15.

Időszerűsítés

15.1. Az időszerűsítés definíciója

Induljunk ki egy olyan adatfile-ból amelyben azonos típusú elemek találhatóak. Ezt a file-t *törzsfile*-nak fogjuk nevezni. Legyen az elemek típusa E, ekkor

$T = seq(E)$

Legyen adott továbbá egy olyan file, amely transzformációk sorozatát tartalmazza. Ezt a file-t fogjuk *módosítófile*-nak nevezni. Legyen $F = \{f \mid f : T \to T$ leképezés $\}$, ekkor

$M = seq(F)$

A feladat az, hogy időszerűsítsük a törzsfile-t a módosítófile-ban leírt transzformációkkal. Jelölje upd az időszerűsítés transzformációt. Ekkor $upd : T \times M \to T$ és

$$upd(t, m) = m_{dom(m)} \circ \cdots \circ m_2 \circ m_1(t)$$

Ahhoz, hogy a feladatra megoldást adjunk ez a leírás még mindig túl általános, ezért további kikötéseket teszünk a file-okra.

1. *Az időszerűsítés kulcsos*

   Legyen $E = (k : K, d : D)$ és $F = (k : K, v : V)$, ahol K egy tetszőleges rendezett halmaz, a rekordok kulcsrésze; D a törzsrekord adatrésze; V pedig az elvégzendő transzformációt definiáló típus. Feltesszük továbbá, hogy mind a tözsfile, mind a módosítófile a kulcsmező (k) szerint rendezett, azaz a T és M típusok invariáns tulajdonságára:

$$\begin{aligned} I_T(t) &\Rightarrow \forall i \in [1..dom(t) - 1] : t_i.k \le t_{i+1}.k \\ I_M(m) &\Rightarrow \forall i \in [1..dom(m) - 1] : m_i.k \le m_{i+1}.k \end{aligned}$$

2. *A törzsfile a kulcsmező szerint egyértelmű*

$$I_T(t) \Rightarrow \forall i, j \in [1..dom(t)], i \neq j : t_i.k \neq t_j.k$$

3. *A transzformáció csak az alábbi három féle lehet*

$$\begin{aligned} V &= (t : W_1; b : W_2; j : W_3) \\ W_1 &= \{\alpha\} \\ W_2 &= (d : D) \\ W_3 &= (g : G), \quad \text{ahol } G = \{\gamma \mid \gamma : D \to D\} \end{aligned}$$

Ahol a jelölés nem rontja el a szelektorfüggvények egyértelműségét, ott az egymás után következő szelektorfüggvény-sorozatból a közbülső elemek – a jelölést egyszerűsítendő – kihagyhatók, ezért pl. ha $dm : F$, akkor $dm.v.g$ helyett csak $dm.g$-t írunk.

Hátra van még annak leírása, hogy hogyan hatnak a fenti transzformációk a törzsfile-ra. Kényelmi okokból a transzformációk eredményét halmazként adjuk meg (ez elegendő, mert a törzsfile kulcs szerint egyértelmű). Ehhez bevezetünk néhány jelölést: legyen $x : seq(k : K, y : Y)$, ahol K rendezett halmaz, Y pedig tetszőleges típus, valamint $k \in K$. Ekkor

$$\begin{aligned} \{x\} &= \{x_i \mid i \in [1..dom(x)]\} \\ \{x.k\} &= \{x_i.k \mid i \in [1..dom(x)]\} \\ \mathcal{K}(x, k) &= \{x_i \mid i \in [1..dom(x)] \wedge x_i.k = k\} \end{aligned}$$

$$\{m_i(t)\} = \begin{cases} \mathcal{T}(m_i, t), & \text{ha } m_i.t \\ \mathcal{B}(m_i, t), & \text{ha } m_i.b \\ \mathcal{J}(m_i, t), & \text{ha } m_i.j \end{cases}$$

ahol

$$\begin{aligned} \mathcal{T}(m_i, t) &= \begin{cases} \{t_j \mid t_j \in t \wedge t_j.k \neq m_i.k\}, & \text{ha } m_i.k \in \{t.k\} \\ \{t\}, & \text{különben} \end{cases} \\ \mathcal{B}(m_i, t) &= \begin{cases} \{t\} \cup \{(m_i.k, m_i.v.d)\}, & \text{ha } m_i.k \notin \{t.k\} \\ \{t\}, & \text{különben} \end{cases} \\ \mathcal{J}(m_i, t) &= \begin{cases} \{t_j \mid t_j \in t \wedge t_j.k \neq m_i.k\} \cup & \\ \{(m_i.k, m_i.g(\mathcal{K}(t, m_i.k)))\}, & \text{ha } m_i.k \in \{t.k\} \\ \{t\}, & \text{különben} \end{cases} \end{aligned}$$

4. *A javítás művelet csere*

$$I_M(m) \Rightarrow \forall i \in [1..dom(m)], m_i.v.j \Rightarrow m_i.g \text{ konstans}$$

Ebben az esetben a W_3 típust úgy szoktuk felírni, hogy a $g : G$ mező helyére $d : D$-t írunk és a $m_i.g(\mathcal{K}(t, k))$ függvényalkalkalmazást a módosítórekord adatrészére ($m_i.d$) cseréljük.

A feladat specifikációja tehát:

$$A = \underset{t_0}{T} \times \underset{m}{M} \times \underset{t}{T}$$

$$B = \underset{t_0'}{T} \times \underset{m'}{M}$$

$Q : (t_0 = t_0' \land m = m'$ és az $1..x$ pontok teljesülnek)

$R : (t = upd(t_0', m'))$

Ha a fenti specifikációban $x = 3$, akkor *közönséges*, ha $x = 4$ akkor *egyszerű* időszerűsítésről beszélünk.

