Wellfounded orderings in constructive type theory

March 28, 2011

1 Preliminaries

Subtypes A type S is a subtype of a type T (written $S \subseteq T$) if every term that is a member of S is a member of T and further, any two terms that denote the same member of S denote the same member of T.

Recursive types in Nuprl A function F from types to types is *monotonic* if

$$S \subseteq T \Rightarrow F(S) \subseteq F(T)$$

If F is a monotonic function on types, then it has a fixedpoint:

`rec(T.F[T])`

This fixedpoint is the least fixedpoint, the union of all the types

$$Void, F[Void], F[F[Void]], \ldots F^n[Void], \ldots F^\omega[Void], \ldots F^\alpha[Void] \ldots$$

The "elimination rule" for $rec(T.F[T])$ lets us prove $P(x)$ for all $x \in rec(T.F[T])$ if from the assumption that $P(x)$ is true for all $x \in T \subseteq rec(T.F[T])$ we can show that $P(x)$ is true for all $x \in F[T]$.

2 Wellfounded trees in constructive type theory

Brouwer based his connstruction of ordinal numbers on well–founded trees. This construction was formaixed in type theory by Martin-Lof as the W-type. Its key property is that it has a constructible well-founded ordering which means that we can derive an induction principle for the W-type.

Definition of the W type

```
W(A;a.B[a]) ==  rec(W.a:A × (B[a] → W))
```

Using the *propositions as types* translation, we can also think of the W-type as (the type of witnesses to) the (self-referential) proposition:

```
X == ∃a:A. B[a]⇒ X
```

The constructor for W and its well formedness lemma

```
Wsup(a;b) ==  <a, b>
∀[A:Type]. ∀[B:A → Type]. ∀[a:A]. ∀[b:B[a] → W(A;a.B[a])].  (Wsup(a;b) ∈ W(A;a.B[a]))
```

Note that we state well–formedness lemmas using the *uniform all quantifer*. It is defined by:

```
∀[x:A]. B[x] ==  ∩x:A. B[x]
```

The definition of two mutually recursive comparisons on W In the following definition (and henceforth) we tell Nuprl's display system to display Wcmp(A;a.B[a];btrue) as $\leq$ and Wcmp(A;a.B[a];bfalse) as $<$.

```
Wcmp(A;a.B[a];leq)
==r λw1,w2.
     if leq
     then let a,f = w1 in
            ∀x:B[a]. ((f x) <  w2)
     else let a,f = w2 in
            ∃x:B[a]. (w1 ≤  (f x))
     fi
```

Here is the well–formedness lemma:

```
∀[A:Type]. ∀[B:A → Type]. ∀[leq:𝔹].  (Wcmp(A;a.B[a];leq) ∈ W(A;a.B[a]) → W(A;a.B[a]) → ℙ)
```

Some properties of the comparisons

```
∀[A:Type]. ∀[B:A → Type].  ∀w1,w2:W(A;a.B[a]).  ((w1 <  w2) ⇒ (w1 ≤  w2))
∀[A:Type]. ∀[B:A → Type].  ∀w1,w2:W(A;a.B[a]).  w1 ≤  w2 supposing w1 = w2
∀[A:Type]. ∀[B:A → Type].
  ∀w1,w2,w3:W(A;a.B[a]).
    ((((w1 <  w2) ⇒ (w2 ≤  w3) ⇒ (w1 <  w3)) ∧ ((w1 ≤  w2) ⇒ (w2 <  w3) ⇒ (w1 <  w3)))
    ∧ ((w1 ≤  w2) ⇒ (w2 ≤  w3) ⇒ (w1 ≤  w3)))
∀[A:Type]. ∀[B:A → Type]. ∀[w1:W(A;a.B[a])].  (¬(w1 <  w1))
```

