rxntoarb

v. 2.22+

Christian Biscombe

8 March 2019

`rxntoarb` (pronounced reaction-to-arb) converts a human-readable system of chemical reactions into a format suitable for use with *arb* finite volume solver. Reading from one or more input files (with the preferred extension .rxn), `rxntoarb` automatically creates

- rate constant definitions (performing unit conversions where necessary),
- reaction rate expressions,
- source terms,
- transport equations for each chemical species (easily customisable by means of a template file),
- magnitude estimates.

Other features include easy switching between ODE and PDE output and the ability to include or exclude reactions based on regular expression matches.

`rxntoarb` offers several benefits:

- the reaction network is represented using intuitive syntax that is easier and faster to read and write,
- reactions may be easily added or removed without the user having to keep track of the effects of these changes on source terms,
- the potential for error is greatly reduced by eliminating the need for manual unit conversions and manual entry of reaction rates and source terms.

`rxntoarb` requires Ruby 1.9.3 or newer. The current version of the template file (see § 2) is designed for *arb* 0.58. (Older versions of the template file are available going back to *arb* 0.56.)

Contents

1 Invocation and command-line options

`rxntoarb` is invoked as follows:

`rxntoarb [options_list] <rxn_file_list>`,

where `rxn_file_list` specifies the path to at least one .rxn file and the optional `options_list` may include any of the following:

`-d|--debug` Write copious debugging output to STDERR. (Intended mainly as a developer option.)

`-i|--interactive` Prompt for confirmation before overwriting an existing .arb output file.

`-l|--alias-labels` Use reaction aliases (§ 3.8) instead of internal reaction labelling scheme.

`-n|--none-centred` Activate ODE mode: generate ODEs (no spatial dependence) rather than PDEs. `rxntoarb` operates in PDE mode by default.

`-o|--outfile <output_file>` Use `output_file` as the .arb output file. This option is ignored if `rxn_file_list` contains more than one .rxn file.

`-t|--template <template_file>` Use `template_file` as the template file.

`-v|--version` Print version information.

As of *arb* 0.58, `rxntoarb` is invoked automatically by *arb* if an `INCLUDE` statement (or similar) in an *arb* input file references a file with the .rxn extension. This mechanism ensures that the .arb output file generated by `rxntoarb` is always up-to-date with the .rxn file. To enable options to be passed to `rxntoarb` via this mechanism, any of the above command-line options may also be set within the .rxn file itself (see § 3.1).

2 File structure

Using `rxntoarb` with *arb* involves the use of four files.

2.1 .rxn file

.rxn files are input files read by `rxntoarb`. They contain a list of chemical reactions together with kinetic data. The syntax used in .rxn files is defined in § 3. An example .rxn file, reproduced in Listing 1, may be found under `examples/reactions.rxn` in the `rxntoarb` root directory. Vim syntax highlighting for .rxn files is provided by the file `rxn.vim` included in the `rxntoarb` root directory (type `:help usr_06` in Vim for instructions on how to set this up).

