The cpsystems LaTeX package

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

James Cooper, University of Auckland

jcoo092@aucklanduni.ac.nz

v0.12, 2019/12/31

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.

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. 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 not necessarily 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

[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.

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.

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.

2.1 Environments

`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'.

`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.

`\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}`

`\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.

`\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 general, experience suggests that it is best to use `\cpfunc` when a functor will contain greater complexity than a single variable or only atoms of the same type. So, something like $a(A)$ would be typeset simply as `\(a(A)\)` and $c(ddddd)$ as `\(c(ddddd)\)`, whereas $b\big(c(C)\ d(D)\ ee\ fff\big)$ should probably be written as `\ensuremath{\cpfunc{b}{c(C)~d(D)~ee~fff}}`.

Do note also, that the size of the parentheses around a `\cpfunc` are slightly enlarged from the usual, using the `\big` bracket modifier. This is mostly only noticeable if one looks closely, but it does seem to make it easier to spot where the containing functor closes, as compared to its contents.

`\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.

`\cpterm` A full description and examples for this item are to come.

2.3 Full examples

2.3.1 A floating cP systems ruleset

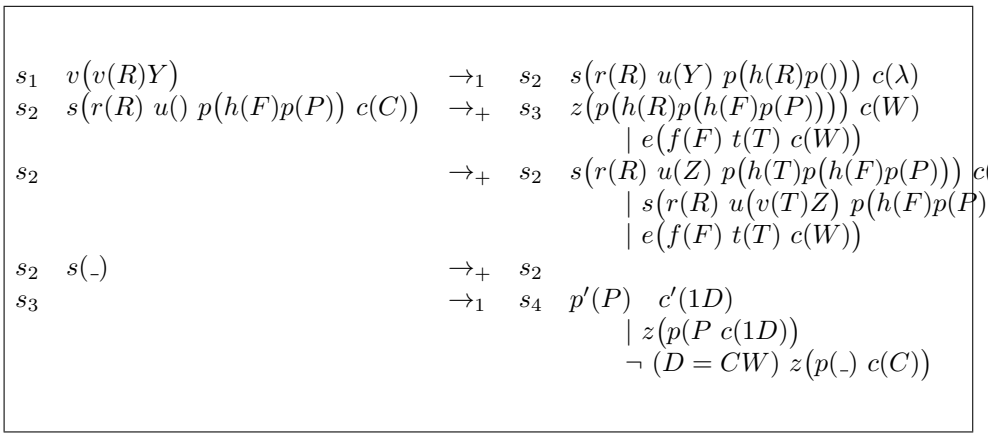
For example, to typeset the ruleset for the solution to the Travelling Salesman Problem in [1], depicted here in Ruleset 1, the following was used:[3]

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)}}
```

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



$$
\begin{array}{llcll}
s_1 & v\big(v(R)Y\big) & \to_1 & s_2 & s\big(r(R)\ u(Y)\ p\big(h(R)p()\big)\big)\ c(\lambda) & (1)\\
s_2 & s\big(r(R)\ u()\ p\big(h(F)p(P)\big)\ c(C)\big) & \to_+ & s_3 & z\big(p\big(h(R)p\big(h(F)p(P)\big)\big)\big)\ c(W) & (2)\\
 & & & & \quad \mid e\big(f(F)\ t(T)\ c(W)\big) & \\
s_2 & & \to_+ & s_2 & s\big(r(R)\ u(Z)\ p\big(h(T)p\big(h(F)p(P)\big)\big)\big)\ c(CW)\big) & (3)\\
 & & & & \quad \mid s\big(r(R)\ u\big(v(T)Z\big)\ p\big(h(F)p(P)\big)\ c(C)\big) & \\
 & & & & \quad \mid e\big(f(F)\ t(T)\ c(W)\big) & \\
s_2 & s(\_) & \to_+ & s_2 & & (4)\\
s_3 & & \to_1 & s_4 & p'(P)\quad c'(1D) & (5)\\
 & & & & \quad \mid z\big(p(P\ c(1D)\big) & \\
 & & & & \quad \neg\ (D = CW)\ z\big(p(\_)\ c(C)\big) &
\end{array}
$$

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

```
    \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}{}

    \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{cpruleset}float}
```

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 lead to either 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{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}{}

    \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 carries on from those in the previous `cpruleset`, Ruleset 1.

$$
\begin{array}{llll}
s_1 & v\big(v(R)Y\big) & \to_1 \; s_2 & s\big(r(R)\ u(Y)\ p\big(h(R)p()\big)\big)\ c(\lambda) \qquad (6)\\
s_2 & s\big(r(R)\ u()\ p\big(h(F)p(P)\big)\ c(C)\big) & \to_+ \; s_3 & z\big(p\big(h(R)p\big(h(F)p(P)\big)\big)\big)\ c(W) \qquad (7)\\
 & & & \quad \mid e\big(f(F)\ t(T)\ c(W)\big)\\
s_2 & & \to_+ \; s_2 & s\big(r(R)\ u(Z)\ p\big(h(T)p\big(h(F)p(P)\big)\big)\ c(CW)\big) \qquad (8)\\
 & & & \quad \mid s\big(r(R)\ u\big(v(T)Z\big)\ p\big(h(F)p(P)\big)\ c(C)\big)\\
 & & & \quad \mid e\big(f(F)\ t(T)\ c(W)\big)\\
s_2 & s(\_) & \to_+ \; s_2 & \qquad (9)\\
s_3 & & \to_1 \; s_4 & p'(P)\quad c'(1D) \qquad (10)\\
 & & & \quad \mid z\big(p(P\ c(1D)\big)\\
 & & & \quad \neg\ (D = CW)\ z\big(p(\_)\ c(C)\big)
\end{array}
$$

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.

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

2.3.2 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() \\ & \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}{}

 \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}
```

