Programok stabil tulajdonságainak megfigyelése

a) Specifikáció, megoldás szuperpozícióval, heurisztikák a egyes tulajdonságok megfigyelésére, megoldás igazolása heurisztikával

A detektálás és a szuperpozíciós ötlet

Az "előrejelzésre" a *detect* tulajdonságot használjuk. Ez a többi tulajdonsághoz hasonlóan programszövegből számolható, az alábbi módon:

$$detect := \{(P \to Q) \in inv_S(\bigwedge_{Q' \in init} Q') \wedge Q \hookrightarrow_S P\}$$

Tehát ez is egy logikai függvény, azokra az állapotokra igaz, amelyre a fenti két állítás igaz. Célja valami olyasmi, hogy azt mondhassuk vele, hogy ha "P teljesül (ami szintén logikai fv), akkor Q is fog". Tehát az, hogy P-ből következik Q, az egy invariáns [talpatlan nyíl: implikáció jele]. Mert P előrejelzi Q-t. A másik fele pedig arra utal, hogy ha Q véletlenül P előrejelzése nélkül lett igaz, akkor visszalépünk az alaphelyzetbe. A detektálást persze egy új változóval tesszük, ezt hozzá fogjuk szuperponálni a programhoz, a cél az, hogy a mérés ne változtassa meg a program viselkedését.

Példa:

$$S = (s_0, \{x := x + 1\})$$

Egy új, b változóval jelezzük előre, ha $x$nagyobb lesz, mint 3, azaz:

$$b\ detects_S(x > 3)$$

Ehhez alakítsuk át S-t, hogy legyen benne b

$$S' = (s_0 \parallel b :=\downarrow, \{x := x + 1 \square b := (x > 3)\})$$

Tehát alapvetően úgy működik ez a detect, hogy hozzárakunk egy olyan logikai változót, ami akkor válik igazzá, amikor igaz lesz az az állítás, amit előrejelezni akarunk.

Bizonyítsuk be, hogy $bdetect_S(x > 3)$. Ehhez a definícióban adott két állítást kell megnézni.

1) $b \to (x > 3)$ invariáns-e?

   a) kezdetben igaz?
   $\uparrow\Rightarrow lf(s_0 \parallel b :=\downarrow, b \to x > 3) =\downarrow\to (x > 0)^{s_0}$✓(hamisból minden következik)

   b) stabil?
   $(b \to x > 3) \Rightarrow lf(S, b \to x > 3) = lf(x := x + 1, b \to x > 3) \wedge lf(b := (x > 3, b \to x > 3) = (b \to x > 3)^{x \leftarrow x+1} \wedge (b \to x > 3)^{b \leftarrow b > 3} = (\neg b \vee x + 1 > 3) \wedge (x > 3 \to x > 3)$ ✓

2) $x > 3 \hookrightarrow_S b$ igaz-e?

   a) Már $x > 3 \mapsto_S b$ is igaz, hiszen van b-t $x > 3$-ra állító utasítás. Ekkor meg a görbe nyíl is igaz lesz def. szerint.

Detektálni nem csak stabil tulajdonságokat lehet!

A $count > 10$ detektálás problémája

Az általános alak innentől: $claim\ detect_S\ W$ ahol a *claim* egy (új, szuperponált) logikai változó, W pedig pedig egy stabil tulajdonság. Jelentse ebben a konkrét példában W az "egy programban a végrehajtott utasítások száma nagyobb mint 10" állítást, ami nyilván stabil (azaz, ha egyszer igaz lett, akkor az is marad. . .). Az utasításszámláló legyen egy új változó, tehát a bizonyítandó állítás ez:

$$claim\ detect_S\ count > 10$$

Legyen a szokásos $S = (s_0, \{s_1, \ldots, s_n\})$. Ebben se *claim* sem a *count* nincs benne, mindkettőt szuperponáljuk. A *count*-ot minden utasításhoz,hiszen minden végrehajtott utasítás után ezt növelni kéne, ez a szuperponálás egyik lehetséges módja, a *claim*-et pedig a fenti egyszerű példában látott "megszorított unió" módszerrel adjuk a programunkhoz, azaz beiktatunk egy új utasítást, amelyben ő szerepel (ennek a baloldalán tehát csak "új" változó állhat, jobboldalán persze bármi). Tehát ez most minőségileg két különböző fajta szuperponálás, mind az n értékadáshoz hozzáveszünk egy újabbat, valamint az egész halmazhoz hozzáveszünk egy $n+1$-ediket. Sőt s_0-hoz is hozzá kell venni az újak inicializálását.

