Formalización en PVS de la buena fundamentación del orden de multiconjuntos

José A. Alonso Jiménez
María J. Hidalgo Doblado
Francisco J. Martín Mateos
José L. Ruiz Reina

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 23 de marzo de 2006

Creo poder hacer muy clara la relación de mi conceptografía con el lenguaje común si la comparo con la que hay entre el microscopio y el ojo. Este último, por el campo de su aplicabilidad y la movilidad con que se sabe adaptar a las más diversas situaciones, posee gran superioridad frente al microscopio. Considerado como aparato óptico, muestra si duda muchas imperfecciones, las cuales pasan desapercibidas, por lo común, sólo como consecuencia de su estrecha conexión con la vida mental. Pero tan pronto como los propósitos científicos establecen mayores exigencia en la precisión de las distinciones, el ojo resulta insuficiente. Por el contrario, el microscopio es de lo más apropiado para tales fines, aunque, por ello, no es utilizable para otros.

Gottlob Frege (1879) *Conceptografía (Un lenguaje de fórmulas, semejante al de la aritmética, para el pensamiento puro)* p. 8–9.

Resumen

Sea T un conjunto y $<$ una relación binaria bien fundamentada, definida en T. El propósito de este trabajo es realizar una formalización en PVS de la prueba de la buena fundamentación de la relación de orden inducida por $<$ entre los multiconjuntos finitos de elementos de T. La prueba que hemos formalizado en PVS está basada en la realizada por W. Buchholz, descrita por T. Nipkow en *An Inductive Proof of the Wellfoundedness of the Multiset Order* ([2]).

En la literatura hay diversas definiciones de buena fundamentación de una relación binaria. En el artículo de P. Rudnicki y A. Trybulec, *On Equivalents of Well–Foundedness (An experiment in MIZAR)* ([3])), se establece la equivalencia entre distintas definiciones de relación bien fundamentada. Concretamente, se prueba que si T es un conjunto y $<$ una relación binaria sobre T, entonces las siguientes afirmaciones son equivalentes:

1. Todo subconjunto no vacío de T tiene elemento $<$–minimal.
2. En $(T, <)$ se tiene el principio de inducción bien fundamentada.
3. En $(T, <)$ se tiene el principio de definición por recursión.
4. Unicidad de las funciones definidas recursivamente sobre T.
5. En $(T, <)$ no existen cadenas descendentes infinitas.

La primera de estas expresiones coincide justamente con la definición de relación bien fundamentada establecida en el preludio de PVS. Ahora bien, la caracterización de buena fundamentación que usaremos no es ninguna de las anteriores, sino la que proporciona P. Aczel en su artículo *An Introduction to Inductive Definitions* ([1]) y que usa el conjunto denominado **parte bien fundamentada** de una relación. Este conjunto se define inductivamente como el menor subconjunto de T, $W(T, <)$ tal que:

$$x \in W(T, <) \leftarrow \forall y < x.y \in W(T, <)$$

Se verifica que la relación $(T, <)$ es bien fundamentada si y sólo si $W(T, <) = T$.

El esquema de la formalización realizada en PVS es el siguiente:

- En el capítulo 1 definimos (inductivamente) la parte bien fundamentada de T con respecto a $<$, y probamos que la relación $(T, <)$ es bien fundamentada si y sólo si $W(T, <) = T$.
- En el capítulo 2 definimos (inductivamente) la clausura transitiva de una relación binaria $(T, <)$ y probamos que es bien fundamentada si $(T, <)$ lo es, usando la caracterización establecida.

- En el capítulo 3 definimos la relación $N <_1 M$ si N se obtiene sustituyendo un elemento a de M por un multiconjunto K de elementos menores que a. Probamos que si $(T, <)$ es bien fundamentada, entonces $(\mathcal{M}(T), <_1)$ también es bien fundamentada, donde $\mathcal{M}(T)$ representa los multiconjuntos finitos de elementos de T. Finalmente, definimos $<_{mul}$ como la clausura transitiva de $<_1$ y probamos que si $(T, <)$ es bien fundamentada, entonces $(\mathcal{M}(T), <_{mul})$ es bien fundamentada.

Índice general

Capítulo 1

Buena fundamentación de una relación binaria

1.1. Introducción

Dada una relación binaria $<$ definida en un conjunto T, la parte bien fundamentada de $<$ es el conjunto $W(T,<)$ de elementos a de T para los que no existe una cadena infinita descendente comenzando en a. Este conjunto se puede definir también, inductivamente, como el menor subconjunto de T cerrado bajo el conjunto de reglas

$$\forall y < a . y \in W(T,<) \rightarrow a \in W(T,<)$$

.

En este capítulo, definimos en PVS el conjunto $W(T,<)$ y probamos que la relación $<$ es bien fundamentada si y sólo si $W(T,<) = T$. También se prueba que $<$ es bien fundamentada si en $(T,<)$ se verifica el principio de inducción bien fundamentada. Por último, establecemos que si existe una aplicación de $(T,<)$ en los ordinales conservando el orden, entonces la relación $<$ es bien fundamentada.

Usamos la definición de relación bien fundamentada del preludio de PVS; es decir, la relación $<$ es bien fundamentada en T si todo subconjunto no vacío de T tiene elemento minimal.

1.2. Parte bien fundamentada de una relación binaria

Nota 1.2.1 En el desarrollo de la teoría PVS condición suficiente de buena fundamentación, `wf_cs`, se considera un conjunto no vacío T y una relación binaria $<$ en T.

```
wf_cs[T: TYPE+, <:pred[[T,T]]]: THEORY
 BEGIN
```

Nota 1.2.2 Se usarán las siguientes variables:

```
  p:   VAR pred[T]
  x,y: VAR T
```

Nota 1.2.3 PVS proporciona el soporte necesario para construir definiciones inductivas. En general, las definiciones inductivas se presentan mediante las reglas que generan los elementos del conjunto. Así, un elemento pertenece a dicho conjunto si y sólo si ha sido generado de acuerdo con las reglas. Es decir, se define el menor conjunto cerrado bajo dichas reglas.

Definición 1.2.4 *La **parte bien fundamentada** de T respecto de $<$ es el menor subconjunto $W(T,<)$ de T tal que para todo x se verifica que si $\forall y < x.y \in W(T,<)$, entonces $x \in W(T,<)$.*

```
well_founded_part(x): INDUCTIVE bool =
  FORALL y: y < x IMPLIES well_founded_part(y)
```

Nota 1.2.5 La definición inductiva de la parte bien fundamentada en PVS genera, de manera automática, los siguientes axiomas de inducción:

- axioma de inducción débil de la parte bien fundamentada:
$$\frac{\forall x.(\forall y.(y < x \rightarrow P(y))) \rightarrow P(x)}{\forall x.x \in W(T,<) \rightarrow P(x)}$$
- axioma de inducción de la parte bien fundamentada:
$$\frac{\forall x.(\forall y.(y < x \rightarrow y \in W(T,<) \wedge P(y))) \rightarrow P(x)}{\forall x.x \in W(T,<) \rightarrow P(x)}$$

1.3. Caracterizaciones de buena fundamentación

Lema 1.3.1 *Si $(T,<)$ es bien fundamentada, entonces $W(T,<) = T$.*

```
well_founded_part_cn: LEMMA
  well_founded?[T](<) IMPLIES FORALL x: well_founded_part(x)
```

Demostración: Por inducción sobre $(T,<)$ con el predicado $x \in W(T,<)$. $\square$

Lema 1.3.2 *Si $W(T,<) = T$, entonces $(T,<)$ es bien fundamentada.*

```
well_founded_part_cs: LEMMA
  (FORALL x: well_founded_part(x)) IMPLIES well_founded?[T](<)
```

Demostración: Sea $P \subseteq T$ no vacío. Tenemos que demostrar que P posee un elemento minimal; es decir,
$$\exists y.y \in P \wedge \forall x \in P.x \not< y$$

Supongamos que
$$\neg\exists y.y \in P \wedge \forall x \in P.x \not< y$$
entonces
$$\forall y.(\forall x < y.x \notin P) \rightarrow y \notin P$$
Luego, $P^c = W(T,<)$ (por el axioma de inducción débil de la parte bien fundamentada), $P^c = T$ (por hipótesis) y $P = \emptyset$ (contradicción con la definición de P). $\square$

Teorema 1.3.3 *$(T, <)$ es bien fundamentada si y sólo si $W(T, <) = T$.*

```
well_founded_part_cns: THEOREM
  well_founded?[T](<) IFF (FORALL x: well_founded_part(x))
```

Demostración: Consecuencia de los lemas previos. □

Teorema 1.3.4 *$(T, <)$ es bien fundamentada si y sólo si se verifica el principio de inducción bien fundamentada en $(T, <)$.*

```
well_founded_induct_cns: THEOREM
  well_founded?[T](<) IFF
  (FORALL (p: pred[T]):
    (FORALL (x: T): (FORALL (y: T): y<x IMPLIES p(y)) IMPLIES p(x))
    IMPLIES
    (FORALL (x:T): p(x)))
```

Demostración:
($\Rightarrow$) Se prueba por inducción bien fundamentada en $(T, <)$.
($\Leftarrow$) Se realiza un razonamiento análogo al del lema 1.3.2. □

Teorema 1.3.5 (Condición suficiente de buena fundamentación) *: Si existe una aplicación de $(T, <)$ en los ordinales conservando el orden, entonces la relación $<$ es bien fundamentada.*

```
well_founded_cs_ordinal: THEOREM
  (EXISTS (f:[T->ordinal]): FORALL x,y: x<y IMPLIES f(x)<f(y))
  IMPLIES
  well_founded?[T](<)
```

Demostración: Sea f una aplicación de $(T, <)$ en los ordinales que conserva el orden y $P \subseteq T$ no vacío. Tenemos que demostrar que P posee un elemento mininimal (i.e. existe un $y \in P$ tal que para todo $x \in P$, $x \not< y$).

Consideremos el conjunto $Q = f[P]$ (i.e. $Q = \{f(x) : x \in P\}$). Se verifica que Q posee un elemento minimal (puesto que $Q \subseteq Ord$, $Q \neq \emptyset$ y Ord está bien fundamentado (`well_founded_le`)). Sea u un elemento minimal de Q. Existe un $y \in P$ tal que $f(y) = u$. Veamos que y es un elemento minimal de P. Para ello consideramos un $x \in P$. Entonces

- $f(x) \in Q$ (por definición de Q),
- $f(x) \not< u$ (por definición de u),
- $f(x) \not< f(y)$ (por definición de y),
- $x \not< y$ (porque f conserva el orden).

□

Nota 1.3.6 Y con esto termina la teoría `wf_cs`.

```
END wf_cs
```

Capítulo 2

Buena fundamentación de la clausura transitiva de una relación

2.1. Introducción

En este capítulo definimos en PVS la clausura transitiva de una relación binaria, de manera inductiva. Y probamos que la clausura transitiva de una relación bien fundamentada también es bien fundamentada. Por último, definimos en PVS la clausura reflexiva transitiva de una relación binaria y probamos sus propiedades fundamentales.

2.2. Clausura transitiva de una relación

En esta sección se define (de manera inductiva) la clausura transitiva ($<^+$) de una relación binaria ($<$) definida en T, y se establecen sus propiedades fundamentales.

Nota 2.2.1 En el desarrollo de esta teoría se considera un conjunto no vacío T.

```
cl_tr[T:TYPE+]: THEORY
 BEGIN
```

Nota 2.2.2 Se usarán las variables x, y, z para los elementos de T y $<$ para representar una relación binaria en T.

```
  x,y,z,t: VAR T
  <,rel  : VAR pred[[T,T]]
```

Definición 2.2.3 *(Definición inductiva de la clausura transitiva de la relación $<$) Sea $<$ una relación binaria en T. Definimos la clausura transitiva de $<$ como la menor relación $<^+$ tal que:*

$$\forall x, y \in T, x < y \vee \exists z.(x <^+ z \wedge z < y) \rightarrow x <^+ y$$

```
  tr_cl(<)(x,y): INDUCTIVE bool =
   x < y OR EXISTS z: tr_cl(<)(x,z) AND z < y
```

Nota 2.2.4 De esta forma, estamos definiendo el conjunto de elementos menores que y mediante $<^+$; es decir, los elementos generados por el conjunto de reglas $\{x < y, \exists z.(x <^+ z \wedge z < y)\}$. La definición inductiva de la clausura transitiva de $<$ genera, de manera automática, los siguientes axiomas de inducción:

- axioma de inducción débil:
$$\frac{\forall y.(x < y \vee \exists z.(z < y \wedge P(z))) \rightarrow P(y)}{\forall y.(x <^+ y \rightarrow P(y))}$$

- axioma de inducción fuerte:
$$\frac{\forall y.(x < y \vee \exists z.(x <^+ z \wedge z < y \wedge P(z))) \rightarrow P(y)}{\forall y.(x <^+ y \rightarrow P(y))}$$

Nota 2.2.5 Veamos que, en efecto, cualquier otra relación que verifica las mismas reglas de formación contiene a la relación $<^+$.

Lema 2.2.6 *La relación $<^+$ es la menor relación binaria cerrada bajo las reglas $\{x < y, \exists z.(x <^+ z \wedge z < y)\}$.*

```
tr_cl_less_rules: LEMMA
  (FORALL x,y : (x < y OR EXISTS z: rel(x,z) AND z < y )
                IMPLIES
                rel(x,y))
  IMPLIES
  FORALL x,y: tr_cl(<)(x,y) IMPLIES rel(x,y)
```

Demostración: Directamente, usando el axioma de inducción débil 2.2.4. □

```
tr_cl_weak_induction_2: LEMMA
  FORALL (<, P: [[T,T] -> boolean]):
    (FORALL x,y: (x < y OR (EXISTS z: P(x,z) AND z < y)) IMPLIES P(x,y))
     IMPLIES (FORALL x,y: tr_cl(<)(x, y) IMPLIES P(x,y))
```

Nota 2.2.7 Con objeto de demostrar la transitividad de la relación $<^+$ hemos de probar que el conjunto de reglas $\{z < x, \exists x.(z < x \wedge x <^+ y)\}$ también genera los pares de la relación $<^+$. Para ello, enunciamos los siguientes lemas.

Lema 2.2.8 *Los pares de elementos que verifican la regla básica $x < y$ pertenecen al conjunto definido por $<^+$.*

```
tr_cl_cs_basic: LEMMA
  x < y IMPLIES tr_cl(<)(x,y)
```

Demostración: Directamente por la definición de $<^+$. □

Lema 2.2.9 *Sean $x, y, z \in T$. Se verifica:* $z < x \wedge x <^+ y \rightarrow z <^+ y$.

```
tr_cl_transitive_l0: LEMMA
  (z < x AND tr_cl(<)(x,y))
  IMPLIES
  tr_cl(<)(z,y)
```

Demostración: Sean x_1, z_1 fijos. Hay que probar

$$\forall y.(z_1 < x_1 \wedge x_1 <^+ y \rightarrow z_1 <^+ y)$$

o, lo que es lógicamente equivalente,

$$\forall y.(x_1 <^+ y \rightarrow z_1 \nless x_1 \vee z_1 <^+ y)$$

Para ello, basta usar el axioma de inducción débil de la clausura transitiva 2.2.4 con el predicado $P(y) \equiv (z_1 \nless x_1) \vee (z_1 <^+ y)$. Así, hay que probar la premisa de dicho axioma:

$$\forall y.(x_1 < y \vee \exists t.(t < y \wedge P(t))) \rightarrow P(y)$$

La disyunción de la hipótesis hace que se distingan dos casos:

- $x_1 < y \rightarrow P(y)$

  Es decir, $x_1 < y \rightarrow z_1 \nless x_1 \vee z_1 <^+ y$, que se transforma en $(z_1 < x_1 \wedge x_1 < y) \rightarrow z_1 <^+ y$, lo que se prueba por la definición de $<^+$ y el lema 2.2.8.

- $\exists t.(t < y \wedge P(t)) \rightarrow P(y)$

  Sea t_1 un tal elemento. Hemos de probar que

  $$t_1 < y \wedge P(t_1) \rightarrow P(y)$$

  Es decir: $t_1 < y \wedge (z_1 \nless x_1 \vee z_1 <^+ t_1) \rightarrow (z_1 \nless x_1 \vee z_1 <^+ y)$

  El caso $(t_1 < y \wedge z_1 \nless x_1) \rightarrow (z_1 \nless x_1 \vee z_1 <^+ y)$ se prueba directamente por simplificación proposicional.

  Finalmente, el caso $(t_1 < y \wedge z_1 <^+ t_1) \rightarrow (z_1 \nless x_1 \vee z_1 <^+ y)$ se tiene directamente por la definición de $<^+$.

□

Teorema 2.2.10 *La relación $<^+$ es transitiva.*

```
tr_cl_transitive: THEOREM
  transitive?(tr_cl(<))
```

Demostración: Por la definición de relación transitiva, si consideramos x_1 y z_1, hay que probar:

$$\forall y.((x_1 <^+ y \wedge y <^+ z_1) \rightarrow x_1 <^+ z_1)$$

O, lo que es equivalente

$$\forall y.(x_1 <^+ y \rightarrow (y \not<^+ z_1 \vee x_1 <^+ z_1)$$

Para ello, basta usar el axioma de inducción débil de la clausura transitiva 2.2.4 con el predicado $P(y) \equiv y \not<^+ z_1 \vee x_1 <^+ z_1$. Así, hay que probar la premisa de dicho axioma:

$$\forall y.(x_1 < y \vee \exists t.(t < y \wedge P(t))) \rightarrow P(y)$$

La disyunción de la hipótesis hace que se distingan dos casos:

- $x_1 < y \rightarrow P(y)$

  Es decir, $x_1 < y \rightarrow y \not<^+ z_1 \vee x_1 <^+ z_1$, que se transforma en $(x_1 < y \wedge y <^+ z_1) \rightarrow x_1 <^+ z_1$, lo que se prueba por el lema 2.2.9.

- $\exists t.(t < y \wedge P(t)) \rightarrow P(y)$

  Sea t_1 un tal elemento. Hemos de probar que

  $$t_1 < y \wedge P(t_1) \rightarrow P(y)$$

  Es decir: $t_1 < y \wedge (t_1 \not<^+ z_1 \vee x_1 <^+ z_1) \rightarrow y \not<^+ z_1 \vee x_1 <^+ z_1$.

  El caso $t_1 < y \wedge t_1 \not<^+ z_1 \rightarrow y \not<^+ z_1 \vee x_1 <^+ z_1$ se prueba usando el lema 2.2.9.

  Finalmente, el caso $t_1 < y \wedge x_1 <^+ z_1 \rightarrow y \not<^+ z_1 \vee x_1 <^+ z_1$ se prueba directamente por simplificación proposicional.

□

Lema 2.2.11 $\forall x, y \in T.x <^+ y \rightarrow (x < y \vee \exists z.(x < z \wedge z <^+ y))$

```
tr_cl_equiv_cn: LEMMA
  tr_cl(<)(x,y) IMPLIES (x < y OR EXISTS z: x < z AND tr_cl(<)(z,y))
```

Demostración: La prueba es análoga a la de los resultados 2.2.10 y 2.2.9 □

Lema 2.2.12 $\forall x, y \in T.(x < y \vee \exists z.(x < z \wedge z <^+ y)) \rightarrow x <^+ y$

```
tr_cl_equiv_cs: LEMMA
  (x < y OR EXISTS z: x < z AND tr_cl(<)(z,y)) IMPLIES tr_cl(<)(x,y)
```

Demostración: Usando el lema 2.2.9. □

Lema 2.2.13 $\forall x, y \in T.x < y \vee \exists z.(x < z \wedge z <^+ y) \leftrightarrow x <^+ y$

```
  tr_cl_equiv: LEMMA
   tr_cl(<)(x,y) IFF (x < y OR EXISTS z: x < z AND tr_cl(<)(z,y))
```

Demostración: Por los lemas 2.2.11 y 2.2.12. □

Teorema 2.2.14 *La clausura transitiva, $<^+$, de una relación binaria $<$, es menor que cualquier relación transitiva que contiene a $<$.*

```
  tr_cl_less_transitive: THEOREM
    transitive?(rel) AND
    (FORALL x,y: x < y IMPLIES rel(x,y))
    IMPLIES
    FORALL x,y: tr_cl(<)(x,y) IMPLIES rel(x,y)
```

Demostración: Sea $<_1$ una relación transitiva que contiene a $<$, y sea x_1 un elemento de T. Hay que probar que

$$\forall y.(x_1 <^+ y \rightarrow x_1 <_1 y)$$

Para ello, basta usar el axioma de inducción débil de la clausura transitiva 2.2.4 con el predicado $P(y) \equiv x_1 <_1 y$. □

Nota 2.2.15 Como consecuencia, se tiene que la clausura transitiva, $<^+$, de una relación binaria $<$ es la menor relación transitiva que contiene a $<$.

```
 END cl_tr
```

2.3. Buena fundamentación de la clausura transitiva de una relación

En esta sección se prueba que la clausura transitiva de una relación bien fundamentada también es bien fundamentada.

Nota 2.3.1 Como en la sección anterior, se considera un conjunto no vacío T. Se usarán las variables x, y, z para los elementos de T y $<$ para representar una relación binaria en T.

```
wf_cl_tr[T:TYPE+]: THEORY
 BEGIN

  <    : VAR pred[[T,T]]
  x,y,z: VAR T
```

Nota 2.3.2 Usamos la teoría *cl_tr* en la que se ha definido la clausura transitiva de una relación binaria; y la teoría *wf_cs* en la que hemos establecido una caracterización de la buena fundamentación de una relación, usando la parte bien fundamentada de dicha relación.

```
IMPORTING cl_tr
IMPORTING wf_cs
```

Lema 2.3.3 *Sea* $x \in T$. *Si* $\forall y.(y < x \to y \in W(T, <^*))$, *entonces* $\forall y.(y <^* x \to y \in W(T, <^*))$.

```
well_founded_part_cl_tr_l1: LEMMA
 (FORALL y: y < x IMPLIES well_founded_part[T,tr_cl(<)](y))
 IMPLIES
 (FORALL z: tr_cl(<)(z,x) IMPLIES well_founded_part[T,tr_cl(<)](z))
```

Demostración: Sea x tal que $\forall y.(y < x \to y \in W(T, <^*))$, y sea z con $z <^* x$. Entonces,

- si $z < x$, $z \in W(T, <^*)$ por hipótesis.
- si $\exists u.(z <^* u \wedge u < x)$, entonces:
  - $u \in W(T, <^*)$ por hipótesis, y
  - $z \in W(T, <^*)$ por la definición de $W(T, <^*)$.

□

Teorema 2.3.4 *Sea* $<$ *una relación binaria en* T. *Se verifica* $W(T, <) \subseteq W(T, <^*)$.

```
well_founded_part_cl_tr: THEOREM
  well_founded_part[T,<](x)
  IMPLIES
  well_founded_part[T,tr_cl(<)](x)
```

Demostración: Usando el axioma de inducción débil (1.2.5) generado por la definición de $W(T, <)$, con el predicado $P(x) \equiv x \in W(T, <^*)$, basta probar lo siguiente:

$$\forall x.(\forall y.(y < x \to y \in W(T, <^*))) \to x \in W(T, <^*)$$

En efecto, sea $a \in T$ tal que $\forall y.(y < a \to y \in W(T, <^*))$. Hay que probar que $a \in W(T, <^*)$. Para ello, por la definición de parte bien fundamentada, es suficiente probar que $\forall z.(z <^* a \to z \in W(T, <^*))$, lo que se tiene por el lema 2.3.3. □

Teorema 2.3.5 *Si* $(T, <)$ *es una relación binaria bien fundamentada, entonces* $(T, <^*)$ *también lo es.*

```
well_founded_cl_tr: THEOREM
  well_founded?[T](<) IMPLIES well_founded?[T](tr_cl(<))
```

Demostración: Usando el teorema 1.3, hay que probar que si $W(T, <) = T$, entonces $W(T, <^*) = T$, lo que se tiene aplicando 2.3.4 . □

```
END wf_cl_tr
```

2.4. Clausura reflexiva transitiva de una relación

En esta sección se define (de manera inductiva) la clausura reflexiva transitiva ($<^*$) de una relación binaria ($<$) definida en T, y se establecen sus propiedades fundamentales.

Nota 2.4.1 En el desarrollo de esta teoría se considera un conjunto no vacío T.

```
cl_rtr[T:TYPE+]: THEORY
 BEGIN
```

Nota 2.4.2 Se usarán las variables x, y, z para los elementos de T y $<$ para representar una relación binaria en T.

```
x,y,z,t: VAR T
<,rel  : VAR pred[[T,T]]
```

Definición 2.4.3 *(Definición inductiva de la clausura reflexiva transitiva de la relación $<$) Sea $<$ una relación binaria en T. Definimos la clausura reflexiva transitiva de $<$ como la menor relación $<^*$ tal que:*

$$\forall x, y \in T, x = y \vee \exists z.(x <^* z \wedge z < y) \rightarrow x <^* y$$

```
rtr_cl(<)(x,y): INDUCTIVE bool =
 x = y  OR EXISTS z: rtr_cl(<)(x,z) AND z < y
```

Nota 2.4.4 De esta forma, estamos definiendo el conjunto de elementos menores que y mediante $<^*$; es decir, los elementos generados por el conjunto de reglas $\{x = y, \exists z.(x <^* z \wedge z < y)\}$. La definición inductiva de la clausura reflexiva transitiva de $<$ genera, de manera automática, los siguientes axiomas de inducción:

- axioma de inducción débil:

$$\frac{\forall y.(x = y \vee \exists z.(z < y \wedge P(z))) \rightarrow P(y)}{\forall y.(x <^* y \rightarrow P(y))}$$

- axioma de inducción fuerte:

$$\frac{\forall y.(x = y \vee \exists z.(x <^* z \wedge z < y \wedge P(z))) \rightarrow P(y)}{\forall y.(x <^* y \rightarrow P(y))}$$

Lema 2.4.5 *Los pares de elementos que verifican la regla básica $x = y$ pertenecen al conjunto definido por $<^*$.*

```
rtr_cl_cs_basic: LEMMA
  x = y IMPLIES rtr_cl(<)(x,y)
```

Demostración: Directamente por la definición de $<^*$. □

Lema 2.4.6 *Los pares de elementos que verifican la relación $x < y$ pertenecen al conjunto definido por $<^*$. Es decir, la relación $<^*$ contiene a la relación $<$.*

```
rtr_cl_cs_rel: LEMMA
  x < y IMPLIES rtr_cl(<)(x,y)
```

Demostración: Usando la definición de $<^*$ y el lema 2.4.5. □

Teorema 2.4.7 *La relación $<^*$ es reflexiva.*

```
rtr_cl_reflexive: THEOREM
  reflexive?(rtr_cl(<))
```

Demostración: Trivial. □

Nota 2.4.8 Con objeto de demostrar la transitividad de la relación $<^*$ probamos que el conjunto de reglas $\{z = x, \exists x.(z < x \wedge x <^* y)\}$ también genera los pares de la relación $<^*$. Análogamente, para el conjunto de reglas $\{z < x, \exists x.(z < x \wedge x <^* y)\}$. Para ello, enunciamos los siguientes lemas.

