Binding and Substitution

Susmit Sarkar and Peter Sewell and Francesco Zappa Nardelli

October 18, 2007

We explain the binding specifications used in Ott in this document. The explanation is for a fragment of Ott, not including the list forms. The concrete substitution and free variable functions generated by Ott are defined in this document. We also give a more mathematical treatment by defining a notion of alpha-equivalence on abstract syntax terms for arbitrary binding specification. We then prove that under appropriate conditions, the concrete substitution functions respect alpha-equivalence. Further, we show that concrete substitutions coincides with capture-avoiding substitutions.

1 Grammar

The metavariables used in the following are:
index, i, i', j, j', k, k', l, l', m, m', n, n', o, o', q, q'
terminal, t
metavarroot, *mvr*
nontermroot, *ntr*
suffix, *suff*
variable, *var*
auxfn, f
prodname, *pn*

The grammar of mini-Ott is:

metavar, *mv*	::=		
	|	*metavarroot suffix*	
nonterm, *nt*	::=		
	|	*nontermroot suffix*	
element, *e*	::=		Elements of production rules
	|	*terminal*	
	|	*metavar*	
	|	*nonterm*	
mse	::=		Metavariable set expressions
	|	$\{\}$	
	|	$mse \cup mse'$	
	|	*metavar*	
	|	*auxfn*(*nonterm*)	
mses	::=		Lists of metavariable set expressions
	|	$[\, mse_1 \,..\, mse_n \,]$	

$$
\begin{array}{llll}
\mathit{bindspec},\ \mathit{bs} & ::= & & \\
 & | & \mathtt{bind}\ \mathit{mse}\ \mathtt{in}\ \mathit{nonterm} & \\
 & | & \mathit{auxfn}\ =\ \mathit{mse} & \\
 & & & \\
\mathit{prod},\ \mathit{p} & ::= & & \text{Grammar productions} \\
 & | & \mathtt{|}\ \mathit{element}_1\ ..\ \mathit{element}_m\ \mathtt{::}\ \mathtt{::}\ \mathit{prodname}\ \mathtt{(+}\ \mathit{bindspec}_1\ ..\ \mathit{bindspec}_n\ \mathtt{+)} & \\
 & & & \\
\mathit{rule},\ \mathit{r} & ::= & & \\
 & | & \mathit{nontermroot}\ \mathtt{::}\ \mathtt{''}\ \mathtt{::=}\ \mathit{prod}_1\ ..\ \mathit{prod}_m & \\
 & & & \\
\mathit{grammar\_rules},\ \mathit{g} & ::= & & \\
 & | & \mathtt{grammar}\ \mathit{rule}_1\ ..\ \mathit{rule}_m &
\end{array}
$$

Abstract syntax terms from the above grammar with concrete variables are defined as

$$
\begin{array}{lll}
\mathit{sorted\_var},\ \mathit{v} & ::= & \\
 & | & \mathit{var}{:}\mathit{mvr} \\
 & & \\
\mathit{concrete\_abstract\_syntax\_term},\ \mathit{cast} & ::= & \\
 & | & \mathit{v} \\
 & | & \mathit{prodname}\ (\ \mathit{cast}_1\ ,\ ..\ ,\ \mathit{cast}_m\ )
\end{array}
$$

We will have a simple type structure on the above grammar to ensure some basic sanity properties. These use the following types:

$$
\begin{array}{lll}
\mathit{auxfn\_type},\ \mathit{aut} & ::= & \\
 & | & \mathit{nontermroot}_1\ ..\ \mathit{nontermroot}_n\ \rightarrow\ \mathit{metavarroot} \\
 & & \\
\mathit{auxfn\_type\_env},\ \Phi & ::= & \\
 & & \\
\mathit{cast\_type},\ \mathit{ct} & ::= & \\
 & | & \mathit{ntr} \\
 & | & \mathit{mvr}
\end{array}
$$

Our substitutions are defined as a list of simultaneous substitutions of *cast* for *var* at particular sorts:

$$
\begin{array}{lll}
\mathit{substitution},\ \mathit{s} & ::= & \\
 & | & \{\ \mathit{cast}_1\ /\ \mathit{v}_1\ ,\ ..\ ,\ \mathit{cast}_n\ /\ \mathit{v}_n\ \}
\end{array}
$$

Finally, our interpretations will use the following auxiliary notions:

$$
\begin{array}{llll}
\mathit{ntmv} & ::= & & \\
 & | & \mathit{nt} & \\
 & | & \mathit{mv} & \\
 & & & \\
\mathit{ntmv\_list} & ::= & & \\
 & | & \mathit{ntmv}_1\ ,\ ..\ ,\ \mathit{ntmv}_l & \\
 & & & \\
\mathit{occurence},\ \mathit{oc} & ::= & & \text{Lists of natural number indices, picking out path in syntax tree} \\
 & | & [\,] & \\
 & | & \mathit{n}\ \mathtt{::}\ \mathit{oc} & \\
 & & & \\
\mathit{oc\_set} & ::= & & \text{Sets of occurences} \\
 & & & \\
\mathit{oc\_reln} & ::= & & \text{PER's over occurences, represented as sets of sets of occurences}
\end{array}
$$

var_set ::= Sets of sorted variables

2 Rules

2.1 Utility functions and relations

We first define some utility relations (many of which are in fact defining primitive recursive partial functions) to work on syntactic objects.

$\boxed{p \in g\,(\,ntr\,)}$

$$\frac{\begin{array}{l} 1:\ g = \texttt{grammar}\, r_1 \,..\, r_l \\ 2:\ r_i = ntr\, \texttt{::}\, \texttt{''}\, \texttt{::=}\, p_1 \,..\, p_m \\ 3:\ j\ \mathbf{INDEXES}\ p_1 \,..\, p_m \end{array}}{p_j \in g\,(\,ntr\,)} \quad \text{FUNSPEC\_LOOKUP\_P\_PRODS}$$

$\boxed{\mathbf{remove\_suffix}\,(\,ntmv\,) = ct}$

$$\frac{}{\mathbf{remove\_suffix}\,(\,ntr\ suff\,) = ntr} \quad \text{FUNSPEC\_REMOVE\_SUFFIX\_NT}$$

$$\frac{}{\mathbf{remove\_suffix}\,(\,mvr\ suff\,) = mvr} \quad \text{FUNSPEC\_REMOVE\_SUFFIX\_MV}$$

$\boxed{\mathbf{remove\_terminals}\,(\,e_1 \,..\, e_n\,) = ntmv\_list}$

$$\frac{}{\mathbf{remove\_terminals}\,(\ ) = [\,]} \quad \text{FUNSPEC\_REMOVE\_TERMINALS\_NIL}$$

$$\frac{1:\ \mathbf{remove\_terminals}\,(\,e_1 \,..\, e_n\,) = ntmv\_list}{\mathbf{remove\_terminals}\,(\,t_m\ e_1 \,..\, e_n\,) = ntmv\_list} \quad \text{FUNSPEC\_REMOVE\_TERMINALS\_TM}$$

$$\frac{1:\ \mathbf{remove\_terminals}\,(\,e_1 \,..\, e_n\,) = ntmv\_list}{\mathbf{remove\_terminals}\,(\,mv\ e_1 \,..\, e_n\,) = mv\,,\ ntmv\_list} \quad \text{FUNSPEC\_REMOVE\_TERMINALS\_MV}$$

$$\frac{1:\ \mathbf{remove\_terminals}\,(\,e_1 \,..\, e_n\,) = ntmv\_list}{\mathbf{remove\_terminals}\,(\,nt\ e_1 \,..\, e_n\,) = nt\,,\ ntmv\_list} \quad \text{FUNSPEC\_REMOVE\_TERMINALS\_NT}$$

$\boxed{\mathbf{binding\_mses}\,\forall j.\,(bs_j) \in ntmv \Rightarrow mses}$

$$\frac{}{\mathbf{binding\_mses}\ \ \in nt \Rightarrow [\ ]} \quad \text{FUNSPEC\_BINDING\_MSES\_NT\_NIL}$$

$$\frac{1:\ \mathbf{binding\_mses}\,\forall j.\,(bs_j) \in nt \Rightarrow mses}{\mathbf{binding\_mses}\,\texttt{bind}\ mse\ \texttt{in}\ nt\,\forall j.\,(bs_j) \in nt \Rightarrow mse\ mses} \quad \text{FUNSPEC\_BINDING\_MSES\_NT\_CONS\_T}$$

$$\frac{\begin{array}{l} 1:\ \mathbf{binding\_mses}\,\forall j.\,(bs_j) \in nt \Rightarrow mses \\ 2:\ \neg\,(\,\exists mse,\ bs = \texttt{bind}\ mse\ \texttt{in}\ nt\,) \end{array}}{\mathbf{binding\_mses}\ bs\,\forall j.\,(bs_j) \in nt \Rightarrow mses} \quad \text{FUNSPEC\_BINDING\_MSES\_NT\_CONS\_F}$$

$$\frac{}{\mathbf{binding\_mses}\,\forall j.\,(bs_j) \in mv \Rightarrow [\ ]} \quad \text{FUNSPEC\_BINDING\_MSES\_MV}$$

$\boxed{cast@oc = cast'}$

$$\frac{}{cast@[\,] = cast} \quad \text{FUNSPEC\_TERM\_AT\_NIL}$$

$$\frac{1:\ cast_i@oc = cast}{pn\,(\,\forall j.\,(\,cast_j\,)\,)@i :: oc = cast} \quad \text{FUNSPEC\_TERM\_AT\_CONS}$$

$\boxed{cast \simeq cast'\ \mathbf{at}\ oc}$

$$\frac{\begin{array}{l}1: \ cast_1 @ oc = v \\ 2: \ cast_2 @ oc = v\end{array}}{cast_1 \ \simeq \ cast_2 \ \textbf{at} \ oc} \quad \text{NODE\_IDENT\_VAR}$$

$$\frac{\begin{array}{l}1: \ cast''_1 @ oc = pn \, (\forall i. \, (cast_i)) \\ 2: \ cast''_2 @ oc = pn \, (\forall i. \, (cast'_i))\end{array}}{cast''_1 \ \simeq \ cast''_2 \ \textbf{at} \ oc} \quad \text{NODE\_IDENT\_APP}$$

We also define several relations that characterise operations on occurences, sets thereof and relations thereupon.

$\boxed{\textbf{head} \ oc \ = \ i}$

$$\frac{}{\textbf{head} \, (\, i :: oc \,) \ = \ i} \quad \text{FUNSPEC\_OC\_HEAD\_T}$$

$\boxed{oc\_set \ \textbf{subset} \ oc\_set'}$
Axiomatic definition of the subset relation on sets of occurences

$$\frac{1: \ \forall oc, \ (\, oc \ \in \ oc\_set \ \Rightarrow \ oc \ \in \ oc\_set' \,)}{oc\_set \ \textbf{subset} \ oc\_set'} \quad \text{OC\_SUBSET\_DEF}$$

$\boxed{oc \ \in \ \textbf{support} \ oc\_reln}$
Whether an occurence is in the support of a relation

$$\frac{\begin{array}{l}1: \ oc \ \in \ oc\_set \\ 2: \ oc\_set \ \in \ oc\_reln\end{array}}{oc \ \in \ \textbf{support} \ oc\_reln} \quad \text{OC\_IN\_IN\_DEF}$$

$\boxed{\textbf{is\_PER} \ oc\_reln}$
A oc_reln is a proper PER (partial equivalence relation) iff it is a set of disjoint sets (these sets are the equivalence classes).

$$\frac{1: \ \forall oc\_set, \ \forall oc\_set', \ (\, (\, oc\_set \ \in \ oc\_reln \ \wedge \ oc\_set' \ \in \ oc\_reln \,) \ \Rightarrow \ \neg \, (\, \exists oc, \ (\, oc \ \in \ oc\_set \ \wedge \ oc \ \in \ oc\_set' \,) \,) \,)}{\textbf{is\_PER} \ oc\_reln} \quad \text{OC\_IS\_PE}$$

$\boxed{oc\_reln \ \textbf{refines} \ oc\_reln'}$
A PER oc_reln refines another oc_reln' iff being related by oc_reln implies being related by oc_reln'. Since oc_reln' relates more pairs of elements than oc_reln, oc_reln' is coarser than oc_reln.

$$\frac{1: \ \forall oc\_set, \ (\, oc\_set \ \in \ oc\_reln \ \Rightarrow \ \exists oc\_set', \ (\, oc\_set' \ \in \ oc\_reln' \ \wedge \ oc\_set \ \textbf{subset} \ oc\_set' \,) \,)}{oc\_reln \ \textbf{refines} \ oc\_reln'} \quad \text{OC\_REFINES\_DEF}$$

$\boxed{\textbf{union\_closure} \ oc\_reln_1 \ oc\_reln_2 \ = \ oc\_reln_3}$
The union-closure of oc_reln_1 and oc_reln_2 is the finest PER that is coarser than both oc_reln_1 and oc_reln_2. In other words (if equivalence relations are considered as traditional sets of pairs), it is the smallest partial equivalence relation containing the union of oc_reln_1 and oc_reln_2.

$$\frac{\begin{array}{l}1: \ \textbf{is\_PER} \ oc\_reln_3 \\ 2: \ oc\_reln_1 \ \textbf{refines} \ oc\_reln_3 \\ 3: \ oc\_reln_2 \ \textbf{refines} \ oc\_reln_3 \\ 4: \ \forall oc\_reln', \ (\, (\, \textbf{is\_PER} \ oc\_reln' \ \wedge \ (\, oc\_reln_1 \ \textbf{refines} \ oc\_reln' \ \wedge \ oc\_reln_2 \ \textbf{refines} \ oc\_reln' \,) \,) \ \Rightarrow \ oc\_reln_3 \ \textbf{refines} \ oc\_reln' \,)\end{array}}{\textbf{union\_closure} \ oc\_reln_1 \ oc\_reln_2 \ = \ oc\_reln_3}$$

`<< no parses (char 48):  :oc_reln_select:  oc_reln = { oc_set in oc_reln' ***'|' formula } >>`

$$\frac{\begin{array}{l}1:\ \forall oc\_set,\ (\,oc\_set\ \in\ oc\_reln\ \Rightarrow\ oc\_set\ \in\ oc\_reln'\,)\\ 2:\ \forall oc\_set,\ (\,oc\_set\ \in\ oc\_reln\ \Rightarrow\ formula\,)\\ 3:\ \forall oc\_set,\ (\,(\,oc\_set\ \in\ oc\_reln'\ \wedge\ formula\,)\ \Rightarrow\ oc\_set\ \in\ oc\_reln\,)\end{array}}{oc\_reln\ =\ \{\,oc\_set\ \in\ oc\_reln'\mid formula\,\}}\quad\text{OC\_RELN\_SELECT\_DEF}$$

$\boxed{oc\_set\ =\ \textbf{eponymous}\ oc\ cast}$

$$\frac{\begin{array}{l}1:\ cast@oc = v\\ 2:\ \forall oc',\ (\,oc'\ \in\ oc\_set\ \Leftrightarrow\ cast@oc' = v\,)\end{array}}{oc\_set\ =\ \textbf{eponymous}\ oc\ cast}\quad\text{OC\_SET\_EPONYMOUS\_DEF}$$

2.2 Sanity checks

We will define typing judgements that check for sanity properties, such as the fact that production names are not repeated for different productions of a non-terminal, that there is a unique definition for a non-terminal, and that if an auxiliary function is defined for a non-terminal, every production clause must define the function.