$$S' = \left(s_0 \parallel count := 0 \parallel claim :=\downarrow \left\{ \begin{array}{l} (\overset{n}{\underset{j=1}{\square}} s_j \parallel count := count + 1)\square \\ claim := (count > 10) \end{array} \right\} \right)$$

Bizonyítás

1) $(claim \to count > 10) \in inv_S(\uparrow)$ igaz-e?

   a) kezdetben igaz, mert a *claim* hamis, és abból meg bármi következik.✓

   b) stabil is, mert az $n+1$-edik utasítás pont egyenlővé teszi a kettőt, tehát következni is fog, a többi utasítások meg *count*-ot növelik, *claim*-et nem módosítják, tehát ha eddig nagyobb volt, most is az marad. ✓

2) $count > 10 \hookrightarrow_S claim$ igaz-e?

   a) Egyenes nyílból. A $count > 10$ egyszer tutira igaz lesz, és mivel feltétlenül pártatlan az ütemezésünk, ezért a $claim := (count > 10)$ is le fog futni. És akkor a *claim* igaz lesz. De amúgy az egyáltalán nem biztos, hogy $count = 11$-nél. Lehet, hogy később. ✓

Mármost ezzel a megoldással van egy kis probléma. A *count* mind az $n+1$ utasítás közös változója, ezért a rendszer nem tud valóban párhuzamos lenni. Ezen lehet javítani, hiszen az "igazi" S-től független a két segédváltozónk.

Heurisztikák

Most megvizsgáljuk, milyen heurisztikákkal lehet bizonyos magától értetődő dolgokat bizonyítani, anélkül, hogy a fentieket végigvinnénk minden alkalommal. A heurisztikákat a későbbiekben használni is fogjuk.

H1 heurisztika

Legyen e stabil tulajdonság, amit x logikai változóval szeretnénk előre jelezni. Ha x kezdetben hamis, azaz a kezdő értékadást így szuperponáljuk: $s_0 \parallel x :=\downarrow$ és amúgy az S-t csak úgy módosítom, hogy hozzá $\Box$-elem az x frissítését, azaz ezt: $x := e$ akkor kimondhatom további vizsgálat nélkül, hogy $x\ detect_S\ e$.

Bizonyítás

Két dolog kell: $(x \to e) \in inv_s(\uparrow)$ és $e \hookrightarrow_S x$.
Az invariáns definíció alapján

1) $\uparrow \Rightarrow lf(s_0 \parallel x :=\downarrow, x \to e) = (x \to e)^{x \leftarrow \downarrow, s_0} = (\neg \downarrow \vee s^{s_0})$ (ahol s_0 nem változtatja sem x-et és e-t) Ez igaz, mert a jobb oldalon azonosan igaz van.

2) $(x \to e) \Rightarrow lf(S, x \to e)$ mivan ha x romlik el? Ekkor igazra váltott (hiszen kezdetben hamis volt), mégpedig az $x := e$ utasítással. Ekkor nyilván e is igaz volt, tehát $\uparrow \to \uparrow$.

Nézzük a görbenyilat úgy, hogy belátjuk az egyenesnyilat és onnan def szerint van görbe is. Tehát kell: $e \mapsto_S x$

1) $e \triangleright_S x$. Tudjuk, hogy $e \triangleright_S \downarrow$ (stabil). Mivel $\downarrow \Rightarrow x$, ezért jobb oldal gyengítésével kapjuk, hogy $e \triangleright_S x$. ✓

2) $\exists s \in S : e \wedge \neg x \Rightarrow lf(s, x)$. Nyilván ha van ilyen, akkor az $x := e$ lesz az. $e \wedge \neg x \Rightarrow lf(x := e, x) = (x)^{x \leftarrow e} = e$ ✓

