The cpsystems LaTeX package

https://github.com/jcoo092/LaTeX-cP-systems

James Cooper, University of Auckland

jcoo092@aucklanduni.ac.nz

v0.21, 2021/08/19

Abstract

A package to assist authors writing about cP systems with typesetting their papers. It comprises a handful of environments and macros that are intended to ease writing about cP systems, and just as importantly, reduce the frequency of errors in the presentation. It is recommended to all authors using LaTeX to write about cP systems. Even if you don't want to use it, looking at the implementation details may give you some ideas for your own style.

Contents

1 Introduction

This is a package to assist authors in writing papers on cP systems, a particular variant of P systems that was created by Dr. Radu Nicolescu, along with a number of collaborators, in the early 2010s. This document assumes you have a working knowledge of cP systems and how they should look when fully typeset. It makes no attempt to explain the theory of cP systems, nor why they are set out in the fashion that they are. If you would like further information on cP systems, please see [2], or [3] for an excellent (albeit somewhat old) introduction to P systems generally.

This package was originally created by James Cooper to help with typesetting a specific paper on cP systems (specifically one about modelling Belief Propagation in cP systems). The same commands had historically been copied from paper to paper, and across sections within papers, as most of them weren't even formed into proper LaTeX macros. This was, of course, extremely error prone, with formatting errors (and worse) sometimes making it into published articles. The commands in this package in many cases are no less verbose than simply typing out the commands inside the macros. They are more 'robust'[1], however, in that by using the defined macros the exact same commands are applied each time so there is greater consistency throughout the paper.[2] If the macro is mistyped, the LaTeX engine itself will report the error. They also hopefully should add greater structure to a paper and prove useful in editing the paper.

Note that, at present, this package is *not* available on CTAN, and instead may only be retrieved from the author's academic GitHub account (see the URL listed in this document's title). Once it has been sufficiently developed and stabilised, and/or there is consistent demand for such a package beyond the author's immediate research group, it likely will be added to CTAN, but there is no timeframe nor guarantee in place for that.

Also be aware that some of the commands included in the package aren't currently documented outside of section 3. The commands exist and are used, but I haven't had the spare time to write about them properly yet.

2 Usage

This section firstly gives a brief overview of the individual usage of each command, and then supplies a few examples showing their combined use. For the full examples, the verbatim LaTeX code needed to typeset the corresponding example in each instance is bounded at the top and bottom by horizontal lines, to try to make it more clear where an example starts and ends. It is the case, however, that sometimes floating environments interrupt them.

[1]Note that LaTeX has its own, different, concept of "robust".

[2]It is also hoped, perhaps vainly, that this package could eventually become *the* standard way to set out cP systems, assuring consistency across different papers by different authors.

2.1 Environments

2.1.1 Floats

`cprulesetfloat` Container for a `cpruleset`. The `cprulesetfloat` wraps a `cpruleset` and provides it with the capability to float in the document, in much the same fashion as image and tables typically do. It also provides the ability to caption the environment, and provide it with a label for cross-referencing purposes. These latter are done in the exact same way as with an `\includegraphics{}` environment from the `graphicx` package. Furthermore, it works with the `hyperref` package's `\autoref{}` command, supplying the environment's number, and the name 'Ruleset'.

`cpobjectsfloat` Container for a `cpobjects`. The `cpobjectsfloat` wraps a `cpobjects` and provides it with the capability to float in the document, in much the same fashion as image and tables typically do. It also provides the ability to caption the environment, and provide it with a label for cross-referencing purposes. These latter are done in the exact same way as with an `\includegraphics{}` environment from the `graphicx` package. Furthermore, it works with the `hyperref` package's `\autoref{}` command, supplying the environment's number, and the name 'Objects Group'.

2.1.2 Maths environment floats

`cpruleset` Goes inside a `cprulesetfloat`. This provides the surrounding environment that `cprule`s, `cppromoter`s and `cpinhibitor`s are written inside. It ensures that the appropriate maths mode is active inside itself, as well as providing the array structure that the rules are set inside and drawing a box around the whole thing. Note that the text and box can become misaligned if they bump into the edge of a page. It seems to be common for the box to break over the two pages and stay inside the usual margins, but the text just carries on as it pleases, going right outside the margins. For this reason, as well as the aforementioned benefits, it is strongly recommended that you always place a `cpruleset` inside a `cprulesetfloat`.

`cpobjects` Goes inside a `cpobjectsfloat`. This provides the surrounding environment that `cpobjectline`s are written inside. It ensures that the appropriate maths mode is active inside itself, as well as drawing a box around the whole thing. Note that the text and box can become misaligned if they bump into the edge of a page. It seems to be common for the box to break over the two pages and stay inside the usual margins, but the text just carries on as it pleases, going right outside the margins. For this reason, as well as the aforementioned benefits, it is strongly recommended that you always place a `cpobjects` inside a `cpobjectsfloat`.

2.2 Macros

Brief descriptions of the rules and their use are provided here for reference purposes, but it is recommended that you take a look at subsection 2.3 for examples on how to use them properly.

2.2.1 Rules

`\cprule` `\cprule` {⟨*Starting state*⟩} {⟨*Input objects*⟩} {⟨*Mode of operation*⟩} {⟨*Ending state*⟩} {⟨*Output objects*⟩}

Goes inside a `cpruleset` environment. This is used to write out individual rules, and takes five mandatory arguments. They are, in order, the starting state for the rule; the set of objects that are matched on and consumed by the rule; the parallelism mode specifier (currently + and 1 are standard for maximum and minimum parallelism); the ending state for the rule; the objects that are output at the end of the rule.

E.g. a rule to move a single a out of a b functor might be written like:

```
\cprule{s_1}{b(aa)}{1}{s_1}{b(a)~a}
```

`\cpruleinline` `\cprule` {⟨*Contained* `cprule*`⟩}

Substitute for the `cpruleset` environment when the full environment is unwanted, and instead one simply wishes to include a single rule in the normal text. This macro takes a single input, a `cprule*` macro, which specifies the actual rule to be typeset. Despite its name, this macro still sets the rule out in its own mini-paragraph because it uses the standalone mathematics delimiters internally.

E.g. a rule to move a single a out of a b functor might be written inline in the text like:

```
\cpruleinline{\cprule*{s_1}{b(aa)}{1}{s_1}{b(a)~a}}
```

`\cprule*` `\cprule` {⟨*Starting state*⟩} {⟨*Input objects*⟩} {⟨*Mode of operation*⟩} {⟨*Ending state*⟩} {⟨*Output objects*⟩}

Goes inside a `cpruleset` environment. Identical to a `\cprule`, except that this version neither shows a rule number, nor increments the rule counter.

`\cppromoter` `\cppromoter` {⟨*The promoting cP systems object*⟩}

Goes immediately beneath a `cprule`, `cpinhibitor` or another `cppromoter`. This command takes a single argument, which is the term(s) to be written out as a promoter for a rule.

