APPLY and rotate[E]

Johan G. F. Belinfante
2003 May 12

\[ \text{In[1]} := \text{\texttt{\textless\textless goedel52.r67; \textless\textless tools.m}} \]

\[ \text{\texttt{\textbackslash :Package Title: goedel52.r67 2003 May 9 at 11:50 p.m.}} \]

\[ \text{\texttt{\textit{It is now: 2003 May 12 at 9:53}}} \]

\[ \text{\texttt{Loading Simplification Rules}} \]

\[ \text{\texttt{TOOLS.M Revised 2003 May 11}} \]

\[ \text{\texttt{weightlimit = 40}} \]

### introduction

The class \texttt{APPLY[x,y]} is a shorthand for

\[ \text{\texttt{In[2]} := \texttt{APPLY[x, y]}} \]

\[ \text{\texttt{Out[2]} = \texttt{A[image[x, singleton[y]]]}} \]

This constructor is mainly useful for applying a function to an argument. Applying \texttt{lambda} to this expression yields a formula involving the rotated membership relation \texttt{rotate[E]}. This formula involves the function \texttt{BIGCAP} which corresponds to the constructor \texttt{A[x]}:

\[ \text{\texttt{In[3]} := \texttt{class[y, forall[z, implies[member[z, x], member[y, z]]]]}} \]

\[ \text{\texttt{Out[3]} = \texttt{A[x]}} \]

\[ \text{\texttt{In[4]} := \texttt{lambda[x, A[x]]}} \]

\[ \text{\texttt{Out[4]} = \texttt{BIGCAP}} \]

For function evaluation, the purpose of \texttt{BIGCAP} is to extract the value of a function from the singleton produced by taking the vertical section \texttt{image[x,singleton[y]]} of a function \texttt{x} at an argument \texttt{y}. The unary intersection operation \texttt{A} is not the only way to extract an element from its singleton. It is often desirable, when dealing with functions, to replace the function \texttt{BIGCAP} with \texttt{inverse[SINGLETON]}. In this notebook some formulas are derived which show how to go about doing this. These formula involve the function \texttt{FUNPART}:

\[ \text{\texttt{In[5]} := \texttt{lambda[x, funpart[x]]}} \]

\[ \text{\texttt{Out[5]} = \texttt{FUNPART}} \]

The constructor \texttt{funpart} is defined by

\[ \text{\texttt{In[6]} := \texttt{intersection[composite[Id, x], complement[composite[Di, x]]]}} \]

\[ \text{\texttt{Out[6]} = \texttt{funpart[x]}} \]
This constructor is useful for eliminating \texttt{FUNCTION} hypotheses from clauses.

\begin{verbatim}
In[7]:= equal[x, funpart[x]]
Out[7]= FUNCTION[x]
\end{verbatim}

One can therefore think of the expression \texttt{funpart[x]} as a generic function.

\section*{applying lambda to APPLY}

Applying \texttt{lambda} to the constructor \texttt{APPLY} yields the binary function

\begin{verbatim}
In[8]:= lambda[pair[x, y], APPLY[x, y]]
Out[8]= composite[
  VERTSECT[complement[composite[complement[inverse[e]], rotate[e]]]], id[cart[V, V]]
]
\end{verbatim}

The rotated membership relation in this formula can be eliminated:

\begin{verbatim}
In[9]:= composite[complement[inverse[e]], BIGCAP, IMG, cross[Id, SINGLETON]] // VSTriNormality // Reverse
Out[9]= composite[complement[inverse[e]], rotate[e]] ==
  composite[complement[inverse[e]], BIGCAP, IMG, cross[Id, SINGLETON]]
In[10]:= composite[complement[inverse[e]], rotate[e]] :=
  composite[complement[inverse[e]], BIGCAP, IMG, cross[Id, SINGLETON]]
\end{verbatim}

The formula for \texttt{lambda APPLY} now produces a more transparent result:

\begin{verbatim}
In[11]:= lambda[pair[x, y], APPLY[x, y]]
Out[11]= composite[BIGCAP, IMG, cross[Id, SINGLETON]]
\end{verbatim}

\section*{special formulas for function evaluation}

For the application to function evaluation, one can replace \texttt{BIGCAP} by \texttt{inverse[SINGLETON]}. In this section, some formulas are derived which show how to do this.

\begin{verbatim}
In[12]:= Map[VERTSECT[complement[#]] &, composite[complement[inverse[e]],
  BIGCAP, IMG, cross[FUNPART, SINGLETON]]] // TriNormality
Out[12]= composite[BIGCAP, IMG, cross[FUNPART, SINGLETON]] ==
  union[cart[intersection[
    composite[inverse[SINGLETON]], composite[IMAGE[inverse[IMG]],
    range[SINGLETON]]],
    composite[inverse[e]], IMAGE[FIRST], FUNPART]], singleton[0]],
  composite[inverse[SINGLETON], IMG, cross[Id, SINGLETON]]]
\end{verbatim}

The add–on appearing on the right side is the empty set in disguise:
The formula derived above now simplifies to

Another interesting formula for this function is obtained by a triple rotation: