Circus Grammar Explained — Processes

Leonardo Freitas

March 2006

Abstract

This article documents the various ways one can declare *Circus* processes using LaTeX markup. It also documents the open issues of the language related to this syntactic category.

1 Introduction

In this file, we explain the various aspect of *Circus* related to process declarations. At each section relevant to the grammar, we include its corresponding CUP rule, as "Section name — `grammarRule`".

Circus productions must be included within the circus LaTeX environment in order to be recognised by the parser. Therefore, to use *Circus* one needs to have the current version of the circus.sty LaTeX style file.

Amongst other commands, it contains all *Circus* keywords, and some useful environments.

Make a suitable `circus.dtx` with nice and up-to-date *Circus* typesetting commands, as well as embedded documentation.

Join version of `circus.sty` and add literate documentation. Also, gather the information for solving the spacing problem of the environment.

2 *Circus* process paragraphs — processPara

In *Circus* processes are global paragraphs at section level. Other global paragraphs are channel, channel sets, and all other Z paragraphs. Firstly, we need to create a new section containing the circus_toolkit as a parent.

$\textbf{section}\ \ circus\_processes\ \textbf{parents}\ \ circus\_toolkit$

```
begin{zsection}
  \SECTION\ circus\_channels \parents\ circus\_toolkit
end{zsection}
```

We include with each typeset text the corresponded LaTeX code.

Circus processes enables encapsulated specification of both state and behavioural aspects of systems. They are defined by a process paragraph that introduces a name for a process description. Like schemas and axiomatic definitions, process paragraphs can also have formal parameters.

$\textbf{process}\ [X, Y]P \mathrel{\widehat=} \cdots$

For the processes below, we define some channels.

$\textbf{channel}\ [X]a, b : X$
$\textbf{channel}\ c, d : \mathbb{N}$
$\textbf{channel}\ s1, s2$

More information on the syntax of channels can be found in the channels.pdf file.

In *Circus* every process paragraph contains a name attached to a process description, where formal parameters are optional.

$$\textbf{process}\ [N^{+}]N \mathrel{\widehat=} \mathit{ProcessDesc}$$

Process names must be unique across the specification and the available process descriptions are defined below. They usually come between `begin{circus}...end{circus}` environments.

2.1 Process definition — process

Normal process definition uses the various CSP operators to combine process. They are detailed below.

2.1.1 Hiding

Process hiding conceals the events from the given channel set from the given process. As mentioned before in the channels.tex file, channel set expressions cannot mix Z expressions with **BasicChannelSetExpr**, hence the need for $HCS0$ and $HCS1$. The typechecker will infer the type types of channels from HCS in the definition of process H_1.

$$\textbf{channelset}\ HCS0 == \{\!|\ a, b\ |\!\}$$
$$\textbf{channelset}\ HCS1 == \{\!|\ c\ |\!\}$$
$$\textbf{channelset}\ HCS == HCS0 \cup HCS1$$
$$\textbf{process}\ [X, Y]H1 \mathrel{\widehat=} \mathit{BASIC}[X \setminus Y] \mathbin{\big\backslash} HCS$$

```
begin{circus}
    \circchannelset\ HCS0 == \lchanset a, b \rchanset \\
    \circchannelset\ HCS1 == \lchanset c \rchanset \\
    \circchannelset\ HCS == HCS0 \cup HCS1 \\
    \circprocess\ H \circdef BASIC[\nat] \circhide
                             \lchanset a[\nat], b[\nat] \rchanset \\
    \circprocess\ [X, Y] H1 \circdef BASIC[X \setminus Y]
                               \circhide HCS
end{circus}
```

Another interesting point is about typesetting. Although, the hiding ($\big\backslash$) keyword operator does look like the set difference ($\setminus$) operator, their Unicode characters ***must*** be different in order to avoid inaccurate scanning. This rule is applied everything. For instance, the prefixing arrow ($\longrightarrow$) is slightly bigger than the else guarded command arrow ($\longrightarrow$), and slightly smaller than the function symbol ($\rightarrow$) from *relation_toolkit*.

2.1.2 Interleaving

If IL does not have generic formals, or H_1 does not receive adequate generic actual parameters, the type checker will issue an error since calls to generic process must contain enough information to infer the needed types.

$$\textbf{process}\ [X, Y]\ IL \mathrel{\widehat=} H \mathbin{|||} H1[X, Y]$$

```
begin{circus}
   \circprocess\ [X, Y]\ IL \circdef H \interleave H1[X, Y]
end{circus}
```

2.1.3 Parallelism

$+$ *HCS* generic actuals

> **process** $P \mathrel{\widehat{=}} BASIC[\mathbb{N}] \llbracket HCS[\mathbb{N}] \rrbracket H$

```
begin{circus}
   \circprocess\ P \circdef BASIC[\nat] \lpar HCS[\nat] \rpar H
end{circus}
```

2.1.4 External choice

> **process** $EC \mathrel{\widehat{=}} IL \Box P$

```
begin{circus}
   \circprocess\ EC \circdef IL \extchoice P
end{circus}
```

2.1.5 Internal choice

> **process** $[X, Y]\ IC \mathrel{\widehat{=}} BASIC[X] \sqcap H1[Y \times X, X \to Y]$

```
begin{circus}
   \circprocess\ [X, Y]\ IC \circdef BASIC[X] \intchoice
                                     H1[Y \cross X, X \fun Y]
end{circus}
```

2.1.6 Sequential composition

> **process** $SC \mathrel{\widehat{=}} EC \ ; \ EC$

```
begin{circus}
   \circprocess\ SC \circdef EC \circseq EC
end{circus}
```

2.1.7 Prefixing

For fields, something very tricky happens: strokes! Channel names are just refName (hence DECORDWORD), which means strokes are scanned accordingly. So, `c?x` is tokenised differently from `c~?x` as: In this situation we run out of luck because all possible solutions around it do create trouble somewhere else.