$\boxed{\Phi\vdash f\ :\ aut}$

$$\frac{}{\Phi, f : aut\ \vdash\ f\ :\ aut}\quad\text{SANITY\_AUXFN\_HEAD}$$

$$\frac{\begin{array}{l}1:\ \Phi\vdash f\ :\ aut\\ 2:\ \neg(\,f\ =\ f'\,)\end{array}}{\Phi, f' : aut'\ \vdash\ f\ :\ aut}\quad\text{SANITY\_AUXFN\_SKIP}$$

$\boxed{\Phi\,;\ e_1\,..\,e_n\ \vdash\ mse\ :\ metavarroot}$

$$\frac{}{\Phi\,;\ e_1\,..\,e_n\ \vdash\ \{\}\ \ :\ mvr}\quad\text{SANITY\_MSE\_EMPTY}$$

$$\frac{\begin{array}{l}1:\ \Phi\,;\ e_1\,..\,e_n\ \vdash\ mse\ :\ mvr\\ 2:\ \Phi\,;\ e_1\,..\,e_n\ \vdash\ mse'\ :\ mvr\end{array}}{\Phi\,;\ e_1\,..\,e_n\ \vdash\ mse\ \cup\ mse'\ :\ mvr}\quad\text{SANITY\_MSE\_UNION}$$

$$\frac{1:\ \exists! j\in 1..n.e_j\ =\ mvr\ suff}{\Phi\,;\ e_1\,..\,e_n\ \vdash\ mvr\ suff\ :\ mvr}\quad\text{SANITY\_MSE\_MV}$$

$$\frac{\begin{array}{l}1:\ \Phi\vdash f\ :\ ntr_1\,..\,ntr_m\ \rightarrow\ mvr\\ 2:\ \exists! j\in 1..n.(\,e_j\ =\ nt\ \wedge\ nt\ =\ ntr_i\ suff\,)\end{array}}{\Phi\,;\ e_1\,..\,e_n\ \vdash\ f\,(nt)\ :\ mvr}\quad\text{SANITY\_MSE\_F}$$

$\boxed{\Phi\,;\ e_1\,..\,e_n\ :\ ntr\ \vdash\ bs\,\textbf{ok}}$

$$\frac{\begin{array}{l}1:\ \Phi\,;\ e_1\,..\,e_n\ \vdash\ mse\ :\ mvr\\ 2:\ \exists! j\in 1..n.e_j\ =\ nt\end{array}}{\Phi\,;\ e_1\,..\,e_n\ :\ ntr\ \vdash\ \texttt{bind}\ \ mse\ \ \texttt{in}\ \ nt\,\textbf{ok}}\quad\text{SANITY\_BS\_BIND}$$

$$\frac{\begin{array}{l}1:\ \Phi\,;\ e_1\,..\,e_n\ \vdash\ mse\ :\ mvr\\ 2:\ \Phi\vdash f\ :\ ntr_1\,..\,ntr_n\ \rightarrow\ mvr\\ 3:\ ntr\ =\ ntr_i\end{array}}{\Phi\,;\ e_1\,..\,e_n\ :\ ntr\ \vdash\ f\ =\ mse\,\textbf{ok}}\quad\text{SANITY\_BS\_AUXFN}$$

$\boxed{\Phi\vdash prod\ :\ ntr}$

$$\frac{\begin{array}{l}1:\ \forall i\in 1..m.\Phi\,;\ e_1\,..\,e_n\ :\ ntr\ \vdash\ bs_i\,\textbf{ok}\\ 2:\ prod\ =\ \mid e_1\,..\,e_n\ \texttt{::}\ \texttt{::}\ prodname\ \texttt{(+}\,bs_1\,..\,bs_m\,\texttt{+)}\\ 3:\ \forall f\in\mathrm{dom}(\textbf{Phi}),\ (\,\Phi\vdash f\ :\ ntr_1\,..\,ntr_q\ ntr\ ntr'_1\,..\,ntr'_{q'}\ \rightarrow\ mvr\ \Rightarrow\ \exists! i\in 1..m.\exists mse,\ bs_i\ =\ f\ =\ mse\,)\end{array}}{\Phi\vdash prod\ :\ ntr}\quad\text{SANITY\_PROD\_}$$

$$\boxed{\Phi \vdash \mathit{rule}\ \mathbf{ok}}$$

$$\frac{\begin{array}{l} 1:\ \mathit{rule} = \mathit{ntr} :: \texttt{''} ::= \mathit{prod}_1 \mathinner{..} \mathit{prod}_m \\ 2:\ \forall i \in 1..m.\Phi \vdash \mathit{prod}_i : \mathit{ntr} \\ 3:\ \forall i \in 1..m.\forall j \in 1..m.(\,(\mathit{prod}_i = \texttt{|}\ e_1 \mathinner{..} e_m :::: \mathit{pn}\ \texttt{(+}\ \mathit{bs}_1 \mathinner{..} \mathit{bs}_n\ \texttt{+)} \wedge \mathit{prod}_j = \texttt{|}\ e'_1 \mathinner{..} e'_{m'} :::: \mathit{pn}\ \texttt{(+}\ \mathit{bs}'_1 \mathinner{..} \mathit{bs}'_{n'}\ \texttt{+)}\,) \Rightarrow i = j \end{array}}{\Phi \vdash \mathit{rule}\ \mathbf{ok}}$$

$$\boxed{\Phi \vdash \mathit{grammar\_rules}\ \mathbf{ok}}$$

$$\frac{\begin{array}{l} 1:\ \mathit{grammar\_rules} = \texttt{grammar}\ \mathit{rule}_1 \mathinner{..} \mathit{rule}_m \\ 2:\ \forall i \in 1..m.\Phi \vdash \mathit{rule}_i\ \mathbf{ok} \\ 3:\ \forall i \in 1..m.\forall j \in 1..m.(\,(\mathit{rule}_i = \mathit{ntr} :: \texttt{''} ::= \mathit{prod}_1 \mathinner{..} \mathit{prod}_m \wedge \mathit{rule}_j = \mathit{ntr} :: \texttt{''} ::= \mathit{prod}'_1 \mathinner{..} \mathit{prod}'_n\,) \Rightarrow i = j\,) \end{array}}{\Phi \vdash \mathit{grammar\_rules}\ \mathbf{ok}} \quad \text{SANIT}$$

