Trenner für Paare

$$d_{\sigma\times\tau} : ((\sigma \times \tau) \times \iota)^* \to \iota^{**}$$

- *Fall* $\sigma \times \tau$:

  In der Eingabe $[(v_1, w_1), \dots, (v_n, w_n)]$ sei $v_i = (v_{i,1}, v_{i,2})$.

  (i) Für $i = 1, \dots, n$ speichere die zweite Komponente $v_{i,2}$ des Werts unter einer frischen Adresse $\overline{v_{i,2}}$, bilde das Paar $(\overline{v_{i,2}}, w_i) : \iota \times \iota$ und speichere diesen Wert in einer ebenfalls neuen Adresse $\overline{(\overline{v_{i,2}}, w_i)} : \iota$.

  (ii) Mit dem induktiv schon definierten Trenner d_σ trenne

  $$\begin{aligned} & d_\sigma([(v_{1,1}, \overline{(\overline{v_{1,2}}, w_1)}), \dots, (v_{n,1}, \overline{(\overline{v_{n,2}}, w_n)})] \\ = \quad & [[\overline{(\overline{v_{1,2}}, w_1)}, \dots], \dots, [\overline{(\overline{v_{k,2}}, w_k)}, \dots]] \end{aligned}$$

  nach gleichen ersten Komponenten $v_{1,i}$ der v_i.

  (iii) Ersetze die Adressen durch die dort gespeicherten Werte, und trenne jede Liste mit dem Trenner d_τ:

  $$[d_\tau([(v_{1,2}, w_1), \dots]),\ \dots,\ d_\tau([(v_{k,2}, w_k), \dots])]$$

  (iv) Verkette die Ergebnislisten: das ergibt eine Trennung der w_j nach gleichen Wertpaaren $(v_{i,1}, v_{i,2})$,

  $$[[w_{1,1} \dots, w_{1,k_1}], \dots, [w_{r,1}, \dots, w_{r,k_r}]].$$

  Die Speicherung in neuen Adressen ist nur wegen der Wahl des Typs ι notwendig.

Trennung nach $\sim_N$-äquivalenten Adressen

$$d_{ref(\tau)} : (ref(\tau) \times \iota)^* \to \iota^{**}$$

Die Eingabeliste $[(n_1, w_1), \ldots, (n_m, w_m)]$ soll bezgl. einer Äquivalenz $\sim_N$ auf dem Speicher N getrennt werden:

$$[[w_{1,1}, \ldots w_{1,k_1}], \ldots, [w_{r,1}, \ldots, w_{r,k_r}]]$$

wobei die $w_{i,j}$ die Informationen aller $n \sim_N n_i$ seien.

Stelle den Speicher $S = (N, E)$ und die Äquivalenz $\sim_N$ durch *disrimination records* $r = \{\texttt{cont} : \texttt{U ref}, \texttt{eq} : \texttt{N ref}\}$ dar:

Jeder Adresse n entspricht ein $r = (E(n), n')$, wobei $n' \in N$ die Klasse $[n]_{\sim_N}$ darstellt, d.h. es ist

$$n_1 \sim_N n_2 \iff n_1' = n_2'$$

(aber $E(n')$ muß nicht eine Liste der Elemente von $[n]_{\sim_N}$ sein).

Die Äquivalenz zweier Adressen n_1, n_2 kann dann in $O(1)$ Schritten festgestellt werden.

Änderungen von E und $\sim_N$ werden durch Änderungen bestehender $r = (E(n), n')$ erreicht, und Änderungen an N durch Allozierung neuer r oder Löschung vorhandener.

Die Äquivalenz $\sim_N$ sei also unter n mit übergeben, d.h. $!n = (E(n), n')$, oBdA mit $n' : ref(\iota^*)$.

Reduziere Trennung modulo $\sim_N$ auf Trennung modulo $=_N$:

$$d_{ref(\tau)}([(n_1, w_1), \ldots]) := d_{ref(\iota^*)}([(n_1', w_1), \ldots)$$

Trennung nach gleichen Adressen

$$d_{ref(\iota^*)} : (ref(\iota^*) \times \iota)^* \to \iota^{**}$$

- *Fall ref*(ι^*): Zur Eingabeliste $[(n_1, w_1), \ldots, (n_m, w_m)]$ sollen die w_i nach $=_N$-gleichen Adressen n_i getrennt werden, unabhängig von den gespeicherten Werten.