15.2. Időszerűsítés egyértelmű módosítófile-lal

Mivel a közönséges és az egyszerű időszerűsítés között tulajdonképpen csak a javítás elvégzésében van különbség, mi a továbbiakban az egyszerű időszerűsítéssel fogunk foglalkozni. Közönséges időszerűsítés esetén az adatrész cseréjének helyére a javító függvény $(m_i.g)$ elvégzése írható.

A feladatot három irányból is megpróbáljuk megoldani: visszavezetjük halmazok uniójára, egyváltozós egyértékű illetve kétváltozós egyértékű elemenkénti feldolgozásra.

15.2.1. Visszavezetés halmazok uniójára

Tekintsük a file-okat halmaznak, és próbáljuk meg halmazokon megoldani a feladatot. Milyen kulcsértékek fordulhatnak elő az új törzsfile-ban? Természetesen csak olyanok, amelyek vagy t_0-ban, vagy m-ben, esetleg mindkettőben szerepeltek.

Terjesszük ki a D halmazt az üres értékkel: $D' = D \cup \{<\!\ddot{u}res\!>\}$ A D' segítségével az egyes transzformációk értelmezési tartománya és értékkészlete az alábbiak szerint alakul:

$$\begin{array}{ll} W_1 : & D \to \{<\!\ddot{u}res\!>\} \\ W_2 : & \{<\!\ddot{u}res\!>\} \to D \\ W_3 : & D \to D \end{array}$$

Ha egy elem adatrésze az $<\!\ddot{u}res\!>$ értéket veszi fel, akkor az azt jelzi, hogy valójában nem létezik, és ezért nem kerül bele az új törzsfile-ba. Idézzük fel az *unió* programját:

$z := x \cup y$

$z := \emptyset$		
$x \neq \emptyset \lor y \neq \emptyset$		
$e :\in (x \cup y)$		
$e \in x \land e \notin y$	$e \in x \land e \in y$	$e \notin x \land e \in y$
$z := z \,\tilde{\cup}\, e$	$z := z \,\tilde{\cup}\, e$	$z := z \,\tilde{\cup}\, e$
$x := x \simeq e$	$x := x \simeq e$	$y := y \simeq e$
	$y := y \simeq e$	

A feladat a következő megfeleltetéssel visszavezethető a fenti programra:

$$
\begin{array}{rcl}
x & \rightarrow & t_0 \\
y & \rightarrow & m \\
z & \rightarrow & t \\
x \neq \emptyset \lor y \neq \emptyset & \rightarrow & t_0 \neq \emptyset \lor m \neq \emptyset \\
e :\in (x \cup y) & \rightarrow & k :\in (\{t_0.k\} \cup \{m.k\}) \\
e \in x & \rightarrow & k \in \{t_0.k\} \\
e \in y & \rightarrow & k \in \{m.k\}
\end{array}
$$

A megfelelő program:

$t := \emptyset$		
$t_0 \neq \emptyset \lor m \neq \emptyset$		
$k :\in (\{t_0.k\} \cup \{m.k\})$		
$k \in \{t_0.k\} \land k \notin \{m.k\}$	$k \in \{t_0.k\} \land k \in \{m.k\}$	$k \notin \{t_0.k\} \land k \in \{m.k\}$
S_1	S_2	S_3

Vizsgáljuk meg most, hogy mit kell tenni az elágazás egyes ágaiban:

1. Ha a k kulcs csak a t_0 eredeti törzsfile-ban szerepelt, akkor a $\mathcal{K}(t_0, k)$ elemre nem vonatkozott módosítás, változtatás nélkül kell kiírni az új törzsfile-ba.

S_1

$t := t \,\tilde{\cup}\, \mathcal{K}(t_0, k)$
$t_0 := t_0 \simeq \mathcal{K}(t_0, k)$

2. Ha a k kulcsérték mind az eredeti törzsfile-ban, mind pedig a módosítófile-ban szerepelt, akkor a törlés és a javítás műveletek végezhetők el. Ebben az ágban a k kulcsú elemet mindkét file-ból el kell hagyni.

S_2

$\mathcal{K}(m,k).t$	$\mathcal{K}(m,k).b$	$\mathcal{K}(m,k).j$
$SKIP$	$HIBA$	$t := t \,\tilde{\cup}\, (k, \mathcal{K}(m,k).d)$
$m := m \simeq \mathcal{K}(m,k)$		
$t_0 := t_0 \simeq \mathcal{K}(t_0,k)$		

3. Ha a k kulcs csak a módosítófile-ban szerepelt, akkor csak a beszúrás műveletet lehet elvégezni, és a k kulcsú elemet ki kell törölni a módosítófile-ból.

S_3

$\mathcal{K}(m,k).t$	$\mathcal{K}(m,k).b$	$\mathcal{K}(m,k).j$
$HIBA$	$t := t \,\tilde{\cup}\, (k, \mathcal{K}(m,k).d)$	$HIBA$
$m := m \simeq \mathcal{K}(m,k)$		

Ha a programnak hibajelzést is kell adnia, akkor azt a fenti struktogramokban $HIBA$-val jelzett helyeken kell megtennie.