$$\boxed{g \vdash \mathit{cast} : \mathit{cast\_type}}$$

$$\frac{}{g \vdash v : \mathit{mvr}} \quad \text{SANITY\_CAST\_VAR}$$

$$\frac{\begin{array}{l} 1:\ \texttt{|}\ e_1 \mathinner{..} e_n :::: \mathit{pn}\ \texttt{(+}\ \mathit{bs}_1 \mathinner{..} \mathit{bs}_o\ \texttt{+)} \in g(\mathit{ntr}) \\ 2:\ \mathbf{remove\_terminals}(e_1 \mathinner{..} e_n) = \mathit{ntmv}_1, .., \mathit{ntmv}_q \\ 3:\ \mathbf{remove\_suffix}(\mathit{ntmv}_1) = \mathit{ct}_1 \quad .. \quad \mathbf{remove\_suffix}(\mathit{ntmv}_q) = \mathit{ct}_q \\ 4:\ g \vdash \mathit{cast}_1 : \mathit{ct}_1 \quad .. \quad g \vdash \mathit{cast}_q : \mathit{ct}_q \end{array}}{g \vdash \mathit{pn}(\mathit{cast}_1, .., \mathit{cast}_q) : \mathit{ntr}} \quad \text{SANITY\_CAST\_APP}$$

$$\boxed{v \in \mathbf{dom}(s)}$$

$$\frac{}{v \in \mathbf{dom}(\{\forall i.(\mathit{cast}_i / v_i), \mathit{cast} / v, \forall j.\left(\mathit{cast}'_j / v'_j\right)\})} \quad \text{SANITY\_INDOM\_LIST}$$

2.3 Concrete substitutions

Now, for well-typed grammar rules, we are ready to give rules for the concrete interpretation of *mse* on a *cast*, which is the set of variables picked out by *mse* on *cast*.

$$\boxed{\mathbf{concrete}[\![\mathit{mse}]\!]\, g(\mathit{cast}) \Rightarrow \mathit{var\_set}}$$

$$\frac{}{\mathbf{concrete}[\![\{\}\ ]\!]\, g(\mathit{cast}) \Rightarrow \{\}} \quad \text{FUNSPEC\_CONCRETE\_EMPTY}$$

$$\frac{\begin{array}{l} 1:\ \mathbf{concrete}[\![\mathit{mse}]\!]\, g(\mathit{cast}) \Rightarrow \mathit{var\_set} \\ 2:\ \mathbf{concrete}[\![\mathit{mse}']\!]\, g(\mathit{cast}) \Rightarrow \mathit{var\_set}' \end{array}}{\mathbf{concrete}[\![\mathit{mse} \cup \mathit{mse}']\!]\, g(\mathit{cast}) \Rightarrow \mathit{var\_set} \cup \mathit{var\_set}'} \quad \text{FUNSPEC\_CONCRETE\_UNION}$$

$$\frac{\begin{array}{l} 1:\ \texttt{|}\ e_1 \mathinner{..} e_n :::: \mathit{pn}\ \texttt{(+}\ \mathit{bs}_1 \mathinner{..} \mathit{bs}_o\ \texttt{+)} \in g(\mathit{ntr}) \\ 2:\ \mathbf{remove\_terminals}(e_1 \mathinner{..} e_n) = \mathit{ntmv}_1, .., \mathit{ntmv}_q \\ 3:\ \mathit{ntmv}_l = \mathit{mv} \\ 4:\ \mathit{cast}_l = v' \end{array}}{\mathbf{concrete}[\![\mathit{mv}]\!]\, g(\mathit{pn}(\mathit{cast}_1, .., \mathit{cast}_q)) \Rightarrow \{v'\}} \quad \text{FUNSPEC\_CONCRETE\_MV}$$

$$\frac{\begin{array}{l} 1:\ \texttt{|}\ e_1 \mathinner{..} e_n :::: \mathit{pn}\ \texttt{(+}\ \mathit{bs}_1 \mathinner{..} \mathit{bs}_o\ \texttt{+)} \in g(\mathit{ntr}) \\ 2:\ \mathbf{remove\_terminals}(e_1 \mathinner{..} e_n) = \mathit{ntmv}_1, .., \mathit{ntmv}_q \\ 3:\ \mathit{ntmv}_l = \mathit{nt} \\ 4:\ \mathit{nt} = \mathit{ntr}'\, \mathit{suff}' \\ 5:\ \mathit{cast}_l = \mathit{pn}'(\mathit{cast}'_1, .., \mathit{cast}'_{q'}) \\ 6:\ \texttt{|}\ e'_1 \mathinner{..} e'_{n'} :::: \mathit{pn}'\ \texttt{(+}\ \mathit{bs}'_1 \mathinner{..} \mathit{bs}'_{o'}\ \texttt{+)} \in g(\mathit{ntr}') \\ 7:\ \mathit{bs}'_k = f = \mathit{mse}' \\ 8:\ \mathbf{concrete}[\![\mathit{mse}']\!]\, g(\mathit{cast}_l) \Rightarrow \mathit{var\_set} \end{array}}{\mathbf{concrete}[\![f\,(\mathit{nt})]\!]\, g(\mathit{pn}(\mathit{cast}_1, .., \mathit{cast}_q)) \Rightarrow \mathit{var\_set}} \quad \text{FUNSPEC\_CONCRETE\_F}$$

In the case that *mse* is a single metavariable *mv* (INTERP_MSE_CONC_3) , we pick out the variable at the corresponding position. In the case that *mse* is $f\,(nt)$ (INTERP_MSE_CONC_4), we look at the production for the non-terminal *nt* that we have, and perform the calculation of the auxfn definition.

With these interpretations in hand, we can define the concrete substitution and free variable functions generated by Ott.

$$\boxed{\mathbf{subst}\, s \in \mathit{cast} = \mathit{cast}''}$$

$$\frac{}{\mathbf{subst}\, s \in v = v} \quad \text{CONCRETE\_SUBST\_VAR}$$

$$\frac{}{\mathbf{subst}\,\{\, \forall i.\,(\mathit{cast}_i / v_i)\ ,\ \mathit{cast} / v\ ,\ \forall j.\,\left(\mathit{cast}'_j / v'_j\right)\,\} \in \mathit{pn}\,(\,v\,) = \mathit{cast}} \quad \text{CONCRETE\_SUBST\_IN}$$

$$\frac{1:\ \neg\,(\,v \in \mathbf{dom}\,(\,s\,)\,)}{\mathbf{subst}\, s \in \mathit{pn}\,(\,v\,) = \mathit{pn}\,(\,v\,)} \quad \text{CONCRETE\_SUBST\_OUT}$$

$$\frac{\begin{array}{ll} 1: & \mathit{cast} = \mathit{pn}\,(\forall k.\,(\mathit{cast}_k)\,) \\ 2: & \neg\,(\,\exists v,\ \mathit{cast} = \mathit{pn}\,(\,v\,)\,) \\ 3: & \mid e_1 \mathrel{..} e_n ::::\ \mathit{pn}\ (\texttt{+}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_m\, \texttt{+}) \in g\,(\,\mathit{ntr}\,) \\ 4: & \mathbf{remove\_terminals}\,(\,e_1 \mathrel{..} e_n\,) = \forall k.\,(\mathit{ntmv}_k) \\ 5: & \forall k.\,(\mathbf{binding\_mses}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_m \in \mathit{ntmv}_k \Rightarrow \mathit{mses}_k) \\ 6: & \mathbf{concrete}\, [\![\bigcup \mathit{mses}_k]\!]\, g\,(\,\mathit{cast}\,) \Rightarrow \mathit{var\_set}_k \\ 7: & \forall k.\,(s_k = \mathbf{filter}\, \mathit{var\_set}_k\, \mathbf{from}\, s) \\ 8: & \forall k.\,(\mathbf{subst}\, s_k \in \mathit{cast}_k = \mathit{cast}'_k) \\ 9: & \mathit{cast}' = \mathit{pn}\,(\forall k.\,(\mathit{cast}'_k)\,) \end{array}}{\mathbf{subst}\, s \in \mathit{cast} = \mathit{cast}'} \quad \text{CONCRETE\_SUBST\_APP}$$