H2 heurisztika

Legyen e most monoton növő egész érték (azaz a program utasításai csak növelni tudják), aminek egy bizonyos felső határon k való túlléptét x egész típusú változóval szeretnénk előre jelezni. Ha x kezdetben e értékét veszi fel, azaz a kezdő értékadást így szuperponáljuk: $s_0 \parallel x := e$ és amúgy az S-t csak úgy módosítom, hogy hozzá $\Box$-elem az x frissítését, azaz ezt: $x := e$ akkor kimondhatom további vizsgálat nélkül, hogy x is monoton növő, valamint hogy $x \geq k\ detect_S\ e \geq k$.

Példa: Az előző S' programban ezt szeretnénk a H1-gyel bizonyítani:
$claim\ detect_S\ count > 10$
Teljesülnek-e H1 feltételei?

- $count > 10$ stabil? ✓
- $claim$ logikai változó? ✓
- $claim$ kezdetben hamis S'-ben? ✓
- $claim$ csak úgy kap új értéket, hogy $claim := (count > 10)$? ✓

Szóval, mivel $e \sim (count > 10)$ és $x \sim claim$ szereposztással teljesülnek H1 feltételei, ezért bebizonyíttatott, hogy $claim\ detect_S\ (count > 10)$.

Bizonyítás

Két dolog kell: $(x \geq k \to e \geq k) \in inv_S(\uparrow)$ és $e \geq k \hookrightarrow_S x \geq k$.
Invariáns:
$\cdots = (x < k \vee e \geq k) \in inv_S(\uparrow) = (x < k \vee k \leq e) \in inv_S(\uparrow)$
Ha $(x \leq e) \in inv_S(\uparrow)$, akkor nem lehet az, hogy x nagyobb, mint e azaz nem történhet meg a fenti bizonyítandó dolog tagadása. Tehát elég ezt bebizonyítani, és akkor a fenti is kijön. De x kezdetben e, tehát kezdetben igaz az egyenlőség. Később pedig a monoton növő e-t követi, néha egyenlő vele, néha kisebb.

Görbe nyilat pedig megint egyenes nyílból.

H3 heurisztika

A detektálás tranzitív, azaz:

$$P\ detect_S\ Q \wedge Q\ detect_S\ R \to P\ detect_S\ R$$

Bizonyítás

Két dologból áll egy ilyen detektálás. Egy görbe nyílból meg egy invariánsból. A görbe nyíl természeténél fogva tranzitív. Míg az invariánsról azt tudom elmondani, hogy ha összeéselek kettőt (márpedig itt összeéselek kettőt), akkor amit kaptam az is invariáns.

b) Aszinkronitás biztosítása finomításokkal, heurisztikák alkalmazása a bizonyításhoz

Az előző példát folytatjuk.

Szekvencializálódás elkerülése

Minden s_j utasításhoz hozzárendelek egy számlálót, ami azt mondja meg, hogy s_j hányszor futott le.

$$S'' = \left(\begin{array}{l} s_0 \parallel N := 0 \parallel claim :=\downarrow \parallel u_1.m := 0 \parallel \ldots \parallel u_n.m := 0 \\ \left\{ \begin{array}{l} \{(\overset{n}{\underset{j=1}{\square}} s_j \parallel u_j.m := u_j.m + 1) \ \square \\ N := \sum_{j=1}^{n} u_j.m \ \square \\ claim := (N > 10) \end{array} \right\} \end{array} \right)$$

Most az n darab folyamatunk csupa különböző változóval játszik, nem blokkol senki semmit. Kivéve, az N-nek értékat adó, amelyik folyamat viszont mindenkit blokkol, de még így is jobb, mint eddig. Most is $claim\ detect_S\ count > 10$ kell a *count* most a $\sum_{j=1}^{n} u_j.m$, ami néha egyenlő N-vel is. De csak néha, mert N csak időnként frissül.

$$claim\ detect_S(\sum_{j=1}^{n} u_j.m) > 10$$

Használjuk H3-at! Ha $claim\ detect_S\ N > 10$ (ami H1 miatt oké), és $N > 10\ detect_S\ (\sum_{j=1}^{n} u_j.m) > 10$ akkor a tranzitivitás miatt igaz a bizonyítandó is. $N > 10\ detect_S\ (\sum_{j=1}^{n} u_j.m) > 10$ pedig H2-vel jön ki, $k = 11$ szereposztással.