Listing 1 An example .rxn file (examples/reactions.rxn)

```
! Blood plasma coagulation biochemistry
! Ref: Jordan and Chaikof (2011), Biophys. J. 101: 276-286
! File created by Christian Biscombe, 2017-08-19

# Fluid-phase reactions
volume_region domain
VIII@v {IIa@v}-> VIIIa@v; kcat=0.9 s-1, KM=0.20 uM
V@v {IIa@v}-> Va@v; kcat=0.23 s-1, KM=0.072 uM
VII@v {Xa@v}-> VIIa@v; kcat=15.2 s-1, KM=1.2 uM # CB corrected erroneous kcat in ref.
VII@v {IIa@v}-> VIIa@v; kcat=0.061 s-1, KM=2.7 uM
IIa@v + ATIII@v -> ; k=6.8d-3 uM-1 s-1
IXa@v + ATIII@v -> ; k=9.9d-6 uM-1 s-1 # CB corrected erroneous k in ref.
Xa@v + ATIII@v -> ; k=2.6d-3 uM-1 s-1 # reaction mistakenly omitted in ref.
Xa@v + TFPI@v <=> TFPI:Xa@v; ka=16 uM-1 s-1, kd=3.3d-4 s-1
Va@v {APC@v}-> ; kcat=0.58 s-1, KM=0.025 uM
  VIIIa@v {APC@v}->
V@v {Xa@v}-> Va@v; kcat=0.046 s-1, KM=0.01 uM

# Reactions occurring on both TF patch and TM patch
surface_regions TF_patch, TM_patch
II@v {Va:Xa@s}-> IIa@v; kcat=33 s-1, KM=0.21 uM
X@v {VIIIa:IXa@s}-> Xa@v; kcat=20 s-1, KM=0.16 uM
Va@v + Xa@v <=> Va:Xa@s; ka=100 uM-1 s-1, kd=0.01 s-1 # units inconsistent!
VIIIa@v + IXa@v <=> VIIIa:IXa@s; ka=100 uM-1 s-1, kd=0.01 s-1 # units inconsistent!

# Reactions occurring on TF patch only
surface_region TF_patch
X@v {TF:VIIa@s}-> Xa@v; kcat=1.2 s-1, KM=0.45 uM
IX@v {TF:VIIa@s}-> IXa@v; kcat=0.34 s-1, KM=0.17 uM
VIIa@v + TF@s <=> TF:VIIa@s; ka=100 uM-1 s-1, kd=0.06 s-1
TF:VIIa@s + TFPI:Xa@v <=> TF:VIIa:TFPI:Xa@s; ka=10 uM-1 s-1, kd=1.1d-3 s-1

# Reactions occurring on TM patch only
surface_region TM_patch
PC@v {IIa:TM@s}-> APC@v; kcat=5.58 s-1, KM=0.7 uM
IIa@v + TM@s <=> IIa:TM@s; ka=100 uM-1 s-1, kd=0.01 s-1

initial_species [II, V, VII, VIIa, VIII, IX, X, PC, ATIII, TFPI]@domain
initial_species TF@TF_patch, TM@TM_patch
```

2.2 Template file (`rxntoarbrc`)

Template files store *arb* code that needs to be included for each chemical species defined in a .rxn file. The extended (non-*arb*) syntax used in template files is defined in § 4. `rxntoarb` uses the following hierarchy to determine the path of the template file:

1. If `rxntoarb` is invoked with the `-t` flag, the specified template file is used.
2. If a file with the name `rxntoarbrc` exists within the working directory from which `rxntoarb` is called, it is used as the template file.
3. The default `rxntoarbrc` located in the `rxntoarb` root directory is used otherwise.

The default `rxntoarbrc` should provide an adequate starting point in most cases. However, the user may wish to alter the (simulation-specific) boundary conditions listed beneath the comment '`# set boundary conditions here`' or to define new per-species `OUTPUT` variables. In this case, the default template should be copied to the working directory and modified there.

2.3 .arb output file

The output of `rxntoarb` is an .arb output file, which contains *arb* code metaprogrammed from the contents of the .rxn file. The .arb output file will have the same basename as the .rxn file unless overridden by the `-o` flag.

2.4 Main .arb file

The .arb output file generated by `rxntoarb` contains *arb* code describing chemical species and the reactions occurring between them, but it is not sufficient to run a complete *arb* simulation. Additional code related to mesh setup, solver options, equations governing fluid flow, time-stepping, etc. should be included in a separate .arb file, referred to as the main .arb file. It is this main .arb file (and not the .arb output file) that should be run through *arb*.

The main .arb file should have the following characteristics to interface cleanly with the .arb output file:

- An `INCLUDE_LOCAL` statement to include the .arb output file must be present.
- Any variables used in the template file but not created by `rxntoarb` must be defined.
- Replacements for any flags (strings delimited by double angle brackets `<<>>`) used in the template file should be defined. The default `rxntoarbrc` contains the following flags:
  - `<<specieshighorderadvection>>`: whether to use high-order advection for chemical species; should be either `1` (true) or `0` (false; default). If false, low-order advection will be used.

