mCRL2 syntax definition

Aad Mathijssen

April 26, 2024

This document describes the syntax of mCRL2 expressions and specifications. We present the syntax in a rich text format. In Section 6 a translation of rich text to plain text is given, which is needed for using the toolset.

Throughout this document, suggestive dots ($\ldots$, $\cdots$) are used to indicate repeating patterns with one or more occurrence. Furthermore, $\mid$ distinguishes alternatives (not to be mistaken with the pipe $|$), $(\mathit{pattern})^+$ indicates one or more occurrences of *pattern*, and $(\mathit{pattern})^*$ indicates zero or more occurrences of *pattern*. As opposed to real EBNF, we do not use quotes to separate the terminals from the non-terminals.

1 Lexical syntax

We defined the notions of identifiers, numbers, whitespace and comments:

- An *identifier* is a string matching the pattern "$[A{-}Za{-}z\_][A{-}Za{-}z\_0{-}9]*$", excluding the following reserved words:

  sort cons map var eqn act glob proc pbes init
  struct Bool Pos Nat Int Real List Set Bag
  true false if div mod in lambda forall exists whr end
  delta tau sum block allow hide rename comm
  val mu nu delay yaled nil

  Identifiers are used for representing sort names b, function names f, data variable names x, action names a, process reference names P, and propositional variable names X.

- A *number* is a string that matches the pattern "0" or "$[1-9][0-9]*$".

- Spaces, tabs and newlines are treated as *whitespace*.

- A %-sign indicates the beginning of a *comment* that extends to the end of the line.

2 Data specifications

Sort expressions s:

$$\begin{array}{lcl}
s & ::= & b \mid s \to s \mid \mathsf{B} \mid \mathsf{N}^+ \mid \mathsf{N} \mid \mathsf{Z} \mid \mathsf{R} \\
 & \mid & \textbf{struct}\ scs \mid \cdots \mid scs \mid \mathsf{List}(s) \mid \mathsf{Set}(s) \mid \mathsf{Bag}(s) \mid (s) \\
scs & ::= & f \mid f(spj, \ldots, spj) \mid f?f \mid f(spj, \ldots, spj)?f \\
spj & ::= & s \mid s \times \cdots \times s \to s \mid f : s \mid f : s \times \cdots \times s \to s
\end{array}$$

Here *scs* and *spj* stand for the constructor and projection functions of a structured sort. The binary operator $\rightarrow$ associates to the right.

Data expressions d:

$$
\begin{array}{lll}
d & ::= & x \mid f \mid d(d,\ldots,d) \mid N \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if} \mid \neg d \mid -d \mid \overline{d} \mid \#d \mid d \oplus d \\
 & & \mid [\,] \mid [d,\ldots,d] \mid \{\,\} \mid \{d,\ldots,d\} \mid \{d:d,\ldots,d:d\} \mid \{x:s \mid d\} \\
 & & \mid \lambda_{mvd,\ldots,mvd} d \mid \forall_{mvd,\ldots,mvd} d \mid \exists_{mvd,\ldots,mvd} d \mid d \;\mathbf{whr}\; x=d,\ldots,x=d \;\mathbf{end} \\
 & & \mid (d) \\
mvd & ::= & x,\ldots,x:s \\
\oplus & ::= & * \mid . \mid \cap \mid / \mid \mathbf{div} \mid \mathbf{mod} \mid + \mid - \mid \cup \mid +\!\!+ \mid \triangleleft \mid \triangleright \\
 & & \mid < \mid \leq \mid \geq \mid > \mid \subset \mid \subseteq \mid \in \mid \approx \mid \not\approx \mid \wedge \mid \vee \mid \Rightarrow
\end{array}
$$

