Circus Grammar Explained — Channels

Leonardo Freitas

March 2006

Abstract

This article documents the various ways one can declare *Circus* channels and channel sets using LaTeX markup. It also documents the open issues of the language related to these two syntactic categories.

1 Introduction

In this file, we explain the various aspect of *Circus* related to channel 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.

2 *Circus* channel paragraphs — channelPara

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

section *circus_channels* **parents** *circus_toolkit*

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

We include with each typeset text the corresponded LaTeX code.

2.1 Common channels — channelDecl

Channels are declared in *Circus* to specify synchronisation points. Channel types are any Z expression, and they define the communication patterns to synchronise.

Untyped channels are also allowed. They represent synchronisation points with the given type *Synch*. Thus, declaring a untyped channel has the same effect as declaring the channel with type *Synch*, as defined in the circus_toolkit.tex file.

Channels are declared within the circus LaTeX environment, which are typeset as Z unboxed items like in the zed LaTeX environment. As the Z standard, various unboxed *Circus* items can be declared within the same environment, as long as they are separated by a **NL** (new-line) token.

2.1.1 Typed channels

Let us define some typed channels,

channel $a, b : \mathbb{N}$

```
begin{circus}
   \circchannel\ a, b: \nat
end{circus}%
```

STROKES ARE NOT ALLOWED IN CHANNEL NAMES - it messes up communication patterns.

2.1.2 Generic channels

Generic channels can also be declared, where generic formal parameters given are right after the **channel** keyword.

channel $[X, Y]\ g, h, i : (X \leftrightarrow Y)$

```
begin{circus}
   \circchannel\ [X, Y]\ g, h, i: (X \rel Y)
end{circus}%
```

As before, names with decorations must have hard spaces before hitting a `COMMA` token, as shown above.

2.1.3 Synchronisation channels

Synchronisation channels are defined without a type expression. The parser introduces a *Synch* **RefExpr** for it automatically, as it is necessary for translation to and from ZML. This name is defined as a given set in the circus_toolkit.tex file.

channel c, d
channel z, w

```
begin{circus}
   \circchannel\ c, d \\
   \circchannel\ z, w
end{circus}%
```

This time, we left two declarations within the same Z box, but separated by a **NL** (new-line) token.

2.2 Channels through schemas — channelFromDecl

Another way of declaring typed channels is through schema references. Let us define a schema named S with two channels a and b

$$\begin{array}{|l} \multicolumn{1}{l}{S} \\ \hline a, b : \mathbb{N} \\ \hline a > b \land b \neq 0 \\ \hline \end{array}$$

```
begin{schema}{S}
    a, b: \nat
\where
    a > b \land b \neq 0
end{schema}
```

With the **channelfrom** keyword, we can declare these channels as well.

channelfrom S

```
begin{circus}
   \circchannelfrom\ S
end{circus}%
```

These declarations include four channels of type $\mathbb{N}$, namely: a, b, a_1, and b_1.

Another option is to have schemas with generic types.

$$
\begin{array}{|l}
\hline
T[X] \\
x, y : \mathbb{P}\ X \\
\hline
x \subseteq y \\
\hline
\end{array}
$$

```
begin{schema}{T}[X]
    x, y: \power~X
\where
    x \subseteq y
end{schema}
```

In this case, one can define the channels from the schema via either: (i) declaring the schema name with corresponding number of generic actuals; (ii) leaving the original generic formal parameters (possibly renaming them); (iii) letting the typechecker decide; or (iii) redefining the generic formal parameters of channel types in terms of a new pattern of formal parameters for the schema generic actuals.

channelfrom $T[\mathbb{N}]$
channelfrom $[Y]\ T[Y]$
channelfrom $T2$
channelfrom $[A, B]\ T3[A \times B]$

```
begin{circus}
    \circchannelfrom\ T[\nat] \\
    \circchannelfrom\ [Y]\ T1[Y] \\
    \circchannelfrom\ T2 \\
    \circchannelfrom\ [A, B]\ T3[A \cross B]
end{circus}%
```

It is the task of the typechecker to guarantee that either the instantiation or (re)definitions are type-correct. For each case we would have:

i **channel** $x, y : \mathbb{P}\ \mathbb{N}$

ii **channel** $[Y]\ x1, y1 : \mathbb{P}\ Y$

iii **channel** $x2, y2 : \mathbb{P}\ X$

iv **channel** $[A, B]\ x3, y3 : \mathbb{P}(A \times B)$

Thus, **channelfrom** is just syntactic sugar for a corresponding **channel** declaration.