$$\boxed{\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ \mathit{cast} = \mathit{var\_set}}$$

$$\frac{\begin{array}{ll} 1: & \mid e_1 \mathrel{..} e_n ::::\ \mathit{pn}\ (\texttt{+}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_o\, \texttt{+}) \in g\,(\,\mathit{ntr}'\,) \\ 2: & \neg\,(\,\mathit{ntr} = \mathit{ntr}'\,) \end{array}}{\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ \mathit{pn}\,(\,v\,) = \{\}} \quad \text{CONCRETE\_FV\_OUT\_NT}$$

$$\frac{\begin{array}{ll} 1: & \mid e_1 \mathrel{..} e_n ::::\ \mathit{pn}\ (\texttt{+}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_o\, \texttt{+}) \in g\,(\,\mathit{ntr}\,) \\ 2: & \neg\,(\,\mathit{mvr} = \mathit{mvr}'\,) \end{array}}{\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ \mathit{pn}\,(\,v'\,) = \{\}} \quad \text{CONCRETE\_FV\_OUT\_MV}$$

$$\frac{1:\ \mid e_1 \mathrel{..} e_n ::::\ \mathit{pn}\ (\texttt{+}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_o\, \texttt{+}) \in g\,(\,\mathit{ntr}\,)}{\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ \mathit{pn}\,(\,v\,) = \{\,v\,\}} \quad \text{CONCRETE\_FV\_IN}$$

$$\frac{}{\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ v = \{\}} \quad \text{CONCRETE\_FV\_VAR}$$

$$\frac{\begin{array}{ll} 1: & \mathit{cast} = \mathit{pn}\,(\forall k.\,(\mathit{cast}_k)\,) \\ 2: & \neg\,(\,\exists v,\ \mathit{cast} = \mathit{pn}\,(\,v\,)\,) \\ 3: & \mid e_1 \mathrel{..} e_n ::::\ \mathit{pn}\ (\texttt{+}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_m\, \texttt{+}) \in g\,(\,\mathit{ntr}\,) \\ 4: & \mathbf{remove\_terminals}\,(\,e_1 \mathrel{..} e_n\,) = \forall k.\,(\mathit{ntmv}_k) \\ 5: & \forall k.\,(\mathbf{binding\_mses}\, \mathit{bs}_1 \mathrel{..} \mathit{bs}_m \in \mathit{ntmv}_k \Rightarrow \mathit{mses}_k) \\ 6: & \forall k.\,(\mathbf{concrete}\, [\![\bigcup \mathit{mses}_k]\!]\, g\,(\,\mathit{cast}\,) \Rightarrow \mathit{var\_set}_k) \\ 7: & \forall k.\,(\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ \mathit{cast}_k = \mathit{var\_set}'_k) \\ 8: & \mathit{var\_set} = \bigcup \{\, \forall k.\,(\mathit{var\_set}'_k - \mathit{var\_set}_k)\,\} \end{array}}{\mathbf{fv}\ \mathit{ntr}\ \mathit{mvr}\ \mathbf{of}\ \mathit{cast} = \mathit{var\_set}} \quad \text{CONCRETE\_FV\_APP}$$

Notice that we treat only variables wrapped in a singleton constructor as free, or subject to substitution.

In the case that this is a compound term (SUBST_3), we first check to see that we do not fall into the singleton variable case. Next, we look up the production in our grammar, and remove the terminals to get elements that can appear in abstract terms. We look at all the bind clauses of the form `bind` mse' `in` *nt*, and filter out the interpretation of mse' from the substitution before applying it to the corresponding subterm.

2.4 Mathematical definition

We now turn to the definition of alpha-equivalence on the concrete terms. This is defined as follows:

$$\boxed{[\![\, mse \,]\!]\, g\,(\, cast \,) \;=\; oc\_set}$$

$$\frac{}{[\![\, \{\} \,]\!]\, g\,(\, cast \,) \;=\; \{\,\}} \quad \textsc{funspec\_interp\_mse\_empty}$$

$$\frac{\begin{array}{l} 1:\; [\![\, mse \,]\!]\, g\,(\, cast \,) \;=\; oc\_set \\ 2:\; [\![\, mse \,]\!]\, g\,(\, cast' \,) \;=\; oc\_set' \end{array}}{[\![\, mse \cup mse' \,]\!]\, g\,(\, cast \,) \;=\; oc\_set \cup oc\_set'} \quad \textsc{funspec\_interp\_mse\_union}$$

$$\frac{\begin{array}{l} 1:\; \mid e_1 \,..\, e_n \;::\;::\; pn\; (+\, bs_1 \,..\, bs_o \,+) \;\in\; g\,(\, ntr \,) \\ 2:\; \mathbf{remove\_terminals}\,(\, e_1 \,..\, e_n \,) \;=\; ntmv_1 \,,\, ..\,,\, ntmv_q \\ 3:\; ntmv_l \;=\; mv \\ 4:\; cast_l \;=\; v' \end{array}}{[\![\, mv \,]\!]\, g\,(\, pn\,(\, cast_1 \,,\, ..\,,\, cast_q \,)\,) \;=\; \{\, l :: [\,] \,\}} \quad \textsc{funspec\_interp\_mse\_mv}$$

$$\frac{\begin{array}{l} 1:\; \mid e_1 \,..\, e_n \;::\;::\; pn\; (+\, bs_1 \,..\, bs_o \,+) \;\in\; g\,(\, ntr \,) \\ 2:\; \mathbf{remove\_terminals}\,(\, e_1 \,..\, e_n \,) \;=\; ntmv_1 \,,\, ..\,,\, ntmv_q \\ 3:\; ntmv_l \;=\; nt \\ 4:\; nt \;=\; ntr'\; suff' \\ 5:\; cast_l \;=\; pn'\,(\, cast'_1 \,,\, ..\,,\, cast'_{q'} \,) \\ 6:\; \mid e'_1 \,..\, e'_{n'} \;::\;::\; pn'\; (+\, bs'_1 \,..\, bs'_{o'} \,+) \;\in\; g\,(\, ntr' \,) \\ 7:\; bs'_k \;=\; f \;=\; mse' \\ 8:\; [\![\, mse' \,]\!]\, g\,(\, cast_l \,) \;=\; oc\_set \end{array}}{[\![\, f\,(nt) \,]\!]\, g\,(\, pn\,(\, cast_1 \,,\, ..\,,\, cast_q \,)\,) \;=\; l :: oc\_set} \quad \textsc{funspec\_interp\_mse\_f}$$

$$\boxed{[\![\, auxfn \,]\!]\, g\,(\, cast \,) \;=\; oc\_set}$$

$$\frac{\begin{array}{l} 1:\; cast \;=\; pn\,(\, cast_1 \,,\, ..\,,\, cast_q \,) \\ 2:\; \mid e_1 \,..\, e_n \;::\;::\; pn\; (+\, bs_1 \,..\, bs_o \,+) \;\in\; g\,(\, ntr \,) \\ 3:\; bs_j \;=\; f \;=\; mse \\ 4:\; [\![\, mse \,]\!]\, g\,(\, cast \,) \;=\; oc\_set \end{array}}{[\![\, f \,]\!]\, g\,(\, cast \,) \;=\; oc\_set} \quad \textsc{funspec\_interp\_auxfn\_def}$$

