Antti Vuorenmaa

WP-LASKENNAN OIKEELLISUUS JA TÄYDELLISYYS

Tietotekniikan pro gradu -tutkielma

Ohjelmistotekniikan linja

24.6.2003

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Antti Vuorenmaa

Yhteystiedot: Puistokatu 1 B 34, 40100 Jyväskylä

Työn nimi: WP-laskennan oikeellisuus ja täydellisyys

Title in English: Soundness and Completeness of WP-calculus

Työ: Pro gradu -tutkielma

Sivumäärä: 42+2 (42 = tutkielman sivumäärä ilman liitteitä; 2 = liitteiden sivumäärä)

Linja: Ohjelmistotekniikka.

Teettäjä: Jyväskylän yliopisto, tietotekniikan laitos

Avainsanat: Aksiomaattinen semantiikka, Heikoimman esiehdon semantiikka, Hoaren logiikka, WP-laskenta, Ohjelmoinnin teoria

Keywords: Axiomatic semantics, Weakest precondition semantics, Hoare logic, WP-calculus, Theory of programming

Tiivistelmä: Tutkielmassa osoitetaan WP-laskennan oikeellisuus ja täydellisyys pysähtyville ohjelmille Cookin formalismin mielessä.

Abstract: The study proves that for terminating programs WP-calculus is sound and complete in the sense of Cook's formalism.

Sisältö

1 Johdanto

Ohjelmistonkehityksen perusongelma on asiakasta tyydyttävien ohjelmien tuottaminen resurssien sallimissa rajoissa. Tilaajan ja alihankkijan vastuut määritellään toimitussopimuksissa, jotka edelleen pohjautuvat kuvauksiin tilatun tuotteen ominaisuuksista. Mitä tarkemmin ja tyhjentävämmin kuvaukset eli spesifikaatiot on laadittu, sitä kiistattomammin lopputuloksen voidaan osoittaa olevan sopimuksen mukainen tai sen vastainen.

Eräs keino ohjelman käyttäytymisen tarkkaan kuvaamiseen on määrittää syötteiden ja vasteiden väliset suhteet. Esimerkiksi reaaliluvun neliöjuuren laskevan ohjelman syötteen tulee olla vasteen neliö.

Periaatteessa on helppoa osoittaa ohjelmiston olevan virheellinen: riittää löytää yksikin syöte, joka tuottaa väärän vasteen. Vaikeampaa on todistaa ohjelman todella olevan spesifikaation mukainen. Tunnettuja keinoja ovat mm. kaikkien syötteiden läpikäynti ja oikeellisuustodistukset ns. Hoaren logiikan [8] avulla.

E. W. Dijkstra esitti vuonna 1975 artikkelissaan [4], että ohjelma ja sen oikeellisuustodistus tulisi kirjoittaa samanaikaisesti; tällöin varmistettaisiin ohjelman todistettavuus. Menetelmäänsä Dijkstra kutsui nimellä "formal derivation of programs" ja sisällytti siihen kehittämänsä Guarded Command -ohjelmointikielen sekä matemaattisen koneiston oikeellisuustodistusten laadintaan.

Dijkstran menetelmän "matemaattinen koneisto" tunnetaan nimellä WP-laskenta. Tämän tutkielman tehtävänä on vastata kysymykseen, onko WP-laskenta teoreettisesti käyttökelpoista, ts. voidaanko WP-laskennalla laadittuihin todistuksiin luottaa ja voiko sitä soveltaa kaikille spesifikaatioille ja ohjelmille. Osoittautuu, että WP-laskenta on luotettavaa ja riittävää tässä tutkielmassa käytetylle yksinkertaiselle ohjelmointikielelle sovellettuna - ainakin kun tiedetään spesifioidun ohjelman pysähtyvän kaikilla syötteillä.

1.1 Tutkielman rakenne ja tulokset

Tässä kappaleessa käydään läpi tutkielman rakenne ja luetellaan kaikki tutkielmassa esitetyt omat määritelmät ja tulokset (näistä käytetään ilmaisua "tutkielman tulos").

Luvussa 2 määritellään Guarded Command -ohjelmointikielen kielioppi ja tyyppijärjestelmä sekä annetaan esimerkkejä kielellä kirjoitetuista ohjelmista. Kielioppi ja tyyppijärjestelmä ovat tutkielman tuloksia, joskin hyvin samankaltaisia Dijkstran artikkelin [4] määritelmien kanssa. Kielioppi on pyritty pitämään mahdollisimman yksinkertaisena (mm. muuttujien esittelyt jätetty pois), jotta luvun 5 oikeellisuus- ja täydellisyystarkastelut pysyisivät lyhyinä ja selkeinä.