`\cpinhibitor` `\cpinhibitor` {⟨*The inhibiting cP systems object*⟩}

Goes immediately beneath a `cprule`, `cpinhibitor` or another `cppromoter`. This command takes a single argument, which is the term(s) to be written out as a inhibitor for a rule.

`\cpsend` `\cpsend` {⟨*The object(s) to be sent*⟩} {⟨*The name of the channel over which to send the object(s)*⟩}

Goes inside a `cprule`. This is as a convenience for writing out parts of rules where one or more objects are sent over a channel. This macro abstracts over the slightly fiddly details of writing it out. This command takes two arguments. The first is the object(s) to be sent, and the second is the name of the channel (as it appears to the current top-level cell) the object(s) are to be sent over.

`\cprecv` `\cprecv` {⟨*The pattern of the object(s) to be received*⟩} {⟨*The name of the channel over which to receive the object(s)*⟩}

Goes inside a `cprule`. This is as a convenience for writing out parts of rules where one or more objects are received over a channel. This macro abstracts over the slightly fiddly details of writing it out. This command takes two arguments.

The first is the object(s) to be received, and the second is the name of the channel (as it appears to the current top-level cell) the object(s) are to be received over.

Note, of course, that if an object is sent to the current top-level either using a non-existent channel or a pattern that is not specified by any receiving rule in the system, that object will not be retrieved from the channel by the current top-level cell.

`\changerulenum` `\changerulenum {`⟨*The number to which you would like to set the rule counter*⟩`}`

Set the rules counter to whatever number you prefer.

`\resetrulenum` `\resetrulenum`

Resets the rules counter back to zero, which means that the next time you use a `\cprule`, that rule will receive the number 1.

2.2.2 Objects

`\cpobjectsline` `\cpobjectsline {`⟨*cP systems objects to be presented as contained within that particular top-level cell*⟩`}`

Goes inside a `cpobjects` environment. Used to set out a full line of inert objects that will be inside a top-level cell. Note that, despite what one may initially assume, there is *no* included ability (currently) for this to re-flow objects across lines. It is up to the author to break up their listing of objects appropriately into separate lines.

2.2.3 Miscellaneous

`\cpfunc` `\cpfunc {`⟨*Outer functor*⟩`} {`⟨*Contents of functor*⟩`}`

Typically used inside a `cprule`, though so long as it is used inside a maths mode environment of some sort (e.g. `\(\_\)` or`\ensuremath{}`) it should still work. This command is used to write out a functor and its contents, where the first argument is the name of the containing functor and the second argument is everything that is contained inside the functor. They can, of course, be nested – in fact, this is encouraged. The implementation of `cpfunc` *should* ensure that the brackets will scale appropriately automatically, so that outer functors have larger brackets than inner ones.

Line breaking is currently, er, broken in this implementation. It didn't play well with the auto-growing brackets. `\\` is completely unusable inside a `cpfunc`. `\linebreak` and `\newline` can be included, but seem to do absolutely nothing. If you have any `&`s for marking out columns, they seem to cause compilation errors and should be removed. It is unclear at the moment how to restore this functionality, but this issue is why the examples below all spill well past the right-hand-side of their frames.

`\cpfuncms` `\cpfuncms {`⟨*Outer functor*⟩`} {`⟨*Contents of functor*⟩`}`

Exactly the same as a regular `cpfunc` except this one is for functors subject to micro-surgeries, and so uses curly braces instead of parentheses. It uses curly braces because currently that is the correct style for typesetting micro-surgeries. That is potentially subject to change in the future, so you should probably opt to use this macro instead of doing it by hand, to save yourself some hassle if/when the style does change. Moreover, using this macro means that you get auto-growing brackets with micro-surgery functors as well as regular ones.

`\cptermdef` This command is used to explain the meaning of a given cP systems term/symbol. E.g. if you have a functor such as $a(b)$, you might use `\cptermdef` entries to explain the meanings of both the a functor and b atom. Note that the current implementation of this command automatically includes a full stop/period at the end of the description – so if you include one inside the description, you will get a double dot. It is generally recommended to encapsulate these definitions inside a `description` environment, which ends up with everything set out pretty well. Further, separating out the atoms from the functors from the states is also recommended. So far, using a `paragraph` command to give the relevant title to each one, followed by a `description` with the `cptermdef`s inside that seems to work out pretty well.

`\cpundig` `\cpundig` A parameter-less convenience macro for inserting the correctly-formatted cP systems 'unitary digit' in rules.

`\cpempty` `\cpempty` A parameter-less convenience macro for inserting the correctly-formatted empty functor symbol (which is actually just a `\lambda`). Mostly used so that intent is clear in the rules specifications, but also partly in case someone ends up changing how empty functors are specified.

2.3 Full examples

2.3.1 A floating cP systems ruleset

For example, to typeset the ruleset for the solution to the Travelling Salesman Problem in [1], depicted here in Ruleset 1, the following was used:[3]

Code to produce Ruleset 1

```
\begin{cprulesetfloat}
  \begin{cpruleset}
    \cprule{s_1}{\cpfunc{v}{v(R)Y}}{1}{s_2}{\cpfunc{s}{r(R)~u(Y)~
    \cpfunc{p}{h(R)p()}}~c(\lambda)}

    \cprule{s_2}{\cpfunc{s}{r(R)~u()~\cpfunc{p}{h(F)p(P)}~c(C)}}
    {+}{s_3}{\cpfunc{z}{\cpfunc{p}{h(R) \cpfunc{p}{h(F)p(P)}}}~c(W)}
    \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

   \cprule{s_2}{}{+}{s_2}{\cpfunc{s}{r(R)~u(Z)~
   \cpfunc{p}{h(T) \cpfunc{p}{h(F) p(P)}}~c(CW)}}
   \cppromoter{\cpfunc{s}{r(R)~\cpfunc{u}{v(T)Z}~
   \cpfunc{p}{h(F) p(P)}~c(C)}}
   \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

   \cprule{s_2}{s(\_)}{+}{s_2}{}
```

[3]The reference to the given label was in turn created using `\autoref{rules:TSP}`, with `autoref` coming from the `hyperref` package.

```
    \cprule{s_3}{}{1}{s_4}{p'(P) \quad c'(1D)}
    \cppromoter{\cpfunc{z}{p(P~c(1D)}}
    \cpinhibitor{(D = CW)~\cpfunc{z}{p(\_)~c(C)}}

    \end{cpruleset}
    \caption{\label{rules:TSP} The five rules from \cite{Cooper2019},
     updated to the latest (at the time of writing) style for cP~systems.}
\end{cprulesetfloat}
```

$$
\begin{array}{llclll}
s_1 & v(v(R)Y) & \rightarrow_1 & s_2 & s\big(r(R)\ u(Y)\ p(h(R)p())\big)\ c(\lambda) & (1) \\
s_2 & s\big(r(R)\ u()\ p(h(F)p(P))\ c(C)\big) & \rightarrow_+ & s_3 & z\Big(p\big(h(R)p(h(F)p(P))\big)\Big)\ c(W) & (2) \\
 & & & & \quad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & & \rightarrow_+ & s_2 & s\Big(r(R)\ u(Z)\ p\big(h(T)p(h(F)p(P))\big)\ c(CW)\Big) & (3) \\
 & & & & \quad \mid s\big(r(R)\ u(v(T)Z)\ p(h(F)p(P))\ c(C)\big) & \\
 & & & & \quad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & s(\_) & \rightarrow_+ & s_2 & & (4) \\
s_3 & & \rightarrow_1 & s_4 & p'(P) \quad c'(1D) & (5) \\
 & & & & \quad \mid z(p(P\ c(1D)) & \\
 & & & & \quad \neg\ (D = CW)\ z(p(\_)\ c(C)) &
\end{array}
$$

Ruleset 1: The five rules from [1], updated to the latest (at the time of writing) style for cP systems.