Lema 2.4.9 *Sean $x, y, z \in T$. Se verifica: $z = x \wedge x <^* y \rightarrow z <^* y$.*

```
rtr_cl_transitive_l0: LEMMA
  (z = x AND rtr_cl(<)(x,y))
  IMPLIES
  rtr_cl(<)(z,y)
```

Lema 2.4.10 *Sean $x, y, z \in T$. Se verifica: $z < x \wedge x <^* y \rightarrow z <^* y$.*

```
rtr_cl_transitive_l1: LEMMA
  (z < x AND rtr_cl(<)(x,y))
  IMPLIES
  rtr_cl(<)(z,y)
```

Teorema 2.4.11 *La relación $<^*$ es transitiva.*

```
rtr_cl_transitive: THEOREM
  transitive?(rtr_cl(<))
```

Lema 2.4.12 $\forall x, y \in T. x <^* y \rightarrow (x = y \vee \exists z.(x < z \wedge z <^* y))$

```
rtr_cl_equiv_cn: LEMMA
  rtr_cl(<)(x,y) IMPLIES (x = y OR EXISTS z: x < z AND rtr_cl(<)(z,y))
```

Lema 2.4.13 $\forall x, y \in T.(x = y \vee \exists z.(x < z \wedge z <^* y)) \rightarrow x <^* y$

```
rtr_cl_equiv_cs: LEMMA
  (x = y OR EXISTS z: x < z AND rtr_cl(<)(z,y)) IMPLIES rtr_cl(<)(x,y)
```

Lema 2.4.14 $\forall x, y \in T.x < y \vee \exists z.(x < z \wedge z <^* y) \leftrightarrow x <^* y$

```
rtr_cl_equiv: LEMMA
 rtr_cl(<)(x,y) IFF (x = y OR EXISTS z: x < z AND rtr_cl(<)(z,y))
```

Teorema 2.4.15 *La clausura reflexiva transitiva, $<^*$, de una relación binaria $<$, es menor que cualquier relación reflexiva transitiva que contiene a $<$.*

```
rtr_cl_less_transitive: THEOREM
  reflexive?(rel) AND transitive?(rel) AND
  (FORALL x,y: x < y IMPLIES rel(x,y))
  IMPLIES
  FORALL x,y: rtr_cl(<)(x,y) IMPLIES rel(x,y)
```

Nota 2.4.16 Como consecuencia, se tiene que la clausura reflexiva transitiva, $<^*$, de una relación binaria $<$ es la menor relación reflexiva transitiva que contiene a $<$.

```
IMPORTING cl_tr
```

Lema 2.4.17 *Dada una relación binaria $<$, la relación $<^*$ contiene a $<^+$.*

```
tr_cl_subset_rtr_cl: LEMMA
  tr_cl(<)(x,y) IMPLIES rtr_cl(<)(x,y)
```

Nota 2.4.18 Las demostraciones de los resultados incluidos en esta teoría son análogas a las de la teoría `cl_tr`.

```
END cl_rtr
```

Capítulo 3

Relación de orden bien fundamentada en multiconjuntos

3.1. Introducción

Sea T un conjunto y $<$ una relación binaria bien fundamentada definida en T. En este capítulo definimos una relación binaria $<_1$, (en PVS, `less_1`), sobre los multiconjuntos finitos de elementos de T, y probamos que es bien fundamentada. La prueba está basada en la realizada por W. Buchholz, descrita por T. Nipkow en "An Inductive Proof of the Wellfoundedness of the Multiset Order". Finalmente, definimos la relación $<_{bag}$ entre multiconjuntos finitos (`less_bag` en PVS), como la clausura transitiva de la relación $<_1$ y probamos que es una relación bien fundamentada. La prueba usa el resultado siguiente: la clausura transitiva de una relación bien fundamentada es bien fundamentada.

3.2. Relación de orden bien fundamentada en multiconjuntos

```
finite_bags_order[T: TYPE+, <: (well_founded?[T])]: THEORY
 BEGIN

  IMPORTING finite_bags_lems[T]
```

Nota 3.2.1 Se usarán $A, B, M, N, K, M_0, K_1, K_2$ como variables sobre multiconjuntos finitos de elementos de T, y x, y, a, b como variables sobre elementos de T.

```
  A,B,M,N,K,M_0,K1,K2,M1: VAR finite_bag
  x,y,a,b:                VAR T
```

Nota 3.2.2 Para evitar la creación de numerosos TCCs análogos, establecemos el siguiente juicio:

```
  JUDGEMENT insert(x,A) HAS_TYPE finite_bag
```

Definición 3.2.3 *Un multiconjunto M es* **menor** *que un elemento a (se notará $M < a$) si todos los elementos de M son menores que a.*

```
less(M,a): bool =
  FORALL b: member(b,M) IMPLIES b < a
```

Definición 3.2.4 *Un multiconjunto N* **es menor que** *M* **según** *$<_1$ ($N <_1 M$) si N se obtiene sustituyendo un elemento a de M por un multiconjunto finito de elementos menores que a. Es decir si existe un elemento a de T, y dos multiconjuntos M_0 y K, tales que:*

- *$M = M_0 \cup \{a\}$,*
- *$N = M_0 \cup K$ y*
- *$K < a$.*

```
less_1(N,M): bool =
  EXISTS M_0,a,K: M = insert(a,M_0) AND
                  N = plus(M_0,K) AND
                  less(K,a)
```

Lema 3.2.5 *Si $N <_1 M \cup \{a\}$, entonces se verifica una de las dos condiciones siguientes:*

- *$\exists M_0 : N = M_0 \cup \{a\} \wedge M_0 <_1 M$.*
- *$\exists K : N = M \cup K \wedge K < a$.*

```
less_1_insert_cn: LEMMA
  less_1(N, insert(a,M))
  IMPLIES
  (EXISTS M_0: N = insert(a,M_0) AND less_1(M_0,M)) OR
  (EXISTS K: N = plus(M,K) AND less(K,a))
```

Demostración: Por ser $N <_1 M \cup \{a\}$, existen M_2, b, K tales que:

- $M \cup \{a\} = M_2 \cup \{b\}$
- $N = M_2 \cup K$
- $K < b$

Consideramos dos casos:

1. $a = b$.
   En este caso, basta probar que $N = M \cup K$, usando las definiciones de las operaciones entre multiconjuntos.

2. $a \neq b$.
   Ahora, tomamos $M_0 = (M_2 \setminus \{a\}) \cup K$ y probamos:

   a) $N = M_0 \cup \{a\}$. Es decir, $M_2 \cup K = ((M_2 \setminus \{a\}) \cup K) \cup \{a\}$, usando las definiciones de las operaciones entre multiconjuntos.

   b) $M_0 <_1 M$. Es decir, $(M_2 \setminus \{a\}) \cup K <_1 M$. Para ello, basta tomar $M_2 \setminus \{a\}$, b y K como los elementos necesarios para probar la relación.

□

Nota 3.2.6 Incluimos aquí algunas propiedades sobre las operaciones con multiconjuntos que hemos necesitado.

Lema 3.2.7 $M \cup \emptyset = M$

```
plus_emptybag: LEMMA
  plus(M,emptybag) = M
```

Lema 3.2.8 $M \cup (N \cup \{x\}) = (M \cup N) \cup \{x\}$

```
insert_plus: LEMMA
  plus(M,insert(x,N)) = insert(x,plus(M,N))
```

Lema 3.2.9 *Si* $(K \cup \{x\}) < a$*, entonces* $x < a$*.*

```
less_insert_cn: LEMMA
  less(insert(x,K),a) IMPLIES x < a
```

Lema 3.2.10 *Si* $(K \cup \{x\}) < a$*, entonces* $K < a$*.*

```
less_insert_cn2: LEMMA
  less(insert(x,K),a) IMPLIES less(K,a)
```

Nota 3.2.11 Importamos la teoría "wf_cs", instanciándola con los multiconjuntos finitos con elementos en T, $\mathcal{M}(T)$, y la relación $<_1$. De esta forma, la prueba de que $(\mathcal{M}(T), <_1)$ es una relación bien fundamentada consistirá en probar que la parte bien fundamentada de $\mathcal{M}(T)$, relativa a la relación $<_1$, coincide con $\mathcal{M}(T)$. Es decir, que $W(\mathcal{M}(T), <_1) = \mathcal{M}(T)$.

```
wf_part: LIBRARY = "../wf"
IMPORTING wf_part@wf_cs[finite_bag[T],less_1]
```

Nota 3.2.12 El esquema de la prueba de $W(\mathcal{M}(T), <_1) = \mathcal{M}(T)$ es el siguiente:

- $W(\mathcal{M}(T), <_1) \subseteq \mathcal{M}(T)$ es trivial.

- La prueba de $\mathcal{M}(T) \subseteq W(\mathcal{M}(T), <_1)$ se realiza por inducción sobre multiconjuntos finitos, según el siguiente esquema de inducción:

$$\frac{P(\emptyset) \wedge \forall a, M.(P(M) \rightarrow P(M \cup \{a\}))}{\forall M.P(M)}$$

siendo $P(M) \equiv M \in W(\mathcal{M}(T), <_1)$.

Por tanto, hay que probar:

1. $\emptyset \in W(\mathcal{M}(T), <_1)$, que se tiene por la definición de $W(\mathcal{M}(T), <_1)$.
2. $\forall a(\forall M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{a\} \in W(\mathcal{M}(T), <_1)$.
   Este resultado está formalizado en el lema 3.2.16. Se demuestra por inducción bien fundamentada en a, (puesto que $[T, <]$ es bien fundamentada), con el predicado:

$$P(a) \equiv \forall M(M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{a\} \in W(\mathcal{M}(T), <_1))$$

   Con ello, la prueba se reduce a demostrar

$$(\forall b.(b < a \rightarrow P(b))) \rightarrow P(a)$$

   Este resultado está formalizado en el lema 3.2.15.
   Hay que demostrar que si se verifica

$$\forall b.[b < a \rightarrow \forall M(M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{b\} \in W(\mathcal{M}(T), <_1))] \qquad (*)$$

   entonces

$$\forall M(M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{a\} \in W(\mathcal{M}(T), <_1)$$

   o, lo que es equivalente,

$$\forall M \in W(\mathcal{M}(T), <_1).M \cup \{a\} \in W(\mathcal{M}(T), <_1)$$

   La prueba se hace usando el axioma de inducción débil de la parte bien fundamentada $W(\mathcal{M}(T), <_1)$, con el predicado

$$Q(M) \equiv M \cup \{a\} \in W(\mathcal{M}(T), <_1) \vee M \notin W(\mathcal{M}(T), <_1)$$

   Por tanto, con la hipótesis $(*)$, hemos de probar $\forall M(\forall N <_1 M.Q(N)) \rightarrow Q(M)$.
   Es decir:
   si $\forall M(\forall N <_1 M(N \in W(\mathcal{M}(T), <_1) \rightarrow N \cup \{a\} \in W(\mathcal{M}(T), <_1)$,
   entonces $M \cup \{a\} \in W(\mathcal{M}(T), <_1) \vee M \notin W(\mathcal{M}(T), <_1)$
   Así, la prueba se reduce al lema 3.2.14, que queda como sigue:
   si

   - $M_0 \in W(\mathcal{M}(T), <_1)$
   - $\forall M <_1 M_0.(M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{a\} \in W(\mathcal{M}(T), <_1))$

- $\forall b < a.(\forall M.M \in W(\mathcal{M}(T), <_1) \to M \cup \{b\} \in W(\mathcal{M}(T), <_1)$

entonces, $M_0 \cup \{a\} \in W(\mathcal{M}(T), <_1)$.

Por último, probamos el lema auxiliar 3.2.13, que establece una condición suficiente para que $M \cup K \in W(\mathcal{M}(T), <_1)$.

Lema 3.2.13 *Sea $a \in T$ y sean M, K multiconjuntos finitos tales que:*

- $M \in W(\mathcal{M}(T), <_1)$
- $K < a$
- $\forall b < a.(\forall M.M \in W(\mathcal{M}(T), <_1) \to M \cup \{b\} \in W(\mathcal{M}(T), <_1)$

Entonces, $M \cup K \in W(\mathcal{M}(T), <_1)$.

```
well_founded_part_less_1_plus: LEMMA
 well_founded_part(M) AND
 less(K,a) AND
 (FORALL b: b<a IMPLIES (FORALL M: well_founded_part(M) IMPLIES
                                   well_founded_part(insert(b,M))))
 IMPLIES
 well_founded_part(plus(M,K))
```

Demostración: Sea $M \in W(\mathcal{M}(T), <_1)$ y a un elemento de T. Probamos el lema, por inducción en K, según el esquema

$$\frac{P(\emptyset) \land \forall b, K.(P(K) \to P(K \cup \{b\}))}{\forall K.P(K)}$$

siendo $P(K) \equiv M \cup K \in W(\mathcal{M}(T), <_1)$.
Tenemos dos casos:

1. $K = \emptyset$. Entonces, $M \cup K = M$ y, por hipótesis, $M \in W(\mathcal{M}(T), <_1)$.
2. $K = X \cup \{b\}$, verificando X la hipótesis de inducción.

   Entonces,

   - por una parte, por el lema 3.2.8, $M \cup K = M \cup (X \cup \{b\}) = (M \cup X) \cup \{b\}$
   - por otra parte, como $K = X \cup \{b\} <_1 a$, por el lema 3.2.10 se tiene $X < a$
   - por tanto, por hipótesis de inducción $M \cup X \in W(\mathcal{M}(T), <_1)$
   - además, por el lema 3.2.9, se tiene que $b < a$
   - luego, por hipótesis del lema, $(X \cup M) \cup \{b\} \in W(\mathcal{M}(T), <_1)$
   - por tanto, $M \cup K \in W(\mathcal{M}(T), <_1)$

$\square$

Lema 3.2.14 *Sea $a \in T$ y sean M, M_0 multiconjuntos finitos tales que:*

- $M_0 \in W(\mathcal{M}(T), <_1)$
- $\forall M <_1 M_0.(M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{a\} \in W(\mathcal{M}(T), <_1))$
- $\forall b < a.(\forall M.M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{b\} \in W(\mathcal{M}(T), <_1)$

Entonces, $M_0 \cup \{a\} \in W(\mathcal{M}(T), <_1)$.

```
well_founded_part_less_1_insert_l0: LEMMA
  well_founded_part(M_0) AND
  (FORALL M: less_1(M,M_0) IMPLIES (well_founded_part(M)
                                    IMPLIES
                                    well_founded_part(insert(a,M)))) AND
  (FORALL b: b < a IMPLIES
           (FORALL M: well_founded_part(M) IMPLIES
               well_founded_part(insert(b,M))))
  IMPLIES
  well_founded_part(insert(a, M_0))
```

Demostración: Considerando la definición inductiva de $W(\mathcal{M}(T), <_1)$, hemos de probar lo siguiente:

$$\forall M_1.(M_1 <_1 M_0 \cup \{a\} \rightarrow M_1 \in W(\mathcal{M}(T), <_1)$$

Sea M_1 tal que $M_1 <_1 M_0 \cup \{a\}$. Por el lema 3.2.5, tenemos dos casos:

1. $\exists N : M_1 = N \cup \{a\}$ y $N <_1 M_0$.

   En este caso, por ser $N <_1 M_0$, de la definición inductiva de $W(\mathcal{M}(T), <_1)$ se tiene que $N \in W(\mathcal{M}(T), <_1)$. Luego, por hipótesis, $N \cup \{a\} \in W(\mathcal{M}(T), <_1)$. Por tanto, $M_1 \in W(\mathcal{M}(T), <_1)$.

2. $\exists K : M_1 = M_0 \cup K$ y $K < a$.

   Este caso se prueba por aplicación directa del lema 3.2.13.

□

Lema 3.2.15 *Si* $\forall b < a.(\forall M \in W(\mathcal{M}(T), <_1).M \cup \{b\} \in W(\mathcal{M}(T))$*, entonces* $\forall M \in W(\mathcal{M}(T), <_1).M \cup \{a\} \in W(\mathcal{M}(T), <_1)$.

```
well_founded_part_less_1_insert_l2: LEMMA
 (FORALL b: b<a IMPLIES
               (FORALL M: well_founded_part(M) IMPLIES
               well_founded_part(insert(b,M))))
 IMPLIES
 (FORALL M: well_founded_part(M) IMPLIES
           well_founded_part(insert(a,M)))
```

Demostración: Usamos el axioma de inducción débil de la parte bien fundamentada $W(\mathcal{M}(T), <_1)$, con el predicado $Q(M) \equiv (M \cup \{a\} \in W(\mathcal{M}(T), <_1) \vee M \notin W(\mathcal{M}(T), <_1)$ y aplicamos el lema 3.2.14. □

Lema 3.2.16 *Si $M \in W(\mathcal{M}(T), <_1)$, entonces $M \cup \{a\} \in W(\mathcal{M}(T), <_1)$.*

```
well_founded_part_less_1_insert: LEMMA
  FORALL M, a : well_founded_part(M) IMPLIES
                well_founded_part(insert(a,M))
```

Demostración: Por inducción bien fundamentada en a, según el esquema "wf_induction[T,<]"(puesto que $<$ es una relación bien fundamentada sobre T):

$$\frac{\forall x \in T.(\forall y \in T.(y < x \rightarrow P(y)) \rightarrow P(x))}{\forall x \in T.P(x)}$$

siendo $P(x) \equiv \forall M(M \in W(\mathcal{M}(T), <_1) \rightarrow M \cup \{x\} \in W(\mathcal{M}(T), <_1)$, usando el lema 3.2.15. □

Lema 3.2.17 *Todo multiconjunto finito con elementos en T pertenece a la parte bien fundamentada de T, relativa a $<_1$. Es decir, $\mathcal{M}(T) \subseteq W(\mathcal{M}(T), <_1)$.*

```
well_founded_part_less_1_all: LEMMA
  FORALL M: well_founded_part(M)
```

Demostración: Por inducción sobre multiconjuntos finitos, según el siguiente esquema de inducción:

$$\frac{P(\emptyset) \wedge \forall a, M.(P(M) \rightarrow P(M \cup \{a\}))}{\forall M.P(M)}$$

con el predicado $P(M) \equiv M \in W(\mathcal{M}(T), <_1)$. Entonces,

- En el caso del multiconjunto vacío, $\emptyset \in W(\mathcal{M}(T), <_1)$, pues ningún multiconjunto N verifica $N <_1 \emptyset$.
- El paso inductivo se tiene de forma inmediata a partir del lema 3.2.16.

□

Teorema 3.2.18 *$(\mathcal{M}(T), <_1)$ es una relación bien fundamentada.*

```
wf_less_1: THEOREM
  well_founded?[finite_bag[T]](less_1)
```

Demostración: Por el teorema 1.3, es suficiente probar que la parte bien fundamentada relativa a la relación $<_1$ coincide con el conjunto de los multiconjuntos finitos sobre T. Es decir, $W(\mathcal{M}(T), <_1) = \mathcal{M}(T)$. Esto se tiene, directamente, por el lema 3.2.17. □

Nota 3.2.19 Importamos la teoría "wf_cl_tr"instanciándola sobre los multiconjuntos finitos de T, en la que se prueba que la clausura transitiva de una relación bien fundamentada también es bien fundamentada.

```
IMPORTING wf_part@wf_cl_tr[finite_bag[T]]
```

Definición 3.2.20 *Definimos la relación de orden entre multiconjuntos finitos $<_{bag}$ como la clausura transitiva de la relación $<_1$.*

```
less_bag: PRED[[finite_bag[T],finite_bag[T]]] = tr_cl(less_1)
```

Teorema 3.2.21 $(\mathcal{M}(T), <_{bag})$ *es una relación bien fundamentada.*

```
wf_less_bag: THEOREM
 well_founded?[finite_bag[T]](less_bag)
```

Demostración: Por definición, $<_{bag}$ es la clausura transitiva de la relación $<_1$, que es bien fundamentada, por el lema 3.2.18. Entonces, por el teorema 2.3.5, se tiene que $<_{bag}$ es bien fundamentada. □

3.3. Equivalencia entre relaciones binarias en $\mathcal{M}(T)$.

Finalmente, queremos establecer la relación que existe entre la relación $N <_{bag} M$, y la definida como el resultado de sustituir un multiconjunto K_1 de elementos de M por otro multiconjunto K_2 formado por elementos acotados superiormente por elementos de K_1. Para ello, en primer lugar, definimos la relación $<_{mult}$ como sigue:

Definición 3.3.1 $N <_{mult} M$ *si existen multiconjuntos* M_0, K_1, K_2 *tales que:*

- $K_1 \neq \emptyset$
- $M = M_0 \cup K_1$
- $N = M_0 \cup K_2$
- $\forall a.(a \in K_2 \rightarrow \exists b.(b \in K_1 \wedge a < b))$

```
less_mult(N,M): bool =
  EXISTS M_0, K1, K2: nonempty_bag?(K1) AND
                      M = plus(M_0, K1) AND
                      N = plus(M_0, K2) AND
                      FORALL a: member(a,K2) IMPLIES
                                EXISTS b: member(b,K1) AND a < b
```

Vamos a probar que, en el caso de que la relación binaria $<$ sea transitiva, las relaciones $<_{bag}$ y $<_{mult}$ coinciden. El esquema de la prueba es el siguiente:

- $<_{bag} \subseteq <_{mult}$

  Como $<_{bag} \equiv <_1^+$ es la menor relación transitiva que contiene a $<_1$, es suficiente probar que $<_{mult}$ es transitiva y contiene a $<_1$.

- $<_{mult} \subseteq <_{bag}$

  Se prueba por inducción en el multiconjunto de elementos del mayor, que se sustituye.

Lema 3.3.2 *Si $<$ es una relación transitiva en T, entonces la relación $<_{mult}$ es transitiva en $\mathcal{M}(T)$.*

```
less_mult_transitive_cs: LEMMA
  transitive?[T](<) IMPLIES transitive?[finite_bag](less_mult)
```

Demostración: Sean X, Y, Z multiconjuntos tales que $X <_{mult} Y$ y $Y <_{mult} Z$. Vamos a probar que $X <_{mult} Z$.

Por definición de $<_{mult}$, existen multiconjuntos $M_0, K_1, K_2, M_1, L_1, L_2$ tales que:

(1)	$K_1 \neq \emptyset$	(5)	$L_1 \neq \emptyset$
(2)	$Y = M_0 \cup K_1$	(6)	$Z = M_1 \cup L_1$
(3)	$X = M_0 \cup K_2$	(7)	$Y = M_1 \cup L_2$
(4)	$\forall a.(a \in K_2 \rightarrow \exists b.(b \in K_1 \wedge a < b))$	(8)	$\forall a.(a \in L_2 \rightarrow \exists b.(b \in L_1 \wedge a < b))$

Hemos de determinar multiconjuntos M_0', K_1', K_2' tales que

- $K_1' \neq \emptyset$
- $Z = M_0' \cup K_1'$
- $X = M_0' \cup K_2'$
- $\forall a.(a \in K_2' \rightarrow \exists b.(b \in K_1' \wedge a < b))$

Para ello, consideremos las siguientes cuestiones:

- ¿Cuáles son los elementos que, con seguridad, se han quitaqdo de Z?: los elementos de L_1 y los elementos de K_1 que sean también de M_1. Por tanto, tomamos

$$K_1' = (K_1 \cap M_1) \cup L_1$$

- ¿Cuáles son los elementos de M_1 que quedan?: los que no se han quitado, es decir, $M_1 \setminus (K_1 \cap M_1)$. Por tanto, tomamos

$$M_0' = M_1 \setminus (K_1 \cap M_1)$$

- ¿Qué elementos le faltan a M_0' para obtener $X = M_0 \cup K_2$?: teniendo en cuenta que $M_1 \setminus (K_1 \cap M_1) \subseteq M_0$, tomamos

$$K_2' = (M_0 \setminus (M_1 \setminus (K_1 \cap M_1))) \cup K_2$$

Así, hemos de probar que estos conjuntos verifican las condiciones para que $X <_{mult} Z$. En efecto:

- $K'_1 \neq \emptyset$. Por (5).
- $Z = M'_0 \cup K'_1$

  $M'_0 \cup K'_1 = (M_1 \setminus (K_1 \cap M_1)) \cup (K_1 \cap M_1) \cup L_1 = M_1 \cup L_1 = Z$, usando el lema "difference_descom_lem", pues $K_1 \cap M_1 \subseteq M_1$.
- $X = M'_0 \cup K'_2$

  Es suficiente comprobar que, para todo elemento se tiene

  $(M'_0 \cup K'_2)(x)$
  $= M'_0(x) + K'_2(x)$
  $= M_1(x) - min(K_1(x), M_1(x)) + M_0(x) - M_1(x) + min(K_1(x), M_1(x)) + K_2(x)$
  $= M_0(x) + K_2(x) = X(x)$
- $\forall a.(a \in K'_2 \rightarrow \exists b.(b \in K'_1 \wedge a < b))$

  Para probarlo, tengamos en cuenta que, por (2) y (7), $M_1 \setminus (K_1 \cap M_1) \subseteq M_0$ y que también $M_0 \setminus (M_1 \setminus (K_1 \cap M_1)) \subseteq L_2$.

  Sea x tal que $x \in K'_2$. Tenemos dos casos:

  1. $x \in M_0 \setminus (M_1 \setminus (K_1 \cap M_1))$
     En este caso, $x \in L_2$ y, por (8) $(\exists y \in L_1).(x < y)$. Como $L_1 \subseteq K'_1$, se tiene que $y \in K'_1$.
  2. $x \in K_2$
     Por (4), $(\exists y \in K_1).(x < y)$. Entonces:
     - si $y \in M_1$, se tiene que $y \in K_1 \cap M_1$ y, por tanto, $y \in K'_1$.
     - si $y \notin M_1$, entonces por (2) y (7), $y \in L_2$ y, por (8), $(\exists z \in L_1).(y < z)$. Luego, $z \in K'_1$ y, como $<$ es **transitiva**, se tiene que $x < z$.

□

Lema 3.3.3 *La relación $<_{mult}$ contiene a la relación $<_1$.*

```
less_1_subset_less_mult: LEMMA
  less_1(N,M) IMPLIES less_mult(N,M)
```

Demostración: Por definición de $<_1$ existe un elemento a de T, y dos multiconjuntos M_0 y K, tales que:

- $M = M_0 \cup \{a\}$,
- $N = M_0 \cup K$ y
- $K < a$.