Here *mvd* stands for a multiple data variable declaration, $\oplus$ for a binary operator, and N for a number. The unary operators have the highest priority, followed by the infix operators, followed by λ, $\forall$ and $\exists$, followed by **whr end**. The descending order of precedence of the infix operators is: $\{*,.,\cap\},\{/,\mathbf{div},\mathbf{mod}\},\{+,-,\cup\},+\!\!+,\triangleleft,\triangleright,\{<,\leq,\geq,>,\subset,\subseteq,\in\},\{\approx,\not\approx\},\{\wedge,\vee\},\Rightarrow$. Of these operators $*$, $.$, $\cap$, $/$, **div**, **mod**, $+$, $-$, $\cup$ and $+\!\!+$ associate to the left and $\approx$, $\not\approx$, $\wedge$, $\vee$ and $\Rightarrow$ associate to the right.

Data specifications *data_spec*:

$$
\begin{array}{lll}
data\_spec & ::= & \mathbf{sort}\;(sd;)^{+} \\
 & & \mid \mathbf{cons}\;(mfd;)^{+} \\
 & & \mid \mathbf{map}\;(mfd;)^{+} \\
 & & \mid \mathbf{var}\;(mvd;)^{+}\;\mathbf{eqn}\;(ed;)^{+} \\
 & & \mid \mathbf{eqn}\;(ed;)^{+} \\
sd & ::= & b \mid b=s \\
mfd & ::= & f,\ldots,f:s \\
ed & ::= & d=d \mid c \rightarrow d=d
\end{array}
$$

Here, *sd* stands for sort declaration, *mfd* for multiple function declaration, *ed* for equation declaration, and *ad* for action declaration.

3 Process specifications

Process expressions p:

$$
\begin{array}{lll}
p & ::= & a \mid \delta \mid \tau \mid p+p \mid p\cdot p \mid P \mid p|p \mid p \parallel p \mid p \,\|\!\_\, p \\
 & & \mid \nabla_{\{as,\ldots,as\}}(p) \mid \partial_{\{a,\ldots,a\}}(p) \mid \tau_{\{a,\ldots,a\}}(p) \mid \rho_{\{ar,\ldots,ar\}}(p) \mid \Gamma_{\{ac,\ldots,ac\}}(p) \\
 & & \mid a(d,\ldots,d) \mid P(d,\ldots,d) \mid P() \mid P(x=d,\ldots,x=d) \\
 & & \mid c \rightarrow p \diamond p \mid c \rightarrow p \mid \sum_{mvd,\ldots,mvd} p \\
 & & \mid p \,{}^{c}\, t \mid t \gg p \mid p \ll q \\
 & & \mid (p) \\
as & ::= & a|\cdots|a \\
ar & ::= & a \rightarrow a \\
ac & ::= & a|as \rightarrow a \mid a|as \rightarrow \tau \mid a|as
\end{array}
$$

Here, c and t stand for data expressions of sort B and R, respectively. For technical reasons, c and t may not have an infix operator, a where clause or a quantifier at the top-level (parentheses should be used instead). *as* represents an action sequence, *ar* an action renaming, and *ac* an action communication. The descending order of precedence of the operators is: $|, {}^{c}, \cdot, \{\gg,\ll\}, \rightarrow, \{\parallel, \|\!\_\}, \sum, +$. Of these operators $+$, $\parallel$, $\|\!\_$, $\cdot$ and $|$ associate to the right.

Process specifications *proc_spec*:

$$
\begin{array}{lll}
proc\_spec & ::= & (proc\_spec\_elt)^* \\
proc\_spec\_elt & ::= & data\_spec \\
 & & \mid \textbf{act}\ (ad;)^+ \\
 & & \mid \textbf{glob}\ (mvd;)^+ \\
 & & \mid \textbf{proc}\ (pd;)^+ \\
 & & \mid \textbf{init}\ p; \\
pd & ::= & P = p \mid P(mvd, \ldots, mvd) = p \\
ad & ::= & a \mid a : s \times \cdots \times s
\end{array}
$$

Here *proc_spec_elt* represents a process specification element, *pd* a process definition, and *ad* an action declaration. Furthermore, we impose the restriction that *proc_spec* should contain precisely one occurrence of the keyword **init**.