At the present point in time, the paragraph following this one is a total lie. It *used* to be the case that everything was re-arranged to make it all fit nicely in the box, but since introducing the auto-growing brackets, this hasn't worked. For some reason I have yet to work out, `\cpfunc` and similar friends hit a compilation error when there is an attempt at a newline inside them. Given that by far the longest parts of some rules are one big long `\cpfunc`, this prevents adjusting things properly. It probably has something to do with the perfectcut package that I use to get the auto-growing brackets, but I haven't figured out the problem yet.

There is, however, an extremely obvious problem with Ruleset 1 – much of it extends beyond the box. That is partly a product of the narrow margins for the main text body of the current document class, but with rules of any real length this is an inevitability really, if the entire thing is set onto a single line. The solution, therefore, is to split the wider parts over multiple lines, thereby permitting the array to narrow each individual field, and thus fit everything inside the box. Unfortunately, at present there is no *good* way to do this. The way to achieve it is simply to include a line break (i.e. \\) and then an appropriate number of ampersands (&) to bring the array back into alignment, as demonstrated in Ruleset 2. It is important (and sometimes slightly tricky) to get the number of ampersands exactly correct. The wrong number can either lead to errors from the LaTeX compiler and/or part of a rule appearing in a different column to the intended one.

Code to produce Ruleset 2
```
\begin{cprulesetfloat}
  \begin{cpruleset}
    \cprule[rule:1]{s_1}{\cpfunc{v}{v(R)Y}}{1}{s_2}{\cpfunc{s}{r(R)~u(Y)~
    \cpfunc{p}{h(R)p()}} &\\&&&& c(\lambda)}

    \cprule{s_2}{\cpfunc{s}{r(R)~u() \newline \cpfunc{p}{h(F)p(P)} \newline c(C)}}
    {+}{s_3}{\cpfunc{z}{\cpfunc{p}{h(R) \cpfunc{p}{h(F)p(P)}}}~c(W)}
    \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

   \cprule[rule:2]{s_2}{}{+}{s_2}{\cpfunc{s}{r(R)~u(Z) \newline
    \cpfunc{p}{h(T) \cpfunc{p}{h(F) p(P)}}
   \newline  c(CW)}}
   \cppromoter{\cpfunc{s}{r(R)~\cpfunc{u}{v(T)Z}~
   \cpfunc{p}{h(F) p(P)}~c(C)}}
   \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

   \cprule{s_2}{s(\_)}{+}{s_2}{}

   \cprule{s_3}{}{1}{s_4}{p'(P) \quad c'(1D)}
   \cppromoter{\cpfunc{z}{p(P~c(1D)}}
   \cpinhibitor{(D = CW)~\cpfunc{z}{p(\_)~c(C)}}

  \end{cpruleset}
  \caption{\label{rules:TSP2} The five rules from \cite{Cooper2019},
   updated to the latest (at the time of writing) style for cP~systems
    -- rewritten to make it fit inside the box.}
\end{cprulesetfloat}
```

Notice that the numbering of the rules in Ruleset 2 *does not* carry on from those in the previous `cprulesetfloat`, Ruleset 1. This is different from the behaviour before version 0.19, where the rules did continue on by default. Generally speaking, this is the desired behaviour. If you do want rules to carry on from earlier, try out the `cprulesetcontnum` environment. Let's give it a looksee now. Check out Ruleset 3.

Code to produce Ruleset 3
```
\begin{cprulesetfloat}
  \begin{cprulesetcontnum}
    \cprule{s_1}{\cpfunc{v}{v(R)Y}}{1}{s_2}{\cpfunc{s}{r(R)~u(Y)~
```

$$
\begin{array}{llclll}
s_1 & v\big(v(R)Y\big) & \rightarrow_1 & s_2 & s\big(r(R)\ u(Y)\ p(h(R)p())\big) & \\
 & & & & c(\lambda) & (1) \\
s_2 & s\big(r(R)\ u()p(h(F)p(P))c(C)\big) & \rightarrow_+ & s_3 & z\Big(p\big(h(R)p(h(F)p(P))\big)\Big)\ c(W) & (2) \\
 & & & & \quad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & & \rightarrow_+ & s_2 & s\Big(r(R)\ u(Z)p\big(h(T)p(h(F)p(P))\big)c(CW)\Big) & (3) \\
 & & & & \quad \mid s\big(r(R)\ u(v(T)Z)\ p(h(F)p(P))\ c(C)\big) & \\
 & & & & \quad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & s(\_) & \rightarrow_+ & s_2 & & (4) \\
s_3 & & \rightarrow_1 & s_4 & p'(P) \quad c'(1D) & (5) \\
 & & & & \quad \mid z(p(P\ c(1D))) & \\
 & & & & \quad \neg\ (D = CW)\ z(p(\_)\ c(C)) &
\end{array}
$$

Ruleset 2: The five rules from [1], updated to the latest (at the time of writing) style for cP systems – rewritten to make it fit inside the box.

```
     \cpfunc{p}{h(R)p()}}~c(\lambda)}

     \cprule[rule:3]{s_2}{\cpfunc{s}{r(R)~u() \newline \cpfunc{p}{h(F)p(P)} \newline c(C
     {+}{s_3}{\cpfunc{z}{\cpfunc{p}{h(R) \cpfunc{p}{h(F)p(P)}}}~c(W)}
     \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

    \cprule{s_2}{}{+}{s_2}{\cpfunc{s}{r(R)~u(Z) \newline
     \cpfunc{p}{h(T) \cpfunc{p}{h(F) p(P)}}
    \newline  c(CW)}}
    \cppromoter{\cpfunc{s}{r(R)~\cpfunc{u}{v(T)Z}~
    \cpfunc{p}{h(F) p(P)}~c(C)}}
    \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

    \cprule{s_2}{s(\_)}{+}{s_2}{}

    \cprule[rule:4]{s_3}{}{1}{s_4}{p'(P) \quad c'(1D)}
    \cppromoter{\cpfunc{z}{p(P~c(1D)}}
    \cpinhibitor{(D = CW)~\cpfunc{z}{p(\_)~c(C)}}

   \end{cprulesetcontnum}
   \caption{\label{rules:TSP3} The five rules from \cite{Cooper2019},
    updated to the latest (at the time of writing) style for cP~systems
     -- rewritten to make it fit inside the box.}
 \end{cprulesetfloat}
```