Luvussa 3 kuvataan Guarded Command -ohjelmointikielellä kirjoitettujen ohjelmien suoritin ja määritellään predikaattimuuntimen käsite, jota käytetään luvussa 4 ohjelmointikielen semantiikan esittämiseen. Sivulla 7 esitetyt kaavat tilojen ja predikaattien lukumäärien laskemiseksi ovat tutkielman tuloksia. Kappaleessa 3.2.7 annettu predikaattimuuntimen määritelmä on tutkielman tulos. Se on "karsittu" versio Dijkstran artikkelissa [4] esitetystä määritelmästä: kaikki predikaattimuuntimen ominaisuudet, jotka voidaan osoittaa seuraavan muista ominaisuuksista on jätetty pois mahdollisimman tiiviin määritelmän aikaansaamiseksi. Myös huomautuksen 3.2.4.1 vastaesimerkki on kuuluu tutkielman tuloksiin.

Luvussa 4 mallinnetaan Guarded Command -ohjelmointikielen semantiikkaa predikaattimuuntimien avulla. Luvussa esitetyt predikaattimuuntimet ovat peräisin Dijkstran kirjasta [5], mutta niiden johdattelut ja todistukset predikaattimuuntimiksi kuuluvat tutkielman tuloksiin, paitsi kappaleen 4.1.1 todistus, joka on esitetty myös lähteessä [5, s. 31].

Luvussa 5 tutkitaan WP-laskennan oikeellisuutta ja täydellisyyttä Cookin formalismin [3] mielessä. Luvun todistukset ja Comp-funktion määritelmä kuuluvat tutkielman tuloksiin.

Luvussa 6 esitetään tutkielman yhteenveto, joka luonnollisesti kuuluu tutkielman tuloksiin.

2 Guarded Command -kieli

Tässä luvussa määritellään Guarded Command -ohjelmointikieli, joka esiintyi ensimmäisen kerran vuonna 1975 Dijkstran artikkelissa [4]. Kieli on ensisijaisesti tarkoitettu algoritmien kuvaamista varten, ei niinkään käännettäväksi tietokoneella suoritettaviksi ohjelmiksi. Kattavampi kuvaus löytyy mm. lähteestä [9].

2.1 Kieliopin BNF-kuvaus

Ensiksi tutustutaan Guarded Command -kielen kielioppiin BNF-metakielen [11] avulla. Produktiot on numeroitu sulkeisiin '(' ja ')' laitetuilla numeroilla viittaamisen helpottamiseksi. Välilyönti-, sarkain- ja rivinvaihtomerkkejä voidaan käyttää ohjelmakoodin muotoiluun kaikkialla muualla paitsi varattujen sanojen ('IF', 'DO', 'FI', 'OD', 'skip' ja 'abort') sekä produktioiden <integer> ja <identifier> sisällä.

(1) <statement> ::= <assignment> | skip | abort

| IF <guarded command set> FI

| DO <guarded command set> OD

| <statement> {; <statement>}

(2) <guarded command set> ::= <guarded command> {□ <guarded command>}

(3) <guarded command> ::= <disjunction> → <statement>

(4) <assignment> ::= <variable> := <addition>

| <variable>, <assignment>, <addition>

(5) <disjunction> ::= <disjunction> ∨ <conjunction> | <conjunction>

(6) <conjunction> ::= <conjunction> ∧ <negation> | <negation>

(7) <negation> ::= ¬ <negation> | (<disjunction>) | <comparison>

(8) <comparison> ::= <addition> = <addition>

| <addition> ≠ <addition>

| <addition> < <addition>

| <addition> > <addition>

| <addition> ≤ <addition>

| <addition> ≥ <addition>

(9) <addition> ::= <addition> + <multiplication>

| <addition> – <multiplication> | <multiplication>

(10) <multiplication> ::= <multiplication> * <term>

| <multiplication> / <term> | <term>

(11) <term> ::= – <term> | + <term> | <integer> | <variable>

| (<addition>)

(12) <integer> ::= <digit> { <digit> }

(13) <variable> ::= <identifier> | <identifier> (<addition>)

(14) <identifier> ::= <letter> { <letter> | <digit> }

(15) <digit> ::= 0|1|2|3|4|5|6|7|8|9

(16) <letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Kielen lähtöproduktio on <statement> (suom. lause), joka voi edelleen olla sijoituslause, IF-lause, DO-silmukka, skip- tai abort-lause tai se voi koostua yhdestä tai useammasta lauseesta, jotka erotetaan toisistaan puolipisteellä (';').

