a well-ordering theorem for strictly progressing functions

Johan G. F. Belinfante and Ming Li
2009 May 11

```
In[1]:= SetDirectory["l:"]; << goedel81.09a; << tools.m

        :Package Title: goedel81.09a          2006 May 9 at 9:35 p.m.

        It is now:  2006 May 11 at 7:42

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 May 8

        weightlimit = 40
```

summary

It is shown in this notebook that minimal invariant subsets for strictly progressing functions are well-ordered by inclusion. The basic observation used is that for an acyclic function **x**, the list **iterate[x, set[y]]** of iterates starting from any point **y** is one-to-one.

```
In[2]:= implies[and[FUNCTION[x], equal[0, fix[trv[x]]], FUNCTION[inverse[iterate[x, set[y]]]]]

Out[2]= True
```

Actually the well-ordering theorem holds for all progressing functions. Roughly speaking, if iteration with a progressing function ever hits a fixed point, the iterates remain constant from there on. This constant therefore contributes only one fixed point, namely the largest one, to the range of iterate, and is therefore relatively harmless. Here this issue is avoided by only considering functions that are strictly progressing.

derivation

For a strictly progressing function one has:

```
In[3]:= SubstTest[implies, and[subclass[u, v], equal[0, fix[trv[v]]],
          equal[0, fix[trv[u]]], {u → funpart[intersection[PS, x]], v → PS}]

Out[3]= equal[0, fix[trv[funpart[intersection[PS, x]]]]] == True

In[4]:= fix[trv[funpart[intersection[PS, x_]]]] := 0
```

It follows that iteration in this case produces a one-to-one function:

```
In[5]:= SubstTest[implies, and[FUNCTION[z], equal[0, fix[trv[z]]]],
         FUNCTION[inverse[iterate[z, set[y]]]], z → funpart[intersection[PS, x]]]

Out[5]= FUNCTION[inverse[iterate[funpart[intersection[PS, x]], set[y]]]] == True

In[6]:= FUNCTION[inverse[iterate[funpart[intersection[PS, x_]], set[y_]]]] := True
```

The iteration is monotone. This result for the case of a strictly progressing function follows from that for progressing functions in general:

```
In[7]:= SubstTest[monotone,
         iterate[funpart[intersection[S, z]], set[y]], S, S, z → intersection[PS, x]]

Out[7]= subclass[composite[iterate[funpart[intersection[PS, x]], set[y]],
           S, inverse[iterate[funpart[intersection[PS, x]], set[y]]]], S] == True

In[8]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

A one-to-one transform of a total order is a total order:

```
In[9]:= SubstTest[TOTALORDER, composite[oopart[u], to[v], inverse[oopart[u]]],
         {u -> iterate[funpart[intersection[PS, x]], set[y]], v → restrict[S, omega, omega]}]

Out[9]= TOTALORDER[composite[iterate[funpart[intersection[PS, x]], set[y]],
           S, inverse[iterate[funpart[intersection[PS, x]], set[y]]]]] == True

In[10]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

At this point one needs to know that a total suborder of a partial order is a restriction. Here the partial order in question is **S**. One needs:

```
In[11]:= SubstTest[implies, subclass[to[u], po[v]],
          equal[to[u], restrict[po[v], fix[to[u]], fix[to[u]]]],
          {u -> composite[iterate[funpart[intersection[PS, x]], set[y]],
             S, inverse[iterate[funpart[intersection[PS, x]], set[y]]]], v → S}]

Out[11]= equal[composite[id[hull[invar[funpart[intersection[PS, x]]], set[y]]],
            S, id[hull[invar[funpart[intersection[PS, x]]], set[y]]]],
           composite[iterate[funpart[intersection[PS, x]], set[y]], S,
            inverse[iterate[funpart[intersection[PS, x]], set[y]]]]] == True

In[12]:= composite[iterate[funpart[intersection[PS, x_]], set[y_]],
           S, inverse[iterate[funpart[intersection[PS, x_]], set[y_]]]] :=
          composite[id[hull[invar[funpart[intersection[PS, x]]], set[y]]],
           S, id[hull[invar[funpart[intersection[PS, x]]], set[y]]]]
```

One now can use the fact that a one-to-one transform of a well-order is a well-order:

```
In[13]:= SubstTest[WELLORDER, composite[oopart[u], wo[v], inverse[oopart[u]]],
          {u -> iterate[funpart[intersection[PS, x]], set[y]], v → restrict[S, omega, omega]}]

Out[13]= subclass[P[hull[invar[funpart[intersection[PS, x]]], set[y]]],
           union[fix[composite[E, BIGCAP]], set[0]]] == True
```

```
In[14]:= subclass[P[hull[invar[funpart[intersection[PS, x_]]], set[y_]]],
           union[fix[composite[E, BIGCAP]], set[0]]] := True
```

The **funpart** wrapper can be removed:

```
In[15]:= SubstTest[implies, equal[x, funpart[intersection[PS, z]]],
          subclass[P[hull[invar[x], set[y]]], union[fix[composite[E, BIGCAP]], set[0]]], z → x]

Out[15]= or[not[equal[0, fix[x]]], not[FUNCTION[x]], not[subclass[x, S]],
           subclass[P[hull[invar[x], set[y]]], union[fix[composite[E, BIGCAP]], set[0]]]] == True

In[16]:= or[not[equal[0, fix[x_]]], not[FUNCTION[x_]], not[subclass[x_, S]], subclass[
            P[hull[invar[x_], set[y_]]], union[fix[composite[E, BIGCAP]], set[0]]]] := True
```

---

application: the power tower

Lemma.

```
In[17]:= equal[intersection[POWER, Di], POWER]

Out[17]= True

In[18]:= intersection[Di, POWER] := POWER
```

Lemma.

```
In[19]:= AssInt[POWER, Di, S]

Out[19]= intersection[POWER, PS] == composite[POWER, id[FULL]]

In[20]:= intersection[POWER, PS] := composite[POWER, id[FULL]]
```

The power tower is well ordered.

```
In[22]:= SubstTest[subclass, P[hull[invar[funpart[intersection[PS, x]]], set[y]]],
          union[fix[composite[E, BIGCAP]], set[0]],
          {x → POWER, y → 0}]

Out[22]= subclass[P[image[IMAGE[inverse[RANK]], omega]],
           union[fix[composite[E, BIGCAP]], set[0]]] == True

In[23]:= subclass[P[image[IMAGE[inverse[RANK]], omega]],
           union[fix[composite[E, BIGCAP]], set[0]]] := True
```