Térjünk most vissza az eredeti feladatra, ahol halmazok helyett szekvenciális file-ok szerepelnek. Legyen az inputfile-okon a $read$ művelet értelmezve. Ekkor a program az alábbiak szerint alakul:

$st_0, dt_0, t_0 : read; sm, dm, m : read$		
$t := \langle\rangle$		
$st = norm \vee sm = norm$		
$(st_0 = sm \wedge dt_0.k < dm.k)$ $\vee sm = abnorm$	$st_0 = sm \wedge$ $dt_0.k = dm.k$	$(st_0 = sm \wedge dt_0.k > dm.k)$ $\vee st_0 = abnorm$
$t : hiext(dt_0)$ $st_0, dt_0, t_0 : read$	S_2^*	S_3^*

ahol

S_2^*

$dm.t$	$dm.b$	$dm.j$
$SKIP$	$HIBA$	$t : hiext(dm.k, dm.d)$
$st_0, dt_0, t_0 : read$		
$sm, dm, m : read$		

S_3^*

$dm.t$	$dm.b$	$dm.j$
$HIBA$	$t : hiext(dm.k, dm.d)$	$HIBA$
$sm, dm, m : read$		

15.2.2. Visszavezetés egyváltozós elemenkénti feldolgozásra

Az időszerűsítés állapottere felfogható úgy is, mint a törzsfile és a módosítófile összefésülése – tegyük fel, hogy egy file-unk van az alábbi szerkezettel:

kulcsérték	t_0-ból vagy m-ből
adatrész	t_0-ból vagy $<\ddot{u}res>$
transzformáció rész	m-ből vagy $<\ddot{u}res>$

Természetesen az adatrész és a transzformáció rész mindegyike nem lehet $<\ddot{u}res>$. Formálisan:

$$\begin{aligned} X &= seq(DX) \\ DX &= (k : K, d : D', v : V') \\ D' &= D \cup \{<\ddot{u}res>\} \\ V' &= V \cup \{<\ddot{u}res>\} \end{aligned}$$

Legyen $x \in X$. Ekkor $\{x.k\} = \{t_0.k\} \cup \{m.k\}$. Jelölje dx az x egy tetszőleges elemét. Ekkor:

$$dx.d = \begin{cases} <\ddot{u}res>, & \text{ha } dx.k \notin \{t_0.k\} \\ \mathcal{K}(t_0, dx.k).d, & \text{ha } dx.k \in \{t_0.k\} \end{cases}$$

$$dx.v = \begin{cases} <\ddot{u}res>, & \text{ha } dx.k \notin \{m.k\} \\ \mathcal{K}(m, dx.k).v, & \text{ha } dx.k \in \{m.k\} \end{cases}$$

Az állapottértranszformációt alkalmazva így eljutottunk az

$$f(\{e\}) = \{(e.k, e.v(e.d))\}$$

elemenként feldolgozható függvényre felírt egyváltozós elemenkénti feldolgozáshoz, ahol a transzformációs rész alkalmazása az adatrészre:

$$e.v(e.d) = \begin{cases} e.d, & \text{ha } e.v = <\ddot{u}res> \\ <\ddot{u}res> & \text{ha } e.v.t \wedge e.d \neq <\ddot{u}res> \\ e.v.d & \text{ha } e.v.b \wedge e.d = <\ddot{u}res> \\ e.v.d & \text{ha } e.v.j \wedge e.d \neq <\ddot{u}res> \end{cases}$$

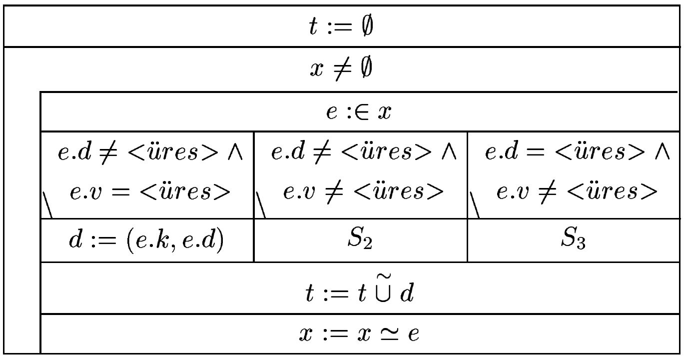
A specifikáció:

$$\begin{array}{ccccc} A = & X & \times & T \\ & x & & t \end{array}$$

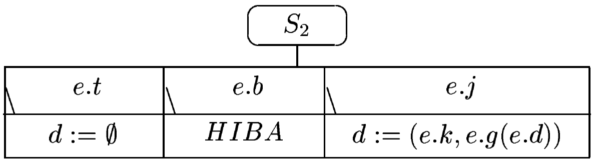
$$B = \underset{x'}{X}$$

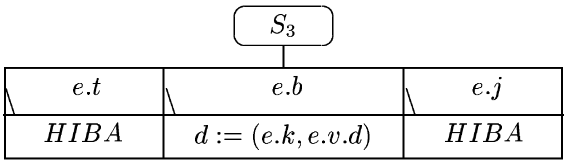
$Q : (x = x'$ és az 1..4 pontok teljesülnek$)$
$R : (t = f(x'))$

A halmazokra felírt megoldóprogram:



ahol





Térjünk vissza most az eredeti állapottérre:

$x \neq \emptyset$	$\Rightarrow$	$t_0 \neq \emptyset \vee m \neq \emptyset$
$e :\in x$	$\Rightarrow$	$k :\in (\{t_0.k\} \cup \{m.k\})$
$e.d \neq <\text{üres}> \wedge e.v = <\text{üres}>$	$\Rightarrow$	$k \in \{t_0.k\} \wedge k \notin \{m.k\}$
$e.d = <\text{üres}> \wedge e.v \neq <\text{üres}>$	$\Rightarrow$	$k \notin \{t_0.k\} \wedge k \in \{m.k\}$
$e.d \neq <\text{üres}> \wedge e.v \neq <\text{üres}>$	$\Rightarrow$	$k \in \{t_0.k\} \wedge k \in \{m.k\}$