$$
\begin{array}{llllr}
s_1 & v(v(R)Y) & \rightarrow_1 & s_2 \;\; s\big(r(R)\ u(Y)\ p(h(R)p())\big)\ c(\lambda) & (6) \\
s_2 & s\big(r(R)\ u()p(h(F)p(P))c(C)\big) & \rightarrow_+ & s_3 \;\; z\Big(p\big(h(R)p(h(F)p(P))\big)\Big)\ c(W) & (7) \\
 & & & \qquad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & & \rightarrow_+ & s_2 \;\; s\Big(r(R)\ u(Z)p\big(h(T)p(h(F)p(P))\big)c(CW)\Big) & (8) \\
 & & & \qquad \mid s\big(r(R)\ u(v(T)Z)\ p(h(F)p(P))\ c(C)\big) & \\
 & & & \qquad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & s(\_) & \rightarrow_+ & s_2 & (9) \\
s_3 & & \rightarrow_1 & s_4 \;\; p'(P) \quad c'(1D) & (10) \\
 & & & \qquad \mid z(p(P\ c(1D)) & \\
 & & & \qquad \neg\ (D = CW)\ z(p(\_)\ c(C)) &
\end{array}
$$

Ruleset 3: The five rules from [1], updated to the latest (at the time of writing) style for cP systems – rewritten to make it fit inside the box.

2.3.2 Rules cross-references

If you were paying close attention, you probably would have noticed the optional parameters to some rules in the previous two rulesets. Something along the lines of `[rule:1]`. These optional arguments are used to give labels to rules in rulesets, if desired — a label is only generated if the optional argument is used (and is non-empty). To use them, you need to use the custom `\cpruleref` command, with the label name, e.g. `\cpruleref{rule:2}` produces the text "rule 3". Currently, there's nothing more to it than that. You could also use `\cpruleref*{rule:2}` to get "3", i.e., the number but not the word rule.

This page has been left blank intentionally (excepting any floats that may end up here). You'll see why in a moment.

2.3.3 A sinking ruleset

If, for some reason, you *don't* want your ruleset floating, or to have a caption or label, you dont actually need to use the `cprulesetfloat` environment. `cpruleset` is all you need for laying out rules. Just, the results probably won't be as good:

Code to produce the unlabelled non-floating cpruleset

```
\begin{cpruleset}
 \cprule{s_1}{\cpfunc{v}{v(R)Y}}{1}{s_2}{\cpfunc{s}{r(R)~u(Y)~
 \cpfunc{p}{h(R)p()}}~c(\lambda)}

 \cprule{s_2}{\cpfunc{s}{r(R)~u() \newline \cpfunc{p}{h(F)p(P)} \newline c(C)}}
 {+}{s_3}{\cpfunc{z}{\cpfunc{p}{h(R) \cpfunc{p}{h(F)p(P)}}}~c(W)}
 \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

 \cprule{s_2}{}{+}{s_2}{\cpfunc{s}{r(R)~u(Z) \newline
  \cpfunc{p}{h(T) \cpfunc{p}{h(F) p(P)}}
 \newline c(CW)}}
 \cppromoter{\cpfunc{s}{r(R)~\cpfunc{u}{v(T)Z}~
 \cpfunc{p}{h(F) p(P)}~c(C)}}
 \cppromoter{\cpfunc{e}{f(F)~t(T)~c(W)}}

 \cprule{s_2}{s(\_)}{+}{s_2}{}

 \cprule{s_3}{}{1}{s_4}{p'(P) \quad c'(1D)}
 \cppromoter{\cpfunc{z}{p(P~c(1D)}}
 \cpinhibitor{(D = CW)~\cpfunc{z}{p(\_)~c(C)}}

\end{cpruleset}
```

$$
\begin{array}{llllll}
s_1 & v(v(R)Y) & \rightarrow_1 & s_2 & s\big(r(R)\ u(Y)\ p(h(R)p())\big)\ c(\lambda) & (1)\\
s_2 & s\big(r(R)\ u()p(h(F)p(P))c(C)\big) & \rightarrow_+ & s_3 & z\Big(p\big(h(R)p(h(F)p(P))\big)\Big)\ c(W) & (2)\\
 & & & & \quad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & & \rightarrow_+ & s_2 & s\Big(r(R)\ u(Z)p\big(h(T)p(h(F)p(P))\big)c(CW)\Big) & (3)\\
 & & & & \quad \mid s\big(r(R)\ u(v(T)Z)\ p(h(F)p(P))\ c(C)\big) & \\
 & & & & \quad \mid e(f(F)\ t(T)\ c(W)) & \\
s_2 & s(\_) & \rightarrow_+ & s_2 & & (4)\\
s_3 & & \rightarrow_1 & s_4 & p'(P) \quad c'(1D) & (5)\\
 & & & & \quad \mid z(p(P\ c(1D)) & \\
 & & & & \quad \neg\ (D = CW)\ z(p(\_)\ c(C)) &
\end{array}
$$

This has been carefully constructed to reveal another problem with this approach. Without the floating environment, the ruleset is printed essentially exactly where it was declared, with the box of the `cpruleset` mostly printing on one page, but breaking and starting again on the next, but the text contained inside the environment is shifted down, overflowing the normal bottom margin for the main text. I.e. the two become out-of-sync. In this instance, whitespace was played around with to ensure this would happen (which is why an earlier page was left blank intentionally), but it happened by chance in an earlier draft of this documentation.

2.3.4 An inline cprule

The `cpruleinline` macro is included to typeset a rule inside the main text without needing to create a full cpruleset environment. The macro takes one parameter, a `cprule*` macro (you could probably use a regular `cprule` macro, but I'm not sure what will happen, except that the rule will probably get a number you could do without). E.g.

Code to produce an inline rule

```
This is some text before the inline
rule \cpruleinline{\cprule*{s_i}{LHS}{+/1}{RHS}{s_j}} and
this is some text after it.
```

This is some text before the inline rule

$$s_i \quad LHS \quad \rightarrow_{+/1} \quad RHS \quad s_j$$

and this is some text after it.

2.3.5 Sending and receiving

Examples are yet to be included.

2.3.6 Listings of objects

Listing the objects contained within a top-level cP systems cell is a fairly common activity, especially when writing out examples of the operation of usage for a given system. Thus, this package includes some assistance for writing those out, also. The style is derived from that used in other recent papers.