Peräkkäiset, puolipistein toisistaan erotetut lauseet suoritetaan järjestyksessä vasemmalta oikealle. Skip-lauseen suorituksen aikana yhdenkään muuttujan arvo ei vaihdu ja ohjelman suoritus jatkuu skip-lausetta seuraavasta lauseesta. Abort-lause aiheuttaa ohjelman suorituksen epänormaalin keskeytyksen, joka voidaan myös käsittää ikuisena silmukkana.

IF-lause sisältää yhden tai useamman <guarded command> -esiintymän, jotka ovat erotettu toisistaan laatikolla ('□'). <guarded command> koostuu suoritusehdosta, nuolimerkistä ('→') ja lauseesta. Kun ohjelman suoritus tulee IF-lauseeseen, valitaan suoritettavaksi yksi niistä lauseista, joiden suoritusehdot ovat voimassa. Kun lause on suoritettu, jatketaan suorittamalla IF-lausetta seuraava lause. Mikäli yksikään suoritusehto ei ole voimassa ohjelman suoritus keskeytyy (vaikutus on siis sama kun abort-lauseella).

DO-lauseen sisältämistä lauselistoista, joiden suoritusehto on voimassa, valitaan suoritettavaksi yksi kunnes kaikki suoritusehdot ovat epätosia. (Jos kaikki suoritusehdot ovat ohjelman joka tilassa epätosia, DO-lauseen vaikutus on sama kuin skip-lauseella.)

2.2 Tyypit, muuttujat, lausekkeet ja taulukot

Tutkielmassa käytetty versio Guarded Command -ohjelmointikielestä tuntee kaksi tyyppiä: kokonaislukutyypin ja totuusarvotyypin. Kokonaislukutyyppi, jota merkitään symbolilla int, koostuu äärettömän monesta kokonaisluvusta ja pohja-alkiosta $\perp_{int}$, jota voidaan pitää määrittelemättömän lausekkeen (esimerkiksi nollalla jako tai määrittelemätön muuttujasymboli) arvona. Vastaavasti totuusarvotyyppi, jota merkitään symbolilla boolean, käsittää arvot true ja false sekä tyypin pohja-alkion $\perp_{boolean}$. Tyyppien määritelmät on laadittu Scottin teorian [7] hengessä.

Kaikki muuttujat ovat tyyppiä int. Muuttujan nimi (produktio 14) alkaa kirjaimella, jota voi seurata numeroita tai kirjaimia. Muuttujien esittelyä ja lohkorakennetta ei ole tähän kielen versioon sisällytetty (kattavampi määrittely on esitetty mm. lähteessä [9]).

Taulukot käsitetään kokonaislukuarvoisina funktioina, joiden määrittelyjoukko koostuu joukosta peräkkäisiä kokonaislukuja. Esimerkiksi f(10) tarkoittaa taulukon (siis funktion) f arvoa pisteessä 10 ja t(–3) := a + b lausekkeen a + b arvon sijoitusta taulukon t paikkaan –3. Funktion arvo määrittelyalueen ulkopuolella on int-tyypin pohja-alkio.

2.3 Esimerkki: kahden luvun maksimi

Ohjelma saa syötteenä kaksi lukua, jotka on talletettu muuttujien x ja y arvoiksi. Tehtävänä on sijoittaa muuttujan m arvoksi suurempi näistä luvuista. Mikäli suoritusehdot ovat voimassa yhtä aikaa, ts. x = y, jompi kumpi lauseista suoritetaan satunnaisen valinnan perusteella.

IF $x \geq y \rightarrow m := x$ □ $y \geq x \rightarrow m := y$ FI

2.4 Esimerkki: kuplalajittelu neljälle luvulle

Ohjelmassa on käytössä neljä muuttujaa: a, b, c ja d. Ohjelman alkaessa muuttujilla on mielivaltaiset lukuarvot. Ohjelman päätyttyä tulee ehdon { $a \leq b \leq c \leq d$ } olla voimassa. Toteutus on tehty DO-silmukalla. DO-silmukan sisällä olevista riveistä suoritetaan se, jonka suoritusehto on tosi. Laatikkomerkki '□' toimii rivien välissä erottimena. Silmukan suoritus loppuu kun kaikkien rivien suoritusehdot ovat epätosia.