Még tutibb megoldás

Most folyamatonként még egy új változót iktatunk be, mert ezekből sose elég...! Sőt, új detektáló folyamatot is, ami kezeli az új változókat.

$$S''' = \left(\begin{array}{l} s_0 \parallel N := 0 \parallel claim :=\downarrow \parallel u_1.m, u_1.r := 0, 0 \parallel \ldots \parallel u_n.m, u_n.r = 0, 0, \\ \left\{ \begin{array}{l} (\overset{n}{\underset{j=1}{\square}} s_j \parallel u_j.m := u_j.m + 1) \ \square \\ (\overset{n}{\underset{j=1}{\square}} u_j.r := u_j.m) \ \square \\ N := \sum_{j=1}^{n} u_j.r \ \square \\ claim := (N > 10) \end{array} \right\} \end{array} \right)$$

Tehát mindenkinek van saját számlálója, ezt néha egy külön folyamat átmásolja egy másik számlálóba ($u_j.r$), és ezeket fogja majd az N összegezni, ez azért jó,

mert így az $u_j.m$-eket nem blokkolja, márpedig azok blokkolása magát az érdemi számítást (s_j) is blokkolná. Bizonyítandó megint a *claim* $detect_S(\sum_{j=1}^{n} u_j.m) > 10$. És ezt megint tranzitívan fogjuk megtenni: *claim* $detect_S\ N > 10$ (H1), $N \geq k\ detect_S\ (\sum_{j=1}^{n} u_j.r) \geq k$ (H2) és $(\sum_{j=1}^{n} u_j.r) \geq k\ detect_S\ (\sum_{j=1}^{n} u_j.m) \geq k$ (ezt nem tudom heurisztikával, de tudom helyette H2-vel egyesével ezeket és ez pont ezt fogja jelenteni: $u_j.r \geq k\ detect_S\ u_j.m \geq k$). És ha ez a három dolog megvan, akkor az első kettőre is nyomok egy H3-at, majd az így kapottra és a harmadikra is nyomok egy H3-at, és meg is van, amit akartam.

c) Inkrementális frissítés, gyenge szuperpozíció alkalmazása, megvalósítás tokennel

Inkrementális frissítés

A cél továbbra is a $claim\ detect_S\ count > 10$ biztosítása. Eddig az összes megoldás olyan volt, hogy néha SOK folyamatot blokkolt. Vagy magukat az eredeti folyamatokat, vagy ha később bevezettünk darab segdéd-folyamatot, akkor azokat. Most olyan megoldást adunk, ahol néha csak 1, de legrosszabb esetben is csak 2 processz áll és vár.

Bevezetünk minden j-vel jelölt ($j \in [1 \dots n]$) folyamathoz egy új változót, az $u_j.delta$-t, ami a j-edik utasítás lefutásainak számát jelöli, de nem összesen, mint eddig, hanem csak a legutóbbi frissítés óta. Tehát a delták értékeit rendszeresen átviszem N-be és ekkor nyilván nullázom a deltát. Ezért van az, hogy egyszerre csak két folyamat áll, mert a deltákat egyesével adogatom N-hez.

$$S'''' = \left(\begin{array}{l} s_0 \parallel N := 0 \parallel claim :=\downarrow \parallel u_1.delta := 0 \parallel \dots \parallel u_n.delta := 0 \\ \left\{ \begin{array}{l} (\overset{n}{\underset{j=1}{\square}} s_j \parallel u_j.delta := u_j.delta + 1)\ \square \\ (\overset{n}{\underset{j=1}{\square}} N : N = u_j.delta \parallel u_j.delta := 0)\ \square \\ claim := (N > 0) \end{array} \right\} \end{array} \right)$$

Előnye még az előző megoldással szemben, hogy kevesebb változót használ.
Bizonyítása. Az kell, hogy $claim\ detect_S\ count > 10$. De mi itt a *count*? Az Nben gyűlik a végrehajtott utasítások száma, de semmi garancia nincs rá, hogy midnen delta lenullázta magát ezért ez az összefüggés érvényes: $count = N + \sum_{j=1}^{n} u_j.delta$. Ezt a programba nem írjuk be, hiszen az pont hogy szekvencializálná azt. A bizonyításhoz viszont használjuk. H1-gyel belátjuk, hogy $claim\ detect_S N > 10$, aztán H2-vel szeretnénk, hogy $N \geq k\ detect_S\ count \geq k$. Ebből a kettőből meg H3-mal kijön, hogy $claim\ detect_S\ count > 10$.