The procedure is much the same as for a `cpruleset`, in that a `cpobjects` is encapsulated inside a `cpobjectsfloat`. The former creates the environment and box for the objects to be typeset in, while the latter wraps that in a floating environment that provides floating, captioning and cross-referencing capabilities. Again borrowing an example from [1] (specifically Figure 7), Objects Group 1 is implemented as:

Code to produce Objects Group 1

```
\begin{cpobjectsfloat}
  \begin{cpobjects}
    \cpobjectsline{\cpfunc{e}{f(1)~t(2)~w(1)}
    \quad \cpfunc{e}{f(1)~t(3)~w(3)}
    \quad \cpfunc{e}{f(1)~t(5)~w(2)}
    \quad \cpfunc{e}{f(2)~t(1)~w(1)}}
    \cpobjectsline{\cpfunc{e}{f(2)~t(4)~w(6)}
    \quad \cpfunc{e}{f(2)~t(5)~w(4)}
    \quad \cpfunc{e}{f(3)~t(1)~w(3)}
    \quad \cpfunc{e}{f(3)~t(4)~w(8)}}
    \cpobjectsline{\cpfunc{e}{f(3)~t(5)~w(5)}
    \quad \cpfunc{e}{f(4)~t(2)~w(6)}
    \quad \cpfunc{e}{f(4)~t(3)~w(8)}
    \quad \cpfunc{e}{f(4)~t(5)~w(7)}}
    \cpobjectsline{\cpfunc{e}{f(5)~t(1)~w(2)}
    \quad \cpfunc{e}{f(5)~t(2)~w(4)}
    \quad \cpfunc{e}{f(5)~t(3)~w(5)}
    \quad \cpfunc{e}{f(5)~t(4)~w(7)}}
    \cpobjectsline{\cpfunc{v}{v(1)\,v(2)\,v(3)\,v(4)\,v(5)}}
  \end{cpobjects}
  \caption{\label{obj:fig7} The first example of an objects group}
\end{cpobjectsfloat}
```

$$e(f(1)~t(2)~w(1)) \quad e(f(1)~t(3)~w(3)) \quad e(f(1)~t(5)~w(2)) \quad e(f(2)~t(1)~w(1))$$

$$e(f(2)~t(4)~w(6)) \quad e(f(2)~t(5)~w(4)) \quad e(f(3)~t(1)~w(3)) \quad e(f(3)~t(4)~w(8))$$

$$e(f(3)~t(5)~w(5)) \quad e(f(4)~t(2)~w(6)) \quad e(f(4)~t(3)~w(8)) \quad e(f(4)~t(5)~w(7))$$

$$e(f(5)~t(1)~w(2)) \quad e(f(5)~t(2)~w(4)) \quad e(f(5)~t(3)~w(5)) \quad e(f(5)~t(4)~w(7))$$

$$v(v(1)\,v(2)\,v(3)\,v(4)\,v(5))$$

Objects Group 1: The first example of an objects group

It is unclear why the first line in the Objects Group 1 appears to have a wider space between it and the next line as compared to the following lines – this has not been observed elsewhere and I can't spot any notable difference between the code for it or any other `cpobjects` environment. This can be overcome by including a blank `\cpobjectsline{}` at the top of the `cpobjects` environment, but that then leaves a larger gap between the top of the box and the start of the objects. An explanation and correction of this would be very welcome.

2.3.7 cP terms

This example demonstrates the use of the `cptermdef` command for stating definitions of the different cP systems terms used in a paper.

It is strongly recommended that you use a `description` environment in which each of the `cptermdef` commands are encapsulated. Further, heading up the collection with a `paragraph` command for providing a heading seems to be a good idea. Such an example is shown below, 'explaining' some of the symbols that appear in Ruleset 2:

Code to produce a brief example of the use of cP terms

```
\paragraph{cP systems terms used}
\begin{description}
    \cptermdef{v}{A functor containing very vacuous varieties of vegetables}
    \cptermdef{s}{A functor containing simply sublime serpentine sausages}
    \cptermdef{f}{A fiendish \& frightening flash of the fraught folly of including you
    \cptermdef{a}{a is for \texttt{atom}.  To quote Radioactive Man:  Up and atom!}
    \cptermdef{s_1}{The starting state, wherein one states the obvious, subtly}
\end{description}
```

cP systems terms used

v A functor containing very vacuous varieties of vegetables

s A functor containing simply sublime serpentine sausages

f A fiendish & frightening flash of the fraught folly of including your own full stop. Except, the behaviour changed and now `cptermdef` no longer puts a '.' at the end of every term's description, since there were occassions when it would be a bad thing.™

a a is for `atom`. To quote Radioactive Man: Up and atom!

s_1 The starting state, wherein one states the obvious, subtly

3 Implementation

This section presents the actual implementation of the package. For the most part you probably won't need to refer to it, but every so often you might, especially to work out some error that LaTeX is throwing at you, based on what the commands defined within become once they have been substituted into your document.

3.1 Preamble

```
\RequirePackage{amsmath}
\RequirePackage{array}
\RequirePackage{etoolbox}
```

```
\RequirePackage{framed}
\RequirePackage{changepage}
\RequirePackage{trimspaces}
\RequirePackage{newfloat}
\RequirePackage{perfectcut}
\RequirePackage[user]{zref}
```

3.2 Environments

3.2.1 Floats

`cprulesetfloat` A floating environment inside which `cpruleset` environments are to be placed. This ‘wrapping’ float provides both the floating capability, as well the ability to caption, label and reference `cprulesets`.

```
\DeclareFloatingEnvironment[name=Ruleset,
listname={List of cP~systems Rulesets},within=none,
placement={htbp},]
{cprulesetfloat}
```

`cpobjectsfloat` A floating environment inside which `cpobjects` environments are to be placed. This ‘wrapping’ float provides both the floating capability, as well the ability to caption, label and reference `cpobjects`.

```
\DeclareFloatingEnvironment[name=Objects Group,
listname={List of cP~systems Objects Groups},within=none,
placement={htbp},]
{cpobjectsfloat}
```

3.2.2 Maths mode environments

`cpruleset` A wrapper environment in which `cprule`s are listed, and which mimics the usual style of presentation for rules: A lined box with the rules inside it.

```
\newenvironment{cpruleset}
{\begin{framed}\begin{adjustwidth}{-1.0em}{-1.0em}
\renewcommand{\arraystretch}{1.1}\[\begin{array}{llllllr}}
{\end{array}\]\end{adjustwidth}\end{framed}
\setcounter{cpsystems@RuleNumAlt}{\value{cpsystems@RuleNum}}}
```

`cprulesetcontnum` Identical to the regular cpruleset environment, except that this one restores the rule number that the previous ruleset had reached. Use of this environment is *usually* not recommended.

```
\newenvironment{cprulesetcontnum}
{\setcounter{cpsystems@RuleNum}{\value{cpsystems@RuleNumAlt}}
\begin{framed}\begin{adjustwidth}{-1.0em}{-1.0em}
\renewcommand{\arraystretch}{1.1}\[\begin{array}{llllllr}}
{\end{array}\]\end{adjustwidth}\end{framed}
\setcounter{cpsystems@RuleNumAlt}{\value{cpsystems@RuleNum}}}
```

