5445159	4.01819×10^{10}	1.5	-87.550471
323	Be9p -> B10g	6.5868101	9.7361439×10^{9}	1.5	-76.436757
324	Be9p -> aad	0.65186643	8.0713026×10^{-11}	-1.5	-7.5645958
325	Be9p -> Li6a	2.1256248	0.61766701	0.	-24.666851
326	Be9a -> C12n	5.7020478	10.2858	0.	-66.169517
327	B10p -> C11g	8.6901821	3.027842×10^{10}	1.5	-100.84538
328	B10p -> Be7a	1.1456662	0.75353379	0.	-13.294904
329	B11p -> C12g	15.956679	7.0136933×10^{10}	1.5	-185.16957
330	B11p -> aaa	8.6819315	3.5009568×10^{-10}	-1.5	-100.74963
331	B11a -> N14n	0.15788897	3.6723556	0.	-1.8322254
332	C13a -> O16n	2.2156092	5.7921384	0.	-25.711077
333	N15p -> C12a	4.9654933	0.70604024	0.	-57.622157
334	Li7t -> Be9n	10.437148	3.5507024	0.	-121.11807
335	B11n -> B12g	3.3696261	2.3383753×10^{10}	1.5	-39.102887
336	C11n -> aaa	11.44597	3.5089529×10^{-10}	-1.5	-132.82497
337	Li7d -> aan	15.121678	9.9441478×10^{-10}	-1.5	-175.47979
338	dn -> tg	6.2572301	1.6364266×10^{10}	1.5	-72.61214
339	tt -> ann	11.33207	3.382618×10^{-10}	-1.5	-131.50321
340	He3n -> ag	20.577621	2.6152357×10^{10}	1.5	-238.79338
341	He3t -> ad	14.320391	1.5981381	0.	-166.18124
342	He3t -> anp	12.095825	3.3886559×10^{-10}	-1.5	-140.36622
343	aan -> Be9g	1.5726998	5.8430987×10^{19}	3.	-18.250424
344	Li7t -> aann	8.864448	1.2153491×10^{-19}	-3.	-102.86765
345	Li7He3 -> aanp	9.628203	6.0875924×10^{-20}	-3.	-111.73066
346	Li8d -> Li9p	1.8376418	4.4398826	0.	-21.324948
347	Li8d -> Li7t	4.224607	1.2509926	0.	-49.024528
348	Be7t -> aanp	10.50869	6.089805×10^{-20}	-3.	-121.94828
349	Be7He3 -> aapp	11.272445	1.220135×10^{-19}	-3.	-130.81129
350	C9a -> N12p	6.7088448	3.4380846	0.	-77.852911
351	Li6n -> ta	4.7834717	1.0691921	0.	-55.509884
352	He3t -> Li6g	15.794149	2.4459923×10^{10}	1.5	-183.28349
353	anp -> Li6g	3.6983246	7.2181786×10^{19}	3.	-42.917275
354	Li6n -> Li7g	7.2510935	1.1903307×10^{10}	1.5	-84.145446
355	Li6d -> Li7p	5.0265272	2.5239503	0.	-58.330426
356	Li6d -> Be7n	3.3822852	2.5185377	0.	-39.24979
357	Li6a -> B10g	4.4611853	1.5762771×10^{10}	1.5	-51.769905
358	Li7a -> B10n	-2.7899082	1.3242346	0.	32.375541
359	Li7n -> Li8g	2.0326231	1.3081025×10^{10}	1.5	-23.587612
360	Li7d -> Li8p	-0.19194317	2.7736709	0.	2.227408
361	Li8n -> Li9g	4.0622081	2.0939116×10^{10}	1.5	-47.139968

362	Li8p -> aan	15.313621	$3.5851938 \times 10^{-10}$	-1.5	-177.7072
363	Li8d -> Be9n	14.661755	4.4419024	0.	-170.1426
364	Be9n -> Be10g	6.8122781	6.8282242×10^{10}	1.5	-79.053205
365	Be9p -> aapn	-1.5726998	$1.7114207 \times 10^{-20}$	-3.	18.250424
366	B11p -> C11n	-2.764039	0.99772122	0.	32.075341
367	Be10n -> Be11g	0.5016381	8.6569962×10^{9}	1.5	-5.8212684
368	Be11n -> Be12g	3.1706881	3.507985×10^{10}	1.5	-36.794308
369	B8p -> C9g	1.2995711	2.0890864×10^{10}	1.5	-15.080896
370	aaa -> C12gg	7.2747476	2.0033647×10^{20}	3.	-84.41994
371	C11p -> N12g	0.60027106	2.333129×10^{10}	1.5	-6.9658564
372	B10a -> N13n	1.0587288	9.3412819	0.	-12.286037
373	B11a -> C14p	0.78375981	11.039007	0.	-9.0951549
374	C11n -> C12g	18.720718	7.0297125×10^{10}	1.5	-217.24491
375	He6 -> Li6Bm	2.7228726	0.33369189	0.	-31.597624
376	Li8 -> aaBm	15.313621	$3.5926093 \times 10^{-10}$	-1.5	-177.7072
377	Li9 -> Be9Bm	12.824113	1.0025242	0.	-148.81765
378	Li9 -> aanBm	11.251413	$1.7157407 \times 10^{-20}$	-3.	-130.56723
379	Be11 -> B11Bm	10.727115	0.50087959	0.	-124.483
380	Be12 -> B12Bm	10.926053	0.33387954	0.	-126.79158
381	B8 -> aaBp	18.854115	$3.5932973 \times 10^{-10}$	-1.5	-218.79292
382	B12 -> C12Bm	12.587053	3.0055912	0.	-146.06668
383	B13 -> C13Bm	12.654544	2.0034564	0.	-146.84988
384	B14 -> C14Bm	19.86176	5.0121716	0.	-230.48615
385	B15 -> C15Bm	18.301553	2.0042066	0.	-212.3807
386	C9 -> aapBp	17.554544	$1.7200328 \times 10^{-20}$	-3.	-203.71203
387	C10 -> B10Bp	4.430406	0.14292924	0.	-51.412727
388	C11 -> B11Bp	2.764039	1.0002152	0.	-32.075341
389	C15 -> N15Bm	8.9893149	1.0011044	0.	-104.31667
390	C16 -> N16Bm	7.227753	0.20017152	0.	-83.87459
391	N12 -> C12Bp	18.120447	3.0067786	0.	-210.27906
392	N13 -> C13Bp	3.0028177	1.0002118	0.	-34.846253
393	N16 -> O16Bm	9.6385551	5.0055055	0.	-111.85079
394	N17 -> O16nBm	3.753337	$1.0969481 \times 10^{-10}$	-1.5	-43.555667
395	O13 -> N13Bp	18.551867	2.0042765	0.	-215.28548
396	O14 -> N14Bp	5.9267112	0.33351101	0.	-68.776628
397	O15 -> N15Bp	3.5365089	1.0002409	0.	-41.039482
398	O19 -> F19Bm	4.0379981	3.0013561	0.	-46.859022
399	O20 -> F20Bm	3.031316	0.20006966	0.	-35.176961
400	F17 -> O17Bp	3.5428112	1.0002132	0.	-41.112617
401	F18 -> O18Bp	2.4382633	3.0003074	0.	-28.294871
402	F20 -> Ne20Bm	6.2421222	5.0030359	0.	-72.43682
403	Ne18 -> F18Bp	5.226847	0.33345067	0.	-60.655041
404	Ne19 -> F19Bp	4.0218421	1.0002313	0.	-46.67154

405	Ne23 -> Na23Bm	3.5934565	1.5005136	0.	-41.700331
406	Na20 -> Ne20Bp	14.674779	5.0053916	0.	-170.29374
407	Na21 -> Ne21Bp	4.329267	1.0002329	0.	-50.239058
408	ann -> He6g	0.97545207	1.0838004×10^{20}	3.	-11.319651
409	O16n -> O17g	4.1430802	3.0413802×10^{9}	1.5	-48.078449
410	N14n -> C14p	0.62587084	3.0059743	0.	-7.2629295
411	O14n -> N14p	5.9267112	0.33420083	0.	-68.776628
412	O14a -> Ne18g	5.1150969	5.420703×10^{10}	1.5	-59.358234
413	C11a -> N14p	2.921928	3.6807432	0.	-33.907566
414	O14a -> F17p	1.1920258	0.49346269	0.	-13.832885
415	O17n -> C14a	1.817745	2.0311807	0.	-21.094055
416	F17n -> N14a	4.7346854	0.67725653	0.	-54.943743
417	F18n -> N15a	6.4180641	0.49910918	0.	-74.478542
418	C14d -> N15n	7.9828597	1.9080971	0.	-92.63724
419	ppn -> dp	2.2245663	2.3580709×10^{9}	1.5	-25.81502
420	C14n -> C15g	1.2181111	9.0075218×10^{9}	1.5	-14.135592
421	O16p -> N13a	-5.2184269	0.17225497	0.	60.55733
422	Li8p -> Be9g	16.886321	2.0948641×10^{10}	1.5	-195.95762
423	B11a -> N15g	10.991186	9.9338436×10^{10}	1.5	-127.54741

We build a function which selects all species having the same mass number A or Z number

In[•]:=
```
NamesMassNumber[A_] :=
 Select[NamesWithWeights, ((Plus @@ (#[[2]])) == A) &][[All, 1]]
 |sélectionne                |plus                    |tout
NamesAtomicNumber[Z_] := Select[NamesWithWeights, (#[[2, 2]] == Z) &][[All, 1]]
                          |sélectionne                                |tout
```

We use two ways of noting the abundance of a species. One notation is Ynipj where i is the number of neutrons and j the number of protons. For instance Yn2p2 is He4.

But we also want to use short names. For instance He4 for Helium4. So we want to relate automatically YHe4 to Yn2p2.

In[•]:=
```
StackY[name_] := "Y" <> ToString[name];
                        |en chaîne de caractères
YName[PostString_][n_, p_] :=
  ToExpression@StackY["n" <> ToString[n] <> "p" <> ToString[p] <> PostString];
  |en expression          |en chaîne de caractères    |en chaîne de caractères
ShortString[nameshort_, np_List] :=
                        |liste
  (Evaluate@ToExpression["Y" <> nameshort] := YName[""] @@ np;);
   |évalue      |en expression
If[Yn =!= Yn1p0, ShortString @@@ NamesWithWeightsAll;];
|si
(*The condition is to avoid infinite
 recursion when loading PRIMAT several times.*)
```

When the function ShortString is evaluated, it defines the relation between the Yshortname and the Ynipj notation

In[ ]:=
```
Yn === Yn1p0
```

Out[ ]=
```
True
```

We define this by default so that the user can uses the names of nuclei. This is overwritten whenever BuildVariables is called.

In[ ]:=
```
ShortNames = NamesWithWeightsAll[[All, 1]];
                                 |tout
KeyVal = Association[# → Position[ShortNames, #][[1, 1]] & /@ ShortNames];
         |association            |position
```

In[ ]:=
```
BuildVariables := (
  ListReactionsNames = ListReactionsUpToChosenMass[[All, 1]];
                                                   |tout
ListNuclearReactionsNames = Select[ListReactionsNames, # =!= "nTOp" &];
                                   |sélectionne
(*We can define an association
   (a dictionary) between reaction names and number*)
KeyNuclearReaction =
   Association[# → Position[ListNuclearReactionsNames, #][[1, 1]] & /@
   |association            |position
     ListNuclearReactionsNames];
  KeyReaction = Association[
                |association
    # → Position[ListReactionsNames, #][[1, 1]] & /@ ListReactionsNames];
        |position
(*Let us collect all the species involved in the reactions. These are
   the species whose abundance we are going to solve numerically.*)
VariablesInEquations =
   Union@Flatten[Join[RemoveNonNuclear[#[[2]]], RemoveNonNuclear[#[[3]]]] & /@
   |union |aplatis |joins
      ListReactionsUpToChosenMass];
(*We can check the number of species
   (59 for the full network of 423 equations).*)
NumberVariable = Length@VariablesInEquations;
                 |longueur
(*Let us collect all the species used together
   with their weights in terms of neutrons and protons*)
NamesWithWeights =
   Select[NamesWithWeightsAll, MemberQ[VariablesInEquations, #[[1]]] &];
   |sélectionne                |appartient ?
(*WeightsNuclear is the list of nuclear weights for
   the variables (their A in nuclear physics notation). It
   is obtained by summing the (n,p) numbers*)
WeightsNuclear = (Plus @@ (#[[2]])) & /@ NamesWithWeights;
                 |plus
  (*The list of all shortnames available is ShortNames,
  and we associate a number through a dictionary KeyVal*)
```

```
  ShortNames = NamesWithWeights[[All, 1]];
                                  tout
  KeyVal = Association[# → Position[ShortNames, #][[1, 1]] & /@ ShortNames];
           association             position

  (*We now build list of variable in standard Ynipj forms are*)
  VarList = ToExpression /@ (StackY /@ ShortNames);
            en expression

  (* Finally for the abstract reactions, we also define rules to replace
   the abstract reaction L[i] or its reverse Lbar[i] by the actual rate.*)
  Li = L[KeyReaction[[#]]] & /@ ListReactionsNames;
  Lbari = Lbar[KeyReaction[[#]]] & /@ ListReactionsNames;

  (*FunList[tv_]:=SetTimeDependence[VarList,tv];
  FunPrimeList[tv_]:=FunList'[tv];*)
 )
```