$$\boxed{\Phi \vdash f\; \mathbf{accepts}\; ntr}$$

$$\frac{1:\; \Phi \vdash f \;:\; ntr_1 \,..\, ntr_m\; ntr\; ntr'_1 \,..\, ntr'_n \;\to\; mvr}{\Phi \vdash f\; \mathbf{accepts}\; ntr} \quad \textsc{f\_accepts\_arg}$$

$$\boxed{\Phi \vdash\; g\,(\, ntr \,)\; \mathbf{at}\; cast\; \mathbf{reveals}\; oc}$$

$$\frac{\begin{array}{l} 1:\; \Phi \vdash f\; \mathbf{accepts}\; ntr \\ 2:\; [\![\, f \,]\!]\, g\,(\, cast \,) \;=\; oc\_set \\ 3:\; oc \;\in\; oc\_set \end{array}}{\Phi \vdash\; g\,(\, ntr \,)\; \mathbf{at}\; cast\; \mathbf{reveals}\; oc} \quad \textsc{ntr\_reveals\_f}$$

$$\boxed{\Phi \vdash \mathbf{equiv\_both}\; g\,(\, cast \,) \;=\; \langle\, \mathbf{closed} \;:\; oc\_reln_1 \,,\, \mathbf{open} \;:\; oc\_reln_2 \,\rangle}$$

1 : $\Phi \vdash g\,\mathbf{ok}$

2 : $cast \;=\; pn\,(\forall i.\,(cast_i)\,)$

3 : $\mid e_1 \mathrel{..} e_n :: :: pn\; (\!+\, bs_1 \mathrel{..} bs_m \,+\!)\; \in\; g\,(\, ntr\,)$

4 : $\mathbf{remove\_terminals}\,(\, e_1 \mathrel{..} e_n\,) \;=\; \forall i.\,(ntmv_i)$

5 : $\forall i.\,(\mathbf{binding\_mses}\, bs_1 \mathrel{..} bs_m \;\in\; ntmv_i \;\Rightarrow\; mses_i)$

6 : $\forall i.\,([\![\bigcup mses_i]\!]\, g\,(\, cast\,) \;=\; oc\_set_i)$

7 : $\forall i.\,(\Phi \vdash \mathbf{equiv\_both}\, g\,(\, cast_i\,) \;=\; \langle\, \mathbf{closed} \;:\; oc\_reln_{1\,i}\,,\, \mathbf{open} \;:\; oc\_reln_{2\,i}\,\rangle)$

8 : $oc\_reln_1 \;=\; \bigcup\{\, \forall i.\,(\, i :: oc\_reln_{1\,i}\,)\;\}$

9 : $oc\_reln_2 \;=\; \bigcup\{\, \forall i.\,(\, i :: oc\_reln_{2\,i}\,)\;\}$

10 : $oc\_reln_3 \;=\; \{\,(\,(\,\mathbf{eponymous}\, oc_0\; cast \,\cap\, \bigcup\{\, \forall i.\,(oc\_set_i)\;\}\,)\,\cup\, \bigcup\{\, \forall i.\,(\{\, oc \;\in\; \mathbf{eponymous}\, oc_0\; cast \mid \mathbf{head}\; oc \;=\; i \;\wedge\; oc$

11 : $\mathbf{union\_closure}\; oc\_reln_2\; oc\_reln_3 \;=\; oc\_reln_4$

12 : $oc\_reln_5 \;=\; \{\, oc\_set' \;\in\; oc\_reln_4 \mid \exists oc,\; (\,\Phi \;\vdash\; g\,(\, ntr\,)\,\mathbf{at}\; cast\; \mathbf{reveals}\; oc \;\wedge\; oc \;\in\; oc\_set'\,)\,\}$

13 : $oc\_reln_6 \;=\; oc\_reln_1 \,\cup\, (\, oc\_reln_4 \;-\; oc\_reln_5\,)$

$$\Phi \vdash \mathbf{equiv\_both}\, g\,(\, cast\,) \;=\; \langle\, \mathbf{closed} \;:\; oc\_reln_6\,,\, \mathbf{ope}$$

For the definition of the equivalence classes (EQUIVS_CAST_1), we start in steps 1−−4 by looking up the production name in our grammar, and removing the terminals. In steps 5−−6, we extract the set of binding occurences as given by the bindspec clauses: oc_set_i is the set of binding occurences of variables that bind in subterm i. In step 7−−9, we recursively calculate closed and binding equivalence relations for all subterms.

In step 10, we iterate over all binding occurences oc_0 of any variable $var{:}mvr$, and build the equivalence class of that occurence. This equivalence class has the form $(C(var{:}mvr) \cup \bigcup_i (i :: D_i(var{:}mvr))) - U$, where:

- $C(var{:}mvr)$ is the set of binding occurences of $var{:}mvr$, computed as the occurences of $var{:}mvr$ (i.e., $\mathbf{eponymous}\; oc_0\; cast$) that are binding;
- $D_i(var{:}mvr)$ is the set of bound occurences of $var{:}mvr$ inside subterm i; $D(var{:}mvr)$ is built directly as the occurences of $var{:}mvr$ whose head is some i such that $var{:}mvr$ is bound in subterm i (note that the bindspec clause that makes $var{:}mvr$ bound in subterm i might mention an occurence oc which is different from oc_0);
- U is the set of occurences that are already bound in the subterm, as recorded in oc_reln_1.

In step 11, we take the equivalence closure of this set with the open binding sets of subterms. Finally, we pick out all equivalence classes such that they pick out something withi the domain of an auxfn (which means they can potentially be bound later), calling that the open bound set, and the remaining equivalence closures of bound variable occurences are called the closed binding set of this term.

$\boxed{\Phi\,;\, g \;\vdash\; cast_1 \;\equiv_\alpha\; cast_2}$

1 : $\Phi \vdash \mathbf{equiv\_both}\, g\,(\, cast_1\,) \;=\; \langle\, \mathbf{closed} \;:\; oc\_reln_1\,,\, \mathbf{open} \;:\; oc\_reln_2\,\rangle$

2 : $\Phi \vdash \mathbf{equiv\_both}\, g\,(\, cast_2\,) \;=\; \langle\, \mathbf{closed} \;:\; oc\_reln_3\,,\, \mathbf{open} \;:\; oc\_reln_4\,\rangle$

3 : $oc\_reln_1 \;=\; oc\_reln_3$

4 : $\forall oc,\; (\,(\,\neg\, oc \;\in\; \mathbf{support}\; oc\_reln_1\,) \;\Rightarrow\; (\,(\,\exists cast_1',\; cast_1 @ oc = cast_1'\,) \;\Leftrightarrow\; (\,\exists cast_2',\; cast_2 @ oc = cast_2'\,)\,)\,)$

5 : $\forall oc,\; (\,(\,(\,\neg\, oc \;\in\; \mathbf{support}\; oc\_reln_1\,) \;\wedge\; (\,(\,\exists cast_1',\; cast_1 @ oc = cast_1'\,) \;\wedge\; (\,\exists cast_2',\; cast_2 @ oc = cast_2'\,)\,)\,) \;\Rightarrow\; cast_1 \;\simeq\; cast_2\; \mathbf{a}$

$$\Phi\,;\, g \;\vdash\; cast_1 \;\equiv_\alpha\; cast_2$$

Two terms are said to be alpha-equivalent if they have the same closed binding sets, and for each occurence not in the closed binding set, the occurence is defined for one term if it is for the other, and the subterms at that occurence are node-identical.

