

# 1 Capture avoiding substitutions

This document specifies how capture avoiding substitutions are currently implemented in mCRL2.

## 1.1 Data expressions

mCRL2 data expressions  $x$  are characterized by the following grammar:

$$x ::= v \mid f \mid x(x) \mid x \text{ whr } v = x \mid \forall_v.x \mid \exists_v.x \mid \lambda_v.x,$$

where  $v$  is a variable and where  $f$  is a function symbol<sup>1</sup>.

## 1.2 Substitutions

A substitution  $\sigma$  is a function that maps variables to expressions. It is assumed that  $\sigma$  has finite support, in other words there is a finite number of variables  $v$  for which  $\sigma(v) \neq v$ . We define the substitution update  $\sigma[v := v']$  as follows:

$$\sigma[v := v'](w) = \begin{cases} v' & \text{if } w = v \\ \sigma(w) & \text{otherwise} \end{cases}$$

## 1.3 Capture avoiding substitutions

Let  $\sigma$  be a substitution that maps variables to data expressions, and let  $x$  be an arbitrary data expression. Let  $FV(x)$  be the free variables in  $x$ , and let  $FV(\sigma)$  be the free variables in the right hand side of  $\sigma$ . More precisely,

$$FV(\sigma) = \bigcup_{v \in \text{domain}(\sigma)} FV(\sigma(v)) \setminus \{v\}.$$

We define a function  $C$  that computes the capture avoiding substitution  $\sigma(x)$  using  $C(x, \sigma, FV(x) \cup FV(\sigma))$ . The function  $C$  is recursively defined as follows:

$$\begin{aligned} C(v, \sigma, V) &= \sigma(v) \\ C(f, \sigma, V) &= f \\ C(x(x_1), \sigma, V) &= C(x, \sigma, V)(C(x_1, \sigma, V)) \\ C(x \text{ whr } v = x_1, \sigma, V) &= \begin{cases} C(x, \sigma, V \cup \{v\}) \text{ whr } v = C(x_1, \sigma, V \cup \{v\}) & \text{if } \sigma(v) = v \text{ and } v \notin V \\ C(x, \sigma', V \cup \{v'\}) \text{ whr } v' = C(x_1, \sigma', V \cup \{v'\}) & \text{otherwise} \end{cases} \\ C(\lambda v.x, \sigma, V) &= \begin{cases} \lambda v.C(x, \sigma, V \cup \{v\}) & \text{if } \sigma(v) = v \text{ and } v \notin V \\ \lambda v'.C(x, \sigma', V \cup \{v'\}) & \text{otherwise,} \end{cases} \end{aligned}$$

---

<sup>1</sup>For simplicity we use only single arguments in function applications, and single variables in binding expressions. It is straightforward to generalize this to multiple arguments and multiple variables.

where  $\Lambda \in \{\forall, \exists, \lambda\}$ , where  $v'$  is an arbitrary variable such that  $\sigma(v') = v'$  and  $v' \notin V$ , and where  $\sigma' = \sigma[v := v']$ . The function  $C$  can be extended to assignments as follows<sup>2</sup>:

$$C(v = x, \sigma, V) = \begin{cases} v = C(x, \sigma, V \cup \{v\}) & \text{if } \sigma(v) = v \text{ and } v \notin V \\ v' = C(x, \sigma', V \cup \{v'\}) & \text{otherwise} \end{cases}$$

**Example** Let  $x = \forall b:\mathbb{B}.b \Rightarrow \forall c:\mathbb{B}.c \Rightarrow d$  and let  $\sigma = [d := b]$ . Then  $C(x, \sigma, FV(x) \cup FV(\sigma)) = \forall b':\mathbb{B}.b' \Rightarrow \forall c:\mathbb{B}.c \Rightarrow b$ .