In[ ]:=
```
BuildVariables
```

Some information on the species and their notation

In[ ]:=
```
(*KeyNuclearReaction*)(*KeyReaction*)
(*VariablesInEquations*)
Grid[NamesWithWeights]
grille
```

Out[ ]=

n	{1, 0}
p	{0, 1}
d	{1, 1}
t	{2, 1}
He3	{1, 2}
a	{2, 2}
Li7	{4, 3}
Be7	{3, 4}
He6	{4, 2}
Li6	{3, 3}
Li8	{5, 3}
Li9	{6, 3}
Be9	{5, 4}
Be10	{6, 4}
Be11	{7, 4}
Be12	{8, 4}
B8	{3, 5}
B10	{5, 5}
B11	{6, 5}
B12	{7, 5}
B13	{8, 5}
B14	{9, 5}
B15	{10, 5}
C9	{3, 6}
C10	{4, 6}
C11	{5, 6}
C12	{6, 6}

```
C13  {7, 6}
C14  {8, 6}
C15  {9, 6}
C16 {10, 6}
N12  {5, 7}
N13  {6, 7}
N14  {7, 7}
N15  {8, 7}
N16  {9, 7}
N17 {10, 7}
O13  {5, 8}
O14  {6, 8}
O15  {7, 8}
O16  {8, 8}
O17  {9, 8}
O18 {10, 8}
O19 {11, 8}
O20 {12, 8}
F17  {8, 9}
F18  {9, 9}
F19 {10, 9}
F20 {11, 9}
Ne18 {8, 10}
Ne19 {9, 10}
Ne20 {10, 10}
Ne21 {11, 10}
Ne22 {12, 10}
Ne23 {13, 10}
Na20 {9, 11}
Na21 {10, 11}
Na22 {11, 11}
Na23 {12, 11}
```

In[•]:= **KeyVal**

Out[•]=

```
⟨| n → 1, p → 2, d → 3, t → 4, He3 → 5, a → 6, Li7 → 7, Be7 → 8, He6 → 9,
 Li6 → 10, Li8 → 11, Li9 → 12, Be9 → 13, Be10 → 14, Be11 → 15, Be12 → 16,
 B8 → 17, B10 → 18, B11 → 19, B12 → 20, B13 → 21, B14 → 22, B15 → 23, C9 → 24,
 C10 → 25, C11 → 26, C12 → 27, C13 → 28, C14 → 29, C15 → 30, C16 → 31,
 N12 → 32, N13 → 33, N14 → 34, N15 → 35, N16 → 36, N17 → 37, O13 → 38,
 O14 → 39, O15 → 40, O16 → 41, O17 → 42, O18 → 43, O19 → 44, O20 → 45,
 F17 → 46, F18 → 47, F19 → 48, F20 → 49, Ne18 → 50, Ne19 → 51, Ne20 → 52,
 Ne21 → 53, Ne22 → 54, Ne23 → 55, Na20 → 56, Na21 → 57, Na22 → 58, Na23 → 59 |⟩
```

Some examples to see how these short names have been related ot the correct names

In[ ]:=
```
Ya
Yp
Yt
YLi7
```

Out[ ]=
```
Yn2p2
```

Out[ ]=
```
Yn0p1
```

Out[ ]=
```
Yn2p1
```

Out[ ]=
```
Yn4p3
```

In[ ]:=
```
VarList
```

Out[ ]=
```
{Yn1p0, Yn0p1, Yn1p1, Yn2p1, Yn1p2, Yn2p2, Yn4p3, Yn3p4, Yn4p2, Yn3p3, Yn5p3,
 Yn6p3, Yn5p4, Yn6p4, Yn7p4, Yn8p4, Yn3p5, Yn5p5, Yn6p5, Yn7p5, Yn8p5, Yn9p5,
 Yn10p5, Yn3p6, Yn4p6, Yn5p6, Yn6p6, Yn7p6, Yn8p6, Yn9p6, Yn10p6, Yn5p7,
 Yn6p7, Yn7p7, Yn8p7, Yn9p7, Yn10p7, Yn5p8, Yn6p8, Yn7p8, Yn8p8, Yn9p8,
 Yn10p8, Yn11p8, Yn12p8, Yn8p9, Yn9p9, Yn10p9, Yn11p9, Yn8p10, Yn9p10,
 Yn10p10, Yn11p10, Yn12p10, Yn13p10, Yn9p11, Yn10p11, Yn11p11, Yn12p11}
```

Numerical abundance functions

We also need the list of functions to store the results of the numerical integrations.
We have divided our BBN period in 3 eras.

High temperature (HT),
Middle temperature (MT)
Low temperature (LT).

So for instance the abundance of a species, say Lithium7 whose shortname is 'Li7', during the HT period at a given cosmological time t, is

YHT["Li7"][t].

Eventually we are interested in the values YLT["species"][tend] where tend is the final time of our numerical integration.

In[ ]:=
```
KeyQ[key_] := MemberQ[NamesWithWeightsAll[[All, 1]], key];
              (appartient ?                    (tout

YHT[key_?KeyQ][t_] := Y["HT"][key][t];
YMT[key_?KeyQ][t_] := Y["MT"][key][t];
YLT[key_?KeyQ][t_] := Y["LT"][key][t];
```

We make a list of the Y_i for a given period at a given

time. This is typically used for initial conditions in Differential Solver.

```
In[•]:= YPeriodTime[period_String][tv_] := Y[period][#][tv] & /@ ShortNames;
```

We also define a function which can choose between previsouly numerically solved function in a given era, or equilibrium values for a list of species

```
In[•]:= NumericalValueOrThermalEquilibriumValue[
          period_, ListThermal_, name_, Tv_, tv_] :=
         Module[
          {Yv = Y[period][name][tv], Yn = Y[period]["n"][tv], Yp = Y[period]["p"][tv]},
          If[ MemberQ[ListThermal, name], YNSE[name, Yn, Yp, Tv], Yv]];
```

We make a list of the Y_i for a given period at a given time. If there is no known numerical value it takes the thermostatistical equilibrium. This is only used for the initial conditions in the middle era, where all species considered start at thermodynamical equilibrium or not very far from it except for neutrons and protons which are computed numerically from the high temperature era.

```
In[•]:= YPeriodTimeOrStateEquilibrium[period_String, ListThermal_List][Tv_, tv_] :=
          NumericalValueOrThermalEquilibriumValue[period, ListThermal, #, Tv, tv] & /@
           ShortNames;
```

CNO

List of CNO nuclei. This includes all N and O, but only C with A>=12. C11 decays into B11 and C10 into B10 and C9 into a+2p.

```
In[•]:= CNONuclei = KeyNucleons /@ Join[Table[{i, 6}, {i, 6, 10}],
                                    Table[{i, 7}, {i, 5, 10}], Table[{i, 8}, {i, 5, 12}]]
```

```
Out[•]= {C12, C13, C14, C15, C16, N12, N13, N14,
         N15, N16, N17, O13, O14, O15, O16, O17, O18, O19, O20}
```

```
In[•]:= YLT["CNO"][t_] := Plus @@ ((YLT[#][t] &) /@ CNONuclei)
```

Coupled systems of differential reactions

Formal construction of r.h.s

This is a function which takes the list of reactions, with the specification of initial and final particles and their multiplicity, and builds the rhs of the differential system.
It builds the rhs of Eq 138 in companion paper.

Abstract abundance functions

We build list of abundance function and abundance function derivatives which are used later to build the system of equations. These stay at an abstract level and they are used only to build the differential system of equations which is later solved by NDSolve.

In[•]:=
```
SetTimeDependence[list_List, tv_] := #[tv] & /@ list;
```
liste

In[•]:=
```
FunList[tv_] := SetTimeDependence[VarList, tv]
FunPrimeList[tv_] := FunList'[tv]
```

In[ ]:=
```
FillReactionMatrix[listreac_List] :=
                   liste
 Module[{Tab, i, j, nvar, TreatReaction, FactorInitialElements},
 module
  nvar = Length@VarList;
         longueur
  Tab = Table[0, {ii, 1, nvar}];
        table
  FactorInitialElements[el_List] :=
                        liste
   Times @@ ((AρB / DensityUnit Y[KeyVal[#[[1]]]]) ^#[[2]] / (#[[2]]!) & /@ el);
   multiplication
  TreatReaction[reaction_List] := Module[{
                  liste                 module
     InitialParticles = Tally[RemoveNonNuclear@reaction[[2]],
                        recense
     FinalParticles = Tally[RemoveNonNuclear@reaction[[3]],
                      recense
     ReactionForward, ReactionBackward,
     FactorInitialForward, FactorInitialBackward},

    ReactionForward = L[KeyReaction[reaction[[1]]]];
    ReactionBackward = Lbar[KeyReaction[reaction[[1]]]];
    FactorInitialForward = FactorInitialElements@InitialParticles;
    (* This computes the product Yi^ni/ni! for initial particles*)
    FactorInitialBackward = FactorInitialElements@FinalParticles;

    (Tab[[KeyVal[#[[1]]]]] = Tab[[KeyVal[#[[1]]]]] - ReactionForward
          FactorInitialForward #[[2]] / AρB * DensityUnit) & /@ InitialParticles;
    (Tab[[KeyVal[#[[1]]]]] = Tab[[KeyVal[#[[1]]]]] + ReactionForward
          FactorInitialForward #[[2]] / AρB * DensityUnit) & /@ FinalParticles;
    (Tab[[KeyVal[#[[1]]]]] = Tab[[KeyVal[#[[1]]]]] - ReactionBackward
          FactorInitialBackward #[[2]] / AρB * DensityUnit) & /@ FinalParticles;
    (Tab[[KeyVal[#[[1]]]]] =
        Tab[[KeyVal[#[[1]]]]] + ReactionBackward FactorInitialBackward
          #[[2]] / AρB * DensityUnit) & /@ InitialParticles;
   ];
  TreatReaction /@ listreac;
  Tab
 ]
```

To check the list of reactions used at this stage, just evaluate the next cell after uncommenting it.

In[ ]:=
```
(*Print[ListReactionsUpToChosenMass];*)
 imprime
```

FormalReactions is the list of rhs of differential equation in an abstract level.
-The species are given by Y[1], Y[2],Y[3], and we need the dictionary KeyVal to know to which species it corresponds.

-The reactions are L[1], L[2],L[3] and we also need a dictionary (KeyReaction) to know to which reaction it corresponds.

We might define compiled version which are used if the $CompileNDSolve option is set to True. This is much faster.
Otherwise if $CompileNDSolve=False, the DY, DY18 and DYPEN are called in DefineEquations further below when it is associated with the l.h.s to form the differential system.