$x := x \simeq e$ $\Rightarrow$

$k \in \{t_0.k\} \wedge k \notin \{m.k\}$	$k \in \{t_0.k\} \wedge k \in \{m.k\}$	$k \notin \{t_0.k\} \wedge k \in \{m.k\}$
$t_0 := t_0 \simeq \mathcal{K}(t_0, k)$	$t_0 := t_0 \simeq \mathcal{K}(t_0, k)$ $m := m \simeq \mathcal{K}(m, k)$	$m := m \simeq \mathcal{K}(m, k)$

Használjuk fel azt a tényt, hogy a $d := f(\{e\})$ értékadást kiszámító programokban és az $x := x \simeq e$ értékadás megfelelőjében szereplő elágazások feltételrendszere megegyezik, továbbá a $t := t \,\tilde{\cup}\, d$ értékadást csak azokba az ágakba írjuk bele, amelyekben $d \neq \emptyset$. Ekkor ugyanazt a programot kapjuk, mint az első megoldásban.

15.2.3. Visszavezetés kétváltozós elemenkénti feldolgozásra

A feladat megoldásának talán legegyszerűbb módja az, ha kétváltozós egyértékű – hibakezelés esetén kétértékű – elemenkénti feldolgozásra vezetjük vissza. Tekintsük a feladat eredeti specifikációját.

Ha a módosítófile kulcs szerint egyértelmű, akkor az időszerűsítés függvénye (upd) a kulcsokra nézve elemenként feldolgozható. A kulcsértékekre felírt függvény:

$$
\begin{aligned}
upd(k, \emptyset) &= \mathcal{K}(t_0, k) \\
upd(\emptyset, k) &= \begin{cases} \emptyset, & \text{ha } \mathcal{K}(m,k).t \\ (k, \mathcal{K}(m,k).d), & \text{ha } \mathcal{K}(m,k).b \\ \emptyset, & \text{ha } \mathcal{K}(m,k).j \end{cases} \\
upd(k, k) &= \begin{cases} \emptyset, & \text{ha } \mathcal{K}(m,k).t \\ \mathcal{K}(t_0, k), & \text{ha } \mathcal{K}(m,k).b \\ (k, \mathcal{K}(m.k).g(\mathcal{K}(t_0,k).d)), & \text{ha } \mathcal{K}(m,k).j \end{cases}
\end{aligned}
$$

Ha ezt a fenti függvényt behelyettesítjük a kétváltozós elemenkénti feldolgozás tételébe, akkor ugyanahhoz a megoldóprogramhoz jutunk – csak sokkal rövidebb úton –, mint az első megoldásban.

15.3. Időszerűsítés nem egyértelmű módosítófilelal

Vajon miben változik a feladat, ha a módosítófile kulcs szerint nem egyértelmű? Ebben az esetben a feladat nem elemenként feldolgozható. Erre a problémára kétféle megoldási módot is megvizsgálunk: az *adatabsztrakciós* és a *függvényabsztrakciós* megközelítést.

15.3.1. Megoldás adatabsztrakcióval

Mint az előbb már megállapítást nyert, ha a módosító file kulcs szerint nem egyértelmű, akkor a feladat nem elemenként feldolgozható. Hát akkor tegyük azzá! Ehhez arra van szükség, hogy a módosító file-t kulcs szerint egyértelművé tegyük. Ezt egy állapottér-transzformáció segítségével könnyen megtehetjük, ugyanis csak annyit kell tennünk, hogy az azonos kulcsú módosító rekordokat egy új rekordba fogjuk össze. így az új módosító rekord a következőképpen fog kinézni:

(kulcs, transzformációsorozat)

Azaz definiáljuk az új módosítófile típusát az alábbi módon:

Legyen $W = seq(V)$; és $F' = (k : K, v : W)$. Ezekkel a típusdefiníciókkal a módosítófile így írható le:

$$X = seq(F');$$

Ezzel a módosítófile-lal tehát eljutottunk a kétváltozós egyértékű (hibafile használata esetén kétértékű) elemenkénti feldolgozáshoz. Az egyetlen különbség csak az, hogy az adott transzformáció-sorozat végrehajtását meg kell valósítanunk az eredeti állapottéren. Ehhez szükségünk lesz az $<\ddot{u}res>$ szibólumra, amely értéket hozzávesszük az adatrész típusához: $D' = D \cup \{<\ddot{u}res>\}$. Az, hogy egy törzsrekord adatrésze $<\ddot{u}res>$ azt jelenti, hogy a rekordot nem kell kiírni az eredményfile-ba.

Az elemenként feldolgozható függvényünk:

$$\begin{aligned}
upd(k, \emptyset) &= \mathcal{K}(t_0, k) \\
upd(\emptyset, k) &= (k, \mathcal{K}(x, k).v(<\ddot{u}res>)) \\
upd(k, k) &= (k, \mathcal{K}(x, k).v(\mathcal{K}(t_0, k).d))
\end{aligned}$$

A transzformációsorozat elvégzése csak a transzformációk megfelelő sorrendje esetén lehetséges. Ha egy transzformációsorozat egy tagját nem lehet elvégezni, akkor azt a sorozatból ki kell hagyni (esetleg hibát kell jelezni). Ezzel az *upd* függvény egyértelműen definiált.