3 *Circus* channel set paragraphs — channelSetPara

Channel sets defines an expression to represent a particular set of channels. They are normally used as reference names for other *Circus* operators that have channel references as parts of their AST, such as the parallel and the hiding operators.

Channel sets are given as a name and a corresponding channel set expression.

$$
\begin{array}{l}
\textbf{channelset}\ \ C == \varnothing \\
\textbf{channelset}\ \ C0 == \{\!|\ |\!\} \\
\textbf{channelset}\ \ C1 == \{\!|\ a, b, c, d\ |\!\} \\
\textbf{channelset}\ \ C2 == \{\!|\ x, y, x1, y1\ |\!\} \\
\textbf{channelset}\ [A, B] C3 == \{\!|\ x2[A \times B], y2[A \to B], x1[A], y1[B]\ |\!\} \\
\textbf{channelset}\ \ C4 == (C1 \cup C2 \cap C3) \setminus (C0 \cup C1 \cup C)
\end{array}
$$

```
begin{circus}
    \circchannelset\ C == \emptyset \\
    \circchannelset\ C0 == \lchanset \rchanset %\\
    \circchannelset\ C1 == \lchanset a, b, c, d \rchanset \\
    \circchannelset\ C2 == \lchanset x, y, x1, y1 \rchanset \\
    \circchannelset\ [A, B] C3 == \lchanset x2[A \cross B],
                                           y2[A \fun B], x_1[A],
                                           y1[B] \rchanset \\
    \circchannelset\ C4 == (C1 \cup C2 \cap C3) \setminus
                           (C0 \cup C1 \cup C)
end{circus}
```

The channel set expressions mostly used are: set union, intersection and difference expressions; set extension expressions; reference expressions; and application expressions.

Considering **BasicChannelSetExpr** a special case of set extension (with special brackets), the channel set expressions are a subset of the whole Z expression tree. More precisely, table **??** shows which one of the available Z expression productions are valid channel set expressions. On the third column of the table table: "**Y(N)**" means the Z expression is (is not) valid; "**Y***" means the Z expression is valid provided one changes the normal ($\{\ \}$) brackets to the special ($\{\!|\ \ |\!\}$) channel set brackets; "**Y!**" means an undecided inclination towards accepting; and "**?**" means yet unknown. At the moment, the parser implementation only filters set extension and comprehension, hence accepting all other Z expression productions.

[Including all Z restrictions] Add the other restrictions from table **??**

[How are the type rules for channel set references?] That is, could channel set references have generic actuals, as in?

$$
\begin{array}{l}
\textbf{channel}\ [X] c, d : X \\
\textbf{channel}\ [Y] e, f : Y \\
\textbf{channelset}\ \ CS1 = \{\!|\ c, d\ |\!\} \\
\textbf{channelset}\ \ CS2 = \{\!|\ e, f\ |\!\} \\
\textbf{channelset}\ \ CS3 = CS1[\mathbb{N}] \cup CS2[\mathbb{P}\ \mathbb{Z}]
\end{array}
$$

This is a typechecking problem that shouldn't affect the parser, unless one wanted to restrict the generic instantiation as a parsing error, something that would require the use of **RefName** rather than **RefExpr**.

Even further, as occurred with channel declaration from schemas, where we could possibly have channels as well as the schema generics, shouldn't we allow this for channel sets as well (*i.e.*, shouldn't it has generic actions?

$$\textbf{channelset}\ [A, B] CS4 = CS1[\mathbb{N} \times A] \cup CS2[A \times B]$$

If this is a desired feature, the AST for ChannelSet do require modification to include the list of **DeclName** for the generic formals.

[decorated channel sets] Decorated channel sets are also valid in the parser at the moment, as it makes the production rules easier accepting strokes.

$$CS1_0 \Rightarrow \{\!| c_0, d_0 |\!\}$$

This could be removed if needed, though. At the moment, I am assuming it as a typechecking rather than a parsing problem.

[Channel set expressions subtree — restrictions] To simplify the implementation of channel set expressions, we simply encapsulate a Z expression (**expression**) filtered "adequately" (*i.e.*, do not containing set extensions or comprehension) and include the production for **BasicChannelSetExpr**.