In[ ]:=
```
CompileFromFormal[FormalReactions_List] := ReleaseHold[
                                  |liste                                |maintiens déverrouillage
  Hold[Compile[{{AρB, _Real}, {L, _Real, 1}, {Lbar, _Real, 1}, {Y, _Real, 1}},
  |main··  |compile              |nombre réel  |nombre réel     |nombre réel     |nombre réel
     inside, CompilationTarget → "C", "RuntimeOptions" → "Speed",
             |cible de compilation     |con···  |options de durée d'exécution
     CompilationOptions → {"InlineExternalDefinitions" → True},
     |options de compilation                                  |vrai
     Parallelization → False]] //.
     |parallélisation  |faux
   {inside → FormalReactions, Y[m_] :↦ Y⟦m⟧, L[m_] :↦ L⟦m⟧, Lbar[m_] :↦ Lbar⟦m⟧}]
```

In[ ]:=
```
BuildFormalReactions := (
  FormalReactions = FillReactionMatrix@ListReactionsUpToChosenMass;
  NReactions = Length[ListReactionsUpToChosenMass];
               |longueur
  (*We also build formally a small network of
   reactions which is used in the middle temperature era.It is
   the same thing but with a reduced number of equations.*)
  NReactionsSmallNetwork =
   If[$ReducedNetwork, SmallNuclearNetworkSize + 1, 18];
   |si
  NReactions18 =
   Min[NReactionsSmallNetwork, Length@ListReactionsUpToChosenMass];
   |minimum                    |longueur
  FormalReactions18 =
   FillReactionMatrix@Take[ListReactionsUpToChosenMass, NReactions18];
                          |prends
  (*We also build a differential system with just neutrons and
   protons and the weak interactions for the high temperature era*)
  FormalReactionsOnlyPEN = FillReactionMatrix[{ReactionPEN}];

  (*We also compile these formal reactions if asked by options.*)
  If[$CompileNDSolve,
  |si
   DYCDef := CompileFromFormal[FormalReactions];
   DY18CDef := CompileFromFormal[FormalReactions18];

   NameFileDYC =
    "DYC_" <> ToString[FileBaseName@ChosenReactionFile] <> "_Nreac_" <>
              |en chaîne ···  |nom de base de fichier
```

```
    ToString[NReactions] <> "_Nnuclei_" <> ToString[Length@VarList];
    NameFileDYC18 =
     "DY18C_" <> ToString[FileBaseName@ChosenReactionFile] <> "_Nreac_" <>
      ToString[NReactions18] <> "_Nnuclei_" <> ToString[Length@VarList];
    DYC := (LoadCompiledFunction[Hold[DYC], Hold[DYCDef], NameFileDYC];
      DYC);
    DY18C :=
     (LoadCompiledFunction[Hold[DY18C], Hold[DY18CDef], NameFileDYC18];
      DY18C);

    DYCN[AρB_?NumericQ, L_, Lbar_, Y_] := DYC[AρB, L, Lbar, Y];
    DY18CN[AρB_?NumericQ, L_, Lbar_, Y_] := DY18C[AρB, L, Lbar, Y];
   ];
  )
```

In[ ]:=
```
BuildFormalReactions
```

In[ ]:=
```
LoadRatesFull := (
  LoadRates;
  BuildVariables;
  BuildFormalReactions;
 )
```

To see how this works we look at a few elements of FormalReactions. For instance the source of Li7 due to nuclear reactions is one element of FormalReactions. It is

In[•]:=
```
FormalReactions[[KeyVal["Li7"]]]
```

Out[•]=
```
AρB L[8] × Y[4] × Y[6] + 1/2 AρB Lbar[13] Y[6]^2 + 1/2 AρB Lbar[14] Y[6]^2 +
 1/2 AρB^2 Lbar[338] × Y[1] Y[6]^2 + 1/4 AρB^3 Lbar[345] Y[1]^2 Y[6]^2 +
 1/2 AρB^3 Lbar[346] × Y[1] × Y[2] Y[6]^2 + 1/2 AρB^2 Lbar[295] × Y[3] Y[6]^2 -
 Lbar[8] × Y[7] - Lbar[355] × Y[7] - AρB L[360] × Y[1] × Y[7] -
 AρB L[13] × Y[2] × Y[7] - AρB L[14] × Y[2] × Y[7] - AρB Lbar[12] × Y[2] × Y[7] -
 AρB Lbar[356] × Y[2] × Y[7] - AρB L[66] × Y[3] × Y[7] - AρB L[338] × Y[3] × Y[7] -
 AρB L[361] × Y[3] × Y[7] - AρB Lbar[293] × Y[3] × Y[7] - AρB L[69] × Y[4] × Y[7] -
 AρB L[335] × Y[4] × Y[7] - AρB L[345] × Y[4] × Y[7] - AρB Lbar[274] × Y[4] × Y[7] -
 AρB Lbar[348] × Y[4] × Y[7] - AρB L[67] × Y[5] × Y[7] - AρB L[68] × Y[5] × Y[7] -
 AρB L[292] × Y[5] × Y[7] - AρB L[295] × Y[5] × Y[7] - AρB L[346] × Y[5] × Y[7] -
 AρB Lbar[302] × Y[5] × Y[7] - AρB L[321] × Y[6] × Y[7] - AρB L[359] × Y[6] × Y[7] -
 AρB Lbar[34] × Y[6] × Y[7] - AρB Lbar[76] × Y[6] × Y[7] - AρB Lbar[95] × Y[6] × Y[7] +
 AρB L[12] × Y[1] × Y[8] + AρB L[302] × Y[4] × Y[8] + AρB L[355] × Y[1] × Y[10] +
 AρB L[356] × Y[3] × Y[10] + AρB L[293] × Y[4] × Y[10] + AρB Lbar[68] × Y[6] × Y[10] +
 Lbar[360] × Y[11] + AρB Lbar[361] × Y[2] × Y[11] + AρB L[348] × Y[3] × Y[11] +
 AρB L[76] × Y[5] × Y[11] + AρB L[274] × Y[2] × Y[12] + Lbar[66] × Y[13] +
 AρB Lbar[335] × Y[1] × Y[13] + AρB Lbar[292] × Y[2] × Y[13] +
 AρB L[95] × Y[3] × Y[13] + Lbar[69] × Y[14] + AρB L[34] × Y[2] × Y[14] +
 Lbar[67] × Y[18] + AρB Lbar[359] × Y[1] × Y[18] + Lbar[321] × Y[19]
```

In[•]:=
```
FormalReactionsOnlyPEN
```

Out[•]=
```
{-L[1] × Y[1] + Lbar[1] × Y[2], L[1] × Y[1] - Lbar[1] × Y[2], 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

* AρB is an abstract variable which needs to be replaced with the appropriate ρB (taking into account that this is not exactly ρ_B but $n_B\, m_a$, see appendix of companion paper).

* The reactions L[i] and Lbar[i] are also the reactions, with i being the i – th line of ListReactions. For instance reaction L[8] is

In[•]:=
```
NiceDisplayReaction[ListReactions[[8]]]
```

Out[•]=
```
{ta -> Li7g, {t, a}, {Li7, g}, 0, DAACV04}
```

The Y[i] are the abstract abundances, corresponding to the i-th elements in VariableList. For instance 9 corresponds to Li7 and so Y[9] is the abstract abundance of Li7.

In[•]:=
```
KeyVal["Li7"]
ShortNames[[9]]
```

Out[•]=
```
7
```

Out[•]=
```
He6
```

We check that formally nucleons are conserved. So we compute formally the $\Sigma_i A_i \, dY_i/dt$ and check that it is 0.

```
In[•]:= WeightsNuclear.FormalReactions // Simplify
                                        simplifie
Out[•]=
0
```

Actual r.h.s of differential system (when compilation is not used)

Now we can build the rhs of the differential equation. This is obtained from its formal expression "FormalReactions" in which time dependence is added thanks to some replacement rules.

First replacement rules for the abstract abundances.

```
In[•]:= Yi := Y[KeyVal[#]] & /@ ShortNames;

RulesY[tv_] := Thread[Rule[Yi, FunList[tv]]];
                       enfile    règle
```

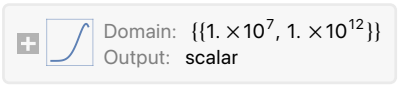
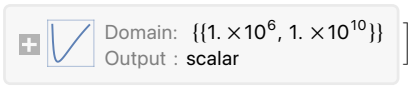
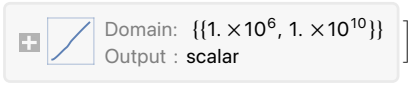
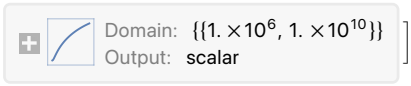
Let us visualize few of these rules to understand

```
In[•]:= Take[RulesY[tv], 8]
         prends
Out[•]=
{Y[1] → Yn1p0[tv], Y[2] → Yn0p1[tv], Y[3] → Yn1p1[tv], Y[4] → Yn2p1[tv],
 Y[5] → Yn1p2[tv], Y[6] → Yn2p2[tv], Y[7] → Yn4p3[tv], Y[8] → Yn3p4[tv]}
```

```
In[•]:= RulesλRHS[Tv_] := Symbol["L" <> #][Tv] & /@ ListReactionsNames;
                          symbole
RulesλRHS[n_, Tv_] := Symbol["L" <> #][Tv] & /@ Take[ListReactionsNames, n];
                      symbole                    prends

RulesλbarRHS[Tv_] := Symbol["Lbar" <> #][Tv] & /@ ListReactionsNames;
                     symbole
RulesλbarRHS[n_, Tv_] :=
  Symbol["Lbar" <> #][Tv] & /@ Take[ListReactionsNames, n];
  symbole                    prends

Rulesλ[Tv_] := Thread[Rule[Li, RulesλRHS[Tv]]];
               enfile    règle
Rulesλbar[Tv_] := Thread[Rule[Lbari, RulesλbarRHS[Tv]]];
                  enfile    règle
```

Let us visualize a few of these rules.

```
In[•]:= Take[Rulesλ[Tv], 4]
        prends
```

```
Out[•]=
{L[1] → 0.0011384335 InterpolatingFunction[  Domain: {{1. ×10^7, 1. ×10^12}} Output: scalar ][Tv],
 L[2] → InterpolatingFunction[  Domain: {{1. ×10^6, 1. ×10^10}} Output : scalar ][Tv],
 L[3] → InterpolatingFunction[  Domain: {{1. ×10^6, 1. ×10^10}} Output : scalar ][Tv],
 L[4] → InterpolatingFunction[  Domain: {{1. ×10^6, 1. ×10^10}} Output: scalar ][Tv]}
```

Let us apply these rules to form the r.h.s

Here is the r.h.s of the differential system for nuclear reactions. We distinguish between high, middle and low temperature system of equations.

```
In[•]:= DYOnlyPEN[Temp_, ρB_, time_] :=
          (FormalReactionsOnlyPEN) /. Dispatch@Rulesλ[Temp] /.
                                      affecte
            Dispatch@Rulesλbar[Temp] /. Dispatch@RulesY[time] /. AρB → ρB;
            affecte                     affecte

        DY18[Temp_, ρB_, time_] :=
          (FormalReactions18) /. Dispatch@Rulesλ[Temp] /. Dispatch@Rulesλbar[Temp] /.
                                 affecte                  affecte
            Dispatch@RulesY[time] /. AρB → ρB;
            affecte

        DY[Temp_, ρB_, time_] :=
          (FormalReactions) /. Dispatch@Rulesλ[Temp] /. Dispatch@Rulesλbar[Temp] /.
                               affecte                  affecte
            Dispatch@RulesY[time] /. AρB → ρB;
            affecte
```

```
In[•]:= (*DY[Tv,rv,tv][[5]];//Timing*)
                             chronométra
```

```
In[•]:= NReactions
```

```
Out[•]=
424
```

Initial nuclear conditions

We use equilibrium solution for the initial condition. This equation A15 of companion paper.

```
In[ ]:= Yni[Tv_] := 1 / (1 + (1 - 3/2 Q/mn) Exp[Q/(kB Tv)]);

        Ypi[Tv_] := 1 - Yni[Tv];
```

Other initial conditions based on the equation and the rates. It is better when including corrections because these are the correct initial conditions whatever the corrections.

```
In[ ]:= Yn2i[Tv_] := LpTOn[Tv] / (LpTOn[Tv] + LnTOp[Tv]);
        Yp2i[Tv_] := 1 - Yn2i[Tv];
```

We see that the difference is very small

```
In[ ]:= Yni[10^11.5]
        Yn2i[10^11.5]
```

```
Out[ ]= 0.48865342

Out[ ]= 0.48898469
```

Actual list of initial conditions. Vanishing for everything except protons and neutrons

```
In[ ]:= CIList[Tv_] :=
         Table[Which[i == 1, Yni[Tv], i == 2, Ypi[Tv], i ≥ 3, 0], {i, 1, NumberVariable}]
```

Construction of differential equations

The function DefineEquations wraps the definition of differential equations. This must be called any time we regenerate the probabilities on the reaction rates when including uncertainties on reaction rates in a Monte-Carlo analysis.