Írjuk fel tehát a fenti megfontolások alapján a kétváltozós elemenkénti feldolgozás programját a megfelelő behelyettesítésekkel:

<table>
<tr><td colspan="4">$t := 0$</td></tr>
<tr><td colspan="4">$st_0, dt_0, t_0 : read$</td></tr>
<tr><td colspan="4">$sx, dx, x : read$</td></tr>
<tr><td colspan="4">$st_0 = norm \vee sx = norm$</td></tr>
<tr><td rowspan="2"></td><td>$sx = abnorm \vee (sx = st_0 \wedge dt_0.k < dx.k)$</td><td>$sx = st_0 \wedge dx.k = dt_0.k$</td><td>$st_0 = abnorm \vee (sx = st_0 \wedge dx.k < dt_0.k)$</td></tr>
<tr><td>$t : hiext(dt_0)$
$st_0, dt_0, t_0 : read$</td><td>$ak := dx.k$
$ad := dx.v(dt_0.d)$
$t : HIEXT(ad, ak)$
$st_0, dt_0, t_0 : read$
$sx, dx, x : read$</td><td>$ak := dx.k$
$ad := dx.v(<\ddot{u}res>)$
$t : HIEXT(ad, ak)$
$sx, dx, x : read$</td></tr>
</table>

Definiálnunk kell még, hogy mit jelent a fenti struktogramban a transzformációsorozat elvégzése. Ehhez bevezetjük az $f : \mathbb{N}_0 \to D'$ rekurzívan definiált függvényt:

$$dx.v(p) = f(dx.v.dom)$$

ahol

$$\begin{aligned} f(0) &= p \\ f(i+1) &= dx.v_{i+1}(f(i)) \end{aligned}$$

A fenti definícióban szereplő $dx.v_{i+1}$ művelet elvégzésén az alábbi függvényértékeket értjük: Legyen $d \in D'$, ekkor:

$$dx.v_{i+1}(d) = \begin{cases} <\ddot{u}res>, & \text{ha } dx.v.t \wedge d \neq <\ddot{u}res> \\ d, & \text{ha } dx.v.t \wedge d = <\ddot{u}res> \\ d, & \text{ha } dx.v.b \wedge d \neq <\ddot{u}res> \\ dx.v.d, & \text{ha } dx.v.b \wedge d = <\ddot{u}res> \\ dx.v.d, & \text{ha } dx.v.j \wedge d \neq <\ddot{u}res> \\ d, & \text{ha } dx.v.j \wedge d = <\ddot{u}res> \end{cases}$$

Az f függvényt kiszámító – a módosítássorozatot elvégző – program:

$ad := dx.v(p)$

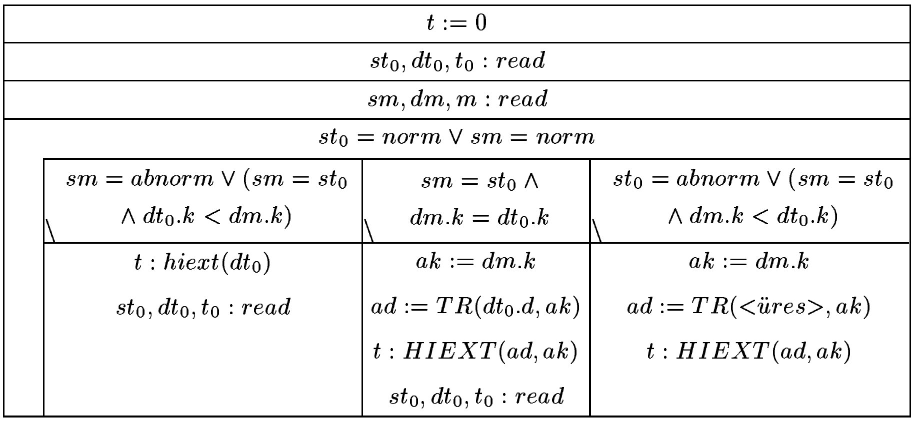
<table>
<tr><td colspan="7">$d := p$</td></tr>
<tr><td colspan="7">$dx.v.dom \neq 0$</td></tr>
<tr><td rowspan="4"></td><td colspan="2">$dx.v.lov.t$</td><td colspan="2">$dx.v.lov.b$</td><td colspan="2">$dx.v.lov.j$</td></tr>
<tr><td colspan="2">$ad = <\ddot{u}res>$</td><td colspan="2">$ad = <\ddot{u}res>$</td><td colspan="2">$ad = <\ddot{u}res>$</td></tr>
<tr><td>$HIBA$</td><td>$ad := <\ddot{u}res>$</td><td>$ad := dx.v.lov.d$</td><td>$HIBA$</td><td>$HIBA$</td><td>$ad := dx.v.lov.d$</td></tr>
<tr><td colspan="6">$dx.v : lorem$</td></tr>
</table>

Mivel az aktuális adat értéke lehet $<\ddot{u}res>$ is, a $hiext$ művelet helyett az alábbi programot használjuk:

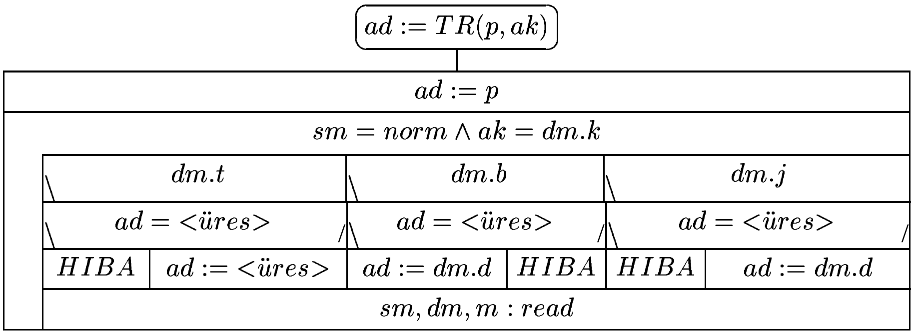
$t : HIEXT(ak, ad)$

<table>
<tr><td colspan="2">$ad = <\ddot{u}res>$</td></tr>
<tr><td>$t : hiext((ak, ad))$</td><td>$SKIP$</td></tr>
</table>