Para probar que $N <_{mult} M$ basta tomar K_1 el multiconjunto cuyo único elemento es a, y comprobar que M_0, K_1, K verifican las condiciones para que $N <_{mult} M$. □

Lema 3.3.4 *Si $<$ es una relación transitiva en T, entonces la relación $<_{mult}$ contiene a $<_{bag}$.*

```
less_bag_subset_less_mult: LEMMA
  transitive?[T](<)
  IMPLIES
  (less_bag(N,M) IMPLIES less_mult(N,M))
```

Demostración: Usando los lemas 2.2.14, 3.3.2 y 3.3.3. □

```
less_bag_subset_less_mult_bis: LEMMA
  transitive?[T](<)
  IMPLIES
  (FORALL M,N: less_bag(N,M) IMPLIES less_mult(N,M))
```

Lema 3.3.5 *Sean M, N, M_0, K_1, K_2 multiconjuntos tales que:*

- $K_1 \neq \emptyset$
- $M = M_0 \cup K_1$
- $N = M_0 \cup K_2$
- $\forall a.(a \in K_2 \to \exists b.(b \in K_1 \wedge a < b))$

Entonces, $N <_{bag} M$.

```
less_mult_subset_less_bag_lem: LEMMA
  (nonempty_bag?(K1) AND
   M = plus(M_0, K1) AND
   N = plus(M_0, K2) AND
   FORALL a: member(a,K2) IMPLIES EXISTS b: member(b,K1) AND a < b)
  IMPLIES
  less_bag(N,M)
```

Demostración: Por inducción en la variable K_1, se tienen dos casos:

- $K_1 = \emptyset$

  Trivial.

- $K_1 = R_1 \cup \{a\}$

  Se distinguen dos casos:

- $R_1 = \emptyset$
  En este caso, es fácil comprobar que $N <_1 M$ y, por tanto, $N <_{bag} M$, pues $<_{bag} \equiv <_1^+$.
- $R_1 \neq \emptyset$
  Por definición de $<_1^+$, hay que probar que existe un multiconjunto Z tal que $N <_{bag} Z$ y $Z <_1 M$. Para determinar Z tal que está a un paso según $<_1$ de M, consideremos el multiconjunto formado por todos los elementos de K_2 que no están acotados por elementos de R_1, es decir,

  $$R_2 = [[x : x \in K_2 \wedge \forall y \in R_1, x \not< y]]$$

  Entonces, tomamos $Z = (M_0 \cup R_2) \cup R_1$, y probamos que verifica lo siguiente:
  - $N <_{bag} Z$
    Tenemos que:
    - $Z = (M_0 \cup R_2) \cup R_1$
    - $N = M_0 \cup K_2 = (M_0 \cup R_2) \cup (K_2 \setminus R_2)$
    - $\forall x \in (K_2 \setminus R_2) \exists y \in R_1 : x < y$

    Por tanto, aplicando la hipótesis de inducción a los multiconjuntos $M'_0 = M_0 \cup R_2$, $K'_1 = R_1$ y $K'_2 = K_2 \setminus R_2$, se verifica que $N <_{bag} Z$.
  - $Z <_1 M$
    Se tiene que

    $$Z = (M_0 \cup R_2) \cup R_1 = (M_0 \cup R_1) \cup R_2 <_1 (M_0 \cup R_1) \cup [[a]]$$

□

Lema 3.3.6 *La relación $<_{bag}$ contiene a la relación $<_{mult}$.*

```
less_mult_subset_less_bag: LEMMA
  less_mult(N,M) IMPLIES less_bag(N,M)
```

Demostración: Inmediato, a partir de 3.3.5 □

Teorema 3.3.7 *Si la relación binaria $<$ es transitiva, entonces las relaciones $<_{bag}$ y $<_{mult}$ coinciden.*

```
less_mult_equiv_less_bag: THEOREM
  transitive?[T](<)
  IMPLIES
  (less_mult(N,M) IFF less_bag(N,M))
```

Demostración: Usando los lemas 3.3.6 y 3.3.4. □

Como consecuencia, se tiene el siguiente resultado.

Corolario 3.3.8 *Si la relación binaria $<$ es transitiva, entonces la relación $(\mathcal{M}(T), <_{mult})$ es bien fundamentada.*

```
less_mult_is_wf: COROLLARY
  transitive?[T](<)
  IMPLIES
  well_founded?[finite_bag[T]](less_mult)
```

```
END finite_bags_order
```

Apéndice A

Teorías PVS de la formalización

A.1. Parte bien fundamentada de una relación binaria

```
wf_cs[T: TYPE+, <: pred[[T, T]]]: THEORY
 BEGIN
  ASSUMING
   T_TCC1: ASSUMPTION EXISTS (x: T): TRUE;
  ENDASSUMING

  p: VAR pred[T]

  x, y: VAR T

  well_founded_part(x):  INDUCTIVE bool =
        FORALL y: y < x IMPLIES well_founded_part(y)

  well_founded_part_weak_induction: AXIOM
    FORALL (P: [T -> boolean]):
      (FORALL (x): (FORALL y: y < x IMPLIES P(y)) IMPLIES P(x)) IMPLIES
       (FORALL (x): well_founded_part(x) IMPLIES P(x));

  well_founded_part_induction: AXIOM
    FORALL (P: [T -> boolean]):
      (FORALL (x):
         (FORALL y: y < x IMPLIES well_founded_part(y) AND P(y)) IMPLIES
          P(x))
       IMPLIES (FORALL (x): well_founded_part(x) IMPLIES P(x));

  well_founded_part_cn: LEMMA
    well_founded?[T](<) IMPLIES (FORALL x: well_founded_part(x))

  well_founded_part_cs: LEMMA
    (FORALL x: well_founded_part(x)) IMPLIES well_founded?[T](<)

  well_founded_part_cns: THEOREM
    well_founded?[T](<) IFF (FORALL x: well_founded_part(x))

  well_founded_induct_cns: THEOREM
```

```
    well_founded?[T](<) IFF
     (FORALL (p: pred[T]):
        (FORALL (x: T): (FORALL (y: T): y < x IMPLIES p(y)) IMPLIES p(x))
         IMPLIES (FORALL (x: T): p(x)))

  well_founded_cs_ordinal: THEOREM
    (EXISTS (f: [T -> ordinal]): FORALL x, y: x < y IMPLIES f(x) < f(y))
     IMPLIES well_founded?[T](<)
 END wf_cs
```

A.2. Clausura transitiva de una relación

```
cl_tr[T: TYPE+]: THEORY
 BEGIN
  ASSUMING
   T_TCC1: ASSUMPTION EXISTS (x: T): TRUE;
  ENDASSUMING

  x, y, z, t: VAR T

  <, rel: VAR pred[[T, T]]

  tr_cl(<)(x, y):  INDUCTIVE bool =
        x < y OR (EXISTS z: tr_cl(<)(x, z) AND z < y)

  tr_cl_weak_induction: AXIOM
    FORALL (<, x, P: [T -> boolean]):
      (FORALL (y): (x < y OR (EXISTS z: P(z) AND z < y)) IMPLIES P(y))
       IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y));

  tr_cl_induction: AXIOM
    FORALL (<, x, P: [T -> boolean]):
      (FORALL (y):
         (x < y OR (EXISTS z: (tr_cl(<)(x, z) AND P(z)) AND z < y)) IMPLIES
          P(y))
       IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y));

  tr_cl_cs_basic: LEMMA x < y IMPLIES tr_cl(<)(x, y)

  tr_cl_transitive_l0: LEMMA
    (z < x AND tr_cl(<)(x, y)) IMPLIES tr_cl(<)(z, y)

  tr_cl_transitive: THEOREM transitive?(tr_cl(<))

  tr_cl_equiv_cn: LEMMA
    tr_cl(<)(x, y) IMPLIES (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y)))

  tr_cl_equiv_cs: LEMMA
    (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y))) IMPLIES tr_cl(<)(x, y)

  tr_cl_equiv: LEMMA
```

```
    tr_cl(<)(x, y) IFF (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y)))

  tr_cl_less_transitive: THEOREM
    transitive?(rel) AND (FORALL x, y: x < y IMPLIES rel(x, y)) IMPLIES
     (FORALL x, y: tr_cl(<)(x, y) IMPLIES rel(x, y))
 END cl_tr
```

A.3. Buena fundamentación de la clausura transitiva de una relación

```
wf_cl_tr[T: TYPE+]: THEORY
 BEGIN
  ASSUMING
   T_TCC1: ASSUMPTION EXISTS (x: T): TRUE;
  ENDASSUMING

  <: VAR pred[[T, T]]

  x, y, z: VAR T

  IMPORTING cl_tr

  IMPORTING wf_cs

  well_founded_part_cl_tr_l1: LEMMA
    (FORALL y: y < x IMPLIES well_founded_part[T, tr_cl(<)](y)) IMPLIES
     (FORALL z: tr_cl(<)(z, x) IMPLIES well_founded_part[T, tr_cl(<)](z))

  well_founded_part_cl_tr: THEOREM
    well_founded_part[T, <](x) IMPLIES well_founded_part[T, tr_cl(<)](x)

  well_founded_cl_tr: THEOREM
    well_founded?[T](<) IMPLIES well_founded?[T](tr_cl(<))
 END wf_cl_tr
```

A.4. Relación de orden bien fundamentada en multiconjuntos

```
finite_bags_order[T: TYPE+, <: (well_founded?[T])]: THEORY
 BEGIN
  ASSUMING
   T_TCC1: ASSUMPTION EXISTS (x: T): TRUE;
  ENDASSUMING

  IMPORTING finite_bags_lems[T]

  finite_emptyset: JUDGEMENT emptyset HAS_TYPE finite_set[T]

  Complement_bijective: JUDGEMENT Complement HAS_TYPE
      (bijective?[setofsets[T], setofsets[T]])
```

```
Intersection_surjective: JUDGEMENT Intersection HAS_TYPE
    (surjective?[setofsets[T], set[T]])

Union_surjective: JUDGEMENT Union HAS_TYPE
    (surjective?[setofsets[T], set[T]])

strict_subset_is_strict_order: JUDGEMENT strict_subset?[T] HAS_TYPE
    (strict_order?[set[T]])

subset_is_partial_order: JUDGEMENT subset?[T] HAS_TYPE
    (partial_order?[set[T]])

JUDGEMENT emptybag HAS_TYPE finite_bag[T]

A, B, M, N, K, M_0, K1, K2: VAR finite_bag

x, y, a, b: VAR T

% Judgement subtype TCC generated (at line 51, column 12) for  insert(x, A)
  % expected type  finite_bag[T]
insert_TCC1: OBLIGATION FORALL (A, x): is_finite[T](insert[T](x, A));

JUDGEMENT insert(x, A) HAS_TYPE finite_bag

less(M, a): bool = FORALL b: member(b, M) IMPLIES b < a

less_1(N, M): bool =
    EXISTS M_0, a, K:
      M = insert(a, M_0) AND N = plus(M_0, K) AND less(K, a)

less_1_insert_cn: LEMMA
  less_1(N, insert(a, M)) IMPLIES
   (EXISTS M_0: N = insert(a, M_0) AND less_1(M_0, M)) OR
    (EXISTS K: N = plus(M, K) AND less(K, a))

plus_emptybag: LEMMA plus(M, emptybag) = M

insert_plus: LEMMA plus(M, insert(x, N)) = insert(x, plus(M, N))

less_insert_cn: LEMMA less(insert(x, K), a) IMPLIES x < a

less_insert_cn2: LEMMA less(insert(x, K), a) IMPLIES less(K, a)

wf_part: LIBRARY = "../wf"

% Existence TCC generated (at line 222, column 12) for
  % wf_part@wf_cs[finite_bag[T], less_1]
IMP_wf_cs_TCC1: OBLIGATION EXISTS (x: finite_bag[T]): TRUE;

IMPORTING wf_part@wf_cs[finite_bag[T], less_1]
```

```
% Subtype TCC generated (at line 326, column 21) for  plus(M, K)
  % expected type  finite_bag[T]
well_founded_part_less_1_plus_TCC1: OBLIGATION
  FORALL (K, M1: finite_bag[T], a: T):
    (FORALL b:
       b < a IMPLIES
        (FORALL M:
           well_founded_part(M) IMPLIES well_founded_part(insert(b, M))))
     AND less(K, a) AND well_founded_part(M1)
     IMPLIES is_finite[T](plus[T](M1, K));

well_founded_part_less_1_plus: LEMMA
  well_founded_part(M) AND
   less(K, a) AND
    (FORALL b:
       b < a IMPLIES
        (FORALL M:
           well_founded_part(M) IMPLIES well_founded_part(insert(b, M))))
   IMPLIES well_founded_part(plus(M, K))

well_founded_part_less_1_insert_l0: LEMMA
  well_founded_part(M_0) AND
   (FORALL M:
      less_1(M, M_0) IMPLIES
       (well_founded_part(M) IMPLIES well_founded_part(insert(a, M))))
    AND
    (FORALL b:
       b < a IMPLIES
        (FORALL M:
           well_founded_part(M) IMPLIES well_founded_part(insert(b, M))))
   IMPLIES well_founded_part(insert(a, M_0))

well_founded_part_less_1_insert_l2: LEMMA
  (FORALL b:
     b < a IMPLIES
      (FORALL M:
         well_founded_part(M) IMPLIES well_founded_part(insert(b, M))))
   IMPLIES
   (FORALL M:
      well_founded_part(M) IMPLIES well_founded_part(insert(a, M)))

well_founded_part_less_1_insert: LEMMA
  FORALL M, a:
    well_founded_part(M) IMPLIES well_founded_part(insert(a, M))

well_founded_part_less_1_all: LEMMA FORALL M: well_founded_part(M)

wf_less_1: THEOREM well_founded?[finite_bag[T]](less_1)

IMPORTING wf_part@wf_cl_tr[finite_bag[T]]

less_bag: PRED[[finite_bag[T], finite_bag[T]]] = tr_cl(less_1)
```

```
  wf_less_bag: THEOREM well_founded?[finite_bag[T]](less_bag)

  less_bag_cn: LEMMA
    less_bag(N, M) IMPLIES
     (EXISTS M_0, K1, K2:
        M = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

  less_bag_cs_l1: LEMMA
    (M = plus(M_0, K1) AND
      N = plus(M_0, K2) AND
       (FORALL a:
          member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
     IMPLIES less_bag(N, M)

  less_bag_cs: LEMMA
    (EXISTS M_0, K1, K2:
       M = plus(M_0, K1) AND
        N = plus(M_0, K2) AND
         (FORALL a:
            member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
     IMPLIES less_bag(N, M)

  less_bag_caract: LEMMA
    less_bag(N, M) IFF
     (EXISTS M_0, K1, K2:
        M = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
 END finite_bags_order
```

Apéndice B

Guiones de pruebas de las teorías PVS de la formalización

B.1. Parte bien fundamentada de una relación binaria

```
Proof scripts for file wf_cs.pvs:


wf_cs.well_founded_part_cn: proved - complete [shostak](0.06 s)

(""
 (prop)
 (use "wf_induction[T,<]")
 (prop)
 (hide -1 2)
 (skosimp)
 (expand "well_founded_part" +)
 (propax))


wf_cs.well_founded_part_cs: proved - complete [shostak](0.11 s)

(""
 (prop)
 (expand "well_founded?")
 (skosimp)
 (lemma "well_founded_part_weak_induction")
 (inst - "lambda(x): NOT p!1(x)")
 (prop)
 (("1" (grind :defs nil))
  ("2"
   (skosimp)
   (inst? +)
   (skolem-typepred)
   (skolem-typepred)
   (inst? -2)
   (prop))))
```

```
wf_cs.well_founded_part_cns: proved - complete [shostak](0.01 s)

(""
 (prop)
 (("1" (rewrite "well_founded_part_cn"))
  ("2" (rewrite "well_founded_part_cs"))))


wf_cs.well_founded_induct_cns: proved - complete [shostak](0.06 s)

(""
 (prop)
 (("1" (use "wf_induction[T,<]"))
  ("2"
   (apply (then (expand "well_founded?") (skosimp)
           (inst - "lambda(x): not p!1(x)") (skosimp) (prop)))
   (("1" (grind :defs nil))
    ("2"
     (skosimp)
     (inst? +)
     (apply (then (skolem-typepred) (inst?) (prop))))))))


wf_cs.well_founded_cs_ordinal: proved - complete [shostak](0.07 s)

(""
 (skosimp*)
 (use "well_founded_le")
 (expand "well_founded?")
 (skosimp*)
 (apply (then
         (inst -1
          "lambda(u:ordinal): Exists (x:T): p!1(x) AND f!1(x) = u")
         (prop)))
 (("1"
   (skosimp - t)
   (skosimp)
   (inst + "x!1")
   (skosimp)
   (inst -5 "x!2" "x!1")
   (prop)
   (inst - "f!1(x!2)")
   (("1" (replace*)) ("2" (typepred "x!2") (inst?))))
  ("2" (inst + "f!1(y!1)") (inst?))))
```

B.2. Clausura transitiva de una relación

```
Proof scripts for file cl_tr.pvs:
```

```
cl_tr.tr_cl_less_rules: proved - complete [shostak](0.95 s)

(""
 (skosimp)
 (skolem 1 ("x!1" "_"))
 (inst?)
 (rewrite "tr_cl_weak_induction"))


cl_tr.tr_cl_weak_induction_2: proved - complete [shostak](0.51 s)

(""
 (skosimp)
 (skolem 1 ("x!1" "_"))
 (inst?)
 (rewrite "tr_cl_weak_induction"))


cl_tr.tr_cl_cs_basic: proved - complete [shostak](0.02 s)

("" (skosimp) (expand "tr_cl") (prop))


cl_tr.tr_cl_transitive_l0: proved - complete [shostak](0.04 s)

(""
 (lemma "tr_cl_weak_induction")
 (skolem 1 ("lessp!1" "x!1" "_" "z!1"))
 (inst - "lessp!1" "x!1"
  "lambda(y): not(lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1,y)")
 (prop)
 (("1" (skosimp) (inst?) (prop))
  ("2"
   (hide 2)
   (skosimp)
   (prop)
   (("1"
     (expand "tr_cl")
     (prop)
     (inst?)
     (prop)
     (rewrite "tr_cl_cs_basic"))
    ("2" (skosimp) (prop) (expand "tr_cl" +) (prop) (inst?) (prop))))))


cl_tr.tr_cl_transitive: proved - complete [shostak](0.06 s)

(""
 (skosimp)
 (expand "transitive?")
 (skolem 1 ("x!1" "y!1" "_"))
 (lemma "tr_cl_weak_induction")
 (inst - "lessp!1" "y!1"
  "lambda(z): not(tr_cl(lessp!1)(x!1,y!1)) or tr_cl(lessp!1)(x!1,z)")
 (prop)
 (("1" (skosimp) (inst - "z!1") (prop))
  ("2"
   (hide 2)
   (skosimp)
   (prop)
   (("1" (expand "tr_cl" +) (prop) (inst?) (prop))
    ("2" (skosimp) (prop) (expand "tr_cl" +) (prop) (inst?) (prop))))))


cl_tr.tr_cl_equiv_cn: proved - complete [shostak](0.03 s)

(""
 (lemma "tr_cl_weak_induction")
 (skolem 1 ("lessp!1" "x!1" "_"))
 (inst - "lessp!1" "x!1"
  "lambda(y): lessp!1(x!1, y) OR exists z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y)")
 (prop)
 (hide 2)
 (skosimp)
 (prop)
 (skosimp)
 (prop)
 (("1" (inst?) (prop) (rewrite "tr_cl_cs_basic"))
  ("2"
   (skosimp)
   (inst + "z!2")
   (prop)
   (expand "tr_cl" +)
   (prop)
   (inst?)
   (prop))))


cl_tr.tr_cl_equiv_cs: proved - complete [shostak](0.01 s)

(""
 (skosimp)
 (prop)
 (("1" (rewrite "tr_cl_cs_basic"))
  ("2" (skosimp) (forward-chain "tr_cl_transitive_l0"))))


cl_tr.tr_cl_equiv: proved - complete [shostak](0.03 s)

(""
 (skosimp)
 (prop)
 (("1" (forward-chain "tr_cl_equiv_cn"))
  ("2" (rewrite "tr_cl_equiv_cs"))
  ("3" (rewrite "tr_cl_equiv_cs") (prop))))


cl_tr.tr_cl_less_transitive: proved - complete [shostak](0.04 s)

(""
 (skosimp)
 (skolem 1 ("x!1" "_"))
 (use "tr_cl_weak_induction")
 (prop)
 (hide 2)
 (skosimp)
 (prop)
 (("1" (inst?) (prop))
  ("2"
   (skosimp)
   (inst - "z!1" "y!1")
   (prop)
   (expand "transitive?")
   (inst - "x!1" "z!1" "y!1")
   (prop))))
```

B.3. Buena fundamentación de la clausura transitiva de una relación

```
Proof scripts for file wf_cl_tr.pvs:


wf_cl_tr.well_founded_part_cl_tr_l1: proved - complete [shostak](0.03 s)

(""
 (apply (then (skosimp*) (expand "tr_cl") (prop)))
 (("1" (apply (then (inst?) (prop))))
  ("2"
   (skosimp)
   (inst - "z!2")
   (apply (then (prop) (expand "well_founded_part" -) (inst?)
           (prop))))))


wf_cl_tr.well_founded_part_cl_tr: proved - complete [shostak](0.04 s)

(""
 (skosimp)
 (apply (then (lemma "well_founded_part_weak_induction[T,lessp!1]")
         (inst - "lambda(x): well_founded_part[T, tr_cl(lessp!1)](x)")
         (prop)))
 (("1" (apply (then (inst?) (prop))))
  ("2"
   (apply (then (hide -1 2) (skosimp*)))
   (expand "well_founded_part" +)
   (use "well_founded_part_cl_tr_l1")
   (grind :defs nil))))


wf_cl_tr.well_founded_cl_tr: proved - complete [shostak](0.05 s)

(""
 (skosimp)
 (forward-chain "well_founded_part_cn[T,lessp!1]")
 (hide -2)
 (rewrite "well_founded_part_cs[T,tr_cl(lessp!1)]")
 (hide 2)
 (use "well_founded_part_cl_tr")
 (grind :defs nil))
```