```
In[ ]:= DefineEquations := (

          (*We build the differential system for the High temperatures *)
          (* So we associate the r.h.s which is constructed thanks to DYOnlyPEN,
          with the l.h.s made of abundances derivatives *)
          SystemEquationsHT[tv_] = Thread[Equal[FunPrimeList[tv],
              (DYOnlyPEN[Tv, ρBv, tv](*/.Dispatch@ReactionProbabilities*))]] /.
            {Tv → Toft@tv, ρBv → ρBForBBN@a@Toft@tv};

          (* For middle and low temperature we distinguish
           between the compiled and the uncompiled method *)
```

```
If[$CompileNDSolve,
 Off[General::munfl];
 (* If $CompileNDSolve=True,
    we reinterpolate the rates for the middle and the low temperatures. *)
 (* The system is a matrix system in
  this case and it is built directly in NDSolve below *)
 RulesλRHSI = MyInterpolationRate@
   Table[{Tv, MyChop@RulesλRHS[Tv]}, {Tv, ListTRange[Tf, T18]}];
 RulesλbarRHSI = MyInterpolationRate@
   Table[{Tv, MyChop@RulesλbarRHS[Tv]}, {Tv, ListTRange[Tf, T18]}];
 RulesλRHS18I =
  MyInterpolationRate@Table[{Tv, RulesλRHS[NReactionsSmallNetwork, Tv]},
    {Tv, ListTRange[T18, TMiddle]}];
 RulesλbarRHS18I = MyInterpolationRate@
   Table[{Tv, RulesλbarRHS[NReactionsSmallNetwork, Tv]},
    {Tv, ListTRange[T18, TMiddle]}];

 On[General::munfl];,

 (* If $CompileNDSolve=False, we (re-)define the
  systems of equations for Middle and Low temperatures*)
 (* We associate the r.h.s formed thanks to DY18 and DY,
 with the l.h.s made of derivatives *)
 SystemEquationsMT[tv_] =
  Thread[Equal[FunPrimeList[tv], (DY18[Tv, ρBv, tv])]] /.
   {Tv → Toft@tv, ρBv → ρBForBBN@a@Toft@tv};
 SystemEquationsLT[tv_] =
  Thread[Equal[FunPrimeList[tv], (DY[Tv, ρBv, tv])]] /.
   {Tv → Toft@tv, ρBv → ρBForBBN@a@Toft@tv};
]
)
```

In[•]:=
```
DefineEquations; // Timing
```

Out[•]=
```
{2.112191, Null}
```

BBN numerical integration

Time delimitation of low, middle and high temperature eras

The time delimitation corresponding to Temperature delimitations of high, middle and low temperature eras.

In[•]:=
```
t0 := tofa@a[Tstart];
tmiddle := tofa@a[TMiddle];
t18 := tofa@a[T18];
tend := tofa@a[Tend];
```

Let us check the values in seconds or t0< tmiddle < t18 < tend

In[•]:=
```
{t0, tmiddle, t18, tend}
```

Out[•]=
```
{0.009947363, 1.0068626, 99.705965, 49 227.562}
```

High temperature integration (n and p only)

The initial conditions at high temperature are found from thermal and chemical equilibrium. This is used to integrate from $10^{11}\,K$ to $10^{10}\,K$. We keep track only of neutrons and protons.

In[•]:=
```
HoldYNames[period_String] := ToExpression /@
                chaîne de caractères   en expression
    ("Hold@Y[\"" <> period <> "\"][\"" <> # <> "\"]" & /@ ShortNames);
      maintiens
```

Initial conditions

In[•]:=
```
InitialConditionsHT[tv_] := Thread[Equal[FunList[tv], CIList[Toft[tv]]]];
                              enfile    égal
```

Actual solver. It solves the system and affects the results to the functions YHT[“n”] (neutrons) and YHT[“p”] (protons) which are functions of time.

```
In[•]:= SolveValueHighTemperatures := (Thread[MySet[Evaluate[HoldYNames["HT"]],
                                       enfile              évalue
      NDSolveValue[
      valeur donnée par solution numérique d'équation différentielle
        Flatten@Join[SystemEquationsHT[tv], InitialConditionsHT[t0]],
        aplatis      joins
        VarList, {tv, t0, tmiddle},
        PrecisionGoal → 8 + PrecisionNDSolve,
        objectif de précision
        AccuracyGoal → 11, InterpolationOrder → InterpOrder]]];
        objectif d'exactitude          ordre d'interpolation
    tHT = Y["HT"]["n"][[3, 1]];
   );
```

This period is very quick to solve as can be checked. The variable tHT stores the time steps used by the solver in case we are interested.

```
In[•]:= AbsoluteTiming[SolveValueHighTemperatures;]
        durée absolue
Out[•]=
{0.088691, Null}
```

We can also extend this integration with only weak interactions to much later times to see what would happen without nuclear reactions.
This is only to perform plots in the paper.

```
In[•]:= SolveValueHighTemperaturesYnOnly :=
   (Thread[MySet[Evaluate[HoldYNames["WeakInteractionsOnly"]],
    enfile              évalue
      NDSolveValue[
      valeur donnée par solution numérique d'équation différentielle
        Flatten@Join[SystemEquationsHT[tv], InitialConditionsHT[t0]],
        aplatis      joins
        VarList,
        {tv, t0, tend},
        PrecisionGoal → 8 + PrecisionNDSolve, AccuracyGoal → 11(* 11*),
        objectif de précision                 objectif d'exactitude
        InterpolationOrder → InterpOrder]]];
        ordre d'interpolation
   );

In[•]:= If[$ResultsPlots, SolveValueHighTemperaturesYnOnly;];
        si
```

Middle temperature integration (n,p,d,t,He3,He4,Be7,Li7,Li6)

The end values for neutrons and protons at high temperatures are used as initial conditions for the middle temperature era.
We use thermostatistical equilibrium for the species defined in the list ListThermalValuesMT

below, and 0 otherwise.

```
In[ ]:= ListThermalValuesMT = If[$ReducedNetwork, {"d", "t", "He3", "a", "Be7", "Li7"},
                                 |si
          {"d", "t", "He3", "a", "Be7", "Li7", "Li6"}];

        ListThermalValuesUsedMT :=
         Intersection[VariablesInEquations, ListThermalValuesMT]
         |intersection
```

We check the initial value used for the middle temperature era.

```
In[ ]:= YPeriodTimeOrStateEquilibrium["HT", ListThermalValuesUsedMT][TMiddle, tmiddle]
```

```
Out[ ]= {0.24028409, 0.75971591, 8.9526347 × 10^-13, 3.2773077 × 10^-23,
         4.2782273 × 10^-23, 1.6184419 × 10^-26, 1.4610691 × 10^-60, 6.867282 × 10^-61,
         0., 9.1609457 × 10^-51, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
```

```
In[ ]:= InitialConditionsMT[Tv_, tv_] := Thread[Equal[FunList[tv],
                                          |enfile    |égal
            YPeriodTimeOrStateEquilibrium["HT", ListThermalValuesUsedMT][Tv, tv]]];
```

We then have the differential equation solver. There are two possibilities depending on if we use the compiled version or not.
The values are then affected to the YMT[“n”], YMT[“n”], YMT[“d”] etc... which are the abundances of species in this era.

In[ ]:=
```
SolveValueMiddleTemperatures := (If[$CompileNDSolve,
                                   si

    (* Compiled version.*)
       compilé
    resMT = NDSolveValue[
            valeur donnée par solution numérique d'équation différentielle
      {Ytab'[tv] == DY18CN[ρBForBBN@a@Toft@tv,
         RulesλRHS18I[Toft@tv], RulesλbarRHS18I[Toft@tv], Ytab[tv]],
       Ytab[tmiddle] == YPeriodTimeOrStateEquilibrium["HT",
          ListThermalValuesUsedMT][TMiddle, tmiddle]},
      Ytab, {tv, tmiddle, t18},
      Method → {"BDF", "MaxDifferenceOrder" → $BDFOrder},
      méthode
      PrecisionGoal → 7 + PrecisionNDSolve, AccuracyGoal → 11,
      objectif de précision                 objectif d'exactitude
      InterpolationOrder → InterpOrder, Compiled → Automatic];
      ordre d'interpolation              compilé      automatique
    Y["MT"][key_][tv_?NumericQ] := resMT[tv][[KeyVal[key]]];
                                   expression numérique ?
    tMT = resMT[[3, 1]];,

    (* Non compiled version. Slightly slower *)
    Thread[MySet[Evaluate[HoldYNames["MT"]], NDSolveValue[
    enfile          évalue                  valeur donnée par solution numérique d'équation di
       Flatten@
       aplatis
        Join[SystemEquationsMT[tv], InitialConditionsMT[TMiddle, tmiddle]],
        joins
       VarList, {tv, tmiddle, t18},
       Method → {"BDF", "MaxDifferenceOrder" → $BDFOrder},
       méthode
       PrecisionGoal → 7 + PrecisionNDSolve, AccuracyGoal → 11,
       objectif de précision                 objectif d'exactitude
       InterpolationOrder → InterpOrder, Compiled → False]]];
       ordre d'interpolation              compilé      faux
    tMT = Y["MT"]["n"][[3, 1]];
   ]);
```

NB: tMT stores the time steps used. Can be used to check the behaviour of the integrator.

In[ ]:=
```
SolveValueMiddleTemperatures
```

```
I load compiled function from lib/DY18C_BBNRatesAC2024_Nreac_18_Nnuclei_59.mx
  and the library linked is lib/DY18C_BBNRatesAC2024_Nreac_18_Nnuclei_59.dylib
```

In[ ]:=
```
AbsoluteTiming[SolveValueMiddleTemperatures;]
durée absolue
```

Out[ ]=
```
{0.232935, Null}
```

For information we plot the result of the integration

In[ ]:=
```
If[$ResultsPlots,
 LogLogPlot[Evaluate[YPeriodTime["MT"][tv]], {tv, tmiddle, t18},
  Frame → True, PlotRange → {10^-40, 10}, FrameLabel → {"t(s)", "Yᵢ"}]]
```

Low temperature integration (All 59 isotopes)

!With the reduce network the list of nuclides is the same as in middle temperature range!

The end values at middle temperatures are then used as initial conditions for the low temperature era.
Now the full system of equations is used.

In[ ]:=
```
InitialConditionsLT[tv_] :=
  Thread[Equal[FunList[tv], YPeriodTime["MT"][tv]]];
```

In[ ]:=
```
SolveValueLowTemperatures := (If[$CompileNDSolve,
                                 |si

    (* Compiled version*)
       |compilé
    resLT = NDSolveValue[
            |valeur donnée par solution numérique d'équation différentielle
      {Ytab'[tv] == DYCN[ρBForBBN@a@Toft@tv,
         RulesλRHSI[Toft@tv], RulesλbarRHSI[Toft@tv], Ytab[tv]],
       Ytab[t18] == YPeriodTime["MT"][t18]},
      Ytab, {tv, t18, tend},
      (*Method→{NDSolve`LSODA,"MaxDifferenceOrder"→1},*)
        |méthode
      Method → {"BDF", "MaxDifferenceOrder" → $BDFOrder},
      |méthode
      PrecisionGoal → 5 + PrecisionNDSolve, AccuracyGoal → AccuracyNDSolve,
      |objectif de précision                |objectif d'exactitude
      InterpolationOrder → InterpOrder,
      |ordre d'interpolation
      StartingStepSize → 10^-4, MaxStepSize → 500];
      |taille d'étape initiale   |taille maximale de pas
    Y["LT"][key_][tv_?NumericQ] := resLT[tv][[KeyVal[key]]];
                                   |expression numérique ?
    tLT = resLT[[3, 1]];,

    (* Uncompiled version. Slower. *)
    Thread[MySet[Evaluate[HoldYNames["LT"]], NDSolveValue[
    |enfile          |évalue                 |valeur donnée par solution numérique d'équation di
        Flatten@Join[SystemEquationsLT[tv], InitialConditionsLT[t18]],
        |aplatis      |joins
        VarList, {tv, t18, tend},
        Method → {"BDF", "MaxDifferenceOrder" → $BDFOrder,
        |méthode
          "EquationSimplification" → "Solve"},
                                    |résous
        PrecisionGoal → 5 + PrecisionNDSolve, AccuracyGoal → AccuracyNDSolve,
        |objectif de précision               |objectif d'exactitude
        InterpolationOrder → InterpOrder, StartingStepSize → 10^-4]]];
        |ordre d'interpolation              |taille d'étape initiale
    tLT = Y["LT"]["n"][[3, 1]];
   ];)
```

In[ ]:=
```
AbsoluteTiming[SolveValueLowTemperatures;]
|durée absolue
```

```
I load compiled function from lib/DYC_BBNRatesAC2024_Nreac_424_Nnuclei_59.mx
  and the library linked is lib/DYC_BBNRatesAC2024_Nreac_424_Nnuclei_59.dylib
```

Out[ ]=
```
{12.312889, Null}
```

We can plot the results

```
In[ ]:= If[$ResultsPlots, LogLogPlot[Evaluate[YPeriodTime["LT"][tv]], {tv, t18, tend},
        si                  tracé log–log  évalue
    Frame → True, PlotRange → {10^-40, 10}, FrameLabel → {"t(s)", "Yᵢ"}]]
    cadre  vrai  zone de tracé              étiquette de cadre
```

Gathering integrations on all periods in one function

We define an interpolation of the results. We join the results from high, middle and low temperature eras. This is joined in the function

YI[“key”][time] where key is the name of the nuclide (e.g. “a” for He4, “t” for tritium and “d” for deuterium but otherwise “Li7”, “C12” etc...).