`cpobjects` A wrapper environment in which `cpobjectline`s are listed, imitating a style used in the past: A lined box with lines of cP systems objects defined inside it. Primarily used for illustrating examples.

```
\newenvironment{cpobjects}{\begin{framed}}{\end{framed}}
```

3.3 Macros

3.3.1 cP systems definitions

\cptuple A convenience for stating the definition of a cP system. The first parameter is the subscript to Π, while the next six are T, A, O, R, S, and $\bar{s}$.

```
\newcommand{\cptuple}[7]{
    \[
        \cpfunc{\Pi_{#1}}{#2, #3, #4, #5, #6, #7}
    \]
}
```

\cptupletemplate A convenience for stating the definition of a cP system. The template tuple that is usually stated as how a cP system is defined.

```
\newcommand*{\cptupletemplate}{
    \cptuple{cP}{T}{A}{O}{R}{S}{\Bar{s}}
}
```

3.3.2 Rules

\cprule The base "selector" macro for choosing between the starred and unstarred `\cprule` variants.

```
\newcommand*{\cprule}{%
            \@ifstar
                \cpruleStar%
                \cpruleNoStar%
             }
```

\cpruleNoStar For writing out a rule inside a `cpruleset` environment, as well as displaying and incrementing the rule number. Required arguments are, in order, beginning state name; LHS of rule; the label to be applied to the arrow; the ending state name; the RHS of the rule. This takes an optional argument, the label which you want to give this particular rule for cross-referencing purposes. No label is set if you do not provide the optional argument.

```
\newcommand{\cpruleNoStar}[6][]{
    \refstepcounter{cpsystems@RuleNum}
    \notblank{#1}{\zref@label{#1}}{}
    \cpsystems@basecprule{#2}{#3}{#4}{#5}{#6}
{\hspace{1.5em}(\thecpsystems@RuleNum)}
}
```

\cpruleStar Same as with a regular `\cpruleNoStar`, but does not show or increment the rule counter. Nor can you give these rules a label (since they don't have any numbers to refer back to).

```
\newcommand{\cpruleStar}[5]{
\cpsystems@basecprule{#1}{#2}{#3}{#4}{#5}{}
}
```

`\cpruleinline` For writing out a single rule inside the text, without creating a cpruleset environment. It still places the rule in its own little paragraph, because it uses the standalone maths environment internally. Takes a single parameter, a `cprule*`.

```
\newcommand{\cpruleinline}[1]{
\[\begin{array}{lllllr}#1\end{array}\]
}
```

`\cprulecustnum` Mostly the same as a regular `\cprule`, but takes the rule number to display as an extra parameter. This also does *not* increment the rule counter. I'm yet to work out how you could make references to these rule numbers. This is currently basically just a quick hack to permit the display of alternatives of pre-existing rules, and right now everything must be updated manually. If you are displaying an alternative to a pre-existing rule, you could *probably* use `\cpruleref*` plus the extra bit to make things work better. E.g. if you want to have rules 3a and 3b, you could do `\cprulecustnum{....}{\cpruleref*{otherrule}a}`.

```
\newcommand{\cprulecustnum}[6]{
\cpsystems@basecprule{#1}{#2}{#3}{#4}{#5}{(#6)}
}
```

`\cppromoter` For specifying promoters as part of a rule.

```
\newcommand*{\cppromoter}[1]{
\cpsystems@cpprominhi{|}{#1}
}
```

`\cpinhibitor` For specifying inhibitors as part of a rule.

```
\newcommand*{\cpinhibitor}[1]{
\cpsystems@cpprominhi{\neg}{#1}
}
```

`\cpsend` Encapsulate a ‘send’ in cP systems. First argument is the object(s) to be sent, and the second argument is the name of the channel the object(s) shall be sent on.

```
\newcommand{\cpsend}[2]{
    \cpsystems@basecpsendrecv{#1}{#2}{!}
}
```

`\cprecv` Encapsulate a ‘receive’ in cP systems. First argument is the object(s) to be received, and the second argument is the name of the channel the object(s) shall be received on.

```
\newcommand{\cprecv}[2]{
    \cpsystems@basecpsendrecv{#1}{#2}{?}
}
```

`\cpantisend` Encapsulate an antiport ‘send’ in cP systems. First argument is the object(s) to be sent, and the second argument is the name of the channel the object(s) shall be sent on.

```
\newcommand{\cpantisend}[2]{
    \cpsystems@basecpsendrecv{#1}{#2}{!!}
}
```

`\cpantirecv` Encapsulate an antiport 'receive' in cP systems. First argument is the object(s) to be received, and the second argument is the name of the channel the object(s) shall be received on.

```
\newcommand{\cpantirecv}[2]{
    \cpsystems@basecpsendrecv{#1}{#2}{??}
}
```

`\cpchangerulenumber` Change the rules counter to whatever positive integer you specify. There's likely an upper limit to what numbers you can set (and you may even be able to use negative numbers (you'd have to look into the details of LaTeX/TeX's counter system to be sure)), but that number has not been found as of yet, and is probably higher than the number of rules you should include in any one paper.

```
\newcommand*{\cpchangerulenumber}[1]{
\setcounter{cpsystems@RuleNum}{#1}
\setcounter{cpsystems@RuleNumAlt}{#1}
}
```

`\cpresetrulenumber` Reset the rules counter, i.e. make it zero again, so that the next rule in the document will receive the number 1.

```
\newcommand*{\cpresetrulenumber}{
    \cpchangerulenumber{0}
}
```

`\cponce` A minor convenience macro, to typeset the 'exactly once' rule mode symbol in the "correct" way in rules.

```
\newcommand*{\cponce}{{\scriptstyle 1}}
```

`\cpmaxpar` A minor convenience macro, to typeset the 'maximally parallel' rule mode symbol in the "correct" way in rules.

```
\newcommand*{\cpmaxpar}{{\scriptstyle +}}
```

`\cpruleref` The base "selector" macro for choosing between the starred and unstarred `\cpruleref` variants.

```
\newcommand*{\cpruleref}{%
            \@ifstar
                \cprulerefStar%
                \cprulerefNoStar%
             }
```

`\cprulerefNoStar` The below is used to enable cross-referencing for rules numbers. It was adapated from this TeX.StackExchange answer: `https://tex.stackexchange.com/a/610616/113430`. Many thanks to Ulrike Fischer for it. At current, I am using a basic implementation which does *not* have any hyperlinking capabilities or anything. I want to figure out how this all works a bit more before I get into anything fancy — and I don't want to introduce a dependency of hyperref just yet.

```
\newcommand*{\cprulerefNoStar}[1]{\zref@extract{#1}{cpsystems@ruleslabel}{rule \zref{#1}}}
```

\cprulerefStar Much the same as `\cprulerefNoStar`, but *doesn't* include the word “rule”.

```
\newcommand*{\cprulerefStar}[1]{\zref@extract{#1}{cpsystems@ruleslabel}{\zref{#1}}}
```