B.4. Relación de orden bien fundamentada en multiconjuntos

```
Proof scripts for file finite_bags_order.pvs:


finite_bags_order.insert_TCC1: proved - complete [shostak](0.03 s)

("" (skosimp) (rewrite "finite_insert"))


finite_bags_order.less_1_insert_cn: proved - complete [shostak](0.65 s)

(""
 (skosimp)
 (expand "less_1" -)
 (skosimp)
 (case-replace "a!2 = a!1")
 (("1"
   (apply (then (hide 1) (inst?) (prop) (replace*) (hide -1 -3 -4)))
   (grind :defs nil :rewrites ("bag_equality" "eqmult" "plus")
    :polarity? t)
   (apply (then (inst?) (expand "insert") (lift-if) (prop)
           (grind :defs nil))))
  ("2"
   (hide 3)
   (inst + "plus(delete(a!1,M!2,1),K!1)")
   (("1"
     (prop)
     (("1"
       (replace*)
       (grind :defs nil :rewrites ("bag_equality" "eqmult" "plus")
        :polarity? t)
       (apply (then (inst?) (inst?) (expand "insert") (lift-if) (prop)
               (grind :defs nil :rewrites ("plus" "delete")))))
```

```
      ("2"
       (expand "less_1")
       (inst + "delete(a!1,M!2,1)" "a!2" "K!1")
       (("1"
         (grind :defs nil :rewrites ("bag_equality" "eqmult" "plus")
          :polarity? t)
         (apply (then (inst?) (inst?) (expand "insert") (lift-if)
                 (prop) (grind :defs nil :rewrites ("delete")))))
        ("2" (rewrite "finite_delete"))))))
    ("2"
     (apply (then (rewrite "finite_bag_plus")
             (rewrite "finite_delete"))))))))
```

```
finite_bags_order.plus_emptybag: proved - complete [shostak](0.02 s)

(""
 (grind :defs nil :rewrites
  ("bag_equality" "eqmult" "plus" "emptybag" "max")))
```

```
finite_bags_order.insert_plus: proved - complete [shostak](0.23 s)

(""
 (grind :defs nil :rewrites ("bag_equality" "eqmult" "plus" "insert"))
 (expand "insert")
 (lift-if)
 (expand "plus")
 (propax))
```

```
finite_bags_order.less_insert_cn: proved - complete [shostak](0.12 s)

(""
 (skosimp)
 (expand "less")
 (inst?)
 (expand "member")
 (expand "insert")
 (assert))
```

```
finite_bags_order.less_insert_cn2: proved - complete [shostak](0.17 s)

(""
 (grind :defs nil :rewrites ("less" "member") :polarity? t)
 (apply (then (expand "insert") (lift-if) (prop) (assert))))
```

```
finite_bags_order.IMP_wf_cs_TCC1: proved - complete [shostak](0.00 s)

("" (inst + "emptybag[T]"))
```

```
finite_bags_order.well_founded_part_less_1_plus_TCC1: proved - complete [shostak](0.03 s)

("" (skolem-typepred) (skosimp) (rewrite "finite_bag_plus"))


finite_bags_order.well_founded_part_less_1_plus: proved - complete [shostak](0.09 s)

(""
 (skolem 1 ("_" "M1" "a"))
 (induct "K" :name "finite_bag_induction")
 (("1" (skosimp*) (rewrite "plus_emptybag"))
  ("2"
   (skosimp*)
   (prop)
   (("1"
     (rewrite "insert_plus")
     (inst -4 "x!1")
     (forward-chain "less_insert_cn")
     (prop)
     (inst -1 "plus(M1, b!1)")
     (prop))
    ("2" (forward-chain "less_insert_cn2"))))
  ("3" (hide 2) (skosimp*) (rewrite "finite_bag_plus"))
  ("4" (hide 2) (skosimp*) (rewrite "finite_insert"))))


finite_bags_order.well_founded_part_less_1_insert_l0: proved - complete [shostak](0.11 s)

(""
 (apply (then (skosimp) (expand "well_founded_part" +) (skosimp)))
 (forward-chain "less_1_insert_cn")
 (("1"
   (skosimp)
   (inst -4 "M!2")
   (prop)
   (("1" (replace*))
    ("2"
     (apply (then (expand "well_founded_part" -3) (inst -3 "M!2")
             (prop))))))
  ("2"
   (grind-with-lemmas :defs nil :lemmas
    ("well_founded_part_less_1_plus")))))


finite_bags_order.well_founded_part_less_1_insert_l2: proved - complete [shostak](0.08 s)

(""
 (apply (then (skosimp) (lemma "well_founded_part_weak_induction")
         (inst -1
          "lambda(M): well_founded_part(insert(a!1, M)) OR not(well_founded_part(M))")))
 (prop)
 (("1" (grind :defs nil))
  ("2"
   (hide 2)
   (skosimp)
   (rewrite "well_founded_part_less_1_insert_l0")
   (hide -3 2)
   (skosimp)
   (inst?)
   (prop))))
```

```
finite_bags_order.well_founded_part_less_1_insert: proved - complete [shostak](0.03 s)

(""
 (induct "a" :name "wf_induction[T,<]")
 (skosimp)
 (use "well_founded_part_less_1_insert_l2")
 (prop))
```

```
finite_bags_order.well_founded_part_less_1_all: proved - complete [shostak](0.13 s)

(""
 (induct "M" :name "finite_bag_induction")
 (("1"
   (expand "well_founded_part")
   (skosimp)
   (expand "less_1")
   (skosimp)
   (hide -2 -3 1)
   (decompose-equality)
   (grind :defs nil :rewrites ("emptybag" "insert")))
  ("2" (skosimp) (rewrite "well_founded_part_less_1_insert"))))
```

```
finite_bags_order.wf_less_1: proved - complete [shostak](0.01 s)

(""
 (use "well_founded_part_less_1_all")
 (rewrite "well_founded_part_cns"))
```

```
finite_bags_order.wf_less_bag: proved - complete [shostak](0.01 s)

(""
 (grind :defs nil :rewrites
  ("less_bag" "well_founded_cl_tr" "wf_less_1")))
```

```
finite_bags_order.less_mult_transitive_cs: proved - complete [shostak](1.58 s)
```

```
(""
 (prop)
 (expand "transitive?")
 (skosimp)
 (expand "less_mult")
 (skolem -2 ("M0" "K1" "K2"))
 (skolem -3 ("M1" "L1" "L2"))
 (name "M0_1" "difference(M1,intersection(K1,M1))")
 (name "K1_1" "plus(intersection(K1,M1),L1)")
 (name "K2_1"
       "plus(difference(M0,difference(M1,intersection(K1,M1))),K2)")
 (inst + "M0_1" "K1_1" "K2_1")
 (("1"
   (prop)
   (("1"
     (replace -2 :dir rl)
     (grind :defs nil :rewrites ("nonempty_bag?" "empty?" "plus")))
    ("2"
     (hide -4 -5 -6 -7 -8 -9 -11 -12)
     (replace*)
     (replace -2 :dir rl)
     (replace -3 :dir rl)
     (hide -1 -2 -3 -4)
     (rewrite "bag_equality")
     (skosimp)
     (grind :defs nil :rewrites
      ("eqmult" "plus" "intersection" "min" "difference")))
    ("3"
     (hide -4 -5 -6 -8 -9 -10 -11 -12)
     (replace*)
     (replace -3 :dir rl)
     (replace -1 :dir rl)
     (hide -1 -2 -3 -4)
     (rewrite "bag_equality")
     (skosimp)
     (expand "eqmult")
     (expand "plus")
     (use "subbag_intersection2")
     (case "subbag?(difference(M1, intersection(K1, M1)),M0)")
     (("1" (grind :defs nil :rewrites ("subbag?" "difference")))
      ("2"
       (hide 2)
       (reveal -8 -12)
       (replace*)
       (hide -1)
       (expand "subbag?")
       (skosimp)
       (rewrite "bag_equality")
       (inst?)
       (expand "eqmult")
       (expand "plus")
       (inst?)
      (expand "difference")
      (lift-if)
      (prop)
      (assert)
      (use "subbag_intersection1")
      (expand "subbag?")
      (inst?)
      (assert)
      (expand "intersection")
      (expand "min")
      (lift-if)
      (prop)
      (("1" (assert)) ("2" (assert))))))
   ("4"
    (hide -5 -9)
    (replace -1 :dir rl)
    (replace -2 :dir rl)
    (hide -1 -2 -3)
    (skosimp)
    (expand "member" -1)
    (expand "plus" -1)
    (case "K2(a!1) > 0")
    (("1"
      (inst? -6)
      (expand "member")
      (prop)
      (skosimp)
      (case "M1(b!1) > 0")
      (("1"
        (inst?)
        (prop)
        (expand "plus")
        (expand "intersection")
        (assert)
        (expand "min")
        (lift-if)
        (prop)
        (("1" (assert)) ("2" (assert))))
       ("2"
        (replace*)
        (hide -6 -7 -8)
        (rewrite "bag_equality")
        (inst -6 "b!1")
        (expand "eqmult")
        (expand "plus" -6)
        (inst -7 "b!1")
        (prop)
        (("1"
          (skosimp)
          (inst 2 "b!2")
          (prop)
          (("1"
             (expand "plus")
             (expand "intersection")
             (expand "min")
             (lift-if)
             (prop)
             (("1" (assert)) ("2" (assert))))
            ("2" (inst -7 "a!1" "b!1" "b!2") (prop))))
          ("2" (assert))))))
      ("2"
       (inst -8 "a!1")
       (prop)
       (("1"
         (skosimp)
         (inst? +)
         (prop)
         (expand "member")
         (expand "plus")
         (assert))
        ("2"
         (hide -2 -5 -6 2 3)
         (replace*)
         (hide -2 -3)
         (expand "member")
         (rewrite "bag_equality")
         (inst?)
         (expand "eqmult")
         (expand "plus")
         (reveal 1)
         (grind :defs nil :rewrites
          ("difference" "intersection" "min")))))))))
  ("2"
   (replace -1 :dir rl)
   (grind :defs nil :rewrites
    ("finite_bag_plus"
     "finite_bag_difference"
     "finite_bag_intersection")))
  ("3"
   (replace -2 :dir rl)
   (grind :defs nil :rewrites
    ("finite_bag_plus" "finite_bag_intersection")))
  ("4"
   (replace -3 :dir rl)
   (grind :defs nil :rewrites
    ("finite_bag_difference" "finite_bag_intersection")))))


finite_bags_order.less_1_subset_less_mult: proved - complete [shostak](0.08 s)

(""
 (skosimp)
 (expand "less_1")
 (expand "less_mult")
 (skosimp)
 (inst + "M!2" "singleton_bag(a!1)" "K!1")
 (("1"
   (prop)
   (("1"
     (hide -1 -2 -3)
     (grind :defs nil :rewrites
      ("nonempty_bag?" "empty?" "singleton_bag")))
    ("2"
     (hide -2 -3)
     (replace*)
     (hide -1)
     (rewrite "bag_equality")
     (skosimp)
     (expand "eqmult")
     (expand "insert")
     (expand "singleton_bag")
     (expand "plus")
     (lift-if)
     (lift-if)
     (assert))
    ("3"
     (hide -1 -2)
     (skosimp)
     (expand "less")
     (inst? -)
     (prop)
     (inst?)
     (prop)
     (expand "member")
     (expand "singleton_bag")
     (assert))))
  ("2" (rewrite "finite_singleton_bag"))))


finite_bags_order.less_bag_subset_less_mult: proved - complete [shostak](0.03 s)

(""
 (skosimp)
 (expand "less_bag")
 (lemma "tr_cl_less_transitive[finite_bag[T]]")
 (inst - "less_1" "less_mult")
 (prop)
 (("1" (inst?) (prop)) ("2" (rewrite "less_mult_transitive_cs"))
  ("3" (skosimp) (rewrite "less_1_subset_less_mult"))))


finite_bags_order.less_mult_subset_less_bag_lem: proved - complete [shostak](0.89 s)

(""
 (induct "K1" :name "finite_bag_induction")
 (("1"
  (skosimp)
  (expand "nonempty_bag?")
  (expand "empty?")
  (skosimp)
  (expand "emptybag")
  (propax))
 ("2"
  (skosimp*)
  (expand "less_bag" +)
  (expand "tr_cl")
  (prop)
  (case "empty?(b!1)")
  (("1"
    (hide -2 2)
    (expand "less_1")
    (inst + "M!2" "x!1" "K2!1")
    (prop)
    (("1"
      (hide -1 -2 -4 -5)
      (replace*)
      (hide -1)
      (reveal -2)
      (rewrite "bag_equality")
      (skosimp)
      (expand "eqmult")
      (expand "plus")
      (expand "empty?")
      (expand "insert")
      (lift-if)
      (prop)
      (("1" (inst?) (assert)) ("2" (inst - "x!2") (assert))))
     ("2"
      (hide -2 -3 -4)
      (expand "less")
      (skosimp)
      (inst?)
      (prop)
      (skosimp)
      (expand "member")
      (expand "insert")
      (expand "empty?")
      (inst?)
      (lift-if)
      (prop)
      (("1" (assert)) ("2" (assert))))))
   ("2"
    (hide -2 2)
    (name "R2"
          "lambda(x): IF (EXISTS y: b!1(y) > 0 AND  x < y) THEN 0 ELSE K2!1(x) ENDIF")
    (inst + "plus(plus(M!2,R2),b!1)")
    (("1"
      (prop)
```

```
(("1"
  (inst -2 "difference(K2!1,R2)" "plus(plus(M!2, R2), b!1)"
   "plus(M!2,R2)" "N!1")
  (("1"
    (prop)
    (("1" (expand "less_bag") (propax))
     ("2" (expand "nonempty_bag?") (propax))
     ("3"
      (replace*)
      (hide -2 -3 -4 2 3)
      (rewrite "difference_descom_plus")
      (hide 2)
      (replace -1 1 :dir rl)
      (hide -1)
      (expand "subbag?")
      (skosimp)
      (lift-if)
      (prop)
      (("1" (skosimp) (assert)) ("2" (assert))))
     ("4"
      (hide -2 -3 2)
      (replace -1 1 :dir rl)
      (hide -1)
      (skosimp)
      (expand "member")
      (expand "difference")
      (lift-if)
      (prop)
      (lift-if)
      (prop)
      (("1"
        (inst? -)
        (prop)
        (("1"
          (skosimp)
          (expand "insert")
          (lift-if)
          (prop)
          (("1" (inst?) (assert)) ("2" (inst?) (assert))))
         ("2" (assert))))
       ("2" (assert))))))
   ("2"
    (rewrite "finite_bag_plus")
    (case "subbag?(R2,K2!1)")
    (("1"
      (hide -2 -3 -4 -5 3 2 4)
      (expand "is_finite")
      (forward-chain "subbag_lem")
      (hide -2)
      (use "finite_subset")
      (("1" (prop))
       ("2" (typepred "K2!1") (expand "is_finite" -) (propax))))
```

```
          ("2"
           (hide -2 -3 -4 -5 2 3 4 5)
           (replace -1 :dir rl)
           (hide -1)
           (expand "subbag?")
           (skosimp)
           (lift-if)
           (prop)
           (("1" (skosimp) (assert)) ("2" (assert))))))
        ("3"
         (use "subbag_difference")
         (hide -2 -3 -4 -5 2 3)
         (forward-chain "subbag_lem")
         (hide -2)
         (expand "is_finite")
         (use "finite_subset")
         (("1" (prop))
          ("2" (typepred "K2!1") (expand "is_finite" -) (propax))))))
      ("2"
       (hide -2)
       (expand "less_1")
       (inst + "plus(M!2,b!1)" "x!1" "R2")
       (("1"
         (prop)
         (("1"
           (lemma "plus_commutative")
           (inst -1 "M!2" "insert(x!1, b!1)")
           (replace -1)
           (replace*)
           (rewrite "bag_plus_insert")
           (hide -1 -2 -3 -4 -5 2)
           (rewrite "plus_commutative"))
          ("2"
           (hide -1 -2 -3 -4 2)
           (rewrite "bag_equality")
           (skosimp)
           (expand "eqmult")
           (expand "plus")
           (propax))
          ("3"
           (replace -1 1 :dir rl)
           (hide -1 -2 -3 2)
           (expand "less")
           (skosimp)
           (expand "member")
           (lift-if)
           (prop)
           (("1" (assert))
            ("2"
             (inst? -)
             (prop)
             (skosimp)
```

```
               (reveal 1)
               (expand "insert")
               (lift-if)
               (prop)
               (("1" (assert)) ("2" (assert) (inst?) (prop))))))))
          ("2"
           (hide -2 -3 -4 -5 2 3 4 5)
           (case "subbag?(R2,K2!1)")
           (("1"
             (forward-chain "subbag_lem")
             (hide -2)
             (expand "is_finite")
             (use "finite_subset")
             (("1" (prop))
              ("2" (typepred "K2!1") (expand "is_finite" -) (propax))))
            ("2"
             (replace -1 :dir rl)
             (hide -1)
             (hide 2)
             (expand "subbag?")
             (skosimp)
             (lift-if)
             (prop)
             (("1" (skosimp) (assert)) ("2" (assert))))))
          ("3" (rewrite "finite_bag_plus"))))))
      ("2"
       (rewrite "finite_bag_plus")
       (rewrite "finite_bag_plus")
       (case "subbag?(R2,K2!1)")
       (("1"
         (forward-chain "subbag_lem")
         (hide -2)
         (expand "is_finite")
         (use "finite_subset")
         (("1" (prop))
          ("2" (typepred "K2!1") (expand "is_finite" -) (propax))))
        ("2"
         (hide -2 -3 -4 -5 2 3 4 5)
         (replace -1 :dir rl)
         (hide -1)
         (expand "subbag?")
         (skosimp)
         (lift-if)
         (prop)
         (("1" (skosimp) (assert)) ("2" (assert)))))))))))


finite_bags_order.less_mult_subset_less_bag: proved - complete [shostak](0.03 s)

(""
 (skosimp)
 (expand "less_mult")
 (skosimp)
 (forward-chain "less_mult_subset_less_bag_lem"))


finite_bags_order.less_mult_equiv_less_bag: proved - complete [shostak](0.02 s)

(""
 (skosimp)
 (prop)
 (("1" (rewrite "less_mult_subset_less_bag"))
  ("2" (rewrite "less_bag_subset_less_mult"))))
```