$\boxed{\Phi\,;\, g \;\vdash\; s\,\mathbf{ok}}$

1 : $\forall i.\,(\Phi \vdash \mathbf{equiv\_both}\, g\,(\, cast_i\,) \;=\; \langle\, \mathbf{closed} \;:\; oc\_reln_{1\,i}\,,\, \mathbf{open} \;:\; \{\,\}\,\rangle)$

2 : $\forall i.\,(\forall oc,\; (\,(\,\exists v,\; cast_i @ oc = v\,) \;\Rightarrow\; oc \;\in\; \mathbf{support}\; oc\_reln_{1\,i}\,))$

3 : $\forall i.\,(\forall pn,\, \forall ntr,\; (\,(\, g \;\vdash\; pn\,(\, var_i{:}mvr_i\,) \;:\; ntr\,) \;\Rightarrow\; (\, g \;\vdash\; cast_i \;:\; ntr \;\vee\; \exists pn',\; g \;\vdash\; pn'\,(\, cast_i\,) \;:\; ntr\,)\,))$

4 : $\forall i.\,(\neg\,(\,\exists ntr',\; (\,(\,\exists pn,\; g \;\vdash\; pn\,(\, cast_i\,) \;:\; ntr'\,) \;\wedge\; (\,(\,\exists f,\; \Phi \;\vdash\; f \;:\; ntr_1 \mathrel{..} ntr_n\; ntr'\; ntr_1' \mathrel{..} ntr_m' \;\rightarrow\; mvr_i\,) \;\wedge\; (\,\exists pn',\; g \;\vdash\; pn'$

$$\Phi\,;\, g \;\vdash\; \{\,\forall i.\,(cast_i / var_i{:}mvr_i)\;\}\,\mathbf{ok}$$

Well-typed substitutions always substitute closed terms (ie all occurences of variables are within the closed binding relation calculated in step 1). Next (step 3), we ensure that for all possible substitution positions, the result makes sense. Finally, we impose a sanity condition which ensures that substitution matches the notion on alpha-equivalence in step 4. This sanity condition says that we disallow substitutions which substitute terms such that the result of the substitution can itself be picked out by auxfns (to be bound later, presumably).

$$\boxed{\Phi\,;\,g\ \vdash\ \mathbf{c\_a\_subst}\,s\ \in\ cast\ =\ cast''}$$

$$\frac{}{\Phi\,;\,g\ \vdash\ \mathbf{c\_a\_subst}\,s\ \in\ v\ =\ v}\quad\text{SUBST\_SPEC\_VAR}$$

$$\frac{}{\Phi\,;\,g\ \vdash\ \mathbf{c\_a\_subst}\,\{\,\forall i.\,(cast_i/v_i)\ ,\ cast\,/\,v\ ,\forall j.\,\left(cast'_j/v'_j\right)\ \}\ \in\ pn\,(\,v\,)\ =\ cast}\quad\text{SUBST\_SPEC\_IN}$$

$$\frac{1:\ \neg\,(\,v\ \in\ \mathbf{dom}\,(\,s\,)\,)}{\Phi\,;\,g\ \vdash\ \mathbf{c\_a\_subst}\,s\ \in\ pn\,(\,v\,)\ =\ pn\,(\,v\,)}\quad\text{SUBST\_SPEC\_OUT}$$

$$\frac{\begin{array}{l}1:\ cast\ =\ pn\,(\forall k.\,(cast_k)\,)\\ 2:\ \neg\,(\,\exists v,\ cast\ =\ pn\,(\,v\,)\,)\\ 3:\ \mid e_1\,..\,e_n\ ::::\ pn\ (\texttt{+}\ bs_1\,..\,bs_m\ \texttt{+})\ \in\ g\,(\,ntr\,)\\ 4:\ \mathbf{remove\_terminals}\,(\,e_1\,..\,e_n\,)\ =\ \forall k.\,(ntmv_k)\\ 5:\ \forall k.\,(\Phi\,;\,g\ \vdash\ cast'_k\ \equiv_\alpha\ cast_k\ \wedge\ (\forall oc,\ \forall v,\ (\,v\ \in\ \mathbf{dom}\,(\,s\,)\ \Rightarrow\ \neg\,(\,cast'_k@oc = v\,)\,)\,))\\ 6:\ \forall k.\,(\mathbf{subst}\,s_k\ \in\ cast'_k\ =\ cast''_k)\\ 7:\ \forall k.\,(\Phi\,;\,g\ \vdash\ cast''_k\ \equiv_\alpha\ cast'''_k)\\ 8:\ cast'\ =\ pn\,(\forall k.\,(cast'''_k)\,)\end{array}}{\Phi\,;\,g\ \vdash\ \mathbf{c\_a\_subst}\,s\ \in\ cast\ =\ cast'}\quad\text{SUBST\_SPEC\_APP}$$

3 Relating concrete binding and alpha-equivalence

Lemma 3.1 (Substitutions on node-identical occurences produce node-identical results)**.** *Suppose* $cast_1\ \simeq\ cast_2$ **at** oc. *Then for any substitution* s*, if* **subst** s ***in*** $cast_1\ =\ cast'_1$*, and* **subst** s ***in*** $cast_1\ =\ cast'_1$*, then* $cast'_1\ \simeq\ cast'_2$ **at** oc*.*

Proof. By induction on the cases for substitution and node-equality. □

We introduce a bit of notation. Call an occurence oc defined for a term $cast$ if there is a subterm at oc, ie if there exists a $cast'$ such that **termat** $oc\ cast\ =\ cast'$.

Also, call an occurence oc closed bound in a term $cast$ if $Phi\ \vdash$ **equiv_both** $g\,(\,cast\,)\ =\ \langle\,$**closed** $:\ oc\_reln_1\,,$ **open** $:\ oc\_reln_2\,\rangle$ and $oc \in$ **union**$\{oc\_set | oc\_set \in oc\_reln_1\}$.

Lemma 3.2 (Binding occurences are not substituted)**.** *For any* s*,* Ψ*,* g*,* $cast_1$*,* $cast_2$ *and* mse*, if* $\Phi\,;\,g\ \vdash\ s\,$**ok***,* $g\ \vdash\ cast_1\ :\ cast\_type$*,* **subst** s ***in*** $cast_1\ =\ cast_2$ *and if* $oc \in [\![mse]\!]\,g\,(\,cast_1\,)$ *then* oc *is defined for* $cast_2$ *and is the same term as in* $cast_1$*.*

Proof. This is proved by induction on the structure of mse. For $mse\ =\ \{\,\}$, this is immediate. For $mse = mse_1$ `union` mse_2, the inductive hypothesis on the two sets $[\![mse_1]\!]$ and $[\![mse_2]\!]$ give us the required results.

Now consider if $mse = mv$. Then the subterm at that occurence to be well-typed must be $var\ :\ mvr$. Notice that bare variables are not substituted for by the concrete substitution function. Thus this case is covered.

The final case is if $mse = f\,\texttt{(}nt\texttt{)}$. Looking at the definition of the concrete substitution function, this occurence is defined in the result. It is always the same subterm, except when this variable occurence is wrapped in a singleton production. Assume then for purpose of contradiction that this variable occurence is wrapped in a singleton production $prodname$ for some nonterminal root ntr. Since it was picked up by an auxfn, there must be a auxfn f which takes ntr to the sort of metavariables in mse, ie mvr. But this is impossible for well-typed substitutions, since then there must be a singleton production for the substituent for the same nonterminal, and this nonterminal is in the domain of an auxfn. Thus we have the required contradiction.