4 Mu-calculus formulae

Multiactions *ma*:

$$
\begin{array}{lll}
ma & ::= & \tau \mid pa \,|\, \cdots \,|\, pa \\
pa & ::= & a \mid a(d, \ldots, d)
\end{array}
$$

Here, *pa* represents a parameterised action.

Action formulae α:

$$
\begin{array}{ll}
\alpha ::= & ma \mid \alpha {}^{\varsigma} t \mid \mathsf{val}(c) \mid (\alpha) \\
 & \mid \mathsf{true} \mid \mathsf{false} \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \Rightarrow \alpha \mid \forall_{mvd,\ldots,mvd}\alpha \mid \exists_{mvd,\ldots,mvd}\alpha
\end{array}
$$

Here, c and t stand for data expressions of sort B and R, respectively. For technical reasons, t may not have an infix operator, a where clause or a quantifier at the top-level (parentheses should be used instead). The descending order of precedence of the operators is: $\neg, {}^{\varsigma}, \{\wedge, \vee\}, \Rightarrow, \{\forall, \exists\}$. Of the infix operators ${}^{\varsigma}$ associates to the left and $\wedge$, $\vee$ and $\Rightarrow$ associate to the right.

Regular formulae φ_r:

$$
\varphi_r ::= \alpha \mid \epsilon \mid \varphi_r \cdot \varphi_r \mid \varphi_r + \varphi_r \mid \varphi_r^* \mid \varphi_r^+ \mid (\varphi_r)
$$

The postfix operators * and $^+$ have the highest priority, followed by $\cdot$, followed by infix $+$. The infix operators associate to the right.

State formulae φ_s:

$$
\begin{array}{lll}
\varphi_s & ::= & [\varphi_r]\varphi_s \mid \langle\varphi_r\rangle\varphi_s \mid \nabla(t) \mid \Delta(t) \mid \nabla \mid \Delta \mid \mathsf{val}(c) \mid (\varphi_s) \\
 & & \mid \nu X.\varphi_s \mid \mu X.\varphi_s \mid \nu X(vdi, \ldots, vdi).\varphi_s \mid \mu X(vdi, \ldots, vdi).\varphi_s \mid X \mid X(d, \ldots, d) \\
 & & \mid \mathsf{true} \mid \mathsf{false} \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \varphi_s \vee \varphi_s \mid \varphi_s \Rightarrow \varphi_s \mid \forall_{mvd,\ldots,mvd}\varphi_s \mid \exists_{mvd,\ldots,mvd}\varphi_s \\
vdi & ::= & x : s = d
\end{array}
$$

Here *vdi* stands for a data variable declaration and initialisation, and c and t stand for data expressions of sort B and R, respectively. For technical reasons, t may not have an infix operator, a where clause or a quantifier at the top-level (parentheses should be used instead). The descending order of precedence of the operators is: $\{\neg, \_[\_], \_\langle\_\rangle\}, \{\wedge, \vee\}, \Rightarrow, \{\forall, \exists, \mu, \nu\}$. The infix operators $\wedge$, $\vee$ and $\Rightarrow$ associate to the right.

5 PBES's

Parameterised boolean expressions φ_e:

$$
\begin{array}{lcl}
\varphi_e & ::= & pvo \mid \mathsf{val}(c) \mid (\varphi_e) \\
 & & \mid \mathsf{true} \mid \mathsf{false} \mid \neg\varphi_e \mid \varphi_e \wedge \varphi_e \mid \varphi_e \vee \varphi_e \mid \varphi_e \Rightarrow \varphi_e \mid \forall_{mvd,\ldots,mvd}\varphi_e \mid \exists_{mvd,\ldots,mvd}\varphi_e \\
pvo & ::= & X \mid X(d,\ldots,d)
\end{array}
$$

Here *pvo* stands for a propositional variable occurrence, The descending order of operator precedence is: $\neg, \{\wedge, \vee\}, \Rightarrow, \{\forall, \exists\}$. The infix operators $\wedge$, $\vee$ and $\Rightarrow$ associate to the right.