- `<<zeroinitialvolumeconcentrations>>`: whether to set species concentrations to zero initially in volume regions; should be either `1` (true) or `0` (false; default). If false, species concentrations will be set equal to their boundary values (see below) throughout volume regions.
- `<<calculatediffusivities>>`: whether species diffusivities should be calculated using a correlation based on molecular weight; should be either `1` (true) or `0` (false; default). If true, molecular weights must be defined for all volume species (see below). If false, diffusivities must be defined for all volume species (see below).

- Species-specific physical parameters, currently only molecular weights or diffusivities, should be defined as `CONSTANT`s. Molecular weight variable names must follow the convention `<MW_speciesname>`. Diffusivity variable names must follow the convention `<D_speciesname>`. Molecular weights of complexes (two or more species bound to each other) need not be defined: they will be calculated automatically as the sum of the molecular weights of the components.

- Boundary concentrations of all species present initially should be defined as `CONSTANT`s *after* the .arb output file is included (because this file sets the boundary concentrations of all species to zero by default). For surface species, the boundary concentration corresponds to the initial concentration of that species. For volume species, the boundary concentration will generally be the concentration of that species on the boundary where it enters the volume. Boundary concentration variable names must follow the convention `<c_speciesname@region_0>` for species located in volume regions and `<s_speciesname@region_0>` for species located on surface regions.

An example (skeleton) main .arb file may be found under `examples/main.arb` in the `rxntoarb` root directory.

3 .rxn file syntax

3.1 `options` statement

The options listed in § 1 may also be set within a .rxn file using an `options` statement, which (if used) should appear near the top of the file. The syntax for setting options exactly mirrors the command-line invocation. For example, the following code activates ODE mode and sets the template file to `my_template_file`:

```
options -n -t my_template_file
```

Note that options set using an `options` statement take precedence over options set on the command line.

3.2 Comments

Comments begin with the # character and continue until the end of the line. Comments may appear anywhere in the .rxn file. Full-line comments (e.g. line 5 in Listing 1) are ignored by `rxntoarb`. Comments appearing after the parameter list of a reaction (e.g. line 9) are carried over into the .arb output file, where they appear after the `CONSTANT` expressions that define the relevant kinetic parameters. Such end-of-line comments are a good place to cite references from which parameter values have been taken.

3.3 Headers

Lines beginning with a bang (!) are called headers (e.g. lines 1–3). Headers are a type of comment designed primarily to store metadata about the reaction network (title, author, date, description etc.). Headers may appear anywhere in the .rxn file (but it is intended that they appear only at the top). All headers will be collected together as comments at the top of the .arb output file. A line showing the date and time at which the .arb output file was generated and the version of `rxntoarb` used will be appended to the header automatically.

3.4 Species and region names

Chemical species have the general form `speciesname@region`. The `speciesname` describes the chemical identity of the species; the `@region` specifies the arb region on/in which that species is present. (Note that `@` is not part of the region name, but rather acts to separate the species and region names.) Chemical species are declared by including them in a reaction (see § 3.6).

Species and region names may not contain any of the characters `<>"#&` or the character sequences `{{` or `}}` (restrictions inherited from *arb*). `rxntoarb` also requires that species and region names not contain any of the characters `,;@` or any of the substitution sequences or extended syntax used in template files (see § 4). Colons (`:`) within species names are intended to separate the components of a complex (e.g. `TFPI:Xa` in line 14 is a complex of `TFPI` and `Xa`), but this interpretation only comes into play in calculating the molecular weight of complexes. The use of *arb*-style angle bracket delimiters (`<>`) is necessary for species names only if they begin with digits (which would otherwise be interpreted as a stoichiometric coefficient), and necessary for region names only if they contain characters other than letters, digits, and underscores.

Two lazy region identifiers, `@s` and `@v`, are predefined (see § 3.5.1). These identifiers interact with `surface_region` and `volume_region` statements. As such, `s` and `v` should not be used as user-defined region names.