Térjünk most vissza az eredeti állapottérre. Ekkor a főprogram:



Az $ad := TR(p, ak)$ a már korábban definiált rekurzív függvényt kiszámító program megvalósítása az eredeti állapottéren:



15.3.2. Kulcsok egyértelműsítése

A gyakorlatban sokszor találkozhatunk olyan file-okkal, amelyek rekordjaiban van valamilyen kulcsmező ami szerint a file rendezett, ám de a file mégsem egyértelmű kulcs szerint, és így a file a kulcsmezőre vonatkoztatva nem elemenként feldolgozható. A következőkben egy olyan technikát fogunk bemutatni, amelyben egy új kulcsot definiálunk a file-ra, és ezen új kulcs szerint a file már egyértelmű.

Tegyük fel, hogy a file $U = (k : K, z : Z)$ típusú rekordokból áll. Az új kulcsot úgy kapjuk, hogy az egymás után levő azonos kulcsokat megsorszámozzuk. Legyen tehát $V = (h : H, z : Z)$, ahol $H = (k : K, s : \mathbb{N})$. Legyen továbbá $g : seq(U) \to seq(V)$:

i) $g(u).dom = u.dom$

ii) $g(u)_1.s = 1$ és $\forall i \in [1..u.dom - 1]$:

$$g(u)_{i+1}.s = \begin{cases} 1, & \text{ha } u_i.k \neq u_{i+1}.k \\ g(u)_i.s + 1, & \text{ha } u_i.k = u_{i+1}.k \end{cases}$$

iii) $\forall i \in [1..u.dom]$:

$$g(u)_i.k = u_i.k \text{ és } g(u)_i.z = u_i.z.$$

Ekkor tetszőleges $u \in seq(U)$ esetén – feltéve, hogy u a k kulcs szerint rendezett – $g(u)$ a h kulcs szerint rendezett és egyértelmű.

Természetesen a fenti megszámozást általában csak az absztrakció leírására használjuk, és csak ritkán fordul elő, hogy a g függvény által definiált absztrakciót meg is valósítjuk.

15.3.3. Megoldás függvényabsztrakcióval

Egy adott feladat megoldását mindig elkerülhetetlenül befolyásolja a specifikáció módja. A függvényabsztrakció lényege abban rejlik, hogy a megoldást egy alkalmasan választott függvény helyettesítési értékének kiszámítására vezetjük vissza.

Induljunk ki egy olyan absztrakt file-ból, mint amilyet az egyváltozós elemenkénti feldolgozásra való visszavezetésben használtunk. Természetesen, mivel most a módosítófile nem egyértelmű kulcs szerint, az X absztrakt file definíciója kissé módosul: ha egy kulcs mindkét file-ban szerepel, akkor a törzsrekordot az első rá vonatkozó módosítórekorddal vonjuk össze, és az esetlegesen előforduló további azonos kulcsú módosítórekordokból pedig egy-egy olyan absztrakt rekordot képezünk, amelynek adatrésze $<\ddot{u}res>$.

Ez az absztrakció az imént bemutatott egyértelműsítő leképezésen keresztül implicit módon írható le: legyen $t_0 \in T$, $m \in M$ és $x \in X$. Ekkor

$$\{g(x).h\} = \{g(t_0).h\} \cup \{g(m).h\}$$

és $\forall i \in [1..x.dom]$:

$$g(x)_i.d = \begin{cases} \mathcal{K}(g(t_0), g(x)_i.h).d, & \text{ha } g(x)_i.h \in \{g(t_0).h\} \\ <\ddot{u}res>, & \text{különben} \end{cases}$$

$$g(x)_i.v = \begin{cases} \mathcal{K}(g(m), g(x)_i.h).v, & \text{ha } g(x)_i.h \in \{g(m).h\} \\ <\ddot{u}res>, & \text{különben} \end{cases}$$

Megoldás függvénykompozícióval

Először egy olyan megoldást adunk a feladatra, amelynek specifikációjában az utófeltétel egy kompozícióval adott függvény kiszámítása, ahol a kompozíció egy rekurzív módon megadott függvényből és egy esetszétválasztással definiált függvényből áll.

$$A = \underset{x}{X} \times \underset{t}{T}$$

$$B = \underset{x'}{X}$$

$$Q : (x = x')$$

$$R : (t = HIEXT(f_1(x'.dom), (f_2(x'.dom), f_3(x'.dom))))$$

ahol $f : \mathbb{N}_0 \to T \times K \times D'$,

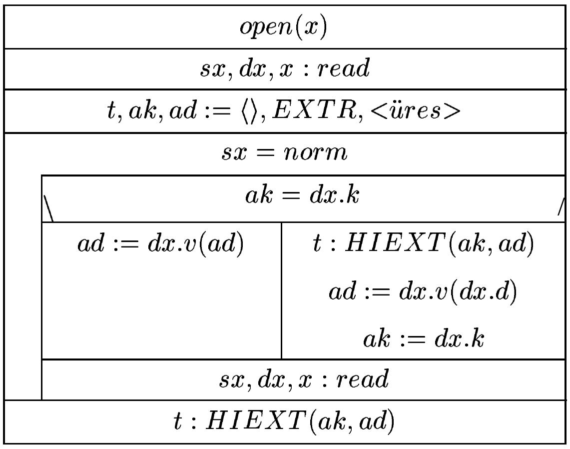
$$
\begin{aligned}
f(0) &= (\langle\rangle, EXTR, <\ddot{u}res>) \\
f(i+1) &= \begin{cases} (f_1(i), f_2(i), x_{i+1}.v(f_3(i))), & \text{ha } x_{i+1}.k = f_2(i) \\ (HIEXT(f_1(i), (f_2(i), f_3(i))), & \\ \quad x_{i+1}.k, x_{i+1}.v(x_{i+1}.d)), & \text{ha } x_{i+1}.k \neq f_2(i) \end{cases}
\end{aligned}
$$