1. New Unicode for "?" (input field)

   That means LaTeX typeset as: `c\commIn x`. This also incurs change in the **KeywordScanner** (or **ContextFreeScanner**) to recognise this new type of "word-glue".

2. Channel names without strokes

   Doesn't change the fact the scanner will recognise $c?$ as one DECORWORD, which means no good will be done here. This in fact is the harder solution, as it demands various changes in both **ContextFree** and **Latex2Unicode** scanners. As a consequence, everywhere that channel appear, require new productions for refName/declName and corresponding lists. ABSOLUTE NIGHTMARE! Forget it.

3. Using INSTROKE to represent input field

   That means disappearing with CIRCCOMMQUERY (easy) and using INSTROKE instead. This does not require any change anywhere else. The price to pay, is that the LaTeX must have a hard space as `c~?x`.

   That is somehow similar to what happens with DOT. The difference is that "?" and "!" are more complicated because they can be lexed as either part of a DECORWORD or as STROKES.

Processes do not have prefixing
Discussed with Ana already: processes do not have prefixing, guards, or recursion.

2.1.8 Some examples on precedences

The following processes are equivalent. The names are given as acronyms of the operator used, where the subscripted version is the parenthesised equivalent. Thus, for instance, *IC_EC_SC* means a tree with $\sqcap$ as its root and $\Box$ and $;$ as its left and right branches respectively.

$\mathbf{process}\ IC\_EC\_SC \mathrel{\widehat=} SC \mathbin{;} SC \mathbin{\Box} EC \sqcap IC[\mathbb{N}, \mathbb{N}]$
$\mathbf{process}\ IC\_EC\_SC0 \mathrel{\widehat=} ((SC \mathbin{;} SC) \mathbin{\Box} EC) \sqcap IC[\mathbb{N}, \mathbb{N}]$

$\mathbf{process}\ SC\_IC\_EC \mathrel{\widehat=} SC \mathbin{;} (SC \mathbin{\Box} EC \sqcap IC[\mathbb{N}, \mathbb{N}])$
$\mathbf{process}\ SC\_IC\_EC0 \mathrel{\widehat=} SC \mathbin{;} ((SC \mathbin{\Box} EC) \sqcap IC[\mathbb{N}, \mathbb{N}])$

$\mathbf{process}\ EC\_SC\_IC \mathrel{\widehat=} SC \mathbin{;} SC \mathbin{\Box} (EC \sqcap IC[\mathbb{N}, \mathbb{N}])$
$\mathbf{process}\ EC\_SC\_IC0 \mathrel{\widehat=} (SC \mathbin{;} SC) \mathbin{\Box} (EC \sqcap IC[\mathbb{N}, \mathbb{N}])$

$\mathbf{process}\ IC\_SC\_EC \mathrel{\widehat=} SC \mathbin{;} (SC \mathbin{\Box} EC) \sqcap IC[\mathbb{N}, \mathbb{N}]$
$\mathbf{process}\ IC\_SC\_EC0 \mathrel{\widehat=} (SC \mathbin{;} (SC \mathbin{\Box} EC)) \sqcap IC[\mathbb{N}, \mathbb{N}]$

$\mathbf{process}\ IL\_\_EC\_SC\_\_IC\_SC \mathrel{\widehat=} SC \mathbin{;} SC \mathbin{\Box} EC \mathbin{|||} EC \sqcap EC \mathbin{;} SC$
$\mathbf{process}\ IL\_\_EC\_SC\_\_IC\_SC0 \mathrel{\widehat=} ((SC \mathbin{;} SC) \mathbin{\Box} EC) \mathbin{|||} (EC \sqcap (EC \mathbin{;} SC))$

2.1.9 Some examples with hard new lines and spaces

Just like in Z, it is desirable to allow special scanning of hard new lines and spaces, such as the `\\` and `\tN` symbols. This enables pretty printing and should not affect parsing.

$\mathbf{process}\ [X, Y]\ IC2 \mathrel{\widehat=} BASIC[X] \sqcap$
$\qquad\qquad H1[Y \times X, X \rightarrow Y]$

```
begin{circus}
   \circprocess\ [X, Y]\ IC2 \circdef BASIC[X] \intchoice \\
                                \t5  H1[Y \cross X, X \fun Y]
end{circus}
```

From the grammar point of view, such tokens are a nuisance: they introduce nasty conflicts, and badly affect the size of the parsing tables. Fortunately,

CZT has a very elegant solution for this, namely a layered architecture involving around 10 different specialised scanners, one of which is responsible for handling new line problems alone! That means, a simple configuration on the **NewLineScanner** terminals telling which tokens allows new lines before and/or after them to be treated specially. This directly (and specifically) implements the specialised handling of new line lexis, as defined in the Z standard Section 7.5.

2.2 Parameterised process — paramProcess

Parameteri

2.3 Indexed process — indexedProcess