The definition of well founded In constructive logic, we say that a relation R is well–founded if there is an induction principle. This follows, classically but not constructively, if there are no infinite R-descending chains.

```
WellFnd(A;x,y.R[x; y]) ==
  ∀P:A → ℙ. ((∀j:A. ((∀k:{k:A| R[k; j]} . P[k]) ⇒ P[j])) ⇒ (∀n:A. P[n]))
```

The definition of uniformly well founded If we change all the forall quantifiers in the definition of well–founded into uniform forall quantifiers, then we get the definition of uniformly well-founded.

```
uWellFnd(A;x,y.R[x; y]) ==
  ∀[P:A → ℙ]. ((∀[j:A]. ((∀[k:{k:A| R[k; j]} ]. P[k]) ⇒ P[j])) ⇒ (∀[n:A]. P[n]))
```

The Y combinator

```
Y ==  λf.((λx.(f (x x))) (λx.(f (x x))))
∀[f:Top]. (Y f ∼ f (Y f))
```

The ordering on W is uniformly well founded

```
∀[A:Type]. ∀[B:A → Type].  uWellFnd(W(A;a.B[a]);w1,w2.w1 <  w2)
Y ∈ ∀[A:Type]. ∀[B:A → Type].  uWellFnd(W(A;a.B[a]);w1,w2.w1 <  w2)
```

Induction on other types uses a measure function that maps into a W type

```
∀[T,A:Type]. ∀[B:A → Type]. ∀[measure:T → W(A;a.B[a])]. ∀[P:T → ℙ].
  ((∀i:T. ((∀j:{j:T| measure[j] <  measure[i]} . P[j]) ⇒ P[i])) ⇒ (∀i:T. P[i]))
∀[T,A:Type]. ∀[B:A → Type]. ∀[measure:T → W(A;a.B[a])]. ∀[P:T → ℙ].
  ((∀[i:T]. ((∀[j:{j:T| measure[j] <  measure[i]} ]. P[j]) ⇒ P[i])) ⇒ (∀[i:T]. P[i]))
```

3 The W type as ordinal numbers (the Brouwer ordinals)

The Brouwer ordinal w is the ordinal zero if it has no immediate prdecessors.

```
isZero(w) ==  ¬B[fst(w)]
```

If we can decide which $a \in A$ are codes for zero and successor, then we can define ordinal addition and multiplication:

```
(w1 + w2)==r let a,f = w2 in if zero a then w1 else Wsup(a;λx.(w1 + f x)) fi
(w1 * w2)==r let a,f = w2 in if succ a then ((w1 * f ·) + w1) else Wsup(a;λx.(w1 * f x)) fi
```

Here is a theorem (proved in Nuprl) that states several properties of the ordinal arithmetic and its ordering properties:

```
∀[A:Type]. ∀[B:A → Type].
  ∀zero,succ:A → 𝔹.
    ((∀a:A. ((↑(succ a)) ⇒ B[a] ≡ Unit))
    ⇒ (∀a:A. (¬↑(zero a) ⟺ B[a]))
    ⇒ (∀a1,a2:A.  ((↑(zero a1)) ⇒ (↑(zero a2)) ⇒ (a1 = a2)))
    ⇒ (∀x,y,z:W(A;a.B[a]).
          (((x + (y + z)) = ((x + y) + z))
          ∧ ((x * (y + z)) = ((x * y) + (x * z)))
          ∧ ((x * (y * z)) = ((x * y) * z))
          ∧ (isZero(z) ⇒ isZero(y) ⇒ (z = y))
          ∧ (isZero(z)
            ⇒ ((((x + z) = x) ∧ ((z + x) = x)) ∧ ((x * z) = z) ∧ (z = (x * z)) ∧ (z ≤  x)))
          ∧ ((x ≤  y) ⇒ (((x + z) ≤  (y + z)) ∧ ((z + x) ≤  (z + y))))
          ∧ ((x <  y) ⇒ ((z + x) <  (z + y)))
          ∧ ((x ≤  y) ⇒ ((x * z) ≤  (y * z))))))
.
```