  (i) Wähle frische Adresse $n_{\texttt{eq}} : ref(\iota^*)$, setze $n_{\texttt{eq}} := []$ und $n_i := []$ für $i = 1, \ldots, m$.

  (ii) Für $i = 1, \ldots, m$ setze

  $$n_{\texttt{eq}} := \begin{cases} n_i :: (!n_{\texttt{eq}}), & \text{falls } !n_i = [], \\ n_{\texttt{eq}}, & \text{sonst} \end{cases}$$

  und anschließend (für ordnungserhaltendes $d_{\texttt{ref}\ \iota^*}$)

  $$n_i := !n_i @ [w_i]$$

  Danach steht unter $n_{\texttt{eq}}$ eine Anordnung der *Menge* $\{n_1, \ldots, n_m\}$, mit jedem n_i genau einmal, und bei n_i die Liste $[w_{i,1}, \ldots, w_{i,k_i}]$ aller $w_{i,j}$, für die $(n_i, w_{i,j})$ in der Eingabeliste vorkommt.

  (iii) Gib für jedes n auf der Liste $!n_{\texttt{eq}}$ die Liste $!n$ aus.

  Achtung:

  Die anfangs unter n_i gespeicherten Werte sind verloren.

  Das ist harmlos, falls die n_i *nur* als Darstellung von $\sim_N$-Klassen benutzt werden: dazu braucht man nur Gleichheiten $n_i =_N n_j$, und die sind von den $E(n_i)$ unabhängig.

  Voraussetung: unter Adressen ι kann man Informationen vom Typ ι^* speichern, also $ref(\iota^*) \subseteq \iota$.

Trenner nach gleichen Adressen

$$d_{ref(\tau)} : (ref(\tau) \times \iota)^* \to \iota^{**}$$

Fall ref(τ): Die Eingabe $[(n_1, w_1), \ldots, (n_m, w_m)]$ soll nach $=_N$-gleichen Adressen getrennt werden, wobei $n_i : ref(\tau)$.[2]

Mit einem universellen Typ U hat, kann man das auf den Fall $ref(\iota^* \times \tau)$ zurückführen:

(i) Ändere die gespeicherten Werte zu $n_i := ([], !n_i)$ für $i = 1, \ldots, m$.

(ii) Trenne $[(n_1, w_1), \ldots, (n_m, w_m)]$ wie im Fall $ref(\iota^*)$, ohne die zweiten Komponenten zu verändern.

(iii) Setze die ursprünglichen Werte mit $n_i := second(!n_i)$ wieder her, für $i = 1, \ldots, m$.

(iv) Gib das Ergebnis von (ii) aus.

Da die Gleichheit der Adressen beibehalten bleibt, erhält man die korrekte Antwort. Damit im ersten Schritt kein Typfehler auftritt, muß man $ref(\tau)$ und $ref(\iota^* \times \tau)$ als Spezialfälle von $\iota = ref(U)$ behandeln können.

---

[2] Es ist zu teuer, zuerst die Gleichheiten unter den n_i zu bestimmen; das geht auch nicht mit

$$\begin{aligned} & d_{\texttt{set}(ref(\tau))}([([n_1], (n_1, w_1)), \ldots, ([n_m], (n_m, w_m))]) \\ = \quad & [[(n_{j_1}, w_{j_1}), \ldots, (n_{j_{k_1}}, w_{j_{k_1}})], \ldots, [(n_{j_r}, w_{j_r}), \ldots, (n_{j_{k_r}}, w_{j_{k_r}})]], \end{aligned}$$

denn dabei würde die schwache Sortierung der Verkettung von Einermengen $[(n_i, \widetilde{w}_i)]$ benötigt, die rekursiv die Trennung nach den n_i erfordert.