```
In[ ]:= InterpolateResults := (
          Clear[Yall, YI];
          efface
          Yall[key_?KeyQ] := Yall[key] = Function[{tv},
                                         fonction
             Piecewise[{{Y["HT"][key][tv], tv < tmiddle}, {Y["MT"][key][tv],
             fonction par morceaux
                 tv < t18 && tv ≥ tmiddle}, {Y["LT"][key][tv], tv ≤ tend && tv ≥ t18}}]];

          YI[key_?KeyQ] := YI[key] = Interpolation[Table[{tv, Yall[key][tv]},
                                     interpolation      table
              {tv, Join[tHT, Rest@tMT, Rest@tLT]}], InterpolationOrder → 1];);
                   joins      reste      reste      ordre d'interpolation
```

```
In[ ]:= InterpolateResults;
```

The function RunNumericalIntegrals below performs the integrations of incomplete neutrino decoupling
and then the Friedmann equation integration.
Then it defines the nuclear reactions, possibly having introduced uncertainty on rates depending on options,
and solves for the high, middle and low temperature era.
This is the Driver of PRIMAT which needs to be called whenever we rerun PRIMAT with new parameters (e.g. exploring dependence in baryons or neutrinos).

In[ ]:=
```
RunNumericalIntegralsNuclearReactions := (
   (* Middle temperature integration *)
   If[$Verbose,
   si
    Print[" Launching middle temperatures, with only the lightest
    imprime
       elements, from T=", TMiddle, "(K) to T=", T18, "(K)"]];
   SolveValueMiddleTemperatures;

   (* Low temperature integration *)
   If[$Verbose,
   si
    Print[" Launching low temperatures, with all elements, from T=",
    imprime
     T18, "(K) to T=", Tend, "(K)"]];
   SolveValueLowTemperatures;
  );
```

In[ ]:=
```
RunBackgroundThermo := (
  time = Timing[LoadPlasma;][[1]];
  (*Print["Timing for plasma is",time];*)

  time = Timing[If[$NEVO, LoadNEVO]][[1]];
  (*Print["Timing for loading NEVO is",time];*)

  LoadDistortions;

  (* In case of incomplete neutrino decoupling, we recompute
   all the integrations a(T) then inversion T(a), then ρν(a).*)
  If[$IncompleteNeutrinoDecoupling,
   RecomputeIncompleteNeutrinoDecoupling;];
  (*In case the plasma conditions have changed in a MC exploration,
  we recompute the inversion of a[T]*)
  (* This is needed if we have recomputed the neutrino decoupling,
  but I am wondering if this is always needed. *)
  InvertaOFT;

  If[$Verbose, Print["Infer a<->t relations"];];
  (* scale factor integration from Friedmann equation. *)
  Computetofa;
  Computeaoft;
 )
```

```
RunNumericalIntegrals := (
   If[$Verbose, "Checking Options."];
   si                          options
   CheckOptions;

   RunBackgroundThermo;

   Off[NIntegrate::izero];
   dé··· intègre numériquement
   LoadWeakRates;
   On[NIntegrate::izero];
   ac·· intègre numériquement

   (* Build equations. Needed since rate are
    modified randomly by the f factor of each reaction*)
   If[$Verbose, Print["Loading nuclear rates and building reactions."];];
   si             imprime
   LoadRatesFull;

   If[$Verbose, Print[MyGrid[Join[{{"Reaction Number", "Reaction Name",
   si             imprime          joins                nombre
          "Initial species", "Final Species", "Reference"}},
       PadLeftNumberRowFromZero[Drop[#, {4}] & /@
                                  laisse tomber
         (NiceDisplayReaction /@ ListReactionsUpToChosenMass)]]]]];

   DefineEquations;

   If[$Verbose, Print["Launching BBN numerics."]];
   si             imprime
   If[$Verbose,
   si
    Print[" Launching high temperatures, with only n/p, from T=",
    imprime
     Tstart, "(K) to T=", TMiddle, "(K)"]];
   (* High temperature integration with only PEN reactions *)
   SolveValueHighTemperatures;

   (* Middle and Low temperature WITH nuclear reactions*)
   RunNumericalIntegralsNuclearReactions;

   (*Interpolating results *)
   InterpolateResults;
  );
```

Gathering the numerical results

We define a pseudo-mass fraction as a function of time using the interpolated results

```
In[•]:= XI[key_?KeyQ][t_] := Ai[key] × YI[key][t]
        XI["CNO"][t_] := Plus @@ ((Ai[#] × YI[#][t] &) /@ CNONuclei)
                         |plus
```

Final abundances are obtained by evaluation at t = tend.

We define a shorthand for the final abundances (Yf) and pseudo mass fraction (Xf).

```
In[•]:= Yf[key_] := YLT[key][tend]
        Xf[key_] := Ai[key] × Yf[key]

        Xf["CNO"] := Plus @@ ((Xf[#] &) /@ CNONuclei)
                     |plus
```

And a shorthand notation for Y_i / H

```
In[•]:= YfH[key_] := Yf[key] / Yf["p"]
```

Results and plots

Checks of the conservation of total number of baryons

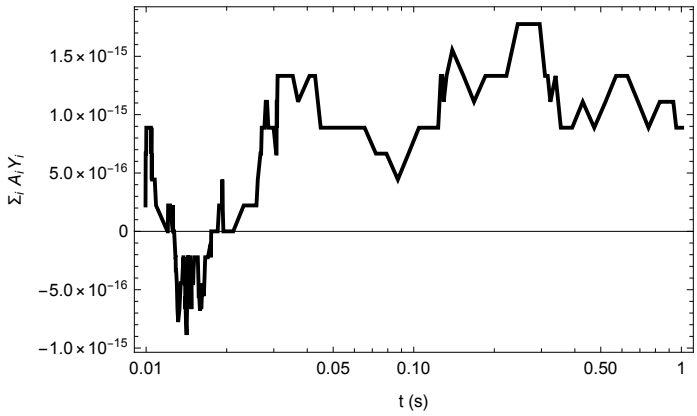
The total number MUST be conserved. We build it from the nuclear weights of species.
At high temperatures we check visually. By construction the quantity Ytot =
$\Sigma_i A_i Y_i$ should be 1 and conserved. We define it and plot the difference with unity.

```
In[•]:= Ytot[period_String] :=
            |chaîne de caractères
          Function[{tv}, Plus @@ (WeightsNuclear * YPeriodTime[period][tv])];
          |fonction      |plus
```

High temperature era

In[•]:=
```
LogLinearPlot[Ytot["HT"][tv] - 1, {tv, t0, tmiddle},
 tracé log-linéaire
  Frame → True, FrameLabel → {"t (s)", "Σi AiYi"}, PlotStyle → Black]
 cadre  vrai  étiquette de cadre  style de tracé  noir
```

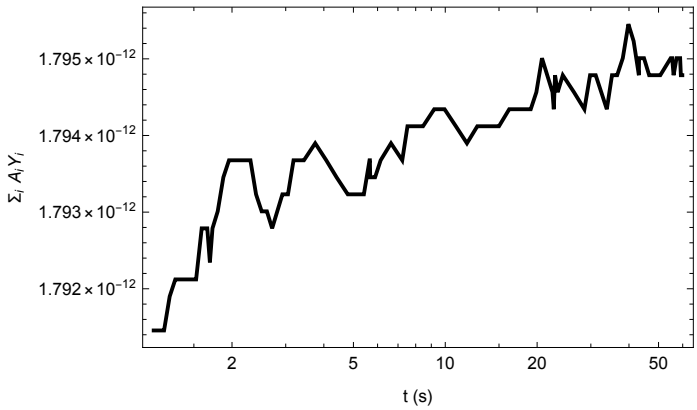
Out[•]=



Middle temperature era

In[•]:=
```
LogLinearPlot[Ytot["MT"][tv] - 1, {tv, tmiddle * 1.1, 0.6 t18},
 tracé log-linéaire
  Frame → True, FrameLabel → {"t (s)", "Σi AiYi"}, PlotStyle → Black]
 cadre  vrai  étiquette de cadre  style de tracé  noir
```

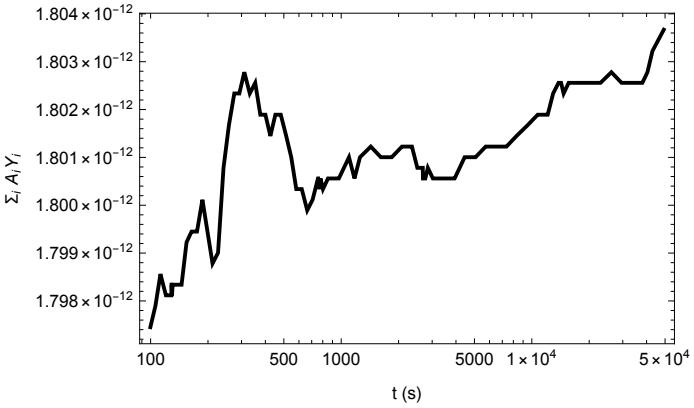
Out[•]=



Low temperature era with the full network.

In[ ]:=
```
LogLinearPlot[Ytot["LT"][t] - 1, {t, t18, tend},
 Frame → True, FrameLabel → {"t (s)", "Σᵢ AᵢYᵢ"}, PlotStyle → Black]
```
tracé log-linéaire · cadre · vrai · étiquette de cadre · style de tracé · noir

Out[ ]=



Time evolution of abundances

Early thermodynamical equilibrium

Estimate of T_{nuc} (see companion paper)

In[ ]:=
```
TFreeze = 0.8 MeV / kB;
tFreeze = tofa@a@TFreeze
YnF[tv_] := 1 / (1 + Exp[Q / kB / TFreeze]) Exp[- (tv - tFreeze) / τneutron ];
```
exponentielle · exponentielle
```
YpF[tv_] := 1 - YnF[tv];
```

Out[ ]=
```
1.1704961
```

In[ ]:=
```
tnuc = FindRoot[YNSE["d", YnF[tv], YpF[tv], Toft[tv]] == YnF[tv], {tv, 100}][[1, 2]]
```
trouve racine
```
Tnuc = Toft[tnuc]
```

Out[ ]=
```
296.70321
```

Out[ ]=
```
7.6959722 × 10^8
```

Abundance of neutrons at T_{nuc} and T_{nuc} in MeV

In[ ]:=
```
YnF[tnuc]
kB * Tnuc / MeV
```

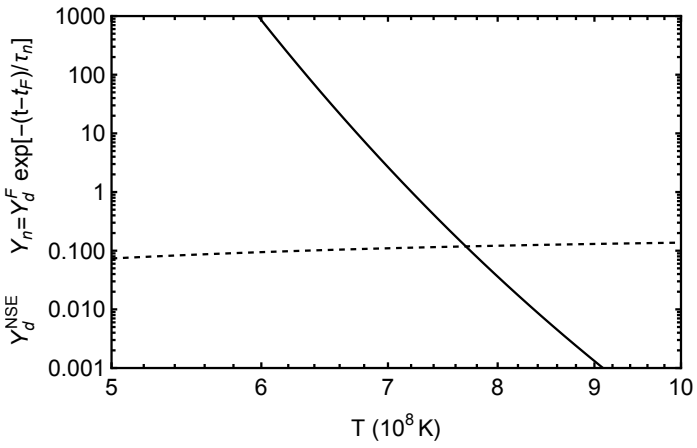
Out[ ]=
```
0.11833563
```

Out[ ]=
```
0.066318757
```

In[ ]:=

```
PlotDeutEq =
 Show[LogLogPlot[{YnF[tofa@a[10^8 Tv]], YNSE["d", YnF[tofa@a[10^8 Tv]],
 mon··· tracé log–log
     YpF[tofa@a[10^8 Tv]], 10^8 Tv]}, {Tv, 0.05 * 10^2, 0.1 * 10^2},
   GridLines → {{{Tnuc, {Gray, Thickness[0.005]}}}, {}}, Frame → True,
   lignes de grille  gris  épaisseur  cadre  vrai
   PlotRange → {{0.05 * 10^2, 0.1 * 10^2}, {10^-3, 10^3}},
   zone de tracé
   PlotRangePadding → None, FrameStyle → Thickness[0.004],
   garnissage de zone de tr···  aucun  style de cadre  épaisseur
   PlotStyle → {{Black, Thickness[0.004], Dashing[0.01]},
   style de tracé  noir  épaisseur  style de tirets
     {Black, Thickness[0.004]}}, PlotRange → {10^-3, 1000},
     noir  épaisseur  zone de tracé
   FrameLabel → {"T (10^8 K)", "Y_d^NSE  Y_n=Y_d^F exp[-(t-t_F)/τ_n]"},
   étiquette de cadre
   LabelStyle → {FontSize → 12}],
   style d'étiquette  taille de police de caractères
  Graphics[{Rotate[Text[Style["0.066 MeV", FontSize → 10, Black],
  graphique  fais pivo··  texte  style  taille de police de ···  noir
      {Log@Tnuc - 0.015, 2}], 90 Degree]}]]
       logarithme  degré