és $HIEXT : T \times (K \times D') \to T$,

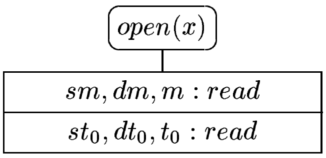
$$
HIEXT(t, k, d) = \begin{cases} hiext(t, (k, d)), & \text{ha } d \neq <\ddot{u}res> \\ t, & \text{ha } d = <\ddot{u}res> \end{cases}
$$

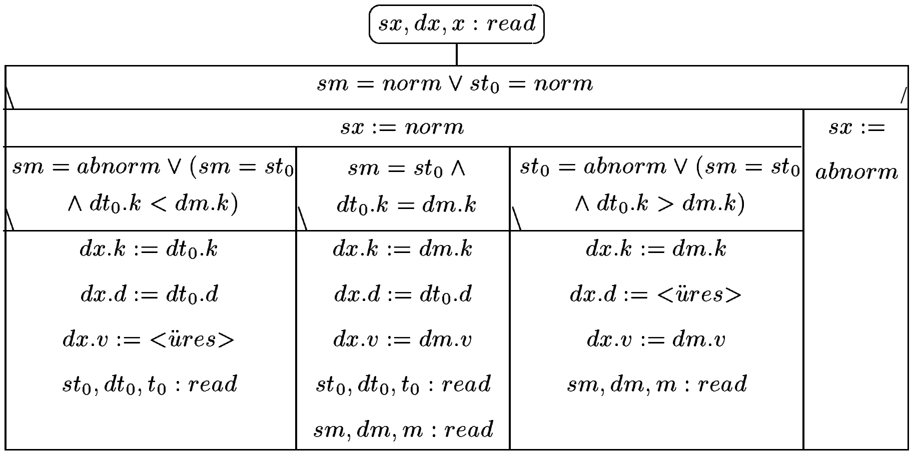
Az f függvény kezdőértékének definíciójában szereplő $EXTR$ kulcsérték egy tetsző-leges olyan kulcsérték lehet, amely egyik file-ban sem fordul elő.

Mivel az utófeltétel függvénykompozícióval adott, a megoldás egy szekvencia lesz, amelynek első része az f, második része pedig a $HIEXT$ függvényt számítja ki. Az f függvény egyes komponenseinek rendre a t, ak, ad változók felelnek meg.

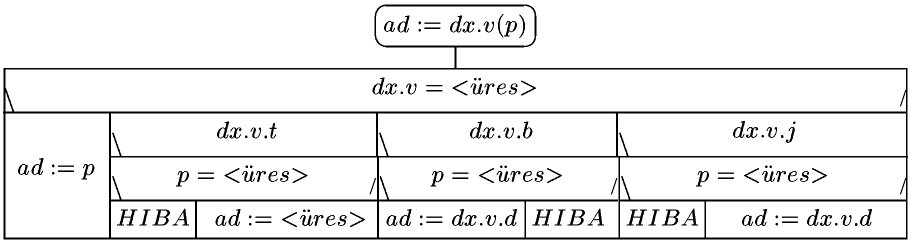


Az x absztrakt file műveleteinek megvalósítása:





Hátra van még a transzformáció elvégzésének megvalósítása:



Megoldás extremális elemmel

Az előző megodás szépséghibája, hogy a feladatot nem egy függvény helyettesítési értékének kiszámításaként specifikáltuk. Ezért adunk most egy másik megoldást is, amely egy a gyakorlatban sokszor hasznos eszközt használ. Ez az utolsó utáni elem bevezetése ($read$ extremális elemmel).

Egészítsük ki az előző megoldásban szereplő X file-t – ha nem üres – egy olyan elemmel, amelynek kulcsa minden lehetséges kulcsértéktől eltér. Jelöljük ezt a típust $\overline{X}$-sal. Ekkor a két típus közötti megfeleltetés: $\eta : X \to \overline{X}$,

$$\eta(x) = \begin{cases} hiext(x, (EXTR, <üres>, <üres>)), & \text{ha } x.dom \neq 0 \\ x, & \text{ha } x.dom = 0 \end{cases}$$

A kiszámolandó rekurzív függvény majdnem teljesen megegyezik az előzővel. A jobb áttekinthetőség érdekében a függvényt most komponensenként fogjuk felírni. Ahhoz, hogy a függvényt egyszerűbben írhassuk fel, tegyük fel, hogy az üres sorozatra alkalmazott lov függvény nem definiált értéket ad vissza (így a lov nem csak parciális függvény, hiszen minden sorozatra alkalmazható). Ekkor $f : \mathbb{N}_0 \to T \times K \times D'$,