Trennung nach gleichen Adressen

⟨*Discriminator for references*⟩≡

```
structure RefDiscr =
  struct
    type discrRef = {content: U ref, eq: class ref} ref

    fun eqClass (v:discrRef) = #eq (!v)
    fun isEmpty (v:discrRef) = (!(eqClass v) = [])
    fun add (v:discrRef) (w:info) =
        let val c = (eqClass v) in c := (!c)@[w] end
    fun getThenResetClass (v:discrRef) =
        let val els = !(eqClass v)
        in (eqClass v) := [] ; els end

    val eqClasses = ref [] : discrRef list ref

    fun addClass (v:discrRef,w:info) =
        (if isEmpty v then
             eqClasses := v::(!eqClasses)
         else ();
         (add v w); ())

    fun discriminate (vws : (discrRef * info) list) =
        let val _ = List.app addClass vws
            val classes =
                map (fn v => getThenResetClass v)
                    (!eqClasses)
        in
            eqClasses := [];  classes
        end
  end
```

Trenner für Listen

$$d_{\tau^*} : (\tau^* \times \iota)^* \to \iota^{**}$$

Fall $\tau^* = \mu\alpha(1 + (\tau \times \alpha))$ (analog: andere Datentypen $\mu\alpha\,\tau$):

(i) Zerlege $[(v_1, w_1), \ldots, (v_n, w_n)]$ in die Paare mit $v_i : 1$ und die mit $v_i : \tau \times \tau^*$, d.h. die mit leerer Liste bzw. nicht-leerer Liste v_i.

(ii) Alle w_i mit $v_i : 1$ bilden eine Klasse A.

(iii) Trenne die Teilliste der übrigen (v_i, w_i) mit $d_{\tau\times\tau^*}$ in

$$\begin{aligned} & d_{\tau\times\tau^*}([(v_1, w_1), \ldots, (v_r, w_r)]) \\ & \quad = \quad [[w_{1,1}, \ldots, w_{1,k_1}], \ldots, [w_{r,1}, \ldots, w_{r,k_r}]] \end{aligned}$$

und füge A hinzu, falls A nicht leer war.

Das ist keine Induktion über die Typausdrücke, terminiert aber, da bei der rekursiven Verwendung von d_{τ^*} nur kürzere Listen in den Eingabetupeln auftreten.

Im allgemeinen Fall muß man für $(\mu\alpha.\tau)(\vec{\beta})$ eine von einer Belegung $[\vec{\beta}/d_{\vec{\beta}}]$ abhängige Trennfunktion rekursiv definieren,

$$d_{\mu\alpha.\tau}[\vec{\beta}/d_{\vec{\beta}}](\vec{v}) := (\mu d.d_\tau[\vec{\beta}/d_{\vec{\beta}}, \alpha/d])(\vec{v}).$$

Trennung modulo $f : \sigma \to \tau$

Trennung einer Eingabe $[(v_1, w_1), \ldots, (v_n, w_n)]$ nicht nach gleichen Werten $v_i =_\sigma v_j$, sondern nach f-gleichen Werten,

$$v_i =^f_\sigma v_j :\iff f(v_i) =_\tau f(v_j).$$

$$\begin{aligned} & d^f_\sigma([(v_1, w_1), \ldots, (v_n, w_n)]) \\ := \quad & d_\tau([(f(v_1), w_1), \ldots, (f(v_n), w_n)]). \end{aligned}$$

Trennung nach *bag*-Gleichheit

$d_{\mathtt{bag}(\tau)} : (\tau^* \times \iota)^* \to \iota^{**}$

Unter $[(\vec{v}_1, w_1), \ldots, (\vec{v}_n, w_n)]$ soll man diejenigen w_i in eine Liste zusammenfassen, die in einem $(\vec{v}_j, w_j)$ mit $\vec{v}_j \simeq_{\mathtt{bag}(\tau)} \vec{v}_i$ als w_j vorkommen.

1.Idee: Zuerst aus $\vec{v}$ eine neue Liste $\mathtt{bag}(\vec{v}) : \tau^*$ konstruieren, in der alle gleichen Elemente aus $\vec{v}$ nebeneinander stehen, und dann $[(\mathtt{bag}(\vec{v}_1), w_1), \ldots, (\mathtt{bag}(\vec{v}_n), w_n)]$ mit d_{τ^*} trennen.

Aber die Konstruktion von $\mathtt{bag}(\vec{v})$ erfordert schon zu viele Vergleiche, und man muß die entstandenen Listen noch nach einer Ordnung auf τ sortieren, damit die Trennung die für $=_{\mathtt{bag}(\tau)}$ richtigen Ergebnisse liefert.

2.Effiziente Lösung (B.Paige): durch *schwache Sortierung* (erfordert keine Ordnung auf τ!)