When `rxntoarb` is called with the `-n` flag, ODE mode is activated. In this mode, all species and reactions are assumed to have no spatial dependence and hence region names effectively become part of the species name.

It is permissible to omit `@region` identifiers throughout the .rxn file under two circumstances:

- No `surface_region` or `volume_region` statements appear. In this case all species and reactions will be assumed to have no spatial dependence, i.e. ODE mode will be assumed.
- Exactly one surface or volume region is specified by either a `surface_region` or `volume_region` statement. In this case all species and reactions will be assumed to be located on/in that region.

3.5 `surface_region` and `volume_region` statements

The `surface_region` and `volume_region` keywords (both optionally plural) are used to specify *arb* regions on/in which chemical species are present and reactions occur. The keywords should be followed by a comma-separated list of region names (conforming to the naming rules stipulated in § 3.4). If `rxntoarb` is called without the `-n` flag (PDE mode), surface regions will be assumed to be two-dimensional and volume regions will be assumed to be three-dimensional. If `rxntoarb` is called with the `-n` flag (ODE mode), then the dimensionality of regions is ignored.

`surface_region` and `volume_region` statements apply to all reactions subsequently defined in the .rxn file, unless and until overridden by another `surface_region` or `volume_region` statement.

3.5.1 Lazy region identifiers `@s` and `@v`

The lazy region identifiers `@s` and `@v` are shorthand references to the surface and volume regions specified by the previous `surface_region` and `volume_region` statements. For example, the `surface_regions` statement in line 20 specifies two surface regions, `TF_patch` and `TM_patch`. References to `@s` in lines 21–24 will therefore be interpreted by `rxntoarb` as references to both `TF_patch` and `TM_patch`, whilst references to `@v` will be interpreted as references to the volume region `domain` specified by the `volume_region` statement in line 6. In line 27, the new `surface_region` statement overrides the previous one so that references to `@s` in lines 28–31 will be interpreted as references to `TF_patch` only.

3.6 Reactions

Four types of reactions are recognised, as described below. Reactions may be entered in any order in the .rxn file, subject to the condition that the lazy region identifiers `@s` and `@v` will take their value(s) from the previous `surface_region` and `volume_region` statements. With the exception of Michaelis–Menten reactions (whose kinetics are governed by the Michaelis–Menten equation), all reaction rates are assumed to be described by mass action kinetics.

3.6.1 Irreversible reactions

Irreversible reactions take the form

```
reactant_list -> product_list.
```

`reactant_list` consists of one or more species, separated by spaces and the character `+` (the spaces are mandatory so that `+` can also be used in species names, e.g. for ions). Stoichiometric coefficients (if any) should immediately precede the species name, optionally separated from it by a space or the `.` or `*` characters. Non-integer stoichiometric coefficients are not supported. product_list is similar except that it may be empty, in which case the products of the reaction are discarded (not tracked in the *arb* simulation). For readability, spaces around the reaction arrow `->` are recommended (but not mandatory). The kinetic parameter `k` describing the rate of the reaction must either be specified or inherited (see § 3.7). Examples of irreversible reactions are shown in lines 11–13.

3.6.2 Reversible reactions

Reversible reactions take the form

```
reactant_list <=> product_list.
```

`product_list` must not be empty for a reversible reaction. The kinetic parameters `ka` and `kd` describing the rates of the forward and reverse reactions must either be specified or inherited (see § 3.7). Examples of reversible reactions are shown in lines 23–24.

3.6.3 Two-step reactions

Two-step reactions consist of a reversible reaction followed by an irreversible reaction:

```
reactant_list <=> intermediate_list -> product_list.
```

`intermediate_list` must not be empty but `product_list` may be empty (which implies that the products are not tracked in the *arb* simulation). The kinetic parameters `ka`, `kd`, and `k` describing the rates of the forward and reverse reactions in the first step and the irreversible reaction in the second step must either be specified or inherited (see § 3.7).