From the expressive options to include Z applications, this is the simplest one. Nevertheless, as **BasicChannelSetExpr** is not embedded into the Z expression subtree, one cannot mix it with normal Z expressions. The consequence is a fairly uncompromising (verboseness) restriction. For example, something like

$$\textbf{channelset}\ CS1 == \{\!| a, b |\!\}$$
$$\textbf{channelset}\ CS2 == \{\!| c |\!\}$$
$$\textbf{channelset}\ CS3 == CS1 \cup CS2$$

is valid because no Z expression is mixed with **BasicChannelSetExpr**, whereas something like

$$\textbf{channelset}\ CS3 == \{\!| a, b |\!\} \cup \{\!| c |\!\}$$

is not valid because such mix occurs.

One first (abandoned) alternative solution was to include the **BasicChannelSetExpr** production under a Z expression and use flags to say when to consider it or not. The consequences are neither elegant, nor desirable: clumsy code, difficulty in properly restricting the channel set expression tree properly, etc.

4 Conclusion

5 Future work

The ideal solution for channel set expressions would be to have their own expression subtree. This would be the neatest choice as it neither requires filtering on the Z expression subtree, nor imposes the restrictions on mixed expressions just mentioned above. Nevertheless, it is a quite hard choice as sorting the precedences and solving the conflicts of such subtree is very difficult (see the parser/tests/circus/cs_expression.Parser.xml file for details).

Another task for the near future is to sort out the JAXB problem from **ChannelDecl**.

Description	Example	Valid CSE?
Conditional	**if** *pred* **then** *expr* **else** *expr*	**Y**
Universal quantification	$\forall\, decl \mid pred \bullet expr$	**N**
Existential quantification	$\exists\, decl \mid pred \bullet expr$	**N**
Unique existential quant.	$\exists_1\, decl \mid pred \bullet expr$	**N**
Function construction	$\lambda\, decl \mid pred \bullet expr$	**Y**
Definite description	$\mu\, decl \mid pred \bullet expr$	**Y**
Substitution expression	**let** *abbrv* $\bullet$ *expr*	**Y**
Schema equivalence	$S \Leftrightarrow T$	**N**
Schema implication	$S \Rightarrow T$	**N**
Schema disjunction	$S \lor T$	**N**
Schema conjunction	$S \land T$	**N**
Schema negation	$\neg\, S$	**N**
Schema composition	$S \fatsemi T$	**N**
Schema piping	$S \gg T$	**N**
Schema hiding	$S \setminus T$	**N**
Schema projection	$S \upharpoonright T$	**N**
Schema precondition	**pre** S	**N**
Powerset	$\mathbb{P}\, X$	**?**
Cartesian product	$X \times Y$	**N**
Prefix application (PRE)	$f\_$	**?**
Prefix application (L, ERE)	$a\_b\_$	**?**
Prefix application (L, SRE)	$a\_, , b\_$	**?**
Postfix application (POST)	$X\ ^*$	**?**
Postfix application (ELP, ERP)	$\_a\_b$	**?**
Postfix application (ELP, SREP)	$\_a\_, , b$	**?**
Infix application (I)	$\_ \cup \_$	**Y**
Infix application (EL, ERE)	$\_a\_b\_$	**?**
Infix application (EL, SRE)	$\_a\_, , b\_$	**?**
Nofix application (L, ER)	a_b	**?**
Nofix application (L, SR)	$\langle\_, , \rangle$	**Y**
Set extension	$\{\, a, b \,\}$	**Y***
Set comprehension	$\{\, decl \mid pred \,\}$	**?**
Characteristic set comp.	$\{\, decl \mid pred \bullet expr \,\}$	**?**
Tuple extension	(x, y, z)	**N**
Characteristic definite desc.	$(\mu\, decl \mid pred)$	**?**
Binding extension	$\langle\!\vert x == 10, y == \{\,\} \vert\!\rangle$	**N**
Empty schema construction	$[\,]$	**N**
Schema construction	$[decl \mid pred]$ or $[\mid pred]$	**N**
Binding selection	$S.x$ or $\langle\!\vert x == 10 \vert\!\rangle$.x	**Y!**
Tuple selection	$t.1$ or $(x, y).1$	**Y!**
Schema decoration	S' or S_1	**Y!**
Binding construction	$\theta\, S$	**Y!**
Function appl with Sch Expr.	$f\, [decl \mid pred]$	**?**
Generic instantiation	$seq[\mathbb{N}, \mathbb{Z}]$, or $(\_ \to \_)[X, Y]$	**Y**
Schema renaming	$S[x/y, a/b]$	**Y!**
Number literal	10	**N**
Reference	$Name$:	**Y**

Table 1: Filtering the Z expression tree for channel sets