\cpRuleref Identical to `\cprulerefNoStar`, except that the word “rule” is capitalised. There is no starred version of this one, because that would end up identical in effect to `\cpruleref*`.

```
\newcommand*{\cpRuleref}[1]{\zref@extract{#1}{cpsystems@ruleslabel}{Rule \zref{#1}}}
```

3.3.3 Objects Groups

\cpobjectsline Used for presenting a group of objects, inside a `cpobjects` environment.

```
\newcommand{\cpobjectsline}[1]{
\[#1\]
}
```

\cpset Used for presenting a list of objects that are meant to be in the same set. Currently doesn't get used much, might be removed in the future.

```
\newcommand*{\cpset}[1]{\cpsystems@perfuncurly{#1}}
```

3.3.4 Miscellaneous

\cpfunc Command for declaring a cP systems functor. The first argument is the symbol for the functor itself, and the second argument is the objects contained inside the functor.

```
\newcommand*{\cpfunc}[2]{
#1\cpsystems@perfunparens{#2}
}
```

\cpfuncms Command for declaring a cP systems functor *which uses micro-surgeries.* The first argument is the symbol for the functor itself, and the second argument is the objects contained inside the functor. Currently the only difference between this and the `cpfunc` command is the nature of the brackets used – regular `cpfunc`s use parentheses, while micro-surgery functors use curly braces. Both (but especially the latter) are potentially subject to change in the future. Thus, one of the reasons why it is recommended to use these commands is that it will be much easier to change all your micro-surgery functors in future if the style changes. Another reason is that these play nicely with the regular `cpfunc`s to give you automatically growing brackets.

```
\newcommand*{\cpfuncms}[2]{
#1\cpsystems@perfuncurly{#2}
}
```

\cpfuncn Command for declaring a three-level cP systems indexed functor. The first argument is the symbol for the functor itself, the second argument is the objects contained inside the first part of the term, and the third argument the bit contained inside the last term.

```
\newcommand*{\cpfuncn}[3]{
```

```
\cpfunc{#1}{#2}\cpsystems@perfunparens{#3}
}
```

`\cpfuncnms` Command for declaring a three-level cP systems indexed functor with a micro-surgery on the end part. The first argument is the symbol for the functor itself, the second argument is the objects contained inside the first part of the term, and the third argument the bit contained inside the last term, which is the micro-surgical part.

```
\newcommand*{\cpfuncnms}[3]{
\cpfunc{#1}{#2}\cpsystems@perfuncurly{#3}
}
```

`\cpfuncnn` Command for declaring a four-level cP systems indexed functor. The first argument is the symbol for the functor itself, the second argument is the objects contained inside the first part of the term, the third argument the bit contained inside next part, and the fourth the bit contained in the last part of the term.

```
\newcommand*{\cpfuncnn}[4]{
\cpfuncn{#1}{#2}{#3}\cpsystems@perfunparens{#4}
}
```

`\cpfuncnnms` Command for declaring a three-level cP systems indexed functor with a micro-surgery on the end part. The first argument is the symbol for the functor itself, the second argument is the objects contained inside the first part of the term, the third argument the bit contained inside next part, and the fourth the bit contained in the last part of the term, which is the micro-surgical part.

```
\newcommand*{\cpfuncnnms}[4]{
\cpfuncn{#1}{#2}{#3}\cpsystems@perfuncurly{#4}
}
```

`\cptermdef` Used when writing a listing describing the cP systems terms used in a given text (which is envisaged to be something that occurs immediately after stating the ruleset). The general suggestion is to use these inside a `description` environment, which tends to leave the whole thing quite nicely set out. The end result is the symbols in math mode, and then their descriptions next to them.

```
\newcommand*{\cptermdef}[2]{
\item[$#1$]#2
}
```

`\cpundig` A parameter-less convenience macro for inserting the correctly-formatted cP systems 'unary digit' in rules.

```
\newcommand*{\cpundig}{\mathit{1}}
```

`\cpempty` A parameter-less convenience macro for inserting the correctly-formatted empty functor symbol (which, currently, is actually just a `\lambda`). Mostly used so that intent is clear in the rules specifications, but also partly in case someone ends up changing how empty functors are specified.

```
\newcommand*{\cpempty}{\lambda}
```

`\cpdiscard` A minor convenience macro, to typeset the 'discard/don't care' symbol used in cP systems in the "correct" way in rules. Honestly, in this instance the underlying command is much shorter, but this is thought of as slightly 'safer' (it's quite easy to muck things up with the underscore), communicates intent better, and moreover use of this makes it much easier to change to using a different symbol if desired.

```
\newcommand*{\cpdiscard}{\_}
```

3.4 Interactions with other packages

`Cleveref and floats` The below snippet checks if the `\crefname` command (which is assumed to be from cleveref) is defined. If the command is defined, then it is used to provide the necessary information for cleveref to do proper cross-references to the `cprulesetfloat` and `cpobjectsfloat` environments.

```
\AfterPreamble{
    \ifdef{\crefname}
        {\crefname{cprulesetfloat}{ruleset}{rulesets}
        \crefname{cpobjectsfloat}{objects group}{objects groups}}
    {}
}
```

3.5 Internal

Macros that are only intended to be used inside the package, and probably shouldn't be used outside of it. Much as if the previous macros are all the public interface of the package, while these ones are the private implementation details.

3.5.1 Counters

`Package-specific counters` A couple of counters used for tracking rules numbers.

```
\newcounter{cpsystems@RuleNum}[cprulesetfloat]
\newcounter{cpsystems@RuleNumAlt}
```

3.5.2 Internal commands for cross-referencing of rules numbers

`Rule references internal defs` The below is used to enable cross-referencing for rules numbers. It was adapated from this TeX.StackExchange answer: `https://tex.stackexchange.com/a/610616/113430`. Many thanks to Ulrike Fischer for it. At current, I am using a basic implementation which does *not* have any hyperlinking capabilities or anything. I want to figure out how this all works a bit more before I get into anything fancy — and I don't want to introduce a dependency of hyperref just yet.

```
\zref@newprop{cpsystems@ruleslabel}[Doc-Start]{}
\zref@addprop{main}{cpsystems@ruleslabel}
```

3.5.3 Internal commands

`\cpsystems@basecprule` For writing out rules inside a `cpruleset` environment. Required arguments are, in order, beginning state name; LHS of rule; the label to be applied to the arrow; the ending state name; the RHS of the rule.

```
\newcommand{\cpsystems@basecprule}[6]{
    #1 & #2 & \rightarrow_{#3} & #4 & #5 & #6\\
}
```

`\cpsystems@cpprominhi` For writing out promoters and inhibitors in a `cpruleset` environment.

```
\newcommand{\cpsystems@cpprominhi}[2]{
    & & & & ~ \hspace{1em} ~ #1 ~ #2 & \\
}
```