Parameterised boolean equations *pb_eqn*:

$$
\begin{array}{lcl}
pb\_eqn & ::= & \sigma\, pvd = \varphi_e \\
\sigma & ::= & \nu \mid \mu \\
pvd & ::= & X \mid X(mvd,\ldots,mvd)
\end{array}
$$

Here σ stands for a fixpoint symbol, and *pvd* for a propositional variable declaration.

PBES specifications *pbes_spec*:

$$
\begin{array}{lcl}
pbes\_spec & ::= & (pbes\_spec\_elt)^* \\
pbes\_spec\_elt & ::= & data\_spec \\
 & & \mid \textbf{glob}\ (mvd;)^+ \\
 & & \mid \textbf{pbes}\ (pb\_eqn;)^+ \\
 & & \mid \textbf{init}\ pvo;
\end{array}
$$

Here *pbes_spec_elt* represents a PBES specification element. We impose the restriction that *pbes_spec* should contain precisely one occurrence of each of the keywords **pbes** and **init**.

6 Table of symbols

In the toolset, a plain text format is used as opposed to the rich text format of the previous section. A mapping from rich text to plain text symbols is provided in Table 1.

Symbol	*Rich*	`Plain`
arrow	$\rightarrow$	`->`
cross	$\times$	`#`
diamond	$\diamond$	`<>`
standard sorts	$\mathsf{B}, \mathsf{N}^+, \mathsf{N}, \mathsf{Z}, \mathsf{R}$	`Bool`, `Pos`, `Nat`, `Int`, `Real`
equality and inequality	$\approx, \not\approx$	`==`, `!=`
logical operators	$\neg, \wedge, \vee, \Rightarrow$	`!`, `&&`, `||`, `=>`
relational numeric operators	$\leq, \geq$	`<=`, `>=`
relational set operators	$\in, \subseteq, \subset$	`in`, `<=`, `<`
set operators	$-, \cup, \cap$	`!`, `+`, `*`
list operators	$\triangleright, \triangleleft, +\!\!+$	`|>`, `<|`, `++`
lambda abstraction	$\lambda_{x:s} d$	`lambda x:s.d`
universal quantification	$\forall_{x:s} \varphi$	`forall x:s.phi`
existential quantification	$\exists_{x:s} \varphi$	`exists x:s.phi`
deadlock	δ	`delta`
internal action	τ	`tau`
left merge	$\lfloor\!\lfloor$	`||_`
sum	$\sum_{x:s} p$	`sum x:s.p`
allow	$\nabla_{\{\, a\|b \,\}}(p)$	`allow({a|b},p)`
block	$\partial_{\{\, a \,\}}(p)$	`block({a},p)`
hide	$\tau_{\{\, a \,\}}(p)$	`hide({a},p)`
rename	$\rho_{\{\, a \rightarrow b \,\}}(p)$	`rename({a -> b},p)`
communication	$\Gamma_{\{\, a\|b \rightarrow c \,\}}(p)$	`comm({a|b -> c},p)`
time	$^{\backprime}, \gg, \ll$	`@`, `>>`, `<<`
negation of ultimate delay	∇	`yaled`
ultimate delay	Δ	`delay`
nil	ϵ	`nil`
fixpoint symbol	ν, μ	`nu`, `mu`
maximal fixpoint	$\nu X(x : s = d).\varphi$	`nu X(x:s = d).phi`
minimal fixpoint	$\mu X(x : s = d).\varphi$	`mu X(x:s = d).phi`

Table 1: Mapping from rich to plain text