If[$ResultsPlots,
si
 Export["Plots/PlotDeutEq.pdf", Style[PlotDeutEq, Magnification → 1], "PDF"];]
 exporte  style  agrandissement  fonction de
```

Out[ ]=



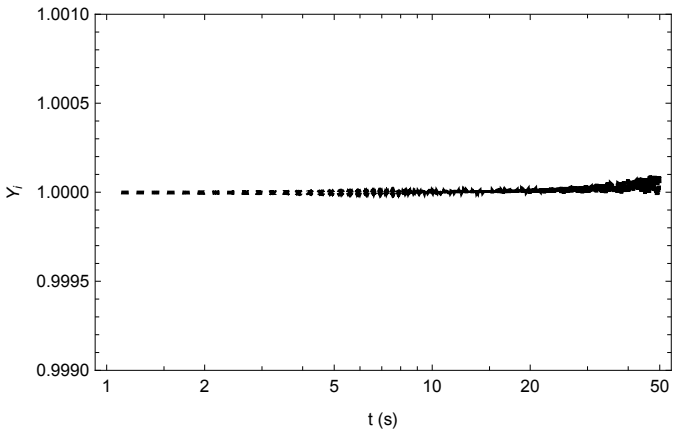
Checks of thermo equilibrium at early times.

We check the accuracy of thermal equilibrium for “d” “t” “a” “Li7”.

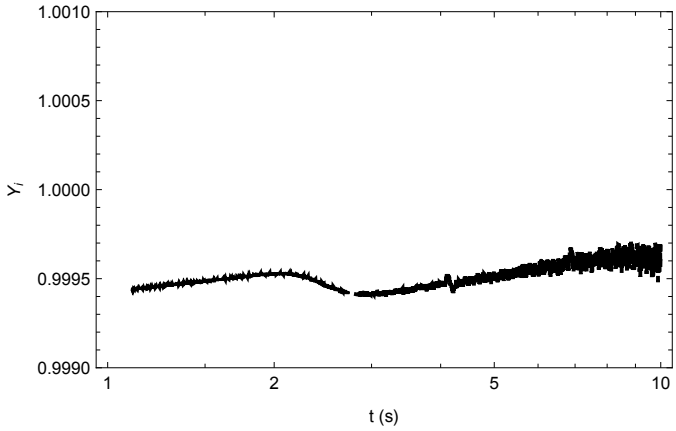
Most important is deuterium because it determines the final Helium abundance.

In[•]:=
```
LogLogPlot[{YI["d"][tv] / YNSE["d", YI["n"][tv], YI["p"][tv], Toft[tv]]},
 {tv, tmiddle * 1.1, 50}, Frame → True, FrameLabel → {"t (s)", "Yᵢ"},
 PlotStyle → {Black, {Black, Dashed}},
 ImagePadding → {{50, 10}, {40, 25}}, PlotRange → {0.999, 1.001}]

LogLogPlot[{YI["t"][tv] / YNSE["t", YI["n"][tv], YI["p"][tv], Toft[tv]]},
 {tv, tmiddle * 1.1, 10}, Frame → True, FrameLabel → {"t (s)", "Yᵢ"},
 PlotStyle → {Black, {Black, Dashed}},
 ImagePadding → {{50, 10}, {40, 25}}, PlotRange → {0.999, 1.001}]

(*LogLogPlot[{YI["a"][tv]/YNSE["a",YI["n"][tv],YI["p"][tv],Toft[tv]]},
 {tv,tmiddle*1.1,1.5},Frame→True,FrameLabel→{"t (s)","Yᵢ"},
 PlotStyle→{Black,{Black,Dashed}},FrameTicks→MyTicks,
 ImagePadding→{{50,10},{40,25}},PlotRange→{0.999,1.001}]

 LogLogPlot[{YI["Li7"][tv]/YNSE["Li7",YI["n"][tv],YI["p"][tv],Toft[tv]]},
 {tv,tmiddle*1.1,1.5},Frame→True,FrameLabel→{"t (s)","Yᵢ"},
 PlotStyle→{Black,{Black,Dashed}},FrameTicks→MyTicks,
 ImagePadding→{{50,10},{40,25}},PlotRange→{0.999,1.001}]*)
```

Out[•]=



Out[ ]=

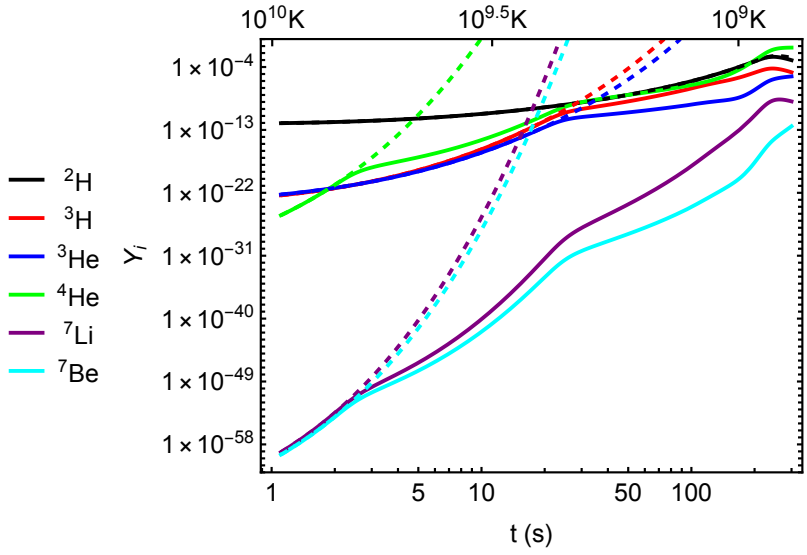


We plot early values together with thermal equilibrium in dashes

In[ ]:=
```
MyTickst = {{Automatic, Automatic},
                automatique   automatique
  {Automatic, {{tofa@a[10^11], "10^11K"}, {tofa@a[10^10.5], "10^10.5K"},
   automatique
    {tofa@a[10^10], "10^10K"}, {tofa@a[10^9.5], "10^9.5K"}, {tofa@a[10^9],
     "10^9K"}, {tofa@a[10^8.5], "10^8.5K"}, {tofa@a[10^8], "10^8K"}}}};
```

In[ ]:=
```
PL1 = LogLogPlot[{YI["d"][tv], YI["t"][tv], YI["He3"][tv], YI["a"][tv],
      YI["Li7"][tv], YI["Be7"][tv], YNSE["d", YI["n"][tv], YI["p"][tv], Toft[tv]],
     YNSE["t", YI["n"][tv], YI["p"][tv], Toft[tv]],
     YNSE["He3", YI["n"][tv], YI["p"][tv], Toft[tv]],
     YNSE["a", YI["n"][tv], YI["p"][tv], Toft[tv]],
     YNSE["Li7", YI["n"][tv], YI["p"][tv], Toft[tv]],
     YNSE["Be7", YI["n"][tv], YI["p"][tv], Toft[tv]]},
    {tv, tmiddle * 1.1, 300}, Frame → True, FrameLabel → {"t (s)", "Yᵢ"},
    FrameTicks → MyTickst, LabelStyle → {FontSize → 13},
    FrameStyle → Thickness[0.004], PlotRange → {10^-62, 1}, AspectRatio → .8,
    PlotStyle → {Black, Red, Blue, Green, Purple, Cyan, {Black, Dashed},
      {Red, Dashed}, {Blue, Dashed}, {Green, Dashed}, {Purple, Dashed},
      {Cyan, Dashed}}, (*ImagePadding→{{50,10},{40,25}},*)
    PlotLegends → Placed[LineLegend[{"²H", "³H", "³He", "⁴He", "⁷Li", "⁷Be"},
      LegendLayout → (Grid[#, Frame → None] &)], Left]]

If[$ResultsPlots, Export["Plots/PlotEarlyEquilibrium.pdf",
    Style[PL1, Magnification → 1], "PDF"];]
```

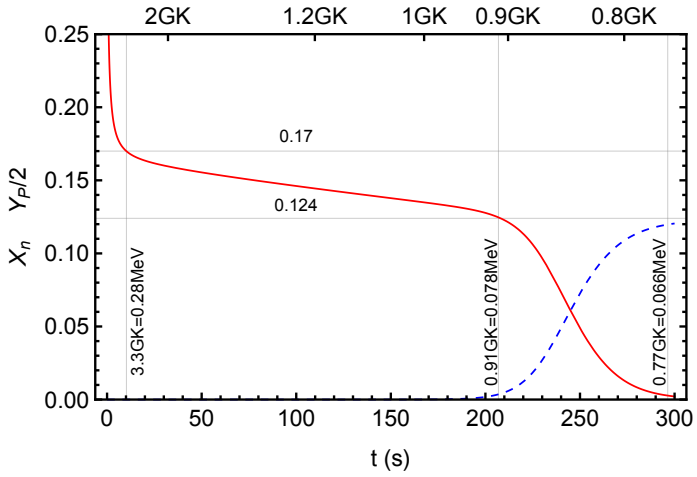
Out[ ]=



Neutrons only evolution

In[•]:=
```
YnCoc =
 Show[Plot[{YI["n"][tv], Y["WeakInteractionsOnly"]["n"][tv], YI["a"][tv] * 2
 mon··  tracé
     (*,1/(1+Exp[Q/kB/Toft[tv]])*)}, {tv, t0, 300}, Frame → True,
              exponentielle                          cadre  vrai
   FrameTicks → {{Automatic, Automatic}, {Automatic, {{tofa@a[10^9], "1GK"},
   graduations de cadre  automatique  automatique  automatique
        {tofa@a[.8 * 10^9], "0.8GK"}, {tofa@a[.9 * 10^9], "0.9GK"},
        {tofa@a[1.2 * 10^9], "1.2GK"}, {tofa@a[2 × 10^9], "2GK"}}}},
   FrameStyle → Thickness[0.004], FrameLabel → {"t (s)", "Xn    YP/2"},
   style de cadre  épaisseur  étiquette de cadre
   LabelStyle → {FontSize → 12},
   style d'étiquette  taille de police de caractères
   PlotStyle → {{Red, Thickness[0.003]}, {Red, Thickness[0.003], Dotted},
   style de tracé  rouge  épaisseur  rouge  épaisseur  en pointillé
     {Blue, Thickness[0.003], Dashed}}, PlotRange → {0, 0.25},
      bleu  épaisseur  en tirets  zone de tracé
   GridLines → {{{tofa@a[3.3 Giga Kelvin], {Gray, Thickness[0.003]}},
   lignes de grille  gris  épaisseur
       {tofa@a[0.91 Giga Kelvin], {Gray, Thickness[0.003]}},
                                  gris  épaisseur
       {tofa@a[0.77 × 10^9], {Gray, Thickness[0.003]}}},
                             gris  épaisseur
     {{0.124, {Gray, Thickness[0.003]}}, {0.17, {Gray, Thickness[0.003]}}}}],
    Graphics[
    graphique
     {Rotate[Text[Style["3.3GK=0.28MeV", FontSize → 9, Black], {16, 0.06}],
      fais pivo··  texte  style  taille de police de··  noir
       90 Degree], Rotate[Text[Style["0.91GK=0.078MeV", FontSize → 9, Black],
        degré  fais pivo··  texte  style  taille de police de··  noir
       {203, 0.06}], 90 Degree],
                      degré
      Rotate[Text[
      fais pivo··  texte
        Style["0.77GK=0.066MeV", FontSize → 9, Black], {292, 0.06}], 90 Degree],
        style  taille de police de··  noir  degré
      Rotate[Text[Style["0.17", FontSize → 9, Black], {100, 0.18}], 0 Degree],
      fais pivo··  texte  style  taille de police de··  noir  degré
      Rotate[Text[Style["0.124", FontSize → 9, Black], {100, 0.133}], 0 Degree]}]]