3.6.4 Michaelis–Menten reactions

Michaelis-Menten reactions take the form

```
substrate {enzyme}-> product_list.
```

Exactly one `substrate` and one `enzyme` must be specified. `product_list` may be empty (which implies that the products are not tracked in the *arb* simulation). The Michaelis–Menten parameters `KM` and `kcat` must either be specified or inherited (see § 3.7). Examples of Michaelis–Menten reactions are shown in lines 7–10.

3.7 Kinetic parameters

Kinetic parameters are entered as a comma-separated list on the same line as the reaction they describe, separated from the reaction itself by a semicolon (`;`). Kinetic parameters may be entered in any order. The following parameter names are recognised:

- `k`,
- `ka` (synonyms `kf` and `kon`),
- `kd` (synonyms `kr` and `koff`),
- `KM` (synonym `Km`),
- `kcat`.

Kinetic parameter values may be entered in two forms. In the first form (used throughout the example .rxn file), a numerical value together with appropriate physical units are specified. The numerical value and each individual unit should be separated by spaces. SI prefixes immediately precede the unit name. Exponents on units should immediately follow the unit name, optionally preceded by the `^` character. More information on unit entry is given in the accompanying documentation for the `convert_units` program, which provides a command-line interface to the `Units` module that accompanies `rxntoarb`. Where necessary, `rxntoarb` performs unit conversions so that parameter values in the .arb output file are expressed in SI base units. Unit consistency checking is also performed with warning messages issued if reaction rates have inconsistent or unexpected units. (Such warnings are produced by lines 23–24.)

Alternatively, kinetic parameter values may be specified as strings, delimited by either single or double quotes. Strings may contain arbitrary *arb* code. This feature is useful in conjunction with aliases (see § 3.8) or for expressing one kinetic parameter in terms of others. The downside is that unit consistency checking is not possible.

3.7.1 Inheritance

Sometimes the same kinetic parameters will apply to a group of closely related reactions. Kinetic parameters from one 'parent' reaction can be assigned to any number of 'child' reactions by placing the children directly below the parent and indenting them by at least one space. An example of inheritance is shown in lines 15–16: the reaction in line 16 inherits the kinetic parameters from line 15. The advantage of inheritance is that kinetic parameters for the group of reactions need only be entered in one place.

3.8 Aliases

Any non-indented reaction may be preceded by an alias, which runs from the beginning of the line until a colon (`:`) followed by a (mandatory) space is encountered. Aliases serve two purposes. Firstly, they may be used to provide descriptive names for reactions. If `rxntoarb` is invoked with the `-l` flag, these names will be used throughout the .arb output file in place of the automatic labels (derived from species names) otherwise generated by `rxntoarb`. Secondly, aliases provide a convenient way to refer to kinetic parameters defined elsewhere in the .rxn file. For example, lines 15–16 could alternatively be written

```
15 Va_inactivation: Va@v {APC@v}-> ; kcat=0.58 s-1, KM=0.025 uM
16 VIIIa@v {APC@v}-> ; kcat="<kcat_Va_inactivation>", KM="<KM_Va_inactivation>"
```

In this case, the values of `kcat` and `KM` specified in line 15 (alias `Va_inactivation`) have been assigned to the reaction in line 16.

3.9 `initial_species` statement

An `initial_species` statement is used to declare which species are present initially in the *arb* simulation. This information is used by `rxntoarb` to estimate the magnitudes of the concentrations of all species based on the concentrations of the precursor species from which they are derived.

The `initial_species` keyword should be followed by a comma-separated list of species together with their associated regions, as shown in line 39. If multiple species on/in the same region are present initially, the array notation shown in line 38 may be used. (Note that lines 38 and 39 have been separated only for aesthetic reasons; they could be combined into a single line.) Any species appearing in an `initial_species` statement but otherwise not appearing in the .rxn file will be ignored.

See § 2.4 for details on how boundary concentrations must be specified in the main .arb file.