Apéndice C

Demostraciones de las teorías PVS de la formalización

```
Loading pvs-load...
Loading pvs-prelude-files-and-regions (source)...
Started initializing ILISP
Finished initializing pvsallegro
Context changed to ~/alonso/estudio/alc/auxiliares/texto/
Context changed to ~/alonso/estudio/alc/auxiliares/wf/
Parsing wf_cs
wf_cs parsed in 0.22 seconds
Typechecking wf_cs
wf_cs typechecked in 0.70s: No TCCs generated
Rerunning proof of well_founded_part_cn


well_founded_part_cn :

  |-------
{1}   well_founded?[T](<) IMPLIES (FORALL x: well_founded_part(x))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
well_founded_part_cn :

{-1}  well_founded?[T](<)
  |-------
{1}   FORALL x: well_founded_part(x)

Rerunning step: (use "wf_induction[T,<]")
Using lemma wf_induction[T,<],
this simplifies to:
well_founded_part_cn :

{-1}  (FORALL (x: T):
         (FORALL (y: T): y < x IMPLIES well_founded_part(y)) IMPLIES
          well_founded_part(x))
```

```
       IMPLIES (FORALL (x: T): well_founded_part(x))
[-2]  well_founded?[T](<)
  |-------
[1]   FORALL x: well_founded_part(x)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
well_founded_part_cn :

[-1]  well_founded?[T](<)
  |-------
{1}   FORALL (x: T):
        (FORALL (y: T): y < x IMPLIES well_founded_part(y)) IMPLIES
         well_founded_part(x)
[2]   FORALL x: well_founded_part(x)

Rerunning step: (hide -1 2)
Hiding formulas:  -1, 2,
this simplifies to:
well_founded_part_cn :

  |-------
[1]   FORALL (x: T):
        (FORALL (y: T): y < x IMPLIES well_founded_part(y)) IMPLIES
         well_founded_part(x)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_cn :

{-1}  FORALL (y: T): y < x!1 IMPLIES well_founded_part(y)
  |-------
{1}   well_founded_part(x!1)

Rerunning step: (expand "well_founded_part" +)
Expanding the definition of well_founded_part,
this simplifies to:
well_founded_part_cn :

[-1]  FORALL (y: T): y < x!1 IMPLIES well_founded_part(y)
  |-------
{1}   FORALL y: y < x!1 IMPLIES well_founded_part(y)

which is trivially true.
Q.E.D.
well_founded_part_cn proved in 0.13 real, 0.13 cpu seconds
Rerunning proof of well_founded_part_cs


well_founded_part_cs :
  |-------
{1}   (FORALL x: well_founded_part(x)) IMPLIES well_founded?[T](<)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
well_founded_part_cs :

{-1}  (FORALL x: well_founded_part(x))
  |-------
{1}   well_founded?[T](<)

Rerunning step: (expand "well_founded?")
Expanding the definition of well_founded?,
this simplifies to:
well_founded_part_cs :

[-1]  (FORALL x: well_founded_part(x))
  |-------
{1}   FORALL (p: pred[T]):
        (EXISTS (y: T): p(y)) IMPLIES
         (EXISTS (y: (p)): FORALL (x: (p)): (NOT x < y))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_cs :

[-1]  (FORALL x: well_founded_part(x))
{-2}  EXISTS (y: T): p!1(y)
  |-------
{1}   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (lemma "well_founded_part_weak_induction")
Applying well_founded_part_weak_induction
this simplifies to:
well_founded_part_cs :

{-1}  FORALL (P: [T -> boolean]):
        (FORALL (x): (FORALL y: y < x IMPLIES P(y)) IMPLIES P(x)) IMPLIES
         (FORALL (x): well_founded_part(x) IMPLIES P(x))
[-2]  (FORALL x: well_founded_part(x))
[-3]  EXISTS (y: T): p!1(y)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (inst - "lambda(x): NOT p!1(x)")
Instantiating the top quantifier in - with the terms:
 lambda(x): NOT p!1(x),
this simplifies to:
well_founded_part_cs :
```

```
{-1}  (FORALL (x_1: T):
         (FORALL y: y < x_1 IMPLIES NOT p!1(y)) IMPLIES NOT p!1(x_1))
       IMPLIES
       (FORALL (x_1: T): well_founded_part(x_1) IMPLIES NOT p!1(x_1))
[-2]  (FORALL x: well_founded_part(x))
[-3]  EXISTS (y: T): p!1(y)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
well_founded_part_cs.1 :

{-1}  FORALL (x_1: T): well_founded_part(x_1) IMPLIES NOT p!1(x_1)
[-2]  (FORALL x: well_founded_part(x))
[-3]  EXISTS (y: T): p!1(y)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (grind :defs nil)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of well_founded_part_cs.1.

well_founded_part_cs.2 :

[-1]  (FORALL x: well_founded_part(x))
[-2]  EXISTS (y: T): p!1(y)
  |-------
{1}   FORALL (x_1: T):
        (FORALL y: y < x_1 IMPLIES NOT p!1(y)) IMPLIES NOT p!1(x_1)
[2]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_cs.2 :

{-1}  FORALL y: y < x!1 IMPLIES NOT p!1(y)
{-2}  p!1(x!1)
[-3]  (FORALL x: well_founded_part(x))
[-4]  EXISTS (y: T): p!1(y)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (inst? +)
Found substitution:
y: (p!1) gets x!1,
Using template: y
Instantiating quantified variables,
this simplifies to:
well_founded_part_cs.2 :

[-1]  FORALL y: y < x!1 IMPLIES NOT p!1(y)
[-2]  p!1(x!1)
[-3]  (FORALL x: well_founded_part(x))
[-4]  EXISTS (y: T): p!1(y)
  |-------
{1}   FORALL (x: (p!1)): (NOT x < x!1)

Rerunning step: (skolem-typepred)
Skolemizing (with typepred on new Skolem constants),
this simplifies to:
well_founded_part_cs.2 :

[-1]  FORALL y: y < x!1 IMPLIES NOT p!1(y)
[-2]  p!1(x!1)
[-3]  (FORALL x: well_founded_part(x))
{-4}  p!1(y!1)
  |-------
[1]   FORALL (x: (p!1)): (NOT x < x!1)

Rerunning step: (skolem-typepred)
Skolemizing (with typepred on new Skolem constants),
this simplifies to:
well_founded_part_cs.2 :

{-1}  p!1(x!2)
[-2]  FORALL y: y < x!1 IMPLIES NOT p!1(y)
[-3]  p!1(x!1)
[-4]  (FORALL x: well_founded_part(x))
[-5]  p!1(y!1)
{-6}  x!2 < x!1
  |-------

Rerunning step: (inst? -2)
Found substitution:
y gets x!2,
Using template: NOT p!1(y)
Instantiating quantified variables,
this simplifies to:
well_founded_part_cs.2 :

[-1]  p!1(x!2)
{-2}  x!2 < x!1 IMPLIES NOT p!1(x!2)
[-3]  p!1(x!1)
[-4]  (FORALL x: well_founded_part(x))
[-5]  p!1(y!1)
[-6]  x!2 < x!1
  |-------

Rerunning step: (prop)
```

```
Applying propositional simplification,

This completes the proof of well_founded_part_cs.2.

Q.E.D.
well_founded_part_cs proved in 0.09 real, 0.09 cpu seconds
Rerunning proof of well_founded_part_cns


well_founded_part_cns :

  |-------
{1}   well_founded?[T](<) IFF (FORALL x: well_founded_part(x))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
well_founded_part_cns.1 :

{-1}  well_founded?[T](<)
  |-------
{1}   (FORALL x: well_founded_part(x))

Rerunning step: (rewrite "well_founded_part_cn")
Found matching substitution:
Rewriting using well_founded_part_cn, matching in *,

This completes the proof of well_founded_part_cns.1.

well_founded_part_cns.2 :

{-1}  (FORALL x: well_founded_part(x))
  |-------
{1}   well_founded?[T](<)

Rerunning step: (rewrite "well_founded_part_cs")
Found matching substitution:
Rewriting using well_founded_part_cs, matching in *,

This completes the proof of well_founded_part_cns.2.

Q.E.D.
well_founded_part_cns proved in 0.02 real, 0.02 cpu seconds
Rerunning proof of well_founded_induct_cns


well_founded_induct_cns :

  |-------
{1}   well_founded?[T](<) IFF
       (FORALL (p: pred[T]):
          (FORALL (x: T): (FORALL (y: T): y < x IMPLIES p(y)) IMPLIES p(x))
```

```
           IMPLIES (FORALL (x: T): p(x)))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
well_founded_induct_cns.1 :

{-1}  well_founded?[T](<)
  |-------
{1}   (FORALL (p: pred[T]):
         (FORALL (x: T): (FORALL (y: T): y < x IMPLIES p(y)) IMPLIES p(x))
          IMPLIES (FORALL (x: T): p(x)))

Rerunning step: (use "wf_induction[T,<]")
Using lemma wf_induction[T,<],

This completes the proof of well_founded_induct_cns.1.

well_founded_induct_cns.2 :

{-1}  (FORALL (p: pred[T]):
         (FORALL (x: T): (FORALL (y: T): y < x IMPLIES p(y)) IMPLIES p(x))
          IMPLIES (FORALL (x: T): p(x)))
  |-------
{1}   well_founded?[T](<)

Rerunning step: (apply (then (expand "well_founded?") (skosimp)
                       (inst - "lambda(x): not p!1(x)") (skosimp)
                       (prop)))
Applying
   (then (expand "well_founded?") (skosimp)
    (inst - "lambda(x): not p!1(x)") (skosimp) (prop)),
this yields  2 subgoals:
well_founded_induct_cns.2.1 :

{-1}  FORALL (x_1: T): NOT p!1(x_1)
{-2}  p!1(y!1)
  |-------
{1}   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (grind :defs nil)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of well_founded_induct_cns.2.1.

well_founded_induct_cns.2.2 :

{-1}  p!1(y!1)
  |-------
{1}   FORALL (x_1: T):
        (FORALL (y: T): y < x_1 IMPLIES NOT p!1(y)) IMPLIES NOT p!1(x_1)
{2}   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)
```

```
Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_induct_cns.2.2 :

{-1}  FORALL (y: T): y < x!1 IMPLIES NOT p!1(y)
{-2}  p!1(x!1)
[-3]  p!1(y!1)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (inst? +)
Found substitution:
y: (p!1) gets x!1,
Using template: y
Instantiating quantified variables,
this simplifies to:
well_founded_induct_cns.2.2 :

[-1]  FORALL (y: T): y < x!1 IMPLIES NOT p!1(y)
[-2]  p!1(x!1)
[-3]  p!1(y!1)
  |-------
{1}   FORALL (x: (p!1)): (NOT x < x!1)

Rerunning step: (apply (then (skolem-typepred) (inst?) (prop)))
Applying
   (then (skolem-typepred) (inst?) (prop)),

This completes the proof of well_founded_induct_cns.2.2.


This completes the proof of well_founded_induct_cns.2.

Q.E.D.
well_founded_induct_cns proved in 0.05 real, 0.05 cpu seconds
Rerunning proof of well_founded_cs_ordinal



well_founded_cs_ordinal :

  |-------
{1}   (EXISTS (f: [T -> ordinal]): FORALL x, y: x < y IMPLIES f(x) < f(y))
       IMPLIES well_founded?[T](<)

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
well_founded_cs_ordinal :

{-1}  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
```

```
  |-------
{1}   well_founded?[T](<)

Rerunning step: (use "well_founded_le")
Using lemma well_founded_le,
this simplifies to:
well_founded_cs_ordinal :

{-1}  well_founded?(LAMBDA (r, s: (ordinal?)): r < s)
[-2]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
  |-------
[1]   well_founded?[T](<)

Rerunning step: (expand "well_founded?")
Expanding the definition of well_founded?,
this simplifies to:
well_founded_cs_ordinal :

{-1}  FORALL (p: pred[(ordinal?)]):
        (EXISTS (y: (ordinal?)): p(y)) IMPLIES
         (EXISTS (y: (p)): FORALL (x: (p)): (NOT x < y))
[-2]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
  |-------
{1}   FORALL (p: pred[T]):
        (EXISTS (y: T): p(y)) IMPLIES
         (EXISTS (y: (p)): FORALL (x: (p)): (NOT x < y))

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
well_founded_cs_ordinal :

[-1]  FORALL (p: pred[(ordinal?)]):
        (EXISTS (y: (ordinal?)): p(y)) IMPLIES
         (EXISTS (y: (p)): FORALL (x: (p)): (NOT x < y))
[-2]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
{-3}  p!1(y!1)
  |-------
{1}   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (apply (then
                        (inst -1
                         "lambda(u:ordinal): Exists (x:T): p!1(x) AND f!1(x) = u")
                        (prop)))
Applying
   (then
    (inst -1 "lambda(u:ordinal): Exists (x:T): p!1(x) AND f!1(x) = u")
    (prop)),
this yields  2 subgoals:
well_founded_cs_ordinal.1 :

{-1}  EXISTS (y:
```

```
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        FORALL (x_1:
                  (LAMBDA (u: ordinal):
                     EXISTS (x: T): p!1(x) AND f!1(x) = u)):
          (NOT x_1 < y)
[-2]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-3]  p!1(y!1)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (skosimp - t)
Skolemizing and flattening,
this simplifies to:
well_founded_cs_ordinal.1 :

{-1}  ordinal?(y!2)
{-2}  EXISTS (x: T): p!1(x) AND f!1(x) = y!2
{-3}  FORALL (x_1:
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        (NOT x_1 < y!2)
[-4]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-5]  p!1(y!1)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_cs_ordinal.1 :

[-1]  ordinal?(y!2)
{-2}  p!1(x!1)
{-3}  f!1(x!1) = y!2
[-4]  FORALL (x_1:
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        (NOT x_1 < y!2)
[-5]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-6]  p!1(y!1)
  |-------
[1]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (inst + "x!1")
Instantiating the top quantifier in + with the terms:
 x!1,
this simplifies to:
well_founded_cs_ordinal.1 :

[-1]  ordinal?(y!2)
[-2]  p!1(x!1)
```

```
[-3]  f!1(x!1) = y!2
[-4]  FORALL (x_1:
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        (NOT x_1 < y!2)
[-5]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-6]  p!1(y!1)
  |-------
{1}   FORALL (x: (p!1)): (NOT x < x!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_cs_ordinal.1 :

[-1]  ordinal?(y!2)
[-2]  p!1(x!1)
[-3]  f!1(x!1) = y!2
[-4]  FORALL (x_1:
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        (NOT x_1 < y!2)
[-5]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-6]  p!1(y!1)
{-7}  x!2 < x!1
  |-------

Rerunning step: (inst -5 "x!2" "x!1")
Instantiating the top quantifier in -5 with the terms:
 x!2, x!1,
this simplifies to:
well_founded_cs_ordinal.1 :

[-1]  ordinal?(y!2)
[-2]  p!1(x!1)
[-3]  f!1(x!1) = y!2
[-4]  FORALL (x_1:
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        (NOT x_1 < y!2)
{-5}  x!2 < x!1 IMPLIES f!1(x!2) < f!1(x!1)
[-6]  p!1(y!1)
[-7]  x!2 < x!1
  |-------

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
well_founded_cs_ordinal.1 :

{-1}  f!1(x!2) < f!1(x!1)
[-2]  ordinal?(y!2)
```

```
[-3]  p!1(x!1)
[-4]  f!1(x!1) = y!2
[-5]  FORALL (x_1:
                (LAMBDA (u: ordinal):
                   EXISTS (x: T): p!1(x) AND f!1(x) = u)):
        (NOT x_1 < y!2)
[-6]  p!1(y!1)
[-7]  x!2 < x!1
  |-------

Rerunning step: (inst - "f!1(x!2)")
Instantiating the top quantifier in - with the terms:
 f!1(x!2),
this yields  2 subgoals:
well_founded_cs_ordinal.1.1 :

[-1]  f!1(x!2) < f!1(x!1)
[-2]  ordinal?(y!2)
[-3]  p!1(x!1)
[-4]  f!1(x!1) = y!2
[-5]  p!1(y!1)
[-6]  x!2 < x!1
  |-------
{1}   f!1(x!2) < y!2

Rerunning step: (replace*)
Repeatedly applying the replace rule,

This completes the proof of well_founded_cs_ordinal.1.1.

well_founded_cs_ordinal.1.2 (TCC):

[-1]  f!1(x!2) < f!1(x!1)
[-2]  ordinal?(y!2)
[-3]  p!1(x!1)
[-4]  f!1(x!1) = y!2
[-5]  p!1(y!1)
[-6]  x!2 < x!1
  |-------
{1}   EXISTS (x: T): p!1(x) AND f!1(x) = f!1(x!2)

Rerunning step: (typepred "x!2")
Adding type constraints for  x!2,
this simplifies to:
well_founded_cs_ordinal.1.2 :

{-1}  p!1(x!2)
[-2]  f!1(x!2) < f!1(x!1)
[-3]  ordinal?(y!2)
[-4]  p!1(x!1)
[-5]  f!1(x!1) = y!2
[-6]  p!1(y!1)
```

```
[-7]  x!2 < x!1
  |-------
[1]   EXISTS (x: T): p!1(x) AND f!1(x) = f!1(x!2)

Rerunning step: (inst?)
Found substitution:
x: T gets x!2,
Using template: p!1(x)
Instantiating quantified variables,

This completes the proof of well_founded_cs_ordinal.1.2.


This completes the proof of well_founded_cs_ordinal.1.

well_founded_cs_ordinal.2 :

[-1]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-2]  p!1(y!1)
  |-------
{1}   EXISTS (y: (ordinal?)): EXISTS (x: T): p!1(x) AND f!1(x) = y
[2]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (inst + "f!1(y!1)")
Instantiating the top quantifier in + with the terms:
 f!1(y!1),
this simplifies to:
well_founded_cs_ordinal.2 :

[-1]  FORALL x, y: x < y IMPLIES f!1(x) < f!1(y)
[-2]  p!1(y!1)
  |-------
{1}   EXISTS (x: T): p!1(x) AND f!1(x) = f!1(y!1)
[2]   EXISTS (y: (p!1)): FORALL (x: (p!1)): (NOT x < y)

Rerunning step: (inst?)
Found substitution:
x: T gets y!1,
Using template: p!1(x)
Instantiating quantified variables,

This completes the proof of well_founded_cs_ordinal.2.

Q.E.D.
well_founded_cs_ordinal proved in 0.09 real, 0.09 cpu seconds
wf_cs: 5 proofs attempted, 5 proved in 0.38 real, 0.38 cpu seconds


 Proof summary for theory wf_cs
    well_founded_part_cn.................proved - complete   [shostak](0.13 s)
    well_founded_part_cs.................proved - complete   [shostak](0.09 s)
    well_founded_part_cns................proved - complete   [shostak](0.02 s)
    well_founded_induct_cns..............proved - complete   [shostak](0.05 s)
    well_founded_cs_ordinal..............proved - complete   [shostak](0.09 s)
    Theory totals: 5 formulas, 5 attempted, 5 succeeded (0.38 s)

Grand Totals: 5 proofs, 5 attempted, 5 succeeded (0.38 s)
Parsing cl_tr
cl_tr parsed in 0.06 seconds
Typechecking cl_tr
cl_tr typechecked in 0.03s: No TCCs generated
Rerunning proof of tr_cl_cs_basic


tr_cl_cs_basic :

  |-------
{1}   FORALL (<: pred[[T, T]], x, y: T): x < y IMPLIES tr_cl(<)(x, y)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_cs_basic :

{-1}  lessp!1(x!1, y!1)
  |-------
{1}   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (expand "tr_cl")
Expanding the definition of tr_cl,
this simplifies to:
tr_cl_cs_basic :

[-1]  lessp!1(x!1, y!1)
  |-------
{1}   lessp!1(x!1, y!1) OR
       (EXISTS z: tr_cl(lessp!1)(x!1, z) AND lessp!1(z, y!1))

Rerunning step: (prop)
Applying propositional simplification,
Q.E.D.
tr_cl_cs_basic proved in 0.00 real, 0.00 cpu seconds
Rerunning proof of tr_cl_transitive_l0


tr_cl_transitive_l0 :

  |-------
{1}   FORALL (<: pred[[T, T]], x, y, z: T):
        (z < x AND tr_cl(<)(x, y)) IMPLIES tr_cl(<)(z, y)

Rerunning step: (lemma "tr_cl_weak_induction")
Applying tr_cl_weak_induction
this simplifies to:
```

```
tr_cl_transitive_l0 :

{-1}  FORALL (<, x, P: [T -> boolean]):
        (FORALL (y): (x < y OR (EXISTS z: P(z) AND z < y)) IMPLIES P(y))
         IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
[1]   FORALL (<: pred[[T, T]], x, y, z: T):
        (z < x AND tr_cl(<)(x, y)) IMPLIES tr_cl(<)(z, y)

Rerunning step: (skolem 1 ("lessp!1" "x!1" "_" "z!1"))
For the top quantifier in 1, we introduce Skolem constants:
  (lessp!1 x!1 _ z!1),
this simplifies to:
tr_cl_transitive_l0 :

[-1]  FORALL (<, x, P: [T -> boolean]):
        (FORALL (y): (x < y OR (EXISTS z: P(z) AND z < y)) IMPLIES P(y))
         IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
{1}   FORALL (y):
        (lessp!1(z!1, x!1) AND tr_cl(lessp!1)(x!1, y)) IMPLIES
         tr_cl(lessp!1)(z!1, y)

Rerunning step: (inst - "lessp!1" "x!1"
                 "lambda(y): not(lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1,y)")
Instantiating the top quantifier in - with the terms:
 lessp!1, x!1, lambda(y): not(lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1,y),
this simplifies to:
tr_cl_transitive_l0 :

{-1}  (FORALL (y_1: T):
         (lessp!1(x!1, y_1) OR
           (EXISTS z:
              (NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, z)) AND
               lessp!1(z, y_1)))
          IMPLIES NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y_1))
       IMPLIES
       (FORALL (y_1: T):
          tr_cl(lessp!1)(x!1, y_1) IMPLIES
           NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y_1))
  |-------
[1]   FORALL (y):
        (lessp!1(z!1, x!1) AND tr_cl(lessp!1)(x!1, y)) IMPLIES
         tr_cl(lessp!1)(z!1, y)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
tr_cl_transitive_l0.1 :

{-1}  FORALL (y_1: T):
        tr_cl(lessp!1)(x!1, y_1) IMPLIES
```

```
         NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y_1)
  |-------
[1]   FORALL (y):
        (lessp!1(z!1, x!1) AND tr_cl(lessp!1)(x!1, y)) IMPLIES
         tr_cl(lessp!1)(z!1, y)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_transitive_l0.1 :

[-1]  FORALL (y_1: T):
        tr_cl(lessp!1)(x!1, y_1) IMPLIES
         NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y_1)
{-2}  lessp!1(z!1, x!1)
{-3}  tr_cl(lessp!1)(x!1, y!1)
  |-------
{1}   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (inst?)
Found substitution:
y_1: T gets y!1,
Using template: tr_cl(lessp!1)(z!1, y_1)
Instantiating quantified variables,
this simplifies to:
tr_cl_transitive_l0.1 :

{-1}  tr_cl(lessp!1)(x!1, y!1) IMPLIES
       NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y!1)
[-2]  lessp!1(z!1, x!1)
[-3]  tr_cl(lessp!1)(x!1, y!1)
  |-------
[1]   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of tr_cl_transitive_l0.1.

tr_cl_transitive_l0.2 :

  |-------
{1}   FORALL (y_1: T):
        (lessp!1(x!1, y_1) OR
          (EXISTS z:
             (NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, z)) AND
              lessp!1(z, y_1)))
         IMPLIES NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y_1)
[2]   FORALL (y):
        (lessp!1(z!1, x!1) AND tr_cl(lessp!1)(x!1, y)) IMPLIES
         tr_cl(lessp!1)(z!1, y)
```

```
Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
tr_cl_transitive_l0.2 :

  |-------
[1]   FORALL (y_1: T):
        (lessp!1(x!1, y_1) OR
          (EXISTS z:
             (NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, z)) AND
              lessp!1(z, y_1)))
         IMPLIES NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, y_1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_transitive_l0.2 :

{-1}  lessp!1(x!1, y!1) OR
       (EXISTS z:
          (NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, z)) AND
           lessp!1(z, y!1))
{-2}  (lessp!1(z!1, x!1))
  |-------
{1}   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
tr_cl_transitive_l0.2.1 :

{-1}  lessp!1(x!1, y!1)
[-2]  (lessp!1(z!1, x!1))
  |-------
[1]   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (expand "tr_cl")
Expanding the definition of tr_cl,
this simplifies to:
tr_cl_transitive_l0.2.1 :

[-1]  lessp!1(x!1, y!1)
[-2]  (lessp!1(z!1, x!1))
  |-------
{1}   lessp!1(z!1, y!1) OR
       (EXISTS z: tr_cl(lessp!1)(z!1, z) AND lessp!1(z, y!1))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_transitive_l0.2.1 :
```

```
[-1]  lessp!1(x!1, y!1)
[-2]  (lessp!1(z!1, x!1))
  |-------
{1}   lessp!1(z!1, y!1)
{2}   EXISTS z: tr_cl(lessp!1)(z!1, z) AND lessp!1(z, y!1)

Rerunning step: (inst?)
Found substitution:
z gets x!1,
Using template: lessp!1(z, y!1)
Instantiating quantified variables,
this simplifies to:
tr_cl_transitive_l0.2.1 :

[-1]  lessp!1(x!1, y!1)
[-2]  (lessp!1(z!1, x!1))
  |-------
[1]   lessp!1(z!1, y!1)
{2}   tr_cl(lessp!1)(z!1, x!1) AND lessp!1(x!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_transitive_l0.2.1 :

[-1]  lessp!1(x!1, y!1)
[-2]  (lessp!1(z!1, x!1))
  |-------
{1}   tr_cl(lessp!1)(z!1, x!1)
[2]   lessp!1(z!1, y!1)

Rerunning step: (rewrite "tr_cl_cs_basic")
Found matching substitution:
y: T gets x!1,
x gets z!1,
<: pred[[T, T]] gets lessp!1,
Rewriting using tr_cl_cs_basic, matching in *,

This completes the proof of tr_cl_transitive_l0.2.1.

tr_cl_transitive_l0.2.2 :

{-1}  EXISTS z:
        (NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, z)) AND
         lessp!1(z, y!1)
[-2]  (lessp!1(z!1, x!1))
  |-------
[1]   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
```

```
tr_cl_transitive_l0.2.2 :

{-1}  NOT (lessp!1(z!1, x!1)) OR tr_cl(lessp!1)(z!1, z!2)
{-2}  lessp!1(z!2, y!1)
[-3]  (lessp!1(z!1, x!1))
  |-------
[1]   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_transitive_l0.2.2 :

{-1}  tr_cl(lessp!1)(z!1, z!2)
[-2]  lessp!1(z!2, y!1)
[-3]  (lessp!1(z!1, x!1))
  |-------
[1]   tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (expand "tr_cl" +)
Expanding the definition of tr_cl,
this simplifies to:
tr_cl_transitive_l0.2.2 :

[-1]  tr_cl(lessp!1)(z!1, z!2)
[-2]  lessp!1(z!2, y!1)
[-3]  (lessp!1(z!1, x!1))
  |-------
{1}   lessp!1(z!1, y!1) OR
       (EXISTS z: tr_cl(lessp!1)(z!1, z) AND lessp!1(z, y!1))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_transitive_l0.2.2 :

[-1]  tr_cl(lessp!1)(z!1, z!2)
[-2]  lessp!1(z!2, y!1)
[-3]  (lessp!1(z!1, x!1))
  |-------
{1}   lessp!1(z!1, y!1)
{2}   EXISTS z: tr_cl(lessp!1)(z!1, z) AND lessp!1(z, y!1)

Rerunning step: (inst?)
Found substitution:
z gets z!2,
Using template: tr_cl(lessp!1)(z!1, z)
Instantiating quantified variables,
this simplifies to:
tr_cl_transitive_l0.2.2 :

[-1]  tr_cl(lessp!1)(z!1, z!2)
```

```
[-2]  lessp!1(z!2, y!1)
[-3]  (lessp!1(z!1, x!1))
  |-------
[1]   lessp!1(z!1, y!1)
{2}   tr_cl(lessp!1)(z!1, z!2) AND lessp!1(z!2, y!1)

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of tr_cl_transitive_l0.2.2.


This completes the proof of tr_cl_transitive_l0.2.

Q.E.D.
tr_cl_transitive_l0 proved in 0.07 real, 0.07 cpu seconds
Rerunning proof of tr_cl_transitive


tr_cl_transitive :

  |-------
{1}   FORALL (<: pred[[T, T]]): transitive?(tr_cl(<))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_transitive :

  |-------
{1}   transitive?(tr_cl(lessp!1))

Rerunning step: (expand "transitive?")
Expanding the definition of transitive?,
this simplifies to:
tr_cl_transitive :

  |-------
{1}   FORALL (x: T), (y: T), (z: T):
        tr_cl(lessp!1)(x, y) & tr_cl(lessp!1)(y, z) => tr_cl(lessp!1)(x, z)

Rerunning step: (skolem 1 ("x!1" "_" "z!1"))
For the top quantifier in 1, we introduce Skolem constants: (x!1 _ z!1),
this simplifies to:
tr_cl_transitive :

  |-------
{1}   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) & tr_cl(lessp!1)(y, z!1) =>
         tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (lemma "tr_cl_weak_induction")
```

```
Applying tr_cl_weak_induction
this simplifies to:
tr_cl_transitive :

{-1}  FORALL (<, x, P: [T -> boolean]):
        (FORALL (y): (x < y OR (EXISTS z: P(z) AND z < y)) IMPLIES P(y))
         IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
[1]   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) & tr_cl(lessp!1)(y, z!1) =>
         tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (inst - "lessp!1" "x!1"
                 "lambda(y): not(tr_cl(lessp!1)(y,z!1)) or tr_cl(lessp!1)(x!1,z!1)")
Instantiating the top quantifier in - with the terms:
 lessp!1, x!1, lambda(y): not(tr_cl(lessp!1)(y,z!1)) or tr_cl(lessp!1)(x!1,z!1),
this simplifies to:
tr_cl_transitive :

{-1}  (FORALL (y_1: T):
         (lessp!1(x!1, y_1) OR
           (EXISTS z:
              (NOT (tr_cl(lessp!1)(z, z!1)) OR tr_cl(lessp!1)(x!1, z!1))
               AND lessp!1(z, y_1)))
          IMPLIES
          NOT (tr_cl(lessp!1)(y_1, z!1)) OR tr_cl(lessp!1)(x!1, z!1))
       IMPLIES
       (FORALL (y_1: T):
          tr_cl(lessp!1)(x!1, y_1) IMPLIES
           NOT (tr_cl(lessp!1)(y_1, z!1)) OR tr_cl(lessp!1)(x!1, z!1))
  |-------
[1]   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) & tr_cl(lessp!1)(y, z!1) =>
         tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
