triple-application formula for CURRY

Johan G. F. Belinfante
2011 July 13

In[1]:= **SetDirectory["l:"]; << goedel.11jul08b**

```
:Package Title: goedel.11jul08b                2011 July 8 at 10:00 p.m.

Loading takes about eleven minutes, half that time due to builtin pauses.

It is now:  2011 Jul 13 at 13:2

Loading Simplification Rules

TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

weightlimit = 40

Loading completed.

It is now:  2011 Jul 13 at 13:12
```

summary

The basic idea of currying is to replace any function **w** with two variables with an equivalent function **APPLY[CURRY, w]** of a single variable, whose values in turn are functions of a single variable. The fundamental idea goes back to a lecture by Moses Schönfinkel (1920) written up for publication in 1924 by Heinruch Behmann, and popularized through the extensive work by Haskell B. Curry in combinatory logic.

In[2]:= **"Moses Schönfinkel, *On the building blocks of mathematical logic*, Math. Annalen, vol. 92, pp. 305–316 (1924). [in German]; reprinted in Jean van Heienoort, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, Harvard University Press, Cambridge, Massachusetts, 1967";**

An **APPLY** formula for curried mappings was derived 2006 November 11 in the posted notebook **cur-left.nb**. In this notebook some related results are derived, including a basic equation that involves triple application.

the basic theorem

The connection between a binary function and its curried counterpart is especially simple when the domain of the binary function is a non-empty cartesian product. The following theorem applies only to this special case.

Theorem. A basic triple application equation for curried mappings.

```
In[3]:= Map[not, SubstTest[and, implies[p3, p4],
           implies[and[p1, p2, p4], p5], implies[p5, p6], not[implies[and[p1, p2, p3], p6]],
           {p1 → member[w, map[cart[x, y], z]], p2 → member[u, x], p3 → member[v, y],
            p4 → not[empty[y]], p5 -> equal[composite[w, LEFT[u]], APPLY[APPLY[CURRY, w], u]],
            p6 -> equal[APPLY[w, PAIR[u, v]], APPLY[APPLY[APPLY[CURRY, w], u], v]]}]] // Reverse

Out[3]= or[equal[APPLY[w, PAIR[u, v]], APPLY[APPLY[APPLY[CURRY, w], u], v]],
          not[member[u, x]], not[member[v, y]], not[member[w, map[cart[x, y], z]]]] == True

In[4]:= or[equal[APPLY[w_, PAIR[u_, v_]], APPLY[APPLY[APPLY[CURRY, w_], u_], v_]],
          not[member[u_, x_]], not[member[v_, y_]],
          not[member[w_, map[cart[x_, y_], z_]]]] := True
```

The following companion theorem involves the inverse process of uncurrying.

Theorem. A basic application equation about uncurrying.

```
In[5]:= Map[not, SubstTest[and, implies[p3, p4], implies[and[p1, p2, p4], p5], implies[p5, p6],
           not[implies[and[p1, p2, p3], p6]], {p1 → member[w, map[x, map[y, z]]],
            p2 → member[u, x], p3 → member[v, y], p4 → not[empty[y]],
            p5 -> equal[composite[APPLY[inverse[CURRY], w], LEFT[u]], APPLY[w, u]], p6 -> equal[
              APPLY[APPLY[inverse[CURRY], w], PAIR[u, v]], APPLY[APPLY[w, u], v]]}]] // Reverse

Out[5]= or[equal[APPLY[APPLY[w, u], v], APPLY[APPLY[inverse[CURRY], w], PAIR[u, v]]],
          not[member[u, x]], not[member[v, y]], not[member[w, map[x, map[y, z]]]]] == True

In[6]:= or[equal[APPLY[APPLY[w_, u_], v_], APPLY[APPLY[inverse[CURRY], w_], PAIR[u_, v_]]],
          not[member[u_, x_]], not[member[v_, y_]],
          not[member[w_, map[x_, map[y_, z_]]]]] := True
```

replacing PAIR with pair

In the **GOEDEL** program **pair[u, v]** is a primitive ordered pair, which agrees with the defined constructor **PAIR[u, v] = A[{u} × {v}]** when both **u** and **v** are sets.

```
In[7]:= A[cart[set[x], set[y]]]

Out[7]= PAIR[x, y]

In[8]:= equal[PAIR[x, y], pair[x, y]]

Out[8]= and[member[x, V], member[y, V]]
```

To provide a measure of flexibility about which of the two ordered pairs is used, the counterparts of the results of the preceding section will be restated here with **pair** in place of **PAIR**.

Lemma. Replacing **PAIR** with **pair** in an **APPLY** expression.

```
In[9]:= Map[not, SubstTest[and, (*implies[p1,p2],implies[p1,p3],*)
           implies[and[p2, p3], p4], not[implies[p1, p4]],
           {p1 -> and[member[u, y], member[v, z]], p2 → member[u, V], p3 → member[v, V],
            p4 -> equal[APPLY[x, PAIR[u, v]], APPLY[x, pair[u, v]]]}]] // Reverse

Out[9]= or[equal[APPLY[x, pair[u, v]], APPLY[x, PAIR[u, v]]],
          not[member[u, y]], not[member[v, z]]] == True

In[10]:= or[equal[APPLY[x_, pair[u_, v_]], APPLY[x_, PAIR[u_, v_]]],
           not[member[u_, y_]], not[member[v_, z_]]] := True
```

Corollary. The basic theorem about curried function application with **pair** in place of **PAIR**.

```
In[11]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 → and[member[u, x], member[v, y], member[w, map[cart[x, y], z]]],
            p2 -> equal[APPLY[w, PAIR[u, v]], APPLY[APPLY[APPLY[CURRY, w], u], v]],
            p3 → equal[APPLY[w, pair[u, v]], APPLY[APPLY[APPLY[CURRY, w], u], v]]}]] // Reverse

Out[11]= or[equal[APPLY[w, pair[u, v]], APPLY[APPLY[APPLY[CURRY, w], u], v]],
           not[member[u, x]], not[member[v, y]], not[member[w, map[cart[x, y], z]]]] == True

In[12]:= or[equal[APPLY[w_, pair[u_, v_]], APPLY[APPLY[APPLY[CURRY, w_], u_], v_]],
           not[member[u_, x_]], not[member[v_, y_]],
           not[member[w_, map[cart[x_, y_], z_]]]] := True
```

Corollary. The basic theorem about un-curried function application with **pair** in place of **PAIR**.

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 → and[member[u, x], member[v, y], member[w, map[x, map[y, z]]]],
            p2 -> equal[APPLY[APPLY[w, u], v], APPLY[APPLY[inverse[CURRY], w], PAIR[u, v]]],
            p3 -> equal[APPLY[APPLY[w, u], v],
              APPLY[APPLY[inverse[CURRY], w], pair[u, v]]]}]] // Reverse

Out[13]= or[equal[APPLY[APPLY[w, u], v], APPLY[APPLY[inverse[CURRY], w], pair[u, v]]],
           not[member[u, x]], not[member[v, y]], not[member[w, map[x, map[y, z]]]]] == True

In[14]:= or[equal[APPLY[APPLY[w_, u_], v_], APPLY[APPLY[inverse[CURRY], w_], pair[u_, v_]]],
           not[member[u_, x_]], not[member[v_, y_]],
           not[member[w_, map[x_, map[y_, z_]]]]] := True
```