### 1.3.1 Capture avoiding substitutions with an identifier generator

Let  $\sigma$  be a substitution that maps variables to data expressions. In this section a substitution is defined that is more efficient than the capture avoiding substitution of section 1.3 because it does not require the calculation of a set  $V$  of variables.

It does require that  $\sigma$  can indicate efficiently whether a variable occurs in the  $\sigma(y)$  (with  $\sigma(y) \neq y$ ) for some variable  $y$ . Furthermore, it requires a identifier generator, that can generate variable names that are guaranteed to be fresh in the sense that they do not occur in any term.

This substitution has been implemented as `replace_variables_capture_avoiding_with_an_identifier_g`. We use  $FV(x)$ ,  $FV(\sigma)$  and  $\sigma[v := v']$  as defined in the previous section.

The substitution is defined as  $\hat{C}$  that calculates  $\sigma(x)$  using  $\hat{C}(x, \sigma)$  recursively as follows:

$$\begin{aligned} \hat{C}(v, \sigma) &= \sigma(v) \\ \hat{C}(f, \sigma) &= f \\ \hat{C}(x(x_1), \sigma) &= \hat{C}(x, \sigma)(\hat{C}(x_1, \sigma)) \\ \hat{C}(x \text{ whr } v = x_1, \sigma) &= \begin{cases} \hat{C}(x, \sigma[v := v]) \text{ whr } v = \hat{C}(x_1, \sigma) & \text{if } v \notin FV(\sigma), \\ \hat{C}(x, \sigma[v := v']) \text{ whr } v' = \hat{C}(x_1, \sigma') & \text{otherwise.} \end{cases} \\ \hat{C}(\Lambda v.x, \sigma, V) &= \begin{cases} \Lambda v.\hat{C}(x, \sigma[v := v]) & \text{if } v \notin FV(\sigma), \\ \Lambda v'.\hat{C}(x, \sigma', V \cup \{v'\}) & \text{otherwise,} \end{cases} \end{aligned}$$

where  $\Lambda \in \{\forall, \exists, \lambda\}$ , where  $v'$  is a fresh variable such that  $\sigma(v') = v'$  and  $v' \notin FV(\sigma) \cup FV(x)$ . The identifier generator is used to generate the name for  $v'$ .

In the examples below  $[]$  is the substitution mapping each variable onto itself and  $[w := v']$  is the substitution mapping all variables onto itself, except that  $w$  is mapped to  $v'$ .

**Example** Let  $x = \forall b:\mathbb{B}.b \Rightarrow \forall c:\mathbb{B}.c \Rightarrow d$  and let  $\sigma = [d := b]$ . Then  $\hat{C}(x, \sigma) = \forall b':\mathbb{B}.b' \Rightarrow \forall c:\mathbb{B}.c \Rightarrow b$  where  $b'$  is a fresh variable.

<sup>2</sup>The definition of  $C$  to assignments is not correct and not how they have been implemented.

**Example** It is necessary that  $v'$  above is chosen such that  $v' \notin FV(\sigma) \cup FV(x)$ . We provide two examples to show what goes wrong if this condition is not satisfied.

1. If  $v' \notin FV(\sigma)$  is not required, the following is possible:  $\hat{C}(\forall v.w, [w := v']) = \forall v'.\hat{C}(w, [w := v']) = \forall v'.v'$ .
2. If  $v' \notin FV(x)$  is not required, it is possible that:  $\hat{C}(\forall v, v', []) = \forall v'.\hat{C}(v', []) = \forall v'.v$ .

**Example** In a where clause the substitutions applied to the equations after the where can remain unchanged. E.g.,  $\hat{C}(f(u, v) \text{ whr } v = v, [u := v]) = \hat{C}(f(u, v), [u := v, v := v']) \text{ whr } v' = \hat{C}(v, [u := v]) = f(v, v') \text{ whr } v' = v$ . In an expression  $f(u, v) \text{ whr } v = v$  the variable  $v$  at the lhs of the '=' is a local variable, whereas the  $v$  at the rhs is globally bound.