(i) Defininiere eine Funktion $\mathit{weaksort} : \tau^{**} \to \tau^{**}$, die jede Liste $\vec{v}_i$ der Eingabe $[\vec{v}_1, \ldots, \vec{v}_n]$ in die Liste $\vec{v}_i'$ umordnet, bei der die Elemente wie ihre ersten Vorkommen in der Verkettung von $[\vec{v}_1, \ldots, \vec{v}_n]$ vorkommen.

(ii) Berechne aus $[(\vec{v}_1, w_1), \ldots, (\vec{v}_n, w_n)]$ zuerst $[\vec{v}_1', \ldots, \vec{v}_n'] = \mathit{weaksort}([\vec{v}_1, \ldots, \vec{v}_n])$ und wende den Trenner d_{τ^*} auf $[(\vec{v}_1', w_1), \ldots, (\vec{v}_n', w_n)]$ an. Das gibt das korrekte Ergebnis, da $\vec{v}_i' =_{\tau^*} \vec{v}_j'$ genau dann gilt, wenn $\vec{v}_i =_{\mathtt{bag}(\tau)} \vec{v}_j$, weil die Komponenten in $\vec{v}_i', \vec{v}_j'$ in schwacher Sortierung auftreten.

Schwache Sortierung von $[\vec{v}_1, \ldots, \vec{v}_n]$
$weaksort : \tau^{**} \to \tau^{**}$

(i) Speichere für spätere Trennung jedes $\vec{v}_i = [c_{i,1}, \ldots, c_{i,k_i}]$ in einer frischen Adresse $\overline{v_i}$ und dann für jedes j das Paar $(\overline{v_i}, c_{i,j})$ in einer frischen Adresse $w_{i,j} := \overline{(\overline{v_i}, c_{i,j})}$. Bilde für jedes i die Liste

$$\tilde{v}_i := [(c_{i,1}, (\overline{v}_i, w_{i,1})), \ldots, (c_{i,k_i}, (\overline{v_i}, w_{i,k_i}))].$$

(ii) Verkette die Ergebnisse $[\tilde{v}_1, \ldots, \tilde{v}_n]$ zu

$$[(c_{1,1}, (\overline{v_1}, w_{1,1})), \ldots, (c_{n,k_n}, (\overline{v_n}, w_{n,k_n}))]$$

und trenne diese Liste mit d_τ nach den Elementen $c_{i,j}$ (entsprechend einer bestimmten Anordnung $<_{[\vec{v}_1,\ldots,\vec{v}_n]}$ der Elemente $c_{i,j}$). Die Ergebnisliste

$$[[(\overline{v_i}, w_{i,j_1}), \ldots,], \ldots, [\ldots]]$$

faßt alle $(\overline{v_i}, w_{i,j})$ in ein Element zusammen, deren Liste v_i ein bestimmtes Element c aus der Verkettung der Argumentlisten $[\vec{v}_1, \ldots, \vec{v}_n]$ enthält.

(iii) Trenne die Verkettung dieser Ergebnisse (ordnungserhaltend) mit $d_{ref(\tau^*)}$ bezüglich der Gleichheit $=_N$ als Äquivalenz $\sim_N$: das Ergebnis

$$d_{ref(\tau^*)}([(\overline{v}_i, w_{i,j_1}), \ldots]) = [[w_{i,j_1}, \ldots, w_{i,j_{k_i}}], \ldots, [\ldots]]$$

enthält in den zweiten Komponenten der $[w_{i,j_1}, \ldots, w_{i,j_{k_i}}]$ genau die Elemente $c_{i,j}$ aus $\vec{v}_i$ in der Ordung $<_{[\vec{v}_1,\ldots,\vec{v}_n]}$ aufgeführt, in den ersten Komponenten jeweils $\overline{v_i}$.

Daraus kann eine Zuordnung $\overline{v_i} \mapsto \vec{v}'_i := \texttt{bag}(\vec{v}_i)$ bestimmt werden, mit der die Eingabe $[\vec{v}_1, \ldots, \vec{v}_n]$ in $[\vec{v}'_1, \ldots, \vec{v}'_n]$ umgewandelt werden kann (in $O(n)$, wenn man sich in a) die Liste $[\overline{v_1}, \ldots, \overline{v_n}]$ merkt.)