If[$ResultsPlots,
si
 Export["Plots/PlotYnCoc.pdf", Style[YnCoc, Magnification → 1], "PDF"];]
 exporte  style  agrandissement  fonction de densité de
```

Out[ ]=

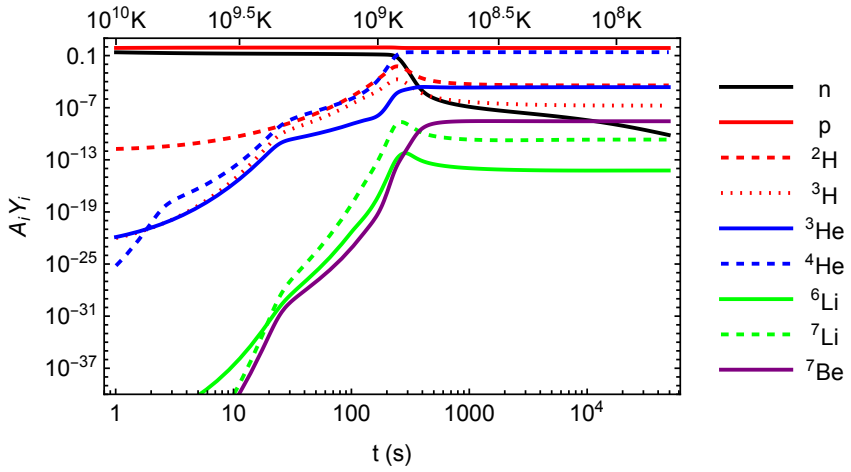


Main species of the small network

The plots assume that we have used the full nuclear network and not the reduced one. Otherwise some nuclides might not exist and Mathematica should complain ...

In[ ]:=
```
BBNsmall = LogLogPlot[
           (tracé log–log)
  {YI["n"][tv], YI["p"][tv], 2 YI["d"][tv], 3 YI["t"][tv], 3 YI["He3"][tv],
   4 YI["a"][tv], YI["Li6"][tv], YI["Li7"][tv], 7 YI["Be7"][tv]},
  {tv, tmiddle, tend}, Frame → True, FrameStyle → Thickness[0.003],
                       (cadre  vrai  style de cadre  épaisseur)
  FrameLabel → {"t (s)", "AᵢYᵢ"}, LabelStyle → {FontSize → 12},
  (étiquette de cadre)                (style d'étiquette  taille de police de caractères)
  PlotRange → {10^-40, 10}, PlotStyle → {Black, Red, {Red, Dashed},
  (zone de tracé)                       (style de tracé  noir  rouge  rouge  en tirets)
    {Red, Dotted}, Blue, {Blue, Dashed}, Green, {Green, Dashed}, Purple},
    (rouge  en pointillé  bleu  bleu  en tirets  vert  vert  en tirets  violet)
  FrameTicks → MyTickst, ImagePadding → {{50, 10}, {40, 25}}, PlotLegends →
  (graduations de cadre)               (garnissage d'image)                (légendes de tracé)
   Placed[LineLegend[{"n", "p", "²H", "³H", "³He", "⁴He", "⁶Li", "⁷Li", "⁷Be"},
   (placé)  (légende de ligne)
     LegendLayout → (Grid[#, Frame → None] &)], Right]]
     (disposition de légende  grille  cadre  aucun  droite)
```

Out[ ]=



In[ ]:=
```
If[$ResultsPlots, Export["Plots/BBNsmall.pdf", BBNsmall, "PDF"];]
(si)              (exporte)                    (fonction de)
```

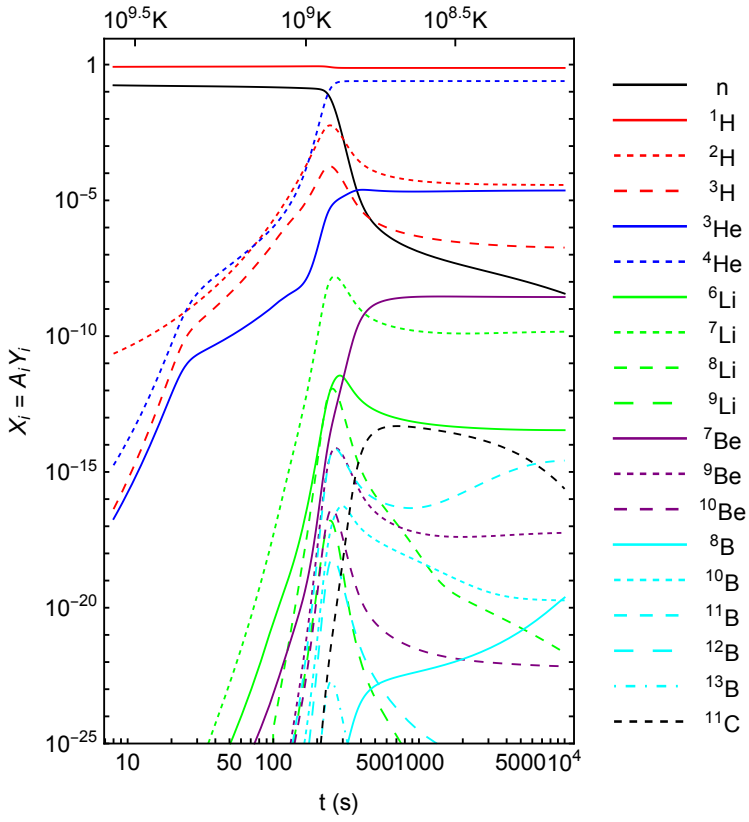
From Hydrogen to Borron

Custom colors for plots of a given element

In[ ]:=
```
ListColor[Color_, n_] := Take[Join[{{Color, Thickness[0.004]}},
                         (prends  joins)  (épaisseur)
  Table[{Color, Thickness[0.004], Dashing[{(i) 0.01}], {i, 1, 3}],
  (table)       (épaisseur)       (style de tirets)
  Table[{Color, Thickness[0.004],
  (table)       (épaisseur)
    Dashing[{0, 0.015 * i, (i) 0.015, i 0.015}]}, {i, 1, 4}]], n]
    (style de tirets)
```

In[•]:=
```
BBNsmall2 =
 LogLogPlot[{XI["n"][tv], XI["p"][tv], XI["d"][tv], XI["t"][tv], XI["He3"][tv],
   XI["a"][tv], XI["Li6"][tv], XI["Li7"][tv], XI["Li8"][tv], XI["Li9"][tv],
   XI["Be7"][tv], XI["Be9"][tv], XI["Be10"][tv], XI["B8"][tv], XI["B10"][tv],
   XI["B11"][tv], XI["B12"][tv], XI["B13"][tv], XI["C11"][tv]}, {tv, 8, 10^4},
  Frame → True, FrameLabel → {"t (s)", "X_i = A_i Y_i"}, LabelStyle → {FontSize → 12},
  FrameStyle → Thickness[0.004], PlotRange → {10^-25, 9},
  PlotStyle → Join[ListColor[Black, 1], ListColor[Red, 3], ListColor[Blue, 2],
    ListColor[Green, 4], ListColor[Purple, 3], ListColor[Cyan, 5],
    {{Black, Thickness[0.004], Dashed}}], AspectRatio → 1.5,
  FrameTicks → MyTickst, ImagePadding → {{50, 10}, {40, 25}},
  PlotLegends → Placed[LineLegend[{"n", "¹H", "²H", "³H", "³He", "⁴He", "⁶Li",
      "⁷Li", "⁸Li", "⁹Li", "⁷Be", "⁹Be", "¹⁰Be", "⁸B", "¹⁰B", "¹¹B", "¹²B",
      "¹³B", "¹¹C"}, LegendLayout → (Grid[#, Frame → None] &)], Right]]

If[$ResultsPlots,
 Export["Plots/PlotBBNLight.pdf", Style[BBNsmall2, Magnification → 1], "PDF"];]
```

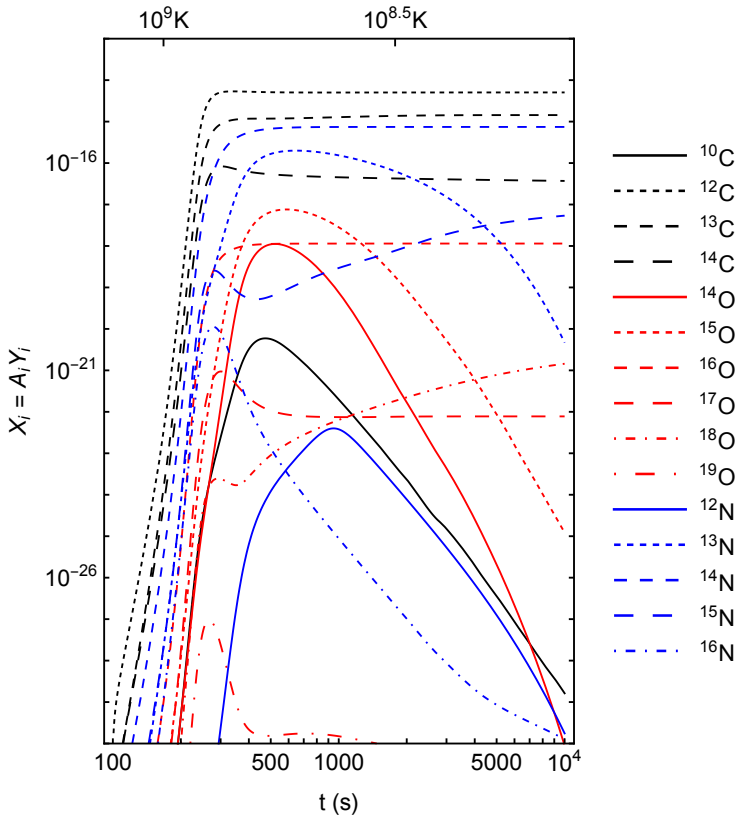
Out[ ]=


$10^{9.5}$K
10^{9}K
$10^{8.5}$K
1
10^{-5}
10^{-10}
10^{-15}
10^{-20}
10^{-25}
$X_i = A_i Y_i$
10
50
100
500
1000
5000
10^4
t (s)
n
^{1}H
^{2}H
^{3}H
^{3}He
^{4}He
^{6}Li
^{7}Li
^{8}Li
^{9}Li
^{7}Be
^{9}Be
^{10}Be
^{8}B
^{10}B
^{11}B
^{12}B
^{13}B
^{11}C

From Carbon to Oxygen

```
In[ ]:= If[Not[$ReducedNetwork],
          BBNCNO = LogLogPlot[{XI["C10"][tv], XI["C12"][tv], XI["C13"][tv],
              XI["C14"][tv], XI["O14"][tv], XI["O15"][tv], XI["O16"][tv],
              XI["O17"][tv], XI["O18"][tv], XI["O19"][tv], XI["N12"][tv],
              XI["N13"][tv], XI["N14"][tv], XI["N15"][tv], XI["N16"][tv]},
            {tv, 1.01 t18, 10^4}, Frame → True, FrameLabel → {"t (s)", "X_i = A_i Y_i"},
            LabelStyle → {FontSize → 12}, FrameStyle → Thickness[0.004],
            PlotRange → {10^-30, 10^-13}, PlotStyle → Join[ListColor[Black, 4],
              ListColor[Red, 6], ListColor[Blue, 5], ListColor[Purple, 1]],
            AspectRatio → 1.5, FrameTicks → MyTickst, ImagePadding → {{50, 10}, {40, 25}},
            PlotLegends → Placed[LineLegend[{"¹⁰C", "¹²C", "¹³C", "¹⁴C", "¹⁴O",
                "¹⁵O", "¹⁶O", "¹⁷O", "¹⁸O", "¹⁹O", "¹²N", "¹³N", "¹⁴N", "¹⁵N", "¹⁶N"},
              LegendLayout → (Grid[#, Frame → None] &)], Right]]
        ]

        If[$ResultsPlots && Not[$ReducedNetwork],
          Export["Plots/PlotBBNHeavy.pdf", Style[BBNCNO, Magnification → 1], "PDF"];];
```

Out[ ]=



Final abundances

Main results

Final time at T = 4×10^7 K in seconds.

In[ ]:=
```
tofa[a[4 * 10^7]]
```

Out[ ]=