tr_cl_transitive.1 :

{-1}  FORALL (y_1: T):
        tr_cl(lessp!1)(x!1, y_1) IMPLIES
         NOT (tr_cl(lessp!1)(y_1, z!1)) OR tr_cl(lessp!1)(x!1, z!1)
  |-------
[1]   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) & tr_cl(lessp!1)(y, z!1) =>
         tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_transitive.1 :
```

```
[-1]  FORALL (y_1: T):
        tr_cl(lessp!1)(x!1, y_1) IMPLIES
         NOT (tr_cl(lessp!1)(y_1, z!1)) OR tr_cl(lessp!1)(x!1, z!1)
{-2}  tr_cl(lessp!1)(x!1, y!1)
{-3}  tr_cl(lessp!1)(y!1, z!1)
  |-------
{1}   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (inst?)
Found substitution:
y_1: T gets y!1,
Using template: NOT (tr_cl(lessp!1)(y_1, z!1))
Instantiating quantified variables,
this simplifies to:
tr_cl_transitive.1 :

{-1}  tr_cl(lessp!1)(x!1, y!1) IMPLIES
       NOT (tr_cl(lessp!1)(y!1, z!1)) OR tr_cl(lessp!1)(x!1, z!1)
[-2]  tr_cl(lessp!1)(x!1, y!1)
[-3]  tr_cl(lessp!1)(y!1, z!1)
  |-------
[1]   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of tr_cl_transitive.1.

tr_cl_transitive.2 :

  |-------
{1}   FORALL (y_1: T):
        (lessp!1(x!1, y_1) OR
          (EXISTS z:
             (NOT (tr_cl(lessp!1)(z, z!1)) OR tr_cl(lessp!1)(x!1, z!1)) AND
              lessp!1(z, y_1)))
         IMPLIES NOT (tr_cl(lessp!1)(y_1, z!1)) OR tr_cl(lessp!1)(x!1, z!1)
[2]   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) & tr_cl(lessp!1)(y, z!1) =>
         tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
tr_cl_transitive.2 :

  |-------
[1]   FORALL (y_1: T):
        (lessp!1(x!1, y_1) OR
          (EXISTS z:
             (NOT (tr_cl(lessp!1)(z, z!1)) OR tr_cl(lessp!1)(x!1, z!1)) AND
```

```
              lessp!1(z, y_1)))
         IMPLIES NOT (tr_cl(lessp!1)(y_1, z!1)) OR tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_transitive.2 :

{-1}  lessp!1(x!1, y!1) OR
       (EXISTS z:
          (NOT (tr_cl(lessp!1)(z, z!1)) OR tr_cl(lessp!1)(x!1, z!1)) AND
           lessp!1(z, y!1))
{-2}  (tr_cl(lessp!1)(y!1, z!1))
  |-------
{1}   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
tr_cl_transitive.2.1 :

{-1}  lessp!1(x!1, y!1)
[-2]  (tr_cl(lessp!1)(y!1, z!1))
  |-------
[1]   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (forward-chain "tr_cl_transitive_l0")
Forward chaining on tr_cl_transitive_l0,

This completes the proof of tr_cl_transitive.2.1.

tr_cl_transitive.2.2 :

{-1}  EXISTS z:
        (NOT (tr_cl(lessp!1)(z, z!1)) OR tr_cl(lessp!1)(x!1, z!1)) AND
         lessp!1(z, y!1)
[-2]  (tr_cl(lessp!1)(y!1, z!1))
  |-------
[1]   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_transitive.2.2 :

{-1}  NOT (tr_cl(lessp!1)(z!2, z!1)) OR tr_cl(lessp!1)(x!1, z!1)
{-2}  lessp!1(z!2, y!1)
[-3]  (tr_cl(lessp!1)(y!1, z!1))
  |-------
[1]   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (prop)
```

```
Applying propositional simplification,
this simplifies to:
tr_cl_transitive.2.2 :

[-1]  lessp!1(z!2, y!1)
[-2]  (tr_cl(lessp!1)(y!1, z!1))
  |-------
{1}   (tr_cl(lessp!1)(z!2, z!1))
[2]   tr_cl(lessp!1)(x!1, z!1)

Rerunning step: (forward-chain "tr_cl_transitive_l0")
Forward chaining on tr_cl_transitive_l0,

This completes the proof of tr_cl_transitive.2.2.


This completes the proof of tr_cl_transitive.2.

Q.E.D.
tr_cl_transitive proved in 0.06 real, 0.06 cpu seconds
Rerunning proof of tr_cl_equiv_cn


tr_cl_equiv_cn :

  |-------
{1}   FORALL (<: pred[[T, T]], x, y: T):
        tr_cl(<)(x, y) IMPLIES
         (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y)))

Rerunning step: (lemma "tr_cl_weak_induction")
Applying tr_cl_weak_induction
this simplifies to:
tr_cl_equiv_cn :

{-1}  FORALL (<, x, P: [T -> boolean]):
        (FORALL (y): (x < y OR (EXISTS z: P(z) AND z < y)) IMPLIES P(y))
         IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
[1]   FORALL (<: pred[[T, T]], x, y: T):
        tr_cl(<)(x, y) IMPLIES
         (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y)))

Rerunning step: (skolem 1 ("lessp!1" "x!1" "_"))
For the top quantifier in 1, we introduce Skolem constants:
  (lessp!1 x!1 _),
this simplifies to:
tr_cl_equiv_cn :

[-1]  FORALL (<, x, P: [T -> boolean]):
        (FORALL (y): (x < y OR (EXISTS z: P(z) AND z < y)) IMPLIES P(y))
         IMPLIES (FORALL (y): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
{1}   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) IMPLIES
         (lessp!1(x!1, y) OR
           (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y)))

Rerunning step: (inst - "lessp!1" "x!1"
                 "lambda(y): lessp!1(x!1, y) OR exists z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y)")
Instantiating the top quantifier in - with the terms:
 lessp!1, x!1, lambda(y): lessp!1(x!1, y) OR exists z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y),
this simplifies to:
tr_cl_equiv_cn :

{-1}  (FORALL (y_1: T):
         (lessp!1(x!1, y_1) OR
           (EXISTS (z_1: T):
              (lessp!1(x!1, z_1) OR
                (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z_1)))
               AND lessp!1(z_1, y_1)))
          IMPLIES
          lessp!1(x!1, y_1) OR
           (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y_1)))
       IMPLIES
       (FORALL (y_1: T):
          tr_cl(lessp!1)(x!1, y_1) IMPLIES
           lessp!1(x!1, y_1) OR
            (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y_1)))
  |-------
[1]   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) IMPLIES
         (lessp!1(x!1, y) OR
           (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y)))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_equiv_cn :

  |-------
{1}   FORALL (y_1: T):
        (lessp!1(x!1, y_1) OR
          (EXISTS (z_1: T):
             (lessp!1(x!1, z_1) OR
               (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z_1)))
              AND lessp!1(z_1, y_1)))
         IMPLIES
         lessp!1(x!1, y_1) OR
          (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y_1))
[2]   FORALL (y: T):
        tr_cl(lessp!1)(x!1, y) IMPLIES
         (lessp!1(x!1, y) OR
           (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y)))
```

```
Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
tr_cl_equiv_cn :

  |-------
[1]   FORALL (y_1: T):
        (lessp!1(x!1, y_1) OR
          (EXISTS (z_1: T):
             (lessp!1(x!1, z_1) OR
               (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z_1)))
              AND lessp!1(z_1, y_1)))
         IMPLIES
         lessp!1(x!1, y_1) OR
          (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y_1))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_equiv_cn :

{-1}  lessp!1(x!1, y!1) OR
       (EXISTS (z_1: T):
          (lessp!1(x!1, z_1) OR
            (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z_1)))
           AND lessp!1(z_1, y!1))
  |-------
{1}   lessp!1(x!1, y!1)
{2}   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_equiv_cn :

{-1}  EXISTS (z_1: T):
        (lessp!1(x!1, z_1) OR
          (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z_1)))
         AND lessp!1(z_1, y!1)
  |-------
[1]   lessp!1(x!1, y!1)
[2]   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_equiv_cn :

{-1}  lessp!1(x!1, z!1) OR
       (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z!1))
{-2}  lessp!1(z!1, y!1)
```

```
  |-------
[1]   lessp!1(x!1, y!1)
[2]   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
tr_cl_equiv_cn.1 :

{-1}  lessp!1(x!1, z!1)
[-2]  lessp!1(z!1, y!1)
  |-------
[1]   lessp!1(x!1, y!1)
[2]   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (inst?)
Found substitution:
z gets z!1,
Using template: lessp!1(x!1, z)
Instantiating quantified variables,
this simplifies to:
tr_cl_equiv_cn.1 :

[-1]  lessp!1(x!1, z!1)
[-2]  lessp!1(z!1, y!1)
  |-------
[1]   lessp!1(x!1, y!1)
{2}   lessp!1(x!1, z!1) AND tr_cl(lessp!1)(z!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_equiv_cn.1 :

[-1]  lessp!1(x!1, z!1)
[-2]  lessp!1(z!1, y!1)
  |-------
{1}   tr_cl(lessp!1)(z!1, y!1)
[2]   lessp!1(x!1, y!1)

Rerunning step: (rewrite "tr_cl_cs_basic")
Found matching substitution:
y: T gets y!1,
x gets z!1,
<: pred[[T, T]] gets lessp!1,
Rewriting using tr_cl_cs_basic, matching in *,

This completes the proof of tr_cl_equiv_cn.1.

tr_cl_equiv_cn.2 :

{-1}  EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, z!1)
```

```
[-2]  lessp!1(z!1, y!1)
  |-------
[1]   lessp!1(x!1, y!1)
[2]   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_equiv_cn.2 :

{-1}  lessp!1(x!1, z!2)
{-2}  tr_cl(lessp!1)(z!2, z!1)
[-3]  lessp!1(z!1, y!1)
  |-------
[1]   lessp!1(x!1, y!1)
[2]   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (inst + "z!2")
Instantiating the top quantifier in + with the terms:
 z!2,
this simplifies to:
tr_cl_equiv_cn.2 :

[-1]  lessp!1(x!1, z!2)
[-2]  tr_cl(lessp!1)(z!2, z!1)
[-3]  lessp!1(z!1, y!1)
  |-------
[1]   lessp!1(x!1, y!1)
{2}   lessp!1(x!1, z!2) AND tr_cl(lessp!1)(z!2, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_equiv_cn.2 :

[-1]  lessp!1(x!1, z!2)
[-2]  tr_cl(lessp!1)(z!2, z!1)
[-3]  lessp!1(z!1, y!1)
  |-------
{1}   tr_cl(lessp!1)(z!2, y!1)
[2]   lessp!1(x!1, y!1)

Rerunning step: (expand "tr_cl" +)
Expanding the definition of tr_cl,
this simplifies to:
tr_cl_equiv_cn.2 :

[-1]  lessp!1(x!1, z!2)
[-2]  tr_cl(lessp!1)(z!2, z!1)
[-3]  lessp!1(z!1, y!1)
  |-------
{1}   lessp!1(z!2, y!1) OR
```

```
       (EXISTS z: tr_cl(lessp!1)(z!2, z) AND lessp!1(z, y!1))
[2]   lessp!1(x!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
tr_cl_equiv_cn.2 :

[-1]  lessp!1(x!1, z!2)
[-2]  tr_cl(lessp!1)(z!2, z!1)
[-3]  lessp!1(z!1, y!1)
  |-------
{1}   lessp!1(z!2, y!1)
{2}   EXISTS z: tr_cl(lessp!1)(z!2, z) AND lessp!1(z, y!1)
[3]   lessp!1(x!1, y!1)

Rerunning step: (inst?)
Found substitution:
z gets z!1,
Using template: tr_cl(lessp!1)(z!2, z)
Instantiating quantified variables,
this simplifies to:
tr_cl_equiv_cn.2 :

[-1]  lessp!1(x!1, z!2)
[-2]  tr_cl(lessp!1)(z!2, z!1)
[-3]  lessp!1(z!1, y!1)
  |-------
[1]   lessp!1(z!2, y!1)
{2}   tr_cl(lessp!1)(z!2, z!1) AND lessp!1(z!1, y!1)
[3]   lessp!1(x!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of tr_cl_equiv_cn.2.

Q.E.D.
tr_cl_equiv_cn proved in 0.04 real, 0.04 cpu seconds
Rerunning proof of tr_cl_equiv_cs


tr_cl_equiv_cs :

  |-------
{1}   FORALL (<: pred[[T, T]], x, y: T):
        (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y))) IMPLIES
         tr_cl(<)(x, y)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
```

```
tr_cl_equiv_cs :

{-1}  (lessp!1(x!1, y!1) OR
        (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)))
  |-------
{1}   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
tr_cl_equiv_cs.1 :

{-1}  lessp!1(x!1, y!1)
  |-------
[1]   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (rewrite "tr_cl_cs_basic")
Found matching substitution:
y: T gets y!1,
x gets x!1,
<: pred[[T, T]] gets lessp!1,
Rewriting using tr_cl_cs_basic, matching in *,

This completes the proof of tr_cl_equiv_cs.1.

tr_cl_equiv_cs.2 :

{-1}  EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)
  |-------
[1]   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_equiv_cs.2 :

{-1}  lessp!1(x!1, z!1)
{-2}  tr_cl(lessp!1)(z!1, y!1)
  |-------
[1]   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (forward-chain "tr_cl_transitive_l0")
Forward chaining on tr_cl_transitive_l0,

This completes the proof of tr_cl_equiv_cs.2.

Q.E.D.
tr_cl_equiv_cs proved in 0.02 real, 0.02 cpu seconds
Rerunning proof of tr_cl_equiv


tr_cl_equiv :
  |-------
{1}   FORALL (<: pred[[T, T]], x, y: T):
        tr_cl(<)(x, y) IFF (x < y OR (EXISTS z: x < z AND tr_cl(<)(z, y)))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
tr_cl_equiv :

  |-------
{1}   tr_cl(lessp!1)(x!1, y!1) IFF
       (lessp!1(x!1, y!1) OR
         (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)))

Rerunning step: (prop)
Applying propositional simplification,
this yields  3 subgoals:
tr_cl_equiv.1 :

{-1}  tr_cl(lessp!1)(x!1, y!1)
  |-------
{1}   lessp!1(x!1, y!1)
{2}   EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)

Rerunning step: (forward-chain "tr_cl_equiv_cn")
Forward chaining on tr_cl_equiv_cn,

This completes the proof of tr_cl_equiv.1.

tr_cl_equiv.2 :

{-1}  lessp!1(x!1, y!1)
  |-------
{1}   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (rewrite "tr_cl_equiv_cs")
Found matching substitution:
y: T gets y!1,
x gets x!1,
<: pred[[T, T]] gets lessp!1,
Rewriting using tr_cl_equiv_cs, matching in *,

This completes the proof of tr_cl_equiv.2.

tr_cl_equiv.3 :

{-1}  EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)
  |-------
{1}   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (rewrite "tr_cl_equiv_cs")
```

```
Found matching substitution:
y: T gets y!1,
x gets x!1,
<: pred[[T, T]] gets lessp!1,
Rewriting using tr_cl_equiv_cs, matching in *,
this simplifies to:
tr_cl_equiv.3 :

[-1]  EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)
  |-------
{1}   (lessp!1(x!1, y!1) OR
        (EXISTS z: lessp!1(x!1, z) AND tr_cl(lessp!1)(z, y!1)))
[2]   tr_cl(lessp!1)(x!1, y!1)

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of tr_cl_equiv.3.

Q.E.D.
tr_cl_equiv proved in 0.02 real, 0.02 cpu seconds
cl_tr: 6 proofs attempted, 6 proved in 0.21 real, 0.21 cpu seconds


 Proof summary for theory cl_tr
    tr_cl_cs_basic.......................proved - complete   [shostak](0.00 s)
    tr_cl_transitive_l0..................proved - complete   [shostak](0.07 s)
    tr_cl_transitive.....................proved - complete   [shostak](0.06 s)
    tr_cl_equiv_cn.......................proved - complete   [shostak](0.04 s)
    tr_cl_equiv_cs.......................proved - complete   [shostak](0.02 s)
    tr_cl_equiv..........................proved - complete   [shostak](0.02 s)
    Theory totals: 6 formulas, 6 attempted, 6 succeeded (0.21 s)

Grand Totals: 6 proofs, 6 attempted, 6 succeeded (0.21 s)
Parsing wf_cl_tr
wf_cl_tr parsed in 0.03 seconds
Typechecking wf_cl_tr
wf_cl_tr typechecked in 0.09s: No TCCs generated
Rerunning proof of well_founded_part_cl_tr_l1



well_founded_part_cl_tr_l1 :

  |-------
{1}   FORALL (<: pred[[T, T]], x: T):
        (FORALL y: y < x IMPLIES well_founded_part[T, tr_cl(<)](y)) IMPLIES
         (FORALL z:
            tr_cl(<)(z, x) IMPLIES well_founded_part[T, tr_cl(<)](z))

Rerunning step: (apply (then (skosimp*) (expand "tr_cl") (prop)))
Applying
   (then (skosimp*) (expand "tr_cl") (prop)),
this yields  2 subgoals:
well_founded_part_cl_tr_l1.1 :

{-1}  lessp!1(z!1, x!1)
{-2}  FORALL y:
        lessp!1(y, x!1) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y)
  |-------
{1}   well_founded_part[T, tr_cl(lessp!1)](z!1)

Rerunning step: (apply (then (inst?) (prop)))
Applying
   (then (inst?) (prop)),

This completes the proof of well_founded_part_cl_tr_l1.1.

well_founded_part_cl_tr_l1.2 :

{-1}  EXISTS (z: T): tr_cl(lessp!1)(z!1, z) AND lessp!1(z, x!1)
{-2}  FORALL y:
        lessp!1(y, x!1) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y)
  |-------
{1}   well_founded_part[T, tr_cl(lessp!1)](z!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_cl_tr_l1.2 :

{-1}  tr_cl(lessp!1)(z!1, z!2)
{-2}  lessp!1(z!2, x!1)
[-3]  FORALL y:
        lessp!1(y, x!1) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y)
  |-------
[1]   well_founded_part[T, tr_cl(lessp!1)](z!1)

Rerunning step: (inst - "z!2")
Instantiating the top quantifier in - with the terms:
 z!2,
this simplifies to:
well_founded_part_cl_tr_l1.2 :

[-1]  tr_cl(lessp!1)(z!1, z!2)
[-2]  lessp!1(z!2, x!1)
{-3}  lessp!1(z!2, x!1) IMPLIES well_founded_part[T, tr_cl(lessp!1)](z!2)
  |-------
[1]   well_founded_part[T, tr_cl(lessp!1)](z!1)

Rerunning step: (apply (then (prop) (expand "well_founded_part" -)
                        (inst?) (prop)))
Applying
   (then (prop) (expand "well_founded_part" -) (inst?) (prop)),
This completes the proof of well_founded_part_cl_tr_l1.2.

Q.E.D.
well_founded_part_cl_tr_l1 proved in 0.05 real, 0.05 cpu seconds
Rerunning proof of well_founded_part_cl_tr


well_founded_part_cl_tr :

  |-------
{1}   FORALL (<: pred[[T, T]], x: T):
        well_founded_part[T, <](x) IMPLIES well_founded_part[T, tr_cl(<)](x)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_cl_tr :

{-1}  well_founded_part[T, lessp!1](x!1)
  |-------
{1}   well_founded_part[T, tr_cl(lessp!1)](x!1)

Rerunning step: (apply (then
                        (lemma
                         "well_founded_part_weak_induction[T,lessp!1]")
                        (inst -
                         "lambda(x): well_founded_part[T, tr_cl(lessp!1)](x)")
                        (prop)))
Applying
   (then (lemma "well_founded_part_weak_induction[T,lessp!1]")
    (inst - "lambda(x): well_founded_part[T, tr_cl(lessp!1)](x)")
    (prop)),
this yields  2 subgoals:
well_founded_part_cl_tr.1 :

{-1}  FORALL (x_1: T):
        well_founded_part(x_1) IMPLIES
         well_founded_part[T, tr_cl(lessp!1)](x_1)
[-2]  well_founded_part[T, lessp!1](x!1)
  |-------
[1]   well_founded_part[T, tr_cl(lessp!1)](x!1)

Rerunning step: (apply (then (inst?) (prop)))
Applying
   (then (inst?) (prop)),

This completes the proof of well_founded_part_cl_tr.1.

well_founded_part_cl_tr.2 :

[-1]  well_founded_part[T, lessp!1](x!1)
  |-------
```

```
{1}   FORALL (x_1: T):
        (FORALL (y: T):
           lessp!1(y, x_1) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y))
         IMPLIES well_founded_part[T, tr_cl(lessp!1)](x_1)
[2]   well_founded_part[T, tr_cl(lessp!1)](x!1)

Rerunning step: (apply (then (hide -1 2) (skosimp*)))
Applying
   (then (hide -1 2) (skosimp*)),
this simplifies to:
well_founded_part_cl_tr.2 :

{-1}  FORALL (y: T):
        lessp!1(y, x!2) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y)
  |-------
{1}   well_founded_part[T, tr_cl(lessp!1)](x!2)

Rerunning step: (expand "well_founded_part" +)
Expanding the definition of well_founded_part,
this simplifies to:
well_founded_part_cl_tr.2 :

[-1]  FORALL (y: T):
        lessp!1(y, x!2) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y)
  |-------
{1}   FORALL (y: T): tr_cl(lessp!1)(y, x!2) IMPLIES well_founded_part(y)

Rerunning step: (use "well_founded_part_cl_tr_l1")
Using lemma well_founded_part_cl_tr_l1,
this simplifies to:
well_founded_part_cl_tr.2 :

{-1}  (FORALL y:
         lessp!1(y, x!2) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y))
       IMPLIES
       (FORALL z:
          tr_cl(lessp!1)(z, x!2) IMPLIES
           well_founded_part[T, tr_cl(lessp!1)](z))
[-2]  FORALL (y: T):
        lessp!1(y, x!2) IMPLIES well_founded_part[T, tr_cl(lessp!1)](y)
  |-------
[1]   FORALL (y: T): tr_cl(lessp!1)(y, x!2) IMPLIES well_founded_part(y)

Rerunning step: (grind :defs nil)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of well_founded_part_cl_tr.2.

Q.E.D.
well_founded_part_cl_tr proved in 0.03 real, 0.03 cpu seconds
Rerunning proof of well_founded_cl_tr
```

```
well_founded_cl_tr :

  |-------
{1}   FORALL (<: pred[[T, T]]):
        well_founded?[T](<) IMPLIES well_founded?[T](tr_cl(<))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_cl_tr :

{-1}  well_founded?[T](lessp!1)
  |-------
{1}   well_founded?[T](tr_cl(lessp!1))

Rerunning step: (forward-chain "well_founded_part_cn[T,lessp!1]")
Forward chaining on well_founded_part_cn[T, lessp!1],
this simplifies to:
well_founded_cl_tr :

{-1}  FORALL (x: T): well_founded_part(x)
[-2]  well_founded?[T](lessp!1)
  |-------
[1]   well_founded?[T](tr_cl(lessp!1))

Rerunning step: (hide -2)
Hiding formulas:  -2,
this simplifies to:
well_founded_cl_tr :

[-1]  FORALL (x: T): well_founded_part(x)
  |-------
[1]   well_founded?[T](tr_cl(lessp!1))

Rerunning step: (rewrite "well_founded_part_cs[T,tr_cl(lessp!1)]")
Found matching substitution:
Rewriting using well_founded_part_cs[T, tr_cl(lessp!1)], matching in *,
this simplifies to:
well_founded_cl_tr :

[-1]  FORALL (x: T): well_founded_part(x)
  |-------
{1}   (FORALL (x: T): well_founded_part(x))
[2]   well_founded?[T](tr_cl(lessp!1))

Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
well_founded_cl_tr :

[-1]  FORALL (x: T): well_founded_part(x)
```

```
  |-------
[1]   (FORALL (x: T): well_founded_part(x))

Rerunning step: (use "well_founded_part_cl_tr")
Using lemma well_founded_part_cl_tr,
this simplifies to:
well_founded_cl_tr :

{-1}  FORALL (<: pred[[T, T]], x: T):
        well_founded_part[T, <](x) IMPLIES well_founded_part[T, tr_cl(<)](x)
[-2]  FORALL (x: T): well_founded_part(x)
  |-------
[1]   (FORALL (x: T): well_founded_part(x))

Rerunning step: (grind :defs nil)
Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.
well_founded_cl_tr proved in 0.04 real, 0.04 cpu seconds
wf_cl_tr: 3 proofs attempted, 3 proved in 0.12 real, 0.12 cpu seconds


 Proof summary for theory wf_cl_tr
    well_founded_part_cl_tr_l1...........proved - complete   [shostak](0.05 s)
    well_founded_part_cl_tr..............proved - complete   [shostak](0.03 s)
    well_founded_cl_tr...................proved - complete   [shostak](0.04 s)
    Theory totals: 3 formulas, 3 attempted, 3 succeeded (0.12 s)

Grand Totals: 3 proofs, 3 attempted, 3 succeeded (0.12 s)
Context changed to ~/alonso/estudio/alc/auxiliares/bags/
Parsing finite_bags_order
finite_bags_order parsed in 0.20 seconds
Typechecking finite_bags_order
Parsing finite_bags_lems
finite_bags_lems parsed in 0.09 seconds
Typechecking finite_bags_lems
Parsing finite_bags
finite_bags parsed in 0.11 seconds
Typechecking finite_bags
Parsing bags
bags parsed in 0.10 seconds
Typechecking bags
bags typechecked in 1.01s: 2 TCCs, 0 proved, 0 subsumed, 2 unproved
Restored theory from prelude_aux.bin in 0.17s (load part took 0.05s)
Restored theory from finite_sets_inductions.bin in 0.15s (load part took 0.11s)
Restored theory from finite_sets_sum.bin in 0.30s (load part took 0.18s)
Restored theory from finite_sets_sum_real.bin in 0.36s (load part took 0.23s)
Parsing bags_to_sets
bags_to_sets parsed in 0.07 seconds
Typechecking bags_to_sets
bags_to_sets typechecked in 0.03s: No TCCs generated

 Judgement at line 81 may lead to unprovable TCCs
```

```
See language document for details.

finite_bags typechecked in 2.99s: 17 TCCs, 0 proved, 0 subsumed, 17 unproved; 1 warning
Parsing finite_bags_inductions
finite_bags_inductions parsed in 0.01 seconds
Typechecking finite_bags_inductions
Parsing finite_bags_aux
finite_bags_aux parsed in 0.07 seconds
Typechecking finite_bags_aux
Parsing bags_aux
bags_aux parsed in 0.02 seconds
Typechecking bags_aux

 % The subtype TCC (at line 26, column 57) in decl choose_gt_zero for  b
    %expected type  (nonempty_bag?)
  % was not generated because it simplifies to TRUE.


 % The subtype TCC (at line 28, column 62) in decl insert_choose_rest for  b
    %expected type  (nonempty_bag?)
  % was not generated because it simplifies to TRUE.

Added 8 proofs from theory bags_aux to orphaned-proofs.prf
bags_aux typechecked in 0.04s: 2 TCCs, 0 proved, 0 subsumed, 2 unproved; 2 msgs
finite_bags_aux typechecked in 0.21s: 4 TCCs, 0 proved, 0 subsumed, 4 unproved
finite_bags_inductions typechecked in 0.36s: 1 TCC, 0 proved, 0 subsumed, 1 unproved

 % The subtype TCC (at line 23, column 41) in decl card_bag_disj_union for  union(A, B)
    %expected type  finite_bag[T]
  % is subsumed by card_bag_union_TCC1

finite_bags_lems typechecked in 3.58s: 4 TCCs, 0 proved, 1 subsumed, 3 unproved; 1 msg
Restored theory from wf_cs.bin in 0.04s (load part took 0.01s)
Restored theory from cl_tr.bin in 0.09s (load part took 0.02s)
Restored theory from wf_cl_tr.bin in 0.01s (load part took 0.01s)
finite_bags_order typechecked in 3.98s: 3 TCCs, 0 proved, 0 subsumed, 3 unproved
Rerunning proof of insert_TCC1


insert_TCC1 :

  |-------
{1}   FORALL (A, x): is_finite[T](insert[T](x, A))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
insert_TCC1 :

  |-------
{1}   is_finite[T](insert[T](x!1, A!1))
```

```
Rerunning step: (rewrite "finite_insert")
Found matching substitution:
B: finite_bag[T] gets A!1,
t: T gets x!1,
Rewriting using finite_insert, matching in *,
Q.E.D.
insert_TCC1 proved in 0.02 real, 0.02 cpu seconds
Rerunning proof of less_1_insert_cn


less_1_insert_cn :

  |-------
{1}   FORALL (M, N: finite_bag[T], a: T):
        less_1(N, insert(a, M)) IMPLIES
         (EXISTS M_0: N = insert(a, M_0) AND less_1(M_0, M)) OR
          (EXISTS K: N = plus(M, K) AND less(K, a))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_1_insert_cn :

{-1}  less_1(N!1, insert(a!1, M!1))
  |-------
{1}   EXISTS M_0: N!1 = insert(a!1, M_0) AND less_1(M_0, M!1)
{2}   EXISTS K: N!1 = plus(M!1, K) AND less(K, a!1)

Rerunning step: (expand "less_1" -)
Expanding the definition of less_1,
this simplifies to:
less_1_insert_cn :

{-1}  EXISTS M_0, a, K:
        insert(a!1, M!1) = insert(a, M_0) AND
         N!1 = plus(M_0, K) AND less(K, a)
  |-------
[1]   EXISTS M_0: N!1 = insert(a!1, M_0) AND less_1(M_0, M!1)
[2]   EXISTS K: N!1 = plus(M!1, K) AND less(K, a!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_1_insert_cn :

{-1}  insert(a!1, M!1) = insert(a!2, M!2)
{-2}  N!1 = plus(M!2, K!1)
{-3}  less(K!1, a!2)
  |-------
[1]   EXISTS M_0: N!1 = insert(a!1, M_0) AND less_1(M_0, M!1)
[2]   EXISTS K: N!1 = plus(M!1, K) AND less(K, a!1)
```

```
Rerunning step: (case-replace "a!2 = a!1")
Assuming and applying a!2 = a!1,
this yields  2 subgoals:
less_1_insert_cn.1 :

{-1}  a!2 = a!1
{-2}  insert(a!1, M!1) = insert(a!1, M!2)
[-3]  N!1 = plus(M!2, K!1)
{-4}  less(K!1, a!1)
  |-------
[1]   EXISTS M_0: N!1 = insert(a!1, M_0) AND less_1(M_0, M!1)
[2]   EXISTS K: N!1 = plus(M!1, K) AND less(K, a!1)

Rerunning step: (apply (then (hide 1) (inst?) (prop) (replace*)
                        (hide -1 -3 -4)))
Applying
   (then (hide 1) (inst?) (prop) (replace*) (hide -1 -3 -4)),
this simplifies to:
less_1_insert_cn.1 :

[-1]  insert(a!1, M!1) = insert(a!1, M!2)
  |-------
{1}   plus(M!2, K!1) = plus(M!1, K!1)

Rerunning step: (grind :defs nil :rewrites
                 ("bag_equality" "eqmult" "plus") :polarity? t)
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
less_1_insert_cn.1 :

{-1}  FORALL (x: T): (insert(a!1, M!1)(x) = insert(a!1, M!2)(x))