3.10 Metaspecies

A 'metaspecies' is a group of entities that behaves like a single entity in a reaction. For instance, when coagulation factors X and Xa bind to phospholipid membranes composed of phosphatidylcholine and phosphatidylserine (70:30 mixture), they occupy 106 and 52 phospholipid head groups, respectively.[1] These binding reactions should *not* be written as

```
X@v + 106PL@s <=> X@s; ka=..., kd=... # INCORRECT!
Xa@v + 52PL@s <=> Xa@s; ka=..., kd=... # INCORRECT!
```

because under mass action kinetics, the rates of the reactions as written above would be proportional to the concentration of `PL@s` raised to the power of 106 and 52, respectively, which is not what is wanted. Instead, binding sites should be entered using the following square bracket notation to represent a metaspecies:

```
X@v + [106PL@s] <=> X@s; ka=..., kd=...
Xa@v + [52PL@s] <=> Xa@s; ka=..., kd=...
```

`rxntoarb` interprets the metaspecies as single entities so that the rates of these reactions are proportional to the concentrations of the respective binding sites (equal to the concentration of `PL@s` divided by 106 and 52, respectively). The source term for `PL@s` is handled sensibly too.

As metaspecies behave like any other chemical species, they may be preceded by a stoichiometric coefficient if necessary (e.g. `2[106PL@s]`).

[1] Hathcock et al. (2005), *Biochemistry* 44: 8187.

3.11 include_only and exclude statements

include_only and exclude are used to selectively include or exclude regions or reactions in the .arb output file. Both keywords must be followed either by (i) a whitespace-free text string for simple text matching or (ii) a valid Ruby regular expression for more complex matches. (Regular expressions are useful for specifying alternate match patterns, forcing the statement to respect word boundaries, or preventing end-of-line comments from being scanned. The Regexp option i, which enables case-insensitive matching, is the only Regexp option recognised by rxntoarb.) For the statement to operate on a region, the region name must be preceded by the @ character. For example, if it is desired to run the example in Listing 1 without the TM_patch region, then the following statement should be issued near the top of the .rxn file:

```
exclude /@TM_patch|APC/
```

This statement excludes all reactions occurring on the TM_patch as well as all fluid-phase reactions involving APC (which is not present initially and is only produced on the TM_patch).

include_only and exclude statements apply to all lines subsequently defined in the .rxn file, unless and until overridden by another include_only or exclude statement. End-of-line comments are scanned by default, effectively enabling the user to define their own keywords to select/deselect subsets of reactions. To prevent end-of-line comments from being scanned, use a regular expression beginning with ^[^#]*.

3.12 substitute statement

A substitute statement[2] performs code replacements. It is particularly useful when metaspecies appear in more than one reaction (although its use is not limited to this application). Carrying on the example from § 3.10, suppose it is decided that factors X and Xa should both bind to 79 phospholipid head groups (79 is the average of 106 and 52). To avoid repeating the constant 79 in multiple locations, a pointer S@s to the binding site metaspecies is defined via a substitute statement as follows:

```
substitute /\bS@s\b/ [79PL@s]
X@v + S@s <=> X@s; ka=..., kd=...
Xa@v + S@s <=> Xa@s; ka=..., kd=...
```

When the above code is parsed, all instances of S@s below the substitute statement will be replaced internally by [79PL@s]. (Including \b in the regular expression ensures that only complete matches are replaced; other species that happen to contain S@s—e.g. PS@s—will not be affected.)

In general, the substitute keyword takes two space-separated arguments. The first argument must either be (i) a whitespace-free text string for simple text matching or (ii) a valid Ruby regular expression for more complex matches. (The Regexp option i, which enables case-insensitive matching, is the only Regexp option recognised by rxntoarb.)

[2] The substitute statement extends and replaces the let statement that appeared in versions 2.7–2.17.

The second argument is taken to be any text following the first argument (excluding end-of-line comments starting with the # character). All subsequent instances of text matching the first argument will be replaced with the second argument.

In principle, `substitute` statements could be used to override any aspect of the syntax described in this manual. Judiciousness is strongly advised.

4 Template file syntax

`rxntoarb` template files (called `rxntoarbrc` by default) store *arb* code that needs to be included for each species. A range of substitutions are performed on the original template file to generate unique code for each species. Template files contain extended (non-*arb*) syntax that is handled by `rxntoarb` as detailed below.