□

Lemma 3.3 (Closed bound occurences are not substituted)**.** *For any s, Ψ, g, $cast_1$ and $cast_2$, if $\Phi\,;\,g\,\vdash\,s\,\mathbf{ok}$, $g\,\vdash\,cast_1\,:\,cast\_type$, $\mathbf{subst}\,s\,\mathbf{in}\,cast_1\,=\,cast_2$, and an occurence oc is closed bound in the term $cast_1$, then oc is defined for $cast_2$ and in fact is closed bound in the term $cast_2$.*

Proof. We perform induction on the structure of the term, and look at the calculation of the closed binding set. We notice that an occurence can turn out to be closed bound in one of three ways.

First, the occurence might be a lift of an occurence already closed bound on a subterm. This case goes through by inductive hypothesis, since the subterm is smaller.

Second, the occurence might be of a variable $var\,:\,mvr$ in the subtree corresponding to nt, where a bindspec clause $\texttt{bind}\,mse\,\texttt{in}\,nt$ present attached to the production. We will call these occurences bound occurences. The variable $var\,:\,mvr$ must lie within the var_set which is the concrete interpretation $\mathbf{concrete}\,[\![\,mse\,]\!]$ of the term $cast_1$. Looking at the definition of concrete substitutions, this set is filtered out from the domain of substitution, and thus the occurence remains unchanged in the result of substitution.

Third, the occurence might lie within $[\![\,mse\,]\!]$ for a bindspec clause $\texttt{bind}\,mse\,\texttt{in}\,nt$. We will call these occurences binding occurences. In this case we defer to the lemma 3.2.

□

Lemma 3.4 (Substitution preserves closed bound equivalence classes)**.** *For any s, Ψ, g, $cast_1$ and $cast_2$, if $\Phi\,;\,g\,\vdash\,s\,\mathbf{ok}$, $g\,\vdash\,cast_1\,:\,cast\_type$, $\mathbf{subst}\,s\,\mathbf{in}\,cast_1\,=\,cast_2$. Say that $\Phi\,\vdash\,\mathbf{equiv\_both}\,g\,(\,cast_1\,)\,=\,\langle\,\mathbf{closed}\,:\,oc\_reln_{11}\,,\,\mathbf{open}\,:\,oc\_reln_{12}\,\rangle$ and $\Phi\,\vdash\,\mathbf{equiv\_both}\,g\,(\,cast_2\,)\,=\,\langle\,\mathbf{closed}\,:\,oc\_reln_{21}\,,\,\mathbf{open}\,:\,oc\_reln_{22}\,\rangle$. Then $oc\_reln_{11}\subseteq oc\_reln_{21}$.*

Proof. We argue by induction on the term, and looking at the calculation of the closed equivalence relation. By induction, the closed equivalence classes of the subterms are still present in the result. Further, for any occurence which is closed bound, the subterm is not changed by substitution, by lemma 3.3. Thus each occurence previously closed bound will still be picked up. No new occurence will be picked up, since well-typedness of substitution ensures that the substituents are closed. Since the set of auxfns have not changed, no occurence which was previously closed bound will become open-bound, or vice versa. □

Theorem 3.5. *For any s, Ψ, g, $cast_1$ and $cast_2$, if $\Phi\,;\,g\,\vdash\,s\,\mathbf{ok}$, $g\,\vdash\,cast_1\,:\,cast\_type$, $g\,\vdash\,cast_2\,:\,cast\_type$, $\Phi\,;\,g\,\vdash\,cast_1\,\equiv_\alpha\,cast_2$, $\mathbf{subst}\,s\,\mathbf{in}\,cast_1\,=\,cast'_1$, and $\mathbf{subst}\,s\,\mathbf{in}\,cast_2\,=\,cast'_2$, then $\Phi\,;\,g\,\vdash\,cast'_1\,\equiv_\alpha\,cast'_2$.*

Proof. We prove this by looking at the definition of $\equiv_\alpha$, knowing that $\Phi\,;\,g\,\vdash\,cast_1\,\equiv_\alpha\,cast_2$.

There are now two cases for an occurence oc which is defined for $cast_1$. Either it is within closed bound in $cast_1$, or it is not. We will look at these cases in turn.

In the case that it is closed bound, since the two closed binding relations have to be the same for alpha-equivalent terms, the same occurence must be defined and indeed closed bound for $cast_2$ (Note that the variable at that occurence need not be identical). We know by lemma 3.3 that the applied substitutions do not touch these occurences, and that the corresponding equivalence class remains within the set of closed equivalence relations for the results of the substitution.

In the case that it is not closed bound, the subterms of the two terms at that occurence are node-identical. The result of substitution on these subterms are therefore themselves node-identical, by lemma 3.1.

Using these facts, we now show that the results are themselves α-equivalent. This involves two steps.

First, we have to show that the closed equivalence relations are identical for the two resultant terms. We follow the construction of the closed equivalence relation at each node of the syntax tree of $cast'_1$. If this subterm was already present in $cast_1$, then all pre-existing closed equivalence classes must still be present in the new equivalence relation, by lemma 3.4. Further, the only new additions can be due to new subterms created by the substitution. Notice however that substitutions only changed the structure of the term at occurences which are not closed bound. In these cases the subterms at occurences were node-identical. Thus only identical equivalence classes of closed occurences are added to the closed equivalence relations of both terms. Further, since well-typed substitutions substitute only closed terms, we get that there are no additional equivalence classes of open bound occurences.

Second, all occurences which are not closed bound in the result are defined for both resultant terms if it is defined for any one resultant term, and the terms at such occurences are node-identical. So consider an occurence which is not closed bound. If this occurence was defined in the initial term, it cannot have been closed bound, since the closed equivalence relation of the result includes that of the initial term, by lemma 3.4. Thus by the previous statements, it is defined and node-identical in both terms. If on the other hand, it is an occurence not defined in the initial term, since the substitution only acted on node-identical terms, it must be defined and identical in both resultantterms.

□

Theorem 3.6 (Correspondence of concrete and capture-avoiding substitution)**.** *For any s, Ψ, g, cast and cast′, if* $\Phi\,;\,g \vdash s\,\mathbf{ok}$*,* $g \vdash \mathit{cast} : \mathit{cast\_type}$*,* $\mathbf{subst}\,s\,\mathbf{in}\,\mathit{cast} = \mathit{cast}'$*, then* $\Phi\,;\,g \vdash \mathbf{c\_a\_subst}\,s\,\mathbf{in}\,\mathit{cast} = \mathit{cast}'$*.*

Proof. This theorem is proved by induction on the structure of the term.

For the cases where the term is a variable, or a singleton production containing a variable, concrete substitution and capture-avoiding substitutions are defined identically.

For a non-singleton production, capture avoiding substitution carries on by alpha-renaming all bound variables to fresh ones. The concrete substitution on the other hand filters out bound variables. We thus pick some fresh variable (that is, one not appearing in the term at all) and performing the alpha-renaming. Since we have countably infinite variables, this can always be done. Now, by theorem 3.5, we get that the result of applying the concrete substitution is alpha-equivalent to the capture-avoiding substitution performed by our chosen renaming. Looking at the rule CA_SUBST_3, this allows us to match up with the premise 7. This completes the case. □