  |-------
{1}   M!2(x!1) = M!1(x!1)

Rerunning step: (apply (then (inst?) (expand "insert") (lift-if) (prop)
                        (grind :defs nil)))
Applying
   (then (inst?) (expand "insert") (lift-if) (prop) (grind :defs nil)),

This completes the proof of less_1_insert_cn.1.

less_1_insert_cn.2 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   a!2 = a!1
[2]   EXISTS M_0: N!1 = insert(a!1, M_0) AND less_1(M_0, M!1)
[3]   EXISTS K: N!1 = plus(M!1, K) AND less(K, a!1)

Rerunning step: (hide 3)
```

```
Hiding formulas:  3,
this simplifies to:
less_1_insert_cn.2 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
[1]   a!2 = a!1
[2]   EXISTS M_0: N!1 = insert(a!1, M_0) AND less_1(M_0, M!1)

Rerunning step: (inst + "plus(delete(a!1,M!2,1),K!1)")
Instantiating the top quantifier in + with the terms:
 plus(delete(a!1,M!2,1),K!1),
this yields  2 subgoals:
less_1_insert_cn.2.1 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
[1]   a!2 = a!1
{2}   N!1 = insert(a!1, plus(delete(a!1, M!2, 1), K!1)) AND
       less_1(plus(delete(a!1, M!2, 1), K!1), M!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
less_1_insert_cn.2.1.1 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   N!1 = insert(a!1, plus(delete(a!1, M!2, 1), K!1))
[2]   a!2 = a!1

Rerunning step: (replace*)
Repeatedly applying the replace rule,
this simplifies to:
less_1_insert_cn.2.1.1 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   plus(M!2, K!1) = insert(a!1, plus(delete(a!1, M!2, 1), K!1))
[2]   a!2 = a!1

Rerunning step: (grind :defs nil :rewrites
                 ("bag_equality" "eqmult" "plus") :polarity? t)
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
less_1_insert_cn.2.1.1 :

{-1}  FORALL (x: T): (insert(a!1, M!1)(x) = insert(a!2, M!2)(x))
{-2}  FORALL (x: T): (N!1(x) = K!1(x) + M!2(x))
[-3]  less(K!1, a!2)
  |-------
{1}   (K!1(x!1) + M!2(x!1) =
        insert(a!1, plus(delete(a!1, M!2, 1), K!1))(x!1))
[2]   a!2 = a!1

Rerunning step: (apply (then (inst?) (inst?) (expand "insert")
                        (lift-if) (prop)
                        (grind :defs nil :rewrites ("plus" "delete"))))
Applying
   (then (inst?) (inst?) (expand "insert") (lift-if) (prop)
    (grind :defs nil :rewrites ("plus" "delete"))),

This completes the proof of less_1_insert_cn.2.1.1.

less_1_insert_cn.2.1.2 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   less_1(plus(delete(a!1, M!2, 1), K!1), M!1)
[2]   a!2 = a!1

Rerunning step: (expand "less_1")
Expanding the definition of less_1,
this simplifies to:
less_1_insert_cn.2.1.2 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   EXISTS M_0, a, K:
        M!1 = insert(a, M_0) AND
         plus(delete(a!1, M!2, 1), K!1) = plus(M_0, K) AND less(K, a)
[2]   a!2 = a!1

Rerunning step: (inst + "delete(a!1,M!2,1)" "a!2" "K!1")
Instantiating the top quantifier in + with the terms:
 delete(a!1,M!2,1), a!2, K!1,
this yields  2 subgoals:
less_1_insert_cn.2.1.2.1 :

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
```

```
  |-------
{1}   M!1 = insert(a!2, delete(a!1, M!2, 1)) AND less(K!1, a!2)
[2]   a!2 = a!1

Rerunning step: (grind :defs nil :rewrites
                 ("bag_equality" "eqmult" "plus") :polarity? t)
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
less_1_insert_cn.2.1.2.1 :

[-1]  less(K!1, a!2)
{-2}  FORALL (x: T): (insert(a!1, M!1)(x) = insert(a!2, M!2)(x))
{-3}  FORALL (x: T): (N!1(x) = K!1(x) + M!2(x))
  |-------
{1}   (M!1(x!1) = insert(a!2, delete(a!1, M!2, 1))(x!1))
[2]   a!2 = a!1

Rerunning step: (apply (then (inst?) (inst?) (expand "insert")
                        (lift-if) (prop)
                        (grind :defs nil :rewrites ("delete"))))
Applying
   (then (inst?) (inst?) (expand "insert") (lift-if) (prop)
    (grind :defs nil :rewrites ("delete"))),

This completes the proof of less_1_insert_cn.2.1.2.1.

less_1_insert_cn.2.1.2.2 (TCC):

[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   is_finite[T](delete[T](a!1, M!2, 1))
[2]   a!2 = a!1

Rerunning step: (rewrite "finite_delete")
Found matching substitution:
n: nat gets 1,
B: finite_bag[T] gets M!2,
t: T gets a!1,
Rewriting using finite_delete, matching in *,

This completes the proof of less_1_insert_cn.2.1.2.2.


This completes the proof of less_1_insert_cn.2.1.2.


This completes the proof of less_1_insert_cn.2.1.

less_1_insert_cn.2.2 (TCC):
```

```
[-1]  insert(a!1, M!1) = insert(a!2, M!2)
[-2]  N!1 = plus(M!2, K!1)
[-3]  less(K!1, a!2)
  |-------
{1}   is_finite[T](plus[T](delete[T](a!1, M!2, 1), K!1))
[2]   a!2 = a!1

Rerunning step: (apply (then (rewrite "finite_bag_plus")
                        (rewrite "finite_delete")))
Applying
   (then (rewrite "finite_bag_plus") (rewrite "finite_delete")),

This completes the proof of less_1_insert_cn.2.2.


This completes the proof of less_1_insert_cn.2.

Q.E.D.
less_1_insert_cn proved in 0.69 real, 0.69 cpu seconds
Rerunning proof of plus_emptybag


plus_emptybag :

  |-------
{1}   FORALL (M: finite_bag[T]): plus(M, emptybag) = M

Rerunning step: (grind :defs nil :rewrites
                 ("bag_equality" "eqmult" "plus" "emptybag" "max"))
Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.
plus_emptybag proved in 0.04 real, 0.04 cpu seconds
Rerunning proof of insert_plus


insert_plus :

  |-------
{1}   FORALL (M, N: finite_bag[T], x: T):
        plus(M, insert(x, N)) = insert(x, plus(M, N))

Rerunning step: (grind :defs nil :rewrites
                 ("bag_equality" "eqmult" "plus" "insert"))
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
insert_plus :

{-1}  is_finite[T](M!1)
{-2}  is_finite[T](N!1)
  |-------
{1}   (insert(x!1, N!1)(x!2) + M!1(x!2) = insert(x!1, plus(M!1, N!1))(x!2))
```

```
Rerunning step: (expand "insert")
Expanding the definition of insert,
this simplifies to:
insert_plus :

[-1]  is_finite[T](M!1)
[-2]  is_finite[T](N!1)
  |-------
{1}   (IF x!1 = x!2 THEN 1 + N!1(x!2) ELSE N!1(x!2) ENDIF + M!1(x!2) =
        IF x!1 = x!2 THEN 1 + plus(M!1, N!1)(x!2)
        ELSE plus(M!1, N!1)(x!2)
        ENDIF)

Rerunning step: (lift-if)
Lifting IF-conditions to the top level,
this simplifies to:
insert_plus :

[-1]  is_finite[T](M!1)
[-2]  is_finite[T](N!1)
  |-------
{1}   IF x!1 = x!2 THEN (1 + N!1(x!2) + M!1(x!2) = 1 + plus(M!1, N!1)(x!2))
      ELSE (N!1(x!2) + M!1(x!2) = plus(M!1, N!1)(x!2))
      ENDIF

Rerunning step: (expand "plus")
Expanding the definition of plus,
this simplifies to:
insert_plus :

[-1]  is_finite[T](M!1)
[-2]  is_finite[T](N!1)
  |-------
{1}   TRUE

which is trivially true.
Q.E.D.
insert_plus proved in 0.23 real, 0.23 cpu seconds
Rerunning proof of less_insert_cn


less_insert_cn :

  |-------
{1}   FORALL (K: finite_bag[T], a, x: T):
        less(insert(x, K), a) IMPLIES x < a

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_insert_cn :
```

```
{-1}  less(insert(x!1, K!1), a!1)
  |-------
{1}   x!1 < a!1

Rerunning step: (expand "less")
Expanding the definition of less,
this simplifies to:
less_insert_cn :

{-1}  FORALL b: member(b, insert(x!1, K!1)) IMPLIES b < a!1
  |-------
[1]   x!1 < a!1

Rerunning step: (inst?)
Found substitution:
b gets x!1,
Using template: b < a!1
Instantiating quantified variables,
this simplifies to:
less_insert_cn :

{-1}  member(x!1, insert(x!1, K!1)) IMPLIES x!1 < a!1
  |-------
[1]   x!1 < a!1

Rerunning step: (expand "member")
Expanding the definition of member,
this simplifies to:
less_insert_cn :

{-1}  insert(x!1, K!1)(x!1) > 0 IMPLIES x!1 < a!1
  |-------
[1]   x!1 < a!1

Rerunning step: (expand "insert")
Expanding the definition of insert,
this simplifies to:
less_insert_cn :

{-1}  1 + K!1(x!1) > 0 IMPLIES x!1 < a!1
  |-------
[1]   x!1 < a!1

Rerunning step: (assert)
Simplifying, rewriting, and recording with decision procedures,
Q.E.D.
less_insert_cn proved in 0.12 real, 0.12 cpu seconds
Rerunning proof of less_insert_cn2


less_insert_cn2 :
  |-------
{1}   FORALL (K: finite_bag[T], a, x: T):
        less(insert(x, K), a) IMPLIES less(K, a)

Rerunning step: (grind :defs nil :rewrites ("less" "member") :polarity?
                 t)
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
less_insert_cn2 :

{-1}  is_finite[T](K!1)
{-2}  K!1(b!1) > 0
  |-------
{1}   insert(x!1, K!1)(b!1) > 0
{2}   b!1 < a!1

Rerunning step: (apply (then (expand "insert") (lift-if) (prop)
                        (assert)))
Applying
   (then (expand "insert") (lift-if) (prop) (assert)),
Q.E.D.
less_insert_cn2 proved in 0.17 real, 0.17 cpu seconds
Rerunning proof of IMP_wf_cs_TCC1


IMP_wf_cs_TCC1 :

  |-------
{1}   EXISTS (x: finite_bag[T]): TRUE

Rerunning step: (inst + "emptybag[T]")
Instantiating the top quantifier in + with the terms:
 emptybag[T],
Q.E.D.
IMP_wf_cs_TCC1 proved in 0.01 real, 0.01 cpu seconds
Rerunning proof of well_founded_part_less_1_plus_TCC1


well_founded_part_less_1_plus_TCC1 :

  |-------
{1}   FORALL (K, M1: finite_bag[T], a: T):
        (FORALL b:
           b < a IMPLIES
            (FORALL M:
               well_founded_part(M) IMPLIES
                well_founded_part(insert(b, M))))
         AND less(K, a) AND well_founded_part(M1)
         IMPLIES is_finite[T](plus[T](M1, K))

Rerunning step: (skolem-typepred)
Skolemizing (with typepred on new Skolem constants),
this simplifies to:
well_founded_part_less_1_plus_TCC1 :

{-1}  is_finite[T](K!1)
{-2}  is_finite[T](M1!1)
  |-------
{1}   (FORALL b:
         b < a!1 IMPLIES
          (FORALL M:
             well_founded_part(M) IMPLIES well_founded_part(insert(b, M))))
       AND less(K!1, a!1) AND well_founded_part(M1!1)
       IMPLIES is_finite[T](plus[T](M1!1, K!1))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_plus_TCC1 :

[-1]  is_finite[T](K!1)
[-2]  is_finite[T](M1!1)
{-3}  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
{-4}  less(K!1, a!1)
{-5}  well_founded_part(M1!1)
  |-------
{1}   is_finite[T](plus[T](M1!1, K!1))

Rerunning step: (rewrite "finite_bag_plus")
Found matching substitution:
B: finite_bag[T] gets K!1,
A gets M1!1,
Rewriting using finite_bag_plus, matching in *,
Q.E.D.
well_founded_part_less_1_plus_TCC1 proved in 0.01 real, 0.01 cpu seconds
Rerunning proof of well_founded_part_less_1_plus


well_founded_part_less_1_plus :

  |-------
{1}   FORALL (K, M1: finite_bag[T], a: T):
        (well_founded_part(M1) AND
          less(K, a) AND
           (FORALL b:
              b < a IMPLIES
               (FORALL M:
                  well_founded_part(M) IMPLIES
                   well_founded_part(insert(b, M)))))
         IMPLIES well_founded_part(plus(M1, K))
```

```
Rerunning step: (skolem 1 ("_" "M1" "a"))
For the top quantifier in 1, we introduce Skolem constants: (_ M1 a),
this simplifies to:
well_founded_part_less_1_plus :

  |-------
{1}   FORALL (K):
        (well_founded_part(M1) AND
          less(K, a) AND
           (FORALL b:
              b < a IMPLIES
               (FORALL M:
                  well_founded_part(M) IMPLIES
                   well_founded_part(insert(b, M)))))
         IMPLIES well_founded_part(plus(M1, K))

Rerunning step: (induct "K" :name "finite_bag_induction")
Inducting on K on formula 1 using induction scheme finite_bag_induction,
this yields  3 subgoals:
well_founded_part_less_1_plus.1 :

  |-------
{1}   (well_founded_part(M1) AND
        less(emptybag[T], a) AND
         (FORALL b:
            b < a IMPLIES
             (FORALL M:
                well_founded_part(M) IMPLIES
                 well_founded_part(insert(b, M)))))
       IMPLIES well_founded_part(plus(M1, emptybag[T]))

***Warning: Fewer subgoals (3) than subproofs (4)
Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_plus.1 :

{-1}  well_founded_part(M1)
{-2}  less(emptybag[T], a)
{-3}  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   well_founded_part(plus(M1, emptybag[T]))

Rerunning step: (rewrite "plus_emptybag")
Found matching substitution:
M: finite_bag[T] gets M1,
Rewriting using plus_emptybag, matching in *,

This completes the proof of well_founded_part_less_1_plus.1.
```

```
well_founded_part_less_1_plus.2 :

  |-------
{1}   FORALL (x: T), (b_1: finite_bag[T]):
        ((well_founded_part(M1) AND
           less(b_1, a) AND
            (FORALL b:
               b < a IMPLIES
                (FORALL M:
                   well_founded_part(M) IMPLIES
                    well_founded_part(insert(b, M)))))
          IMPLIES well_founded_part(plus(M1, b_1)))
         IMPLIES
         (well_founded_part(M1) AND
           less(insert(x, b_1), a) AND
            (FORALL b:
               b < a IMPLIES
                (FORALL M:
                   well_founded_part(M) IMPLIES
                    well_founded_part(insert(b, M)))))
          IMPLIES well_founded_part(plus(M1, insert(x, b_1)))

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_plus.2 :

{-1}  (well_founded_part(M1) AND
        less(b!1, a) AND
         (FORALL b:
            b < a IMPLIES
             (FORALL M:
                well_founded_part(M) IMPLIES
                 well_founded_part(insert(b, M)))))
       IMPLIES well_founded_part(plus(M1, b!1))
{-2}  well_founded_part(M1)
{-3}  less(insert(x!1, b!1), a)
{-4}  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   well_founded_part(plus(M1, insert(x!1, b!1)))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
well_founded_part_less_1_plus.2.1 :

{-1}  well_founded_part(plus(M1, b!1))
[-2]  well_founded_part(M1)
```

```
[-3]  less(insert(x!1, b!1), a)
[-4]  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
[1]   well_founded_part(plus(M1, insert(x!1, b!1)))

Rerunning step: (rewrite "insert_plus")
Found matching substitution:
N: finite_bag[T] gets b!1,
x: T gets x!1,
M gets M1,
Rewriting using insert_plus, matching in *,
this simplifies to:
well_founded_part_less_1_plus.2.1 :

[-1]  well_founded_part(plus(M1, b!1))
[-2]  well_founded_part(M1)
[-3]  less(insert(x!1, b!1), a)
[-4]  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   well_founded_part(insert(x!1, plus(M1, b!1)))

Rerunning step: (inst -4 "x!1")
Instantiating the top quantifier in -4 with the terms:
 x!1,
this simplifies to:
well_founded_part_less_1_plus.2.1 :

[-1]  well_founded_part(plus(M1, b!1))
[-2]  well_founded_part(M1)
[-3]  less(insert(x!1, b!1), a)
{-4}  x!1 < a IMPLIES
       (FORALL M:
          well_founded_part(M) IMPLIES well_founded_part(insert(x!1, M)))
  |-------
[1]   well_founded_part(insert(x!1, plus(M1, b!1)))

Rerunning step: (forward-chain "less_insert_cn")
Forward chaining on less_insert_cn,
this simplifies to:
well_founded_part_less_1_plus.2.1 :

{-1}  x!1 < a
[-2]  well_founded_part(plus(M1, b!1))
[-3]  well_founded_part(M1)
[-4]  less(insert(x!1, b!1), a)
[-5]  x!1 < a IMPLIES
```

```
       (FORALL M:
          well_founded_part(M) IMPLIES well_founded_part(insert(x!1, M)))
  |-------
[1]   well_founded_part(insert(x!1, plus(M1, b!1)))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
well_founded_part_less_1_plus.2.1 :

{-1}  FORALL M:
        well_founded_part(M) IMPLIES well_founded_part(insert(x!1, M))
[-2]  x!1 < a
[-3]  well_founded_part(plus(M1, b!1))
[-4]  well_founded_part(M1)
[-5]  less(insert(x!1, b!1), a)
  |-------
[1]   well_founded_part(insert(x!1, plus(M1, b!1)))

Rerunning step: (inst -1 "plus(M1, b!1)")
Instantiating the top quantifier in -1 with the terms:
 plus(M1, b!1),
this simplifies to:
well_founded_part_less_1_plus.2.1 :

{-1}  well_founded_part(plus(M1, b!1)) IMPLIES
       well_founded_part(insert(x!1, plus(M1, b!1)))
[-2]  x!1 < a
[-3]  well_founded_part(plus(M1, b!1))
[-4]  well_founded_part(M1)
[-5]  less(insert(x!1, b!1), a)
  |-------
[1]   well_founded_part(insert(x!1, plus(M1, b!1)))

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of well_founded_part_less_1_plus.2.1.

well_founded_part_less_1_plus.2.2 :

[-1]  well_founded_part(M1)
[-2]  less(insert(x!1, b!1), a)
[-3]  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   less(b!1, a)
[2]   well_founded_part(plus(M1, insert(x!1, b!1)))

Rerunning step: (forward-chain "less_insert_cn2")
```

```
Forward chaining on less_insert_cn2,

This completes the proof of well_founded_part_less_1_plus.2.2.


This completes the proof of well_founded_part_less_1_plus.2.

well_founded_part_less_1_plus.3 :

{-1}  well_founded_part(M1)
{-2}  less(K!1, a)
{-3}  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   FORALL (K):
        (FORALL b:
           b < a IMPLIES
            (FORALL M:
               well_founded_part(M) IMPLIES
                well_founded_part(insert(b, M))))
         AND less(K, a) AND well_founded_part(M1)
         IMPLIES is_finite[T](plus[T](M1, K))
{2}   well_founded_part(plus(M1, K!1))

Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
well_founded_part_less_1_plus.3 :

[-1]  well_founded_part(M1)
[-2]  less(K!1, a)
[-3]  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
[1]   FORALL (K):
        (FORALL b:
           b < a IMPLIES
            (FORALL M:
               well_founded_part(M) IMPLIES
                well_founded_part(insert(b, M))))
         AND less(K, a) AND well_founded_part(M1)
         IMPLIES is_finite[T](plus[T](M1, K))

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_plus.3 :
```

```
{-1}  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
{-2}  less(K!2, a)
{-3}  well_founded_part(M1)
[-4]  well_founded_part(M1)
[-5]  less(K!1, a)
[-6]  FORALL b:
        b < a IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   is_finite[T](plus[T](M1, K!2))

Rerunning step: (rewrite "finite_bag_plus")
Found matching substitution:
B: finite_bag[T] gets K!2,
A gets M1,
Rewriting using finite_bag_plus, matching in *,

This completes the proof of well_founded_part_less_1_plus.3.

Q.E.D.
well_founded_part_less_1_plus proved in 0.11 real, 0.11 cpu seconds
Rerunning proof of well_founded_part_less_1_insert_l0


well_founded_part_less_1_insert_l0 :

  |-------
{1}   FORALL (M_0: finite_bag[T], a: T):
        (well_founded_part(M_0) AND
          (FORALL M:
             less_1(M, M_0) IMPLIES
              (well_founded_part(M) IMPLIES
                well_founded_part(insert(a, M))))
           AND
           (FORALL b:
              b < a IMPLIES
               (FORALL M:
                  well_founded_part(M) IMPLIES
                   well_founded_part(insert(b, M)))))
          IMPLIES well_founded_part(insert(a, M_0))

Rerunning step: (apply (then (skosimp) (expand "well_founded_part" +)
                        (skosimp)))
Applying
   (then (skosimp) (expand "well_founded_part" +) (skosimp)),
this simplifies to:
well_founded_part_less_1_insert_l0 :
```

```
{-1}  well_founded_part(M!1)
{-2}  FORALL M:
        less_1(M, M!1) IMPLIES
         (well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M)))
{-3}  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
{-4}  less_1(y!1, insert(a!1, M!1))
  |-------
{1}   well_founded_part(y!1)

Rerunning step: (forward-chain "less_1_insert_cn")
Forward chaining on less_1_insert_cn,
this yields  2 subgoals:
well_founded_part_less_1_insert_l0.1 :

{-1}  EXISTS M_0: y!1 = insert(a!1, M_0) AND less_1(M_0, M!1)
[-2]  well_founded_part(M!1)
[-3]  FORALL M:
        less_1(M, M!1) IMPLIES
         (well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M)))
[-4]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
[-5]  less_1(y!1, insert(a!1, M!1))
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_insert_l0.1 :

{-1}  y!1 = insert(a!1, M!2)
{-2}  less_1(M!2, M!1)
[-3]  well_founded_part(M!1)
[-4]  FORALL M:
        less_1(M, M!1) IMPLIES
         (well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M)))
[-5]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
[-6]  less_1(y!1, insert(a!1, M!1))
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (inst -4 "M!2")
Instantiating the top quantifier in -4 with the terms:
 M!2,
this simplifies to:
well_founded_part_less_1_insert_l0.1 :

[-1]  y!1 = insert(a!1, M!2)
[-2]  less_1(M!2, M!1)
[-3]  well_founded_part(M!1)
{-4}  less_1(M!2, M!1) IMPLIES
       (well_founded_part(M!2) IMPLIES well_founded_part(insert(a!1, M!2)))
[-5]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
[-6]  less_1(y!1, insert(a!1, M!1))
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
well_founded_part_less_1_insert_l0.1.1 :

{-1}  well_founded_part(insert(a!1, M!2))
[-2]  y!1 = insert(a!1, M!2)
[-3]  less_1(M!2, M!1)
[-4]  well_founded_part(M!1)
[-5]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
[-6]  less_1(y!1, insert(a!1, M!1))
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (replace*)
Repeatedly applying the replace rule,

This completes the proof of well_founded_part_less_1_insert_l0.1.1.

well_founded_part_less_1_insert_l0.1.2 :

[-1]  y!1 = insert(a!1, M!2)
[-2]  less_1(M!2, M!1)
[-3]  well_founded_part(M!1)
[-4]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
[-5]  less_1(y!1, insert(a!1, M!1))
  |-------
{1}   well_founded_part(M!2)
[2]   well_founded_part(y!1)
```

```
Rerunning step: (apply (then (expand "well_founded_part" -3)
                        (inst -3 "M!2") (prop)))
Applying
   (then (expand "well_founded_part" -3) (inst -3 "M!2") (prop)),

This completes the proof of well_founded_part_less_1_insert_l0.1.2.


This completes the proof of well_founded_part_less_1_insert_l0.1.

well_founded_part_less_1_insert_l0.2 :

{-1}  EXISTS K: y!1 = plus(M!1, K) AND less(K, a!1)
[-2]  well_founded_part(M!1)
[-3]  FORALL M:
        less_1(M, M!1) IMPLIES
         (well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M)))
[-4]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
[-5]  less_1(y!1, insert(a!1, M!1))
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (grind-with-lemmas :defs nil :lemmas
                 ("well_founded_part_less_1_plus"))

Grinding away with the supplied lemmas,,

This completes the proof of well_founded_part_less_1_insert_l0.2.

Q.E.D.
well_founded_part_less_1_insert_l0 proved in 0.14 real, 0.14 cpu seconds
Rerunning proof of well_founded_part_less_1_insert_l2


well_founded_part_less_1_insert_l2 :

  |-------
{1}   FORALL (a: T):
        (FORALL b:
           b < a IMPLIES
            (FORALL M:
               well_founded_part(M) IMPLIES
                well_founded_part(insert(b, M))))
         IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(a, M)))

Rerunning step: (apply (then (skosimp)
                        (lemma "well_founded_part_weak_induction")
                        (inst -1
                         "lambda(M): well_founded_part(insert(a!1, M)) OR not(well_founded_part(M))")))
Applying
   (then (skosimp) (lemma "well_founded_part_weak_induction")
    (inst -1
     "lambda(M): well_founded_part(insert(a!1, M)) OR not(well_founded_part(M))")),
this simplifies to:
well_founded_part_less_1_insert_l2 :

{-1}  (FORALL (x: finite_bag[T]):
         (FORALL (y: finite_bag[T]):
            less_1(y, x) IMPLIES
             well_founded_part(insert(a!1, y)) OR
              NOT (well_founded_part(y)))
          IMPLIES
          well_founded_part(insert(a!1, x)) OR NOT (well_founded_part(x)))
       IMPLIES
       (FORALL (x: finite_bag[T]):
          well_founded_part(x) IMPLIES
           well_founded_part(insert(a!1, x)) OR NOT (well_founded_part(x)))
{-2}  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   FORALL M:
        well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
well_founded_part_less_1_insert_l2.1 :

{-1}  FORALL (x: finite_bag[T]):
        well_founded_part(x) IMPLIES
         well_founded_part(insert(a!1, x)) OR NOT (well_founded_part(x))
[-2]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
[1]   FORALL M:
        well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M))

Rerunning step: (grind :defs nil)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of well_founded_part_less_1_insert_l2.1.

well_founded_part_less_1_insert_l2.2 :

[-1]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   FORALL (x: finite_bag[T]):
        (FORALL (y: finite_bag[T]):
           less_1(y, x) IMPLIES
            well_founded_part(insert(a!1, y)) OR
             NOT (well_founded_part(y)))
         IMPLIES
         well_founded_part(insert(a!1, x)) OR NOT (well_founded_part(x))
[2]   FORALL M:
        well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M))

Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
well_founded_part_less_1_insert_l2.2 :

[-1]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
[1]   FORALL (x: finite_bag[T]):
        (FORALL (y: finite_bag[T]):
           less_1(y, x) IMPLIES
            well_founded_part(insert(a!1, y)) OR
             NOT (well_founded_part(y)))
         IMPLIES
         well_founded_part(insert(a!1, x)) OR NOT (well_founded_part(x))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_insert_l2.2 :

{-1}  FORALL (y: finite_bag[T]):
        less_1(y, x!1) IMPLIES
         well_founded_part(insert(a!1, y)) OR NOT (well_founded_part(y))
{-2}  (well_founded_part(x!1))
[-3]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   well_founded_part(insert(a!1, x!1))

Rerunning step: (rewrite "well_founded_part_less_1_insert_l0")
Found matching substitution:
M_0: finite_bag[T] gets x!1,
a: T gets a!1,
Rewriting using well_founded_part_less_1_insert_l0, matching in *,
this simplifies to:
well_founded_part_less_1_insert_l2.2 :

[-1]  FORALL (y: finite_bag[T]):
        less_1(y, x!1) IMPLIES
         well_founded_part(insert(a!1, y)) OR NOT (well_founded_part(y))
[-2]  (well_founded_part(x!1))
[-3]  FORALL b:
        b < a!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(b, M)))
  |-------
{1}   FORALL M:
        less_1(M, x!1) IMPLIES
         (well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M)))
[2]   well_founded_part(insert(a!1, x!1))

Rerunning step: (hide -3 2)
Hiding formulas:  -3, 2,
this simplifies to:
well_founded_part_less_1_insert_l2.2 :

[-1]  FORALL (y: finite_bag[T]):
        less_1(y, x!1) IMPLIES
         well_founded_part(insert(a!1, y)) OR NOT (well_founded_part(y))
[-2]  (well_founded_part(x!1))
  |-------
[1]   FORALL M:
        less_1(M, x!1) IMPLIES