`\cpsystems@basecpsendrecv` Base rule which both `\cpsend` and `\cprecv` make use of. This is a “single source of truth”, and means only one place needs to be changed to change the brackets which encapsulate messages.

```
\newcommand{\cpsystems@basecpsendrecv}[3]{
    \cpsystems@perfunangles{#1}{#3}_{#2}
}
```

`\cpsystems@perfunparens` Convenience rule used to provide the wrapping of other things in auto-growing parentheses. Introduced because I found I was starting to copy-paste the raw command frequently.

```
\newcommand{\cpsystems@perfunparens}[1]{
    \perfectunary{IncreaseHeight}{(}{)}{#1}
}
```

`\cpsystems@perfuncurly` Convenience rule used to provide the wrapping of other things in auto-growing curly braces. Introduced because I found I was starting to copy-paste the raw command frequently.

```
\newcommand{\cpsystems@perfuncurly}[1]{
    \perfectunary{IncreaseHeight}{\{}{\}}{#1}
}
```

`\cpsystems@perfunangles` Convenience rule used to provide the wrapping of other things in auto-growing curly braces. Introduced because I found I was starting to copy-paste the raw command frequently.

```
\newcommand{\cpsystems@perfunangles}[1]{
    \perfectunary{IncreaseHeight}{\langle}{\rangle}{#1}
}
```

4 Possible improvements

A handful of possible improvements have been thought of already, though in most cases it is entirely unclear how to achieve them at this point. They include:

- A command to create line breaks in `cpruleset` environments, without requiring the user to fill in all the &s required by the contained `array` environment to make it work.
- Restoring support for line breaks inside `cpfunc` commands – it seems to be unavoidable at times, but doesn't play nicely with the resizing brackets.
- Change to a different, more modern way of declaring `cpruleset` environments that doesn't rely on the `array` environment. There's nothing wrong with `array`, but it's probably more low-level than I actually need, and doesn't come with some conveniences (I imagine).
- The ability to specify optional parameters to the `cpruleset` stating the desired amount of arraystretch and adjustwidth to use. Currently they are hard-coded as `1.1em` and `-1.0em`, respectively.
- Provide the ability to specify a printed name for `cprulesetfloat` and `cpobjectsfloat` besides 'Ruleset' and 'Objects Group'.
- Make the package available via CTAN.
- Eliminate the extraneous symbols in the index (it is unclear why they are appearing, when they have been specifically excluded using the normal method for `.dtx` files).
- Change the implementation of the package to using LaTeX 3.
- Versions of `cpruleset` and `cpobjects` that *don't* draw boxes around themselves. Perhaps, e.g. `cpruleset*`.
- A way to break up one ruleset or objects group into multiple boxes and/or across multiple pages, cleanly. Though, to be honest, if this is becoming an issue for a ruleset, it probably means that the ruleset has grown large enough that it could be logically split into subsets. Objects groups may be a different story.
- Seek to integrate this package into David Orellana Martín's 'membranecomputing' package (`https://ctan.org/pkg/membranecomputing`).
- Modify the objectsgroup environment to show somehow the state that the system or containing top-level cell is in at the moment of the object snapshot.
- Use the indexing function properly, so that it appears in the table of contents, and also includes all the various types of numbers pointing to different things that it talks about in its preamble.

Note that there is absolutely no time-frame currently for the completion of any of these, and at least two of them are probably in conflict with each other.

References

[1] Cooper, J., and Nicolescu, R. The Hamiltonian Cycle and Travelling Salesman Problems in cP Systems. *Fundamenta Informaticae 164*, 2-3 (Jan 2019), 157–180.

[2] Nicolescu, R., and Henderson, A. An Introduction to cP Systems. In *Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*, C. Graciani, A. Riscos-Núñez, G. Păun, G. Rozenberg, and A. Salomaa, Eds., no. 11270 in Lecture Notes in Computer Science. Springer International Publishing, Cham, 2018, pp. 204–227.

[3] Păun, G. *Membrane Computing.* Natural Computing Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History

0.11
General: Converted to DTX file . . 1

0.12
General: Completed documentation for first DTX format attempt 1

0.13
General: Added in the commands for `cprulenonum`, `changerulenum` and `resetrulenum`. Modified the four major pre-existing rule typesetting macros all to use an internal macro, reducing duplication of code. Also corrected a handful of typos in the document. 1

0.13.1
General: Fix problem with arrows appearing in the `cppromoter` and `cpinhibitor` results. 1

0.14
General: Add minimum LaTeX2e version to package specification. Change many of the `newcommand` declarations to use the `newcommand*` style, making them "short commands", which (apparently) throw warnings if there is a paragraph break detected in their use – the ones modified aren't expected to have paragraph breaks in them. Also, added in the `cps`, `cpundig` and `cpempty` convenience macros. 1

0.15
General: Introduced use of perfectcut to provide automatically-resizing brackets. Also added an explanation and example for `cpterm`. Unfortunately, this also breaks linebreaks inside `cpfunc` macros for the time being... . . . 1

0.16
General: Added in the `cpruleinline` and `cpfuncms` macros, since I had already created them and am using them in a paper I am writing. . 1

0.17
General: Remove the ending dot from `cpterm`. Adjust `cpsend` and `cprecv` to use a single internal base rule. Removed the `cps` command, to avoid potential conflicts with other packages. I have found defining cP systems as a glossary term to be a superior alternative anyway. Changed `cpresetrulenumber` to make use of `cpchangerulenumber`, instead of replicating the functionality. 1

0.18
General: Added in `cprulecustnum`, `cpantirecv`, `cpantisend`, `cptuple`, `cptupetemplate`, `cponce`, `cpmaxpar`, `cpset`. Adjusted the arraystretch for rules slightly (from 1.0 to 1.1). Prefixed `changerulenumber` and `resetrulenumber` with `cp`. 1

0.19
General: Most importantly, changed the default behaviour of rules numbers — now, they no longer continue on from one ruleset to the next by default. Instead, they reset at each new `cprulesetfloat`. Added commands for the `cpfuncn` and `cpfuncnms` stuff. Modified the

`cprule` definition to permit the optional width adjustment. Introduced the (basically untested) `cprulsetcontnum` environment, and the `cpdiscard` command. Also added in the internal `perfunXXX` macros to tidy up the definitions of other macros. 1

0.20

General: Introduce the `ruleref` and `rulerefalt` commands. This (sadly) means that, at present, you *cannot* adjust a rule's spacing between the RHS and the rule number. Rename `cpterm` to `cptermdef` to reflect the fact that it used to provide the definition of a cP systems term, and not somehow declare one or anything. Add in some setup stuff so that this package's floating environments work with basic usage of the cleveref package. 1

0.21

General: Change the bracket type for inter-top-level-cell communication from curly braces to angle brackets. Add starred and capitalised variants of the `ruleref` macro — the former replaces `rulerefalt`. Rename it to `cpruleref` for consistency purposes. Convert `cprulenonum` to `cprule*`. 1