4.1 Substitution sequences

`rxntoarb` performs substitutions on the following character sequences (all delimited by forward slashes), which must therefore not appear in any species or region names:

- `/c/` (concentration of the species),
- `/species/` (species name),
- `/region/` (region name),
- `/source_region/` (region name),
- `/CENTRING/` (`NONE|FACE|CELL` depending on species location),
- `/centring/` (`none|face|cell` depending on species location),
- `/units/` (`''|mol m-2|mol m-3` depending on species location),
- `/MW/` (molecular weight of the species),
- `/associatedfaces(region)/` (region consisting of faces `associatedwith` the region),
- `/associatedcells(region)/` (region consisting of cells `associatedwith` the region),
- `/domainof(region)/` (region consisting of cells in `domainof` the region).

Additionally, all *arb* statements of the form `ON <region>` are deleted in ODE mode.

4.2 if_rxn statement

if_rxn statements are used to conditionally include *arb* code depending on whether a species is located on a surface, in a volume, or has no location (ODE mode). if_rxn statements take the form

```
if_rxn(location=region_list){if_clause}{else_clause}.
```

The term in parentheses determines the scope of the if_rxn statement. `location` must be one of `none`, `surface`, or `volume`. The optional `region_list` limits the scope of the if_rxn statement to only the named regions, and should be a comma-separated list of region names (*arb*-style angle bracket delimiters are optional). The code contained in the mandatory `if_clause` is retained for all species defined within the scope of the if_rxn statement and deleted otherwise. The code contained in the optional `else_clause` is retained for all species outside the scope of the if_rxn statement and deleted otherwise. Both `if_clause` and `else_clause` may span multiple lines. Nested brace pairs within clauses are allowed but nested if_rxn statements are not permitted.

4.3 each blocks

each blocks are used to apply a particular boundary condition to multiple surface regions bounding a volume region, and hence must reside within if_rxn(volume) statements. each blocks take the Ruby-inspired form

```
[region_list].each { |loopvar| expression }.
```

The mandatory `region_list`, which specifies the volume-region-bounding surface regions for which the `expression` in the block will be retained, should be a comma-separated list of region names (*arb*-style angle bracket delimiters are optional). Each of these region names (stripped of angle brackets, if any) will be passed to the block, where they are accessed through the (user-named) variable `|loopvar|` (vertical bars mandatory). In `region_list`, the shorthand `*` may be used to represent all volume-region-bounding surface regions on which reactions are occurring.

each blocks may span multiple lines. Nested brace pairs within each blocks are allowed but nested each blocks are not permitted.

5 Future development ideas

No guarantees, just some ideas! Contact the author if there is a capability you think `rxntoarb` should have.

- Support (two-way) conversion between .rxn format and SBML and/or CellML. As of 25 October 2017:
  - It is very difficult to translate certain features of .rxn syntax into SBML (e.g. string parameters, aliases, metaspecies).

  - Converting from SBML to .rxn is also tricky because of the relatively unconstrained format of KineticLaw objects.
  - Installing the Ruby bindings for libSBML involves hacking with the (broken) libSBML makefile.

  For these reasons, support for SBML is unlikely in the near future unless a real need arises.

- Allow subsets of reactions to occur at different 'runlevels', e.g. to simulate sequential reaction steps. Would be handy if pre-equilibration steps could be run through *arb* automatically. (Currently this can be achieved by making use of `include_only` and writing a wrapper script.)
- LaTeX output of reactions. (Could defer to COPASI or SBML2LaTeX if SBML support implemented.)
- Graphical output of reaction network diagrams. (Could defer to COPASI or Cytoscape if SBML support implemented.)

6 Copyright and licence

`rxntoarb` source code and documentation © 2016–2019 Christian Biscombe.

`rxntoarb` is contributed to *arb* finite volume solver (in which copyright is held by Dalton Harvie) under the same licence terms as that project. At the time of writing, *arb* is released under the terms of the GNU General Public License (version 3) as published by the Free Software Foundation.