         (well_founded_part(M) IMPLIES well_founded_part(insert(a!1, M)))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_insert_l2.2 :

{-1}  less_1(M!1, x!1)
{-2}  well_founded_part(M!1)
[-3]  FORALL (y: finite_bag[T]):
        less_1(y, x!1) IMPLIES
         well_founded_part(insert(a!1, y)) OR NOT (well_founded_part(y))
[-4]  (well_founded_part(x!1))
  |-------
{1}   well_founded_part(insert(a!1, M!1))

Rerunning step: (inst?)
Found substitution:
y: finite_bag[T] gets M!1,
Using template: well_founded_part(insert(a!1, y))
Instantiating quantified variables,
this simplifies to:
```

```
well_founded_part_less_1_insert_l2.2 :

[-1]  less_1(M!1, x!1)
[-2]  well_founded_part(M!1)
{-3}  less_1(M!1, x!1) IMPLIES
       well_founded_part(insert(a!1, M!1)) OR NOT (well_founded_part(M!1))
[-4]  (well_founded_part(x!1))
  |-------
[1]   well_founded_part(insert(a!1, M!1))

Rerunning step: (prop)
Applying propositional simplification,

This completes the proof of well_founded_part_less_1_insert_l2.2.

Q.E.D.
well_founded_part_less_1_insert_l2 proved in 0.11 real, 0.11 cpu seconds
Rerunning proof of well_founded_part_less_1_insert


well_founded_part_less_1_insert :

  |-------
{1}   FORALL M, a:
        well_founded_part(M) IMPLIES well_founded_part(insert(a, M))

Rerunning step: (induct "a" :name "wf_induction[T,<]")
Inducting on a on formula 1 using induction scheme wf_induction[T,<],
this simplifies to:
well_founded_part_less_1_insert :

  |-------
{1}   FORALL (x: T):
        (FORALL (y: T):
           y < x IMPLIES
            (FORALL M:
               well_founded_part(M) IMPLIES
                well_founded_part(insert(y, M))))
         IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(x, M)))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_insert :

{-1}  FORALL (y: T):
        y < x!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(y, M)))
  |-------
```

```
{1}   FORALL M:
        well_founded_part(M) IMPLIES well_founded_part(insert(x!1, M))

Rerunning step: (use "well_founded_part_less_1_insert_l2")
Using lemma well_founded_part_less_1_insert_l2,
this simplifies to:
well_founded_part_less_1_insert :

{-1}  (FORALL b:
         b < x!1 IMPLIES
          (FORALL M:
             well_founded_part(M) IMPLIES well_founded_part(insert(b, M))))
       IMPLIES
       (FORALL M:
          well_founded_part(M) IMPLIES well_founded_part(insert(x!1, M)))
[-2]  FORALL (y: T):
        y < x!1 IMPLIES
         (FORALL M:
            well_founded_part(M) IMPLIES well_founded_part(insert(y, M)))
  |-------
[1]   FORALL M:
        well_founded_part(M) IMPLIES well_founded_part(insert(x!1, M))

Rerunning step: (prop)
Applying propositional simplification,
Q.E.D.
well_founded_part_less_1_insert proved in 0.05 real, 0.05 cpu seconds
Rerunning proof of well_founded_part_less_1_all


well_founded_part_less_1_all :

  |-------
{1}   FORALL M: well_founded_part(M)

Rerunning step: (induct "M" :name "finite_bag_induction")
Inducting on M on formula 1 using induction scheme finite_bag_induction,
this yields  2 subgoals:
well_founded_part_less_1_all.1 :

  |-------
{1}   well_founded_part(emptybag[T])

Rerunning step: (expand "well_founded_part")
Expanding the definition of well_founded_part,
this simplifies to:
well_founded_part_less_1_all.1 :

  |-------
{1}   FORALL (y: finite_bag[T]):
        less_1(y, emptybag[T]) IMPLIES well_founded_part(y)
```

```
Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_all.1 :

{-1}  less_1(y!1, emptybag[T])
  |-------
{1}   well_founded_part(y!1)

Rerunning step: (expand "less_1")
Expanding the definition of less_1,
this simplifies to:
well_founded_part_less_1_all.1 :

{-1}  EXISTS M_0, a, K:
        emptybag[T] = insert(a, M_0) AND y!1 = plus(M_0, K) AND less(K, a)
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_all.1 :

{-1}  emptybag[T] = insert(a!1, M!1)
{-2}  y!1 = plus(M!1, K!1)
{-3}  less(K!1, a!1)
  |-------
[1]   well_founded_part(y!1)

Rerunning step: (hide -2 -3 1)
Hiding formulas:  -2, -3, 1,
this simplifies to:
well_founded_part_less_1_all.1 :

[-1]  emptybag[T] = insert(a!1, M!1)
  |-------

Rerunning step: (decompose-equality)
Applying decompose-equality,
this simplifies to:
well_founded_part_less_1_all.1 :

{-1}  FORALL (x: T): emptybag[T](x) = insert(a!1, M!1)(x)
  |-------

Rerunning step: (grind :defs nil :rewrites ("emptybag" "insert"))
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of well_founded_part_less_1_all.1.

well_founded_part_less_1_all.2 :
  |-------
{1}   FORALL (x: T), (b: finite_bag[T]):
        well_founded_part(b) IMPLIES well_founded_part(insert(x, b))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
well_founded_part_less_1_all.2 :

{-1}  well_founded_part(b!1)
  |-------
{1}   well_founded_part(insert(x!1, b!1))

Rerunning step: (rewrite "well_founded_part_less_1_insert")
Found matching substitution:
M gets b!1,
a gets x!1,
Rewriting using well_founded_part_less_1_insert, matching in *,

This completes the proof of well_founded_part_less_1_all.2.

Q.E.D.
well_founded_part_less_1_all proved in 0.15 real, 0.15 cpu seconds
Rerunning proof of wf_less_1


wf_less_1 :

  |-------
{1}   well_founded?[finite_bag[T]](less_1)

Rerunning step: (use "well_founded_part_less_1_all")
Using lemma well_founded_part_less_1_all,
this simplifies to:
wf_less_1 :

{-1}  FORALL (M): well_founded_part(M)
  |-------
[1]   well_founded?[finite_bag[T]](less_1)

Rerunning step: (rewrite "well_founded_part_cns")
Found matching substitution:
Rewriting using well_founded_part_cns, matching in *,
Q.E.D.
wf_less_1 proved in 0.01 real, 0.01 cpu seconds
Rerunning proof of wf_less_bag


wf_less_bag :

  |-------
```

```
{1}   well_founded?[finite_bag[T]](less_bag)

Rerunning step: (grind :defs nil :rewrites
                 ("less_bag" "well_founded_cl_tr" "wf_less_1"))
Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.
wf_less_bag proved in 0.01 real, 0.01 cpu seconds
Rerunning proof of less_bag_cn


less_bag_cn :

  |-------
{1}   FORALL (M, N: finite_bag[T]):
        less_bag(N, M) IMPLIES
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (expand "less_bag")
Expanding the definition of less_bag,
this simplifies to:
less_bag_cn :

  |-------
{1}   FORALL (M, N: finite_bag[T]):
        tr_cl(less_1)(N, M) IMPLIES
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (lemma "tr_cl_weak_induction[finite_bag[T]]")
Applying tr_cl_weak_induction[finite_bag[T]]
this simplifies to:
less_bag_cn :

{-1}  FORALL (<: pred[[finite_bag[T], finite_bag[T]]], x,
              P: [finite_bag[T] -> boolean]):
        (FORALL (y: finite_bag[T]):
           (x < y OR (EXISTS (z: finite_bag[T]): P(z) AND z < y)) IMPLIES
            P(y))
         IMPLIES (FORALL (y: finite_bag[T]): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
[1]   FORALL (M, N: finite_bag[T]):
        tr_cl(less_1)(N, M) IMPLIES
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
```

```
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (skolem 1 ("_" "N"))
For the top quantifier in 1, we introduce Skolem constants: (_ N),
this simplifies to:
less_bag_cn :

[-1]  FORALL (<: pred[[finite_bag[T], finite_bag[T]]], x,
              P: [finite_bag[T] -> boolean]):
        (FORALL (y: finite_bag[T]):
           (x < y OR (EXISTS (z: finite_bag[T]): P(z) AND z < y)) IMPLIES
            P(y))
         IMPLIES (FORALL (y: finite_bag[T]): tr_cl(<)(x, y) IMPLIES P(y))
  |-------
{1}   FORALL (M):
        tr_cl(less_1)(N, M) IMPLIES
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (inst - "less_1" "N" "lambda(M):EXISTS M_0, K1, K2:
              M = plus(M_0, K1) AND
               N = plus(M_0, K2) AND
                (FORALL a:
                   member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))")
Instantiating the top quantifier in - with the terms:
 less_1, N, lambda(M):EXISTS M_0, K1, K2:
              M = plus(M_0, K1) AND
               N = plus(M_0, K2) AND
                (FORALL a:
                   member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)),
this simplifies to:
less_bag_cn :

{-1}  (FORALL (y: finite_bag[T]):
          (less_1(N, y) OR
            (EXISTS (z: finite_bag[T]):
               (EXISTS M_0, K1, K2:
                  z = plus(M_0, K1) AND
                   N = plus(M_0, K2) AND
                    (FORALL a:
                       member(a, K2) IMPLIES
                        (EXISTS b: member(b, K1) AND a < b)))
                AND less_1(z, y)))
           IMPLIES
           (EXISTS M_0, K1, K2:
              y = plus(M_0, K1) AND
               N = plus(M_0, K2) AND
                (FORALL a:
                  member(a, K2) IMPLIES
                   (EXISTS b: member(b, K1) AND a < b))))
       IMPLIES
       (FORALL (y: finite_bag[T]):
          tr_cl(less_1)(N, y) IMPLIES
           (EXISTS M_0, K1, K2:
              y = plus(M_0, K1) AND
               N = plus(M_0, K2) AND
                (FORALL a:
                   member(a, K2) IMPLIES
                    (EXISTS b: member(b, K1) AND a < b))))
  |-------
[1]   FORALL (M):
        tr_cl(less_1)(N, M) IMPLIES
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
less_bag_cn :

  |-------
{1}   FORALL (y: finite_bag[T]):
        (less_1(N, y) OR
          (EXISTS (z: finite_bag[T]):
             (EXISTS M_0, K1, K2:
                z = plus(M_0, K1) AND
                 N = plus(M_0, K2) AND
                  (FORALL a:
                     member(a, K2) IMPLIES
                      (EXISTS b: member(b, K1) AND a < b)))
              AND less_1(z, y)))
         IMPLIES
         (EXISTS M_0, K1, K2:
            y = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
[2]   FORALL (M):
        tr_cl(less_1)(N, M) IMPLIES
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (hide 2)
Hiding formulas:  2,
this simplifies to:
less_bag_cn :

  |-------
[1]   FORALL (y: finite_bag[T]):
        (less_1(N, y) OR
          (EXISTS (z: finite_bag[T]):
             (EXISTS M_0, K1, K2:
                z = plus(M_0, K1) AND
                 N = plus(M_0, K2) AND
                  (FORALL a:
                     member(a, K2) IMPLIES
                      (EXISTS b: member(b, K1) AND a < b)))
              AND less_1(z, y)))
         IMPLIES
         (EXISTS M_0, K1, K2:
            y = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
less_bag_cn :

{-1}  less_1(N, y!1) OR
       (EXISTS (z: finite_bag[T]):
          (EXISTS M_0, K1, K2:
             z = plus(M_0, K1) AND
              N = plus(M_0, K2) AND
               (FORALL a:
                  member(a, K2) IMPLIES
                   (EXISTS b: member(b, K1) AND a < b)))
           AND less_1(z, y!1))
  |-------
{1}   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
less_bag_cn.1 :

{-1}  less_1(N, y!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
```

```
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (expand "less_1")
Expanding the definition of less_1,
this simplifies to:
less_bag_cn.1 :

{-1}  EXISTS M_0, a, K:
        y!1 = insert(a, M_0) AND N = plus(M_0, K) AND less(K, a)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_bag_cn.1 :

{-1}  y!1 = insert(a!1, M!1)
{-2}  N = plus(M!1, K!1)
{-3}  less(K!1, a!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (inst + "M!1" "singleton_bag(a!1)" "K!1")
Instantiating the top quantifier in + with the terms:
 M!1, singleton_bag(a!1), K!1,
this yields  2 subgoals:
less_bag_cn.1.1 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   y!1 = plus(M!1, singleton_bag(a!1)) AND
       N = plus(M!1, K!1) AND
        (FORALL a:
           member(a, K!1) IMPLIES
            (EXISTS b: member(b, singleton_bag(a!1)) AND a < b))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
less_bag_cn.1.1.1 :
```

```
[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   y!1 = plus(M!1, singleton_bag(a!1))

Rerunning step: (rewrite "bag_equality")
Found matching substitution:
b: bag[T] gets plus(M!1, singleton_bag(a!1)),
a gets y!1,
Rewriting using bag_equality, matching in *,
this simplifies to:
less_bag_cn.1.1.1 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   FORALL (x: T): eqmult(x, y!1, plus(M!1, singleton_bag(a!1)))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_bag_cn.1.1.1 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   eqmult(x!1, y!1, plus(M!1, singleton_bag(a!1)))

Rerunning step: (expand "eqmult")
Expanding the definition of eqmult,
this simplifies to:
less_bag_cn.1.1.1 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   (y!1(x!1) = plus(M!1, singleton_bag(a!1))(x!1))

Rerunning step: (expand "singleton_bag")
Expanding the definition of singleton_bag,
this simplifies to:
less_bag_cn.1.1.1 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
```

```
{1}   (y!1(x!1) =
        plus(M!1, LAMBDA (x: T): IF x = a!1 THEN 1 ELSE 0 ENDIF)(x!1))

Rerunning step: (expand "plus")
Expanding the definition of plus,
this simplifies to:
less_bag_cn.1.1.1 :

[-1]  y!1 = insert(a!1, M!1)
{-2}  N = (LAMBDA (t: T): K!1(t) + M!1(t))
[-3]  less(K!1, a!1)
  |-------
{1}   (y!1(x!1) = M!1(x!1) + IF x!1 = a!1 THEN 1 ELSE 0 ENDIF)

Rerunning step: (lift-if)
Lifting IF-conditions to the top level,
this simplifies to:
less_bag_cn.1.1.1 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = (LAMBDA (t: T): K!1(t) + M!1(t))
[-3]  less(K!1, a!1)
  |-------
{1}   IF x!1 = a!1 THEN (y!1(x!1) = M!1(x!1) + 1)
      ELSE (y!1(x!1) = M!1(x!1) + 0)
      ENDIF

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
less_bag_cn.1.1.1.1 :

{-1}  x!1 = a!1
[-2]  y!1 = insert(a!1, M!1)
[-3]  N = (LAMBDA (t: T): K!1(t) + M!1(t))
[-4]  less(K!1, a!1)
  |-------
{1}   (y!1(x!1) = M!1(x!1) + 1)

Rerunning step: (expand "insert")
Expanding the definition of insert,
this simplifies to:
less_bag_cn.1.1.1.1 :

[-1]  x!1 = a!1
{-2}  y!1 = (LAMBDA (t: T): IF a!1 = t THEN 1 + M!1(t) ELSE M!1(t) ENDIF)
[-3]  N = (LAMBDA (t: T): K!1(t) + M!1(t))
[-4]  less(K!1, a!1)
  |-------
[1]   (y!1(x!1) = M!1(x!1) + 1)

Rerunning step: (grind)
```

```
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of less_bag_cn.1.1.1.1.

less_bag_cn.1.1.1.2 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = (LAMBDA (t: T): K!1(t) + M!1(t))
[-3]  less(K!1, a!1)
  |-------
{1}   x!1 = a!1
{2}   (y!1(x!1) = M!1(x!1) + 0)

Rerunning step: (grind)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of less_bag_cn.1.1.1.2.


This completes the proof of less_bag_cn.1.1.1.

less_bag_cn.1.1.2 :

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   FORALL a:
        member(a, K!1) IMPLIES
         (EXISTS b: member(b, singleton_bag(a!1)) AND a < b)

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_bag_cn.1.1.2 :

{-1}  member(a!2, K!1)
[-2]  y!1 = insert(a!1, M!1)
[-3]  N = plus(M!1, K!1)
[-4]  less(K!1, a!1)
  |-------
{1}   EXISTS b: member(b, singleton_bag(a!1)) AND a!2 < b

Rerunning step: (inst + "a!1")
Instantiating the top quantifier in + with the terms:
 a!1,
this simplifies to:
less_bag_cn.1.1.2 :

[-1]  member(a!2, K!1)
[-2]  y!1 = insert(a!1, M!1)
[-3]  N = plus(M!1, K!1)
```

```
[-4]  less(K!1, a!1)
  |-------
{1}   member(a!1, singleton_bag(a!1)) AND a!2 < a!1

Rerunning step: (grind)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of less_bag_cn.1.1.2.


This completes the proof of less_bag_cn.1.1.

less_bag_cn.1.2 (TCC):

[-1]  y!1 = insert(a!1, M!1)
[-2]  N = plus(M!1, K!1)
[-3]  less(K!1, a!1)
  |-------
{1}   is_finite[T](singleton_bag[T](a!1))

Rerunning step: (rewrite "finite_singleton_bag")
Found matching substitution:
t: T gets a!1,
Rewriting using finite_singleton_bag, matching in *,

This completes the proof of less_bag_cn.1.2.


This completes the proof of less_bag_cn.1.

less_bag_cn.2 :

{-1}  EXISTS (z: finite_bag[T]):
        (EXISTS M_0, K1, K2:
           z = plus(M_0, K1) AND
            N = plus(M_0, K2) AND
             (FORALL a:
                member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
         AND less_1(z, y!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
less_bag_cn.2 :

{-1}  z!1 = plus(M!1, K1!1)
```

```
{-2}  N = plus(M!1, K2!1)
{-3}  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
{-4}  less_1(z!1, y!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (expand "less_1")
Expanding the definition of less_1,
this simplifies to:
less_bag_cn.2 :

[-1]  z!1 = plus(M!1, K1!1)
[-2]  N = plus(M!1, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
{-4}  EXISTS M_0, a, K:
        y!1 = insert(a, M_0) AND z!1 = plus(M_0, K) AND less(K, a)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_bag_cn.2 :

[-1]  z!1 = plus(M!1, K1!1)
[-2]  N = plus(M!1, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
{-4}  y!1 = insert(a!1, M!2)
{-5}  z!1 = plus(M!2, K!1)
{-6}  less(K!1, a!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Rerunning step: (postpone)
Postponing less_bag_cn.2.

less_bag_cn.2 :
```

```
[-1]  z!1 = plus(M!1, K1!1)
[-2]  N = plus(M!1, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
{-4}  y!1 = insert(a!1, M!2)
{-5}  z!1 = plus(M!2, K!1)
{-6}  less(K!1, a!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))

Postponing less_bag_cn.2.

less_bag_cn.2 :

[-1]  z!1 = plus(M!1, K1!1)
[-2]  N = plus(M!1, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
{-4}  y!1 = insert(a!1, M!2)
{-5}  z!1 = plus(M!2, K!1)
{-6}  less(K!1, a!1)
  |-------
[1]   EXISTS M_0, K1, K2:
        y!1 = plus(M_0, K1) AND
         N = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))
less_bag_cn unproved in 0.40 real, 0.40 cpu seconds
Rerunning proof of less_bag_cs_l1


less_bag_cs_l1 :

  |-------
{1}   FORALL (K1, K2, M, M_0, N: finite_bag[T]):
        (M = plus(M_0, K1) AND
          N = plus(M_0, K2) AND
           (FORALL a:
              member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
         IMPLIES less_bag(N, M)

Rerunning step: (induct "K1" :name "finite_bag_induction")
Inducting on K1 on formula 1 using induction scheme finite_bag_induction,
this yields  2 subgoals:
less_bag_cs_l1.1 :

  |-------
```

```
{1}   FORALL (K2, M, M_0, N: finite_bag[T]):
        (M = plus(M_0, emptybag[T]) AND
          N = plus(M_0, K2) AND
           (FORALL a:
              member(a, K2) IMPLIES
               (EXISTS b: member(b, emptybag[T]) AND a < b)))
         IMPLIES less_bag(N, M)

Rerunning step: (postpone)
Postponing less_bag_cs_l1.1.

less_bag_cs_l1.2 :

  |-------
{1}   FORALL (x: T), (b_1: finite_bag[T]):
        (FORALL (K2, M, M_0, N: finite_bag[T]):
           (M = plus(M_0, b_1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES
                  (EXISTS b: member(b, b_1) AND a < b)))
            IMPLIES less_bag(N, M))
         IMPLIES
         (FORALL (K2, M, M_0, N: finite_bag[T]):
            (M = plus(M_0, insert(x, b_1)) AND
              N = plus(M_0, K2) AND
               (FORALL a:
                  member(a, K2) IMPLIES
                   (EXISTS b: member(b, insert(x, b_1)) AND a < b)))
             IMPLIES less_bag(N, M))

Rerunning step: (postpone)
Postponing less_bag_cs_l1.2.

less_bag_cs_l1.1 :

  |-------
{1}   FORALL (K2, M, M_0, N: finite_bag[T]):
        (M = plus(M_0, emptybag[T]) AND
          N = plus(M_0, K2) AND
           (FORALL a:
              member(a, K2) IMPLIES
               (EXISTS b: member(b, emptybag[T]) AND a < b)))
         IMPLIES less_bag(N, M)

Postponing less_bag_cs_l1.1.

less_bag_cs_l1.2 :

  |-------
{1}   FORALL (x: T), (b_1: finite_bag[T]):
        (FORALL (K2, M, M_0, N: finite_bag[T]):
           (M = plus(M_0, b_1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES
                  (EXISTS b: member(b, b_1) AND a < b)))
            IMPLIES less_bag(N, M))
         IMPLIES
         (FORALL (K2, M, M_0, N: finite_bag[T]):
            (M = plus(M_0, insert(x, b_1)) AND
              N = plus(M_0, K2) AND
               (FORALL a:
                  member(a, K2) IMPLIES
                   (EXISTS b: member(b, insert(x, b_1)) AND a < b)))
             IMPLIES less_bag(N, M))

Postponing less_bag_cs_l1.2.

less_bag_cs_l1.1 :

  |-------
{1}   FORALL (K2, M, M_0, N: finite_bag[T]):
        (M = plus(M_0, emptybag[T]) AND
          N = plus(M_0, K2) AND
           (FORALL a:
              member(a, K2) IMPLIES
               (EXISTS b: member(b, emptybag[T]) AND a < b)))
         IMPLIES less_bag(N, M)
less_bag_cs_l1 unproved in 0.06 real, 0.06 cpu seconds
Rerunning proof of less_bag_cs


less_bag_cs :

  |-------
{1}   FORALL (M, N: finite_bag[T]):
        (EXISTS M_0, K1, K2:
           M = plus(M_0, K1) AND
            N = plus(M_0, K2) AND
             (FORALL a:
                member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))
         IMPLIES less_bag(N, M)

Rerunning step: (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
less_bag_cs :

{-1}  M!1 = plus(M!2, K1!1)
{-2}  N!1 = plus(M!2, K2!1)
{-3}  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
  |-------
```

```
{1}   less_bag(N!1, M!1)

Rerunning step: (expand "less_bag")
Expanding the definition of less_bag,
this simplifies to:
less_bag_cs :

[-1]  M!1 = plus(M!2, K1!1)
[-2]  N!1 = plus(M!2, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
  |-------
{1}   tr_cl(less_1)(N!1, M!1)

Rerunning step: (expand "tr_cl")
Expanding the definition of tr_cl,
this simplifies to:
less_bag_cs :

[-1]  M!1 = plus(M!2, K1!1)
[-2]  N!1 = plus(M!2, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
  |-------
{1}   less_1(N!1, M!1) OR
       (EXISTS (z: finite_bag[T]): tr_cl(less_1)(N!1, z) AND less_1(z, M!1))

Rerunning step: (prop)
Applying propositional simplification,
this simplifies to:
less_bag_cs :

[-1]  M!1 = plus(M!2, K1!1)
[-2]  N!1 = plus(M!2, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
  |-------
{1}   less_1(N!1, M!1)
{2}   EXISTS (z: finite_bag[T]): tr_cl(less_1)(N!1, z) AND less_1(z, M!1)

Rerunning step: (postpone)
Postponing less_bag_cs.

less_bag_cs :

[-1]  M!1 = plus(M!2, K1!1)
[-2]  N!1 = plus(M!2, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
  |-------
{1}   less_1(N!1, M!1)
{2}   EXISTS (z: finite_bag[T]): tr_cl(less_1)(N!1, z) AND less_1(z, M!1)
```

```
Postponing less_bag_cs.

less_bag_cs :

[-1]  M!1 = plus(M!2, K1!1)
[-2]  N!1 = plus(M!2, K2!1)
[-3]  FORALL a:
        member(a, K2!1) IMPLIES (EXISTS b: member(b, K1!1) AND a < b)
  |-------
{1}   less_1(N!1, M!1)
{2}   EXISTS (z: finite_bag[T]): tr_cl(less_1)(N!1, z) AND less_1(z, M!1)
less_bag_cs unproved in 0.04 real, 0.04 cpu seconds
Rerunning proof of less_bag_caract


less_bag_caract :

  |-------
{1}   FORALL (M, N: finite_bag[T]):
        less_bag(N, M) IFF
         (EXISTS M_0, K1, K2:
            M = plus(M_0, K1) AND
             N = plus(M_0, K2) AND
              (FORALL a:
                 member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (skosimp)
Skolemizing and flattening,
this simplifies to:
less_bag_caract :

  |-------
{1}   less_bag(N!1, M!1) IFF
       (EXISTS M_0, K1, K2:
          M!1 = plus(M_0, K1) AND
           N!1 = plus(M_0, K2) AND
            (FORALL a:
               member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b)))

Rerunning step: (prop)
Applying propositional simplification,
this yields  2 subgoals:
less_bag_caract.1 :

{-1}  less_bag(N!1, M!1)
  |-------
{1}   EXISTS M_0, K1, K2:
        M!1 = plus(M_0, K1) AND
         N!1 = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))
```

```
Rerunning step: (rewrite "less_bag_cn")
Found matching substitution:
N: finite_bag[T] gets N!1,
M gets M!1,
Rewriting using less_bag_cn, matching in *,

This completes the proof of less_bag_caract.1.

less_bag_caract.2 :

{-1}  EXISTS M_0, K1, K2:
        M!1 = plus(M_0, K1) AND
         N!1 = plus(M_0, K2) AND
          (FORALL a:
             member(a, K2) IMPLIES (EXISTS b: member(b, K1) AND a < b))
  |-------
{1}   less_bag(N!1, M!1)

Rerunning step: (rewrite "less_bag_cs")
Found matching substitution:
M gets M!1,
N: finite_bag[T] gets N!1,
Rewriting using less_bag_cs, matching in *,

This completes the proof of less_bag_caract.2.

Q.E.D.
less_bag_caract proved in 0.02 real, 0.02 cpu seconds
finite_bags_order: 19 proofs attempted, 16 proved in 2.39 real, 2.39 cpu seconds


 Proof summary for theory finite_bags_order
    insert_TCC1..........................proved - incomplete [shostak](0.02 s)
    less_1_insert_cn.....................proved - incomplete [shostak](0.69 s)
    plus_emptybag........................proved - incomplete [shostak](0.04 s)
    insert_plus..........................proved - incomplete [shostak](0.23 s)
    less_insert_cn.......................proved - incomplete [shostak](0.12 s)
    less_insert_cn2......................proved - incomplete [shostak](0.17 s)
    IMP_wf_cs_TCC1.......................proved - incomplete [shostak](0.01 s)
    well_founded_part_less_1_plus_TCC1....proved - incomplete [shostak](0.01 s)
    well_founded_part_less_1_plus........proved - incomplete [shostak](0.11 s)
    well_founded_part_less_1_insert_l0....proved - incomplete [shostak](0.14 s)
    well_founded_part_less_1_insert_l2....proved - incomplete [shostak](0.11 s)
    well_founded_part_less_1_insert......proved - incomplete [shostak](0.05 s)
    well_founded_part_less_1_all.........proved - incomplete [shostak](0.15 s)
    wf_less_1............................proved - incomplete [shostak](0.01 s)
    wf_less_bag..........................proved - incomplete [shostak](0.01 s)
    less_bag_cn..........................unfinished          [shostak](0.40 s)
    less_bag_cs_l1.......................unfinished          [shostak](0.06 s)
    less_bag_cs..........................unfinished          [shostak](0.04 s)
    less_bag_caract......................proved - incomplete [shostak](0.02 s)
    Theory totals: 19 formulas, 19 attempted, 16 succeeded (2.39 s)

Grand Totals: 19 proofs, 19 attempted, 16 succeeded (2.39 s)
```

Bibliografía

[1] P. Aczel. *Handbook of mathematical logic*, chapter An introduction to inductive definitions. North–Holland, 1977.

[2] T. Nipkow. An inductive proof of the wellfoundedness of the multiset order.

[3] P. Rudnicki and A. Trybulec. On equivalents of well–foundedness (an experiment in mizar. *Journal of Automated Reasoning*, 1992.