```
110 753.28
```

Standard abundances as reported in most BBN papers.
Note the definition $Y_P = 4\, Y_{He4}$. Since the atomic mass
of Helium is not exactly 4 this is not exactly Helium mass abundance

In[ ]:=
```
MyGrid@Transpose[
                transpose
  {{"H", "Y_P=4Y_He", "D/H x 10^5", "^3He/H x 10^5", "T/H x 10^8", "(^3He+T)/H x 10^5",
                                  dérivée d
    "^7Li/H x 10^11", "^7Be/H x 10^10", "(^7Li+^7Be)/H x 10^10", "^6Li/H x 10^14",
    "^9Be/H x 10^19", "^10B/H x 10^21", "^11B/H x 10^16", "CNO/H x 10^16", "N_eff"},
   {Yf["p"], 4 Yf["a"], YfH["d"] 10^5, YfH["He3"] 10^5, YfH["t"] 10^8,
    (YfH["t"] + YfH["He3"]) 10^5, YfH["Li7"] 10^11, YfH["Be7"] 10^10,
    (YfH["Li7"] + YfH["Be7"]) 10^10, YfH["Li6"] 10^14, YfH["Be9"] 10^19,
    YfH["B10"] 10^21, YfH["B11"] 10^16, YfH["CNO"] 10^16, Neff}}]
```

Out[ ]=

H	0.75292973
$Y_P=4Y_{He}$	0.2470101
$D/H \times 10^5$	2.4365041
$^3He/H \times 10^5$	1.031397
$T/H \times 10^8$	7.7566149
$(^3He+T)/H \times 10^5$	1.0391536
$^7Li/H \times 10^{11}$	2.8408514
$^7Be/H \times 10^{10}$	5.2224322
$(^7Li+^7Be)/H \times 10^{10}$	5.5065173
$^6Li/H \times 10^{14}$	0.77927636
$^9Be/H \times 10^{19}$	8.7639613
$^{10}B/H \times 10^{21}$	2.4449084
$^{11}B/H \times 10^{16}$	3.2516885
$CNO/H \times 10^{16}$	7.8057991
N_{eff}	3.0439773

All final abundances

Final number fractions

In[ ]:=
```
MyGrid[{#, Yf[#]} & /@ ShortNames]
```

Out[ ]=

n	$7.1588061 \times 10^{-11}$
p	0.75292973
d	0.000018345164
t	5.840186×10^{-8}
He3	7.7656948×10^{-6}
a	0.061752525
Li7	$2.1389615 \times 10^{-11}$
Be7	$3.9321245 \times 10^{-10}$
He6	$4.4957773 \times 10^{-44}$
Li6	$5.8674034 \times 10^{-15}$
Li8	$4.9881683 \times 10^{-26}$
Li9	$3.3334707 \times 10^{-41}$
Be9	$6.5986471 \times 10^{-19}$
Be10	$6.4509515 \times 10^{-24}$
Be11	$2.4539972 \times 10^{-38}$

Be12	$1.0235161 \times 10^{-55}$
B8	$1.0283055 \times 10^{-23}$
B10	$1.8408442 \times 10^{-21}$
B11	2.448293×10^{-16}
B12	$1.8044084 \times 10^{-32}$
B13	$1.5543563 \times 10^{-48}$
B14	$2.4533269 \times 10^{-63}$
B15	$1.3703606 \times 10^{-81}$
C9	2.054702×10^{-40}
C10	$8.4775263 \times 10^{-37}$
C11	$4.6781285 \times 10^{-27}$
C12	$4.2105773 \times 10^{-16}$
C13	$1.1048876 \times 10^{-16}$
C14	$2.4200729 \times 10^{-18}$
C15	$5.3172726 \times 10^{-34}$
C16	$2.0127112 \times 10^{-49}$
N12	$1.5583642 \times 10^{-45}$
N13	1.451336×10^{-28}
N14	5.309502×10^{-17}
N15	$5.8857772 \times 10^{-19}$
N16	$3.1681132 \times 10^{-34}$
N17	$2.3682892 \times 10^{-44}$
O13	$-1.7680891 \times 10^{-58}$
O14	$3.1948854 \times 10^{-43}$
O15	$6.5545944 \times 10^{-32}$
O16	$7.1532754 \times 10^{-20}$
O17	$4.5679785 \times 10^{-24}$
O18	$1.3320716 \times 10^{-22}$
O19	$6.6615972 \times 10^{-36}$
O20	$6.2975606 \times 10^{-49}$
F17	$5.4577358 \times 10^{-36}$
F18	$8.6690587 \times 10^{-25}$
F19	$6.3396576 \times 10^{-26}$
F20	$3.8443257 \times 10^{-38}$
Ne18	$3.6964668 \times 10^{-49}$
Ne19	$1.3000192 \times 10^{-41}$
Ne20	$6.8049224 \times 10^{-28}$
Ne21	$5.3126351 \times 10^{-30}$
Ne22	$8.8170553 \times 10^{-32}$
Ne23	$-4.7427338 \times 10^{-74}$
Na20	$8.2274167 \times 10^{-64}$
Na21	$-1.9908151 \times 10^{-77}$
Na22	$1.7675041 \times 10^{-36}$
Na23	1.08235×10^{-37}

Table in mass fraction instead of number fraction. So we use $X_i = A_i Y_i$.

In[•]:=
```
TableXMass = {#, Xf[#]} & /@ Join[ShortNames, {"CNO"}];
```
joins

```
MyGrid[TableXMass]
```

Out[•]=

n	$7.1588061 \times 10^{-11}$
p	0.75292973
d	0.000036690327
t	1.7520558×10^{-7}
He3	0.000023297084
a	0.2470101
Li7	1.497273×10^{-10}
Be7	2.7524871×10^{-9}
He6	$2.6974664 \times 10^{-43}$
Li6	$3.5204421 \times 10^{-14}$
Li8	$3.9905346 \times 10^{-25}$
Li9	$3.0001236 \times 10^{-40}$
Be9	$5.9387823 \times 10^{-18}$
Be10	$6.4509515 \times 10^{-23}$
Be11	2.699397×10^{-37}
Be12	$1.2282193 \times 10^{-54}$
B8	$8.2264444 \times 10^{-23}$
B10	$1.8408442 \times 10^{-20}$
B11	$2.6931223 \times 10^{-15}$
B12	$2.1652901 \times 10^{-31}$
B13	$2.0206631 \times 10^{-47}$
B14	$3.4346577 \times 10^{-62}$
B15	$2.0555409 \times 10^{-80}$
C9	$1.8492318 \times 10^{-39}$
C10	$8.4775263 \times 10^{-36}$
C11	$5.1459413 \times 10^{-26}$
C12	$5.0526927 \times 10^{-15}$
C13	$1.4363539 \times 10^{-15}$
C14	3.388102×10^{-17}
C15	7.975909×10^{-33}
C16	3.220338×10^{-48}
N12	1.870037×10^{-44}
N13	$1.8867367 \times 10^{-27}$
N14	$7.4333028 \times 10^{-16}$
N15	$8.8286659 \times 10^{-18}$
N16	$5.0689811 \times 10^{-33}$
N17	$4.0260916 \times 10^{-43}$
O13	$-2.2985158 \times 10^{-57}$
O14	$4.4728396 \times 10^{-42}$
O15	$9.8318916 \times 10^{-31}$
O16	$1.1445241 \times 10^{-18}$
O17	$7.7655634 \times 10^{-23}$
O18	$2.3977288 \times 10^{-21}$
O19	$1.2657035 \times 10^{-34}$

O20	1.2595121×10^{-47}
F17	9.2781508×10^{-35}
F18	1.5604306×10^{-23}
F19	1.2045349×10^{-24}
F20	7.6886515×10^{-37}
Ne18	6.6536403×10^{-48}
Ne19	2.4700365×10^{-40}
Ne20	1.3609845×10^{-26}
Ne21	1.1156534×10^{-28}
Ne22	1.9397522×10^{-30}
Ne23	-1.0908288×10^{-72}
Na20	1.6454833×10^{-62}
Na21	-4.1807117×10^{-76}
Na22	3.888509×10^{-35}
Na23	2.489405×10^{-36}
CNO	7.2762335×10^{-15}

Uncomment if you want to output results in external file

```
In[ ]:= (*SetDirectory[NotebookDirectory[]]
          alloue répertoire   répertoire de notebook
         Export["TableXMass.dat",TableXMass,"TSV"]*)
         exporte
```

```
In[ ]:= (*TableXMassT8={#,XI[#][tofa[a[10^8]]]}&/@Join[ShortNames,{"CNO"}]*)
                                                    joins
```

```
In[ ]:= (*SetDirectory[NotebookDirectory[]]
          alloue répertoire   répertoire de notebook
         Export["TableXMassT8.dat",TableXMassT8,"TSV"]*)
         exporte
```

Tools for Monte-Carlo on nuclear rates

We gather some tools for Monte-Carlo estimation of uncertainties from nuclear rates. Some Examples are provided in the Example folder.

Initialize Kernels (if parallelization this is called. It just distributes definitions)

In[ ]:=
```
InitializeKernels := (
   LaunchKernels[];
   Print["Number of Kernels ", $KernelCount];
   DistributeDefinitions[ReshapedTabulatedReactions,
    ListReactionsUpToChosenMass, LoadRates, DefineEquations,
    SystemEquationsHT, SystemEquationsMT, SystemEquationsLT,
    LoadPlasma, LoadNEVO, LoadDistortions, LoadWeakRates,
    LoadRatesFull, DY, DY18, DYOnlyPEN, LbarnTOp, LnTOp, dFDneu];
  );
```

We define a function which launches the Monte-Carlo on subKernels and collects the results

In[ ]:=
```
RunPRIMATMonteCarlo[number_] := Module[{res, time, timref, Abundances,
    mytabfunctions, sss, RandomVariables, CosmoParametersList},
   If[$Verbose,
    Print["Reaction file used is : ", ChosenReactionFile];
    (*Print[StringJoin@@(#<>" "&/@SafeImport[ChosenReactionFile][[1]])];*)

    (*Print[MyGrid[Join[{{"Reaction Number","Reaction Name",
          "Initial species","Final Species","Reference"}},
       PadLeftNumberRowFromZero[Drop[#,{4}]&/@
         (NiceDisplayReaction/@ListReactionsUpToChosenMass)]]]];*)
   ];

   If[number > 1,
    Print["Running a Monte-Carlo with ", number, " points."];];
   Off[CompiledFunction::cfta];
   mytabfunctions = If[$ParallelBool, ParallelTable, Table];
   If[$ParallelBool, InitializeKernels;
    ParallelEvaluate[$HistoryLength = 0;]];

   (* We always use the same seed so that we always use the same sequence
    of random number as advocated in [Cyburt et al. 2015].*)
   timref = AbsoluteTime[];
   res = mytabfunctions[
```

```
      $Seed := i;
      (* We use a different seed so that for each MC
       point we have a different sequence of reaction rates *)
      InitializeRandom[$Seed];
      (* We restart our random list from the beginning *)

      h2Ωb0 = Meanh2Ωb0 + If[$Randomh2Ωb, σh2Ωb0 NormalRealisation, 0];
                          |si

      τneutron =
       Meanτneutron + If[$Randomτneutron, στneutron NormalRealisation, 0];
                     |si

      CosmoParametersList = {h2Ωb0, τneutron};

      time = AbsoluteTiming[RunNumericalIntegrals][[1]];
             |durée absolue

      RandomVariables = Rest@ListReactionsUpToChosenMass[[All, 4]];
                        |reste                            |tout

      (*Share[];*)
        |partage

      Print["Iteration ", i, "  Memory usage = ",
     |imprime

       MemoryInUse[], " time = ", time, "  Kernel : ", $KernelID];
       |mémoire en utilisation                        |identifiant de noyau

      Abundances = YPeriodTime["LT"][tend];

      (*ClearSystemCache[];*)
        |efface mémoire cache de système

      If[$Verbose, Print[Abundances(*," ",RandomVariables*)]];
     |si            |imprime

      {Abundances, RandomVariables, CosmoParametersList}, {i, 1, number}];


   If[$ParallelBool, CloseKernels[]];
  |si                |ferme noyaux

   Share[];
  |partage

   ClearSystemCache[];
  |efface mémoire cache de système

   Print["Time in the MC for that cosmology was ", AbsoluteTime[] - timref];
  |imprime                                        |temps absolu

   h2Ωb0 = Meanh2Ωb0;

   τneutron = Meanτneutron;

   MC = res[[All, 1]];
            |tout

   RV = res[[All, 2]];
            |tout

   Cosmo = res[[All, 3]];
               |tout


   res];

RunPRIMAT := RunPRIMATMonteCarlo[1];
```

The results of RunPRIMAT are gathered in the files MC (abundances) RV (rates variations) Cosmo (List of cosmological parameters, limited to baryons abundances and neutron lifetime)

We define functions to dump the results of a Monte-Carlo in a file and also the converse to load the result so as to analyze and use it.

In[ ]:=
```
Clear[LoadMC, DumpMC]
```
efface

In[ ]:=
```
DumpMC[File_String] := (
    Print["Exporting ", "Results/MC" <> File <> ".dat"];
    Export["Results/MC" <> File <> ".dat", MC];

    Print["Exporting ", "Results/RV" <> File <> ".dat"];
    Export["Results/RV" <> File <> ".dat", RV];

    Print["Exporting ", "Results/Cosmo" <> File <> ".dat"];
    Export["Results/Cosmo" <> File <> ".dat", Cosmo];);

LoadMC[File_String] := (
    MCfile = "Results/MC" <> File <> ".dat";
    RVfile = "Results/RV" <> File <> ".dat";
    Cosmofile = "Results/Cosmo" <> File <> ".dat";
    MC = Import[MCfile];
    RV = Import[RVfile];
    Cosmo = Import[Cosmofile];
    TMC = Transpose[MC];
  );
```
chaîne de caractères imprime fichier exporte fichier imprime fichier exporte fichier imprime fichier exporte fichier chaîne de caractères fichier fichier fichier importe importe importe transpose

Whenever a Monte-Carlo is finished, or loaded, we can obtain the table of values for a given element, or a given reaction

In[ ]:=
```
ElementColumn[el_] := MC[[All, KeyVal[[el]]]];
                                  tout
ReactionColumn[el_] := RV[[All, KeyNuclearReaction[el]]];
                                  tout
h2Ωb0List := Cosmo[[All, 1]];
                        tout
τneutronList := Cosmo[[All, 2]];
                            tout
```

In[ ]:=
```
(* I think this function is never used ! *)
   unité imaginaire
ReSetCosmology := (
    Meanh2Ωb0 = Meanh2Ωb0Planck;
    NeutrinosGenerations = 3;
    Nrelat = 0;(* Not sure this is needed. *)
                    négation
  );
```