H1: kell hozzá, hogy $N > 10$ stabil (nyilván az, mert N nő), hogy *claim* logikai változó (az), hogy *claim* kezdetben hamis (az) és hogy *claim* csak úgy változik, hogy $N > 10$ tulajdonság értékét kapja (csak úgy változik). Ezek mind stimmelnek, tehát H1 alkalmazható.

$N \geq k\ detect_S\ count \geq k$ bizonyítása. Na ez mégse megy H2-vel, mert *count* nem egy darab változó.
Úgyhogy def. szerint kell.

1) $\forall j \in [1 \dots n] : u_j.delta \geq 0$, ez biztos, mondhatni $(u_j.delta \geq 0) \in inv_s(\uparrow)$. Meg az is igaz, hogy $(count = N + \sum_{j=1}^{n}) \in inv_S(\uparrow)$. Innen (invariánsok éselése invariáns) adódik, hogy: $(count \geq N) \in inv_S(\uparrow)$. Ebből pedig következik, ami kell: $(N \geq k \rightarrow count \geq k) \in inv_S(\uparrow)$.

2) A másik fele pedig, hogy $(count \geq k \hookrightarrow_S N \geq k)$ szintén igaz, mert N előbb-utóbb frissíti magát és utoléri *count* jelenlegi értékét.

Token-alapú megoldás

Ez egy implementációs módszer egy adott erőforrásra kölcsönös kizárás biztosítására. Körbe jár egy token, és akinél épp van, az dolgozhat, a többi vár. A program gyenge szuperpozíciót fog használni. Ennek következik most a definíciója: Adott az S program, annak egy s_j utasításához szuperponáljuk az r utasítást.Így fog kinézni az új j-edik utasítás:

$$s'_j = (s_j \ ha \ b \parallel r, \ ha \ b)$$

De csak akkor, ha az alábbiak teljesülnek:

- Igazak a rendes szuperpozíció feltételei (a kiterjesztésben részt vevő feltétel és az új értékadásban levő változók nem macerálják az erdeti S-t).
- Egyszer b igaz lesz. ($\uparrow \hookrightarrow_S b$).
- b az eredetire nézve stabil ($b \triangleright_{S' \setminus \{s'\}} \downarrow$).

Viselkedését tekintve hasonló, mint a sima szuperpozíció, kivéve, hogy az egyenesnyíl tulajdonság a feltétel miatt nem marad meg.De mindenesetre a tokenes megoldáshoz ezt a módszert alkalmazzuk: Legyen $token \in [0 \dots n]$ egy új változó, ha a $token$ értéke j, akkor a j-edik futtathat.

$$S = \left(\begin{array}{l} s_0 \parallel N := 0 \parallel token := 1 \parallel claim :=\downarrow \parallel u_1.delta := 0 \parallel \dots \parallel u_n.delta := 0, \\ \left\{ \begin{array}{l} (\overset{n}{\underset{j=1}{\square}} s_j, \ ha \ token = j \parallel u_j.delta := u_j.delta + 1, \ ha \ token = j) \ \square \\ (\overset{n}{\underset{j=1}{\square}} N := N + u_j.delta \parallel u_j.delta := 0, \ hatoken = j) \ \square \\ claim := (N + u_j.delta > 10) \parallel token := (token \oplus 1) \ hatoken = 1 \end{array} \right\} \end{array} \right)$$

Ez tehát más implementáció, mint az eddigiek. Jó kevés pluszváltozó van. Viszont mivel egyszerre csak egy folyamat dolgozhat, talán valamelyest rosszabb megoldás, mint az inkrementális frissítéses.