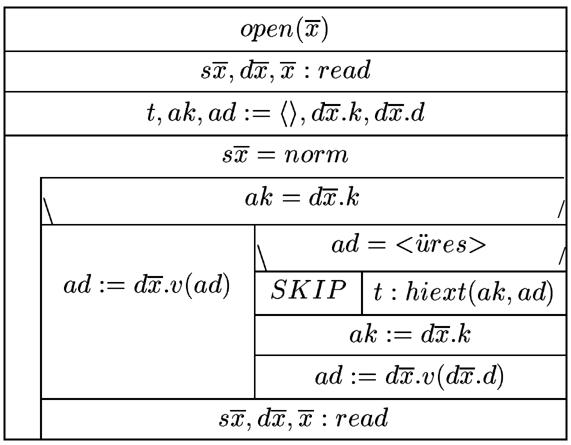
$f = (f_1, f_2, f_3)$

$$\begin{aligned}
f(0) &= (\langle\rangle, \overline{x}.lov.k, \overline{x}.lov.d) \\
f_1(i+1) &= \begin{cases} f_1(i), & \text{ha } f_2(i) = \overline{x}_{i+1}.k \lor \\ & \quad (f_2(i) \neq \overline{x}_{i+1}.k \land f_3(i) = <\ddot{u}res>) \\ hiext(f_1(i), (f_2(i), f_3(i))), & \text{ha } f_2(i) \neq \overline{x}_{i+1}.k \land f_3(i) \neq <\ddot{u}res> \end{cases} \\
f_3(i+1) &= \begin{cases} \overline{x}_{i+1}.v(f_3(i)), & \text{ha } f_2(i) = \overline{x}_{i+1}.k \\ \overline{x}_{i+1}.v(\overline{x}_{i+1}.d), & \text{ha } f_2(i) \neq \overline{x}_{i+1}.k \end{cases} \\
f_2(i+1) &= \begin{cases} f_2(i), & \text{ha } f_2(i) = \overline{x}_{i+1}.k \\ \overline{x}_{i+1}.k, & \text{ha } f_2(i) \neq \overline{x}_{i+1}.k \end{cases}
\end{aligned}$$

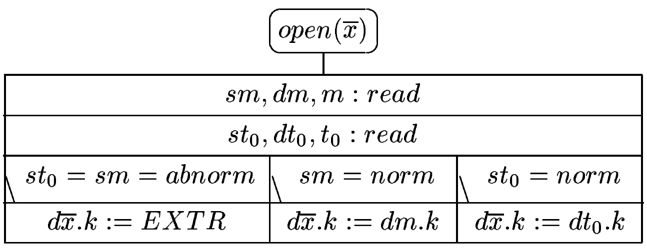
Ezt a függvényt használva a feladat specifikációja:

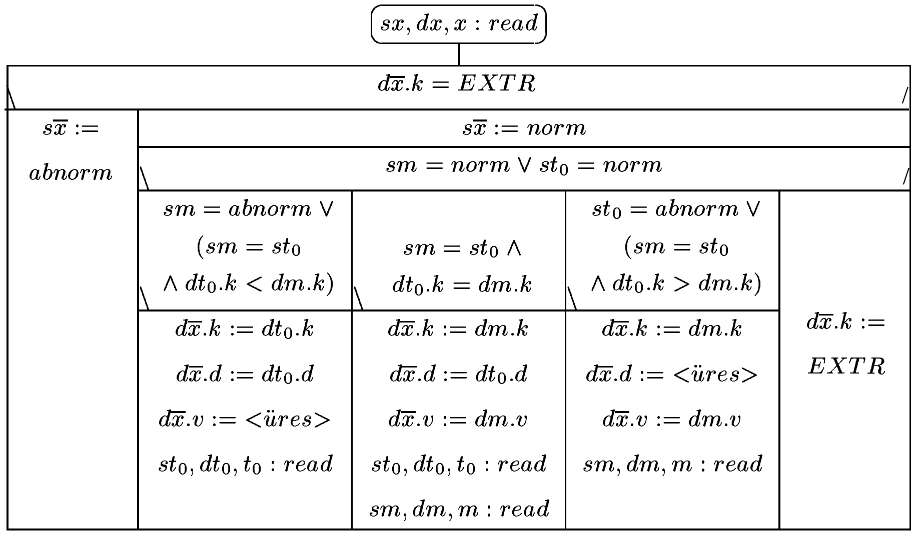
$$\begin{aligned}
A &= \underset{\overline{x}}{\overline{X}} \times \underset{t}{T} \\
B &= \underset{\overline{x}'}{\overline{X}} \\
Q &: (\overline{x} = \overline{x}') \\
R &: (t = f_1(\overline{x}'.dom))
\end{aligned}$$

Ez a feladat visszavezethető file-ra felírt rekurzívan megadott függvény helyettesítési értékének kiszámítására:

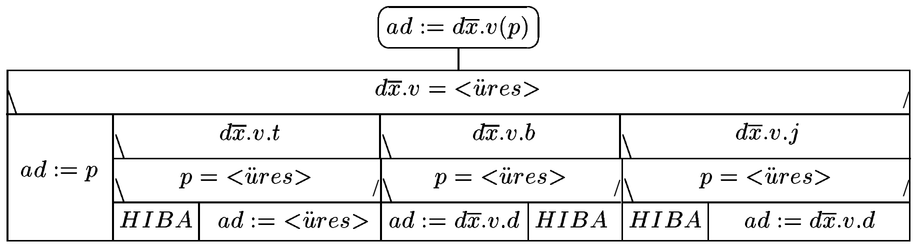


Az $\overline{x}$ absztrakt file műveleteinek megvalósítása:





A transzformáció elvégzésének megvalósítása tulajdonképpen megegyezik az előző esetnél felírttal, csak most az $\overline{x}$ absztrakt file-t használjuk:



A fenti megoldási módokat összehasonlítva látható, hogy minél magasabb absztrakciós szintű fogalmakat használunk, annál egyszerűbben tudjuk kezelni a feladatot.

Nagyon sok esetben az adat- és függvényabsztrakció is alkalmazható egy feladat megoldásakor, sőt mint azt az iménti példa mutatja a kettő kombinációja is egy lehetséges út.