DO

$a > b \rightarrow a, b := b, a$

□ $b > c \rightarrow b, c := c, b$

□ $c > d \rightarrow c, d := d, c$

OD

3 Suoritin, tilat, tilapredikaatit ja predikaattimuuntimet

Guarded Command -kielen suoritin suorittaa kielellä kirjoitettuja ohjelmia. Suorittimen tila määritellään (kuten esimerkiksi lähteessä [1, s. 435]) funktiona ohjelman muuttujilta arvojoukolle[1], ts. jos ohjelmassa on #V muuttujaa käytössä ja arvojoukon koko on #D, niin suorittimella on $\#D^{\#V}$ mahdollista tilaa. Jos suoritettava ohjelma vaihtaa muuttujan arvoa, niin suoritin siirtyy uuteen tilaan. Ohjelman suoritus on epädeterministististä mikäli emme aina varmuudella pysty sanomaan, mihin tilaan suoritin päätyy ohjelman suorituksen jälkeen. Tähän tilanteeseen joudutaan suoritettaessa ohjelmia, jotka sisältävät IF- tai DO-lauseita, joiden suoritusehdoista useampi kuin yksi on totta.

Tilapredikaatti on funktio tiloilta totuusarvoille; se on totta jollekin suorittimen tilojen osajoukolle (siis tilojen potenssijoukon alkiolle) ja epätotta muille tiloille, ts. #S tilalle voidaan muotoilla $2^{\#S}$ eri tilapredikaattia. Sanotaan, että "predikaatti P on vahvempi kuin predikaatti Q", mikäli kaikissa tiloissa, joissa P on totta, myös Q on totta. Tällä tavoin ajateltuna predikaatti, joka on totta kaikille tiloille, on kaikkia muita predikaatteja heikompi. Vastaavasti predikaatti, joka on epätotta kaikille tiloille, on vahvin mahdollinen. Mikäli predikaatit ovat totta täsmälleen samoissa tiloissa, ne ovat yhtä vahvoja ja merkitään P = Q.

Tilapredikaattien tarkkailulla pyritään tulkitsemaan suoritettavien lauseiden vaikutuksia suorittimen tilaan: ollaan kiinnostuneita siitä, mitkä predikaatit ovat totta ennen ohjelman suoritusta ja mitkä suorituksen loputtua. Esiehto (engl. pre-condition, kts. [5, s. 16]) on sellainen predikaatti, jonka voimassaolo ennen lauseen suoritusta takaa toisen predikaatin, jälkiehdon (engl. post-condition, kts. [5, s. 16]), toteutumisen lauseen suorituksen jälkeen. Vakiintunut termi esiehdon, ohjelman ja jälkiehdon muodostamalle kolmikolle on "Hoaren tripla" ja sitä merkitään kolmikolla {P} S {Q}, missä P tarkoittaa esiehtoa, S suoritettavaa ohjelmaa ja Q jälkiehtoa. Mikäli kolmikko {P} S {Q} on totta kaikissa tiloissa, sanotaan, että esiehto ja jälkiehto muodostavat yhdessä ohjelman spesifikaation.

---

[1] Esimerkiksi luonnolliset luvut tai totuusarvot.

Ollaan siis tekemisissä ohjelmointikielten, tilakoneiden ja logiikan kanssa. Aihepiirin ensimmäisenä julkaisuna pidetään Floydin artikkelia [6] vuodelta 1967, missä käsiteltiin ohjelmien oikeellisuustodistuksia vuokaavioiden perusteella. Hoaren artikkelissa [8] vuodelta 1969 hahmoteltiin todistusjärjestelmää yksinkertaiselle ohjelmointikielelle (esikuvana Algol 60 [11]) ja todistettiin tällä kielellä kirjoitettuja ohjelmia oikeiksi. Hoaren järjestelmä vaatii esiehdolta, että sen täyttymisestä ja ohjelman pysähtymisestä seuraa jälkiehto. Dijkstra esittelee vuoden 1975 artikkelissaan [4] "heikoimman esiehdon" (engl. weakest pre-condition) käsitteen, jolla tarkoitetaan esiehtoa, joka on paitsi riittävä mutta myös välttämätön. Erona on myös suhtautuminen ohjelman pysähtymiseen: Alkuperäinen Hoaren logiikka ei esitä mitään keinoja ohjelman pysähtymisen toteamiseen, mutta Dijkstran heikoimmalta esiehdolta vaaditaan