$$
\begin{array}{llll}
s_1 & v\big(v(R)Y\big) & \to_1 \; s_2 & s\big(r(R)\ u(Y)\ p\big(h(R)p()\big)\big)\ c(\lambda) \qquad (11)\\
s_2 & s\big(r(R)\ u() & & \\
 & \quad p\big(h(F)p(P)\big) & & \\
 & \quad c(C)\big) & \to_+ \; s_3 & z\big(p\big(h(R)p\big(h(F)p(P)\big)\big)\big)\ c(W) \qquad (12)\\
 & & & \quad \mid e\big(f(F)\ t(T)\ c(W)\big)\\
s_2 & & \to_+ \; s_2 & s\big(r(R)\ u(Z)\\
 & & & \quad p\big(h(T)p\big(h(F)p(P)\big)\big)\\
 & & & \quad c(CW)\big) \qquad (13)\\
 & & & \quad \mid s\big(r(R)\ u\big(v(T)Z\big)\ p\big(h(F)p(P)\big)\ c(C)\big)\\
 & & & \quad \mid e\big(f(F)\ t(T)\ c(W)\big)\\
s_2 & s(\_) & \to_+ \; s_2 & \qquad (14)\\
s_3 & & \to_1 \; s_4 & p'(P) \quad c'(1D) \qquad (15)\\
 & & & \quad \mid z\big(p(P\ c(1D)\big)\\
 & & & \quad \neg\ (D = CW)\ z\big(p(\_)\ c(C)\big)
\end{array}
$$

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.

2.3.3 Sending and receiving

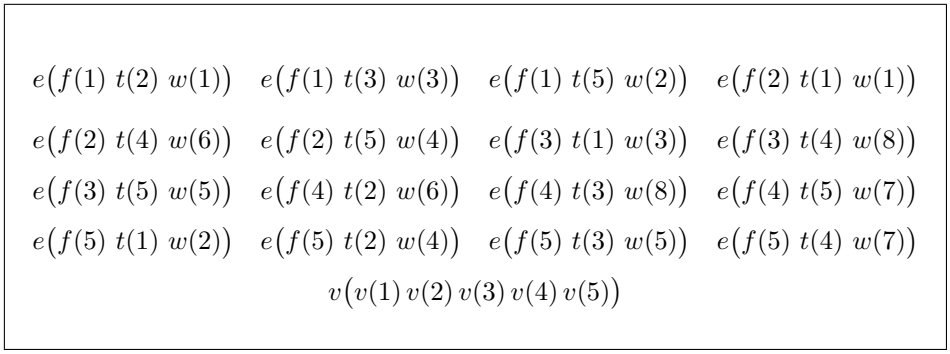
2.3.4 Listings of objects

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)}}
```



Objects Group 1: The first example of an objects group

```
    \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}
```

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. 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.

3 Implementation

This section presents the actual implementation of the package. For the most part you probaly 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.

```
%

\RequirePackage{array}
\RequirePackage{framed}
\RequirePackage{changepage}
\RequirePackage{amsmath}
\RequirePackage{trimspaces}
\RequirePackage{newfloat}

\newcounter{cpsystems@RuleNum}
```

`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,within=none]{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,within=none]{cpobjectsfloat}
```

`cpruleset` A wrapper environment in which `cprules` 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}{1.0}\[\begin{array}{llllr}}
{\end{array}\]\end{adjustwidth}\end{framed}}
```

`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}}
```

`\cprule` For writing out a rule 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{\cprule}[5]{
    \refstepcounter{cpsystems@RuleNum}
    \trim@spaces@noexp{#1 & #2 & \rightarrow_{#3} & #4 & #5
     & (\arabic{cpsystems@RuleNum})\\}
}
```

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

```
\newcommand{\cppromoter}[1]{
\trim@spaces@noexp{& & & ~ \hspace{0.5cm} ~ | ~ #1 & \\}
}
```

`\cpinhibitor`

```
% For specifying inhibitors as part of a rule.
\newcommand{\cpinhibitor}[1]{
\trim@spaces@noexp{& & & ~ \hspace{0.5cm} ~ \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]{
\trim@spaces@noexp{\{#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]{
\trim@spaces@noexp{\{#1\}?_{#2}}
}
```

`\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]{
\trim@spaces@noexp{#1\big(#2\big)}
}
```

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

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

`\cpterm` Explanation to be completed.

```
\newcommand{\cpterm}[2]{
\item[$#1$]#2.
}
```

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.
- User-facing commands to reset the `cpsystems@RuleNum` counter, or set it to a particular number.
- Make brackets automatically grow or shrink, depending on how highly-nested they are.
- Commands to write an individual rule inline in text, or in a separate paragraph. Either way, outside of a `cpruleset` environment.
- The ability to specify an optional parameter to the `cpruleset` stating the desired amount of array stretch to use. Currently it is hard-coded as `-1.0em`.
- 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 LaTeX3's approach.
- Versions of `cpruleset` and `cpobjects` that *don't* draw boxes around themselves. Perhaps, e.g. `cprulset*`.
- 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.

Note that there is absolutely no time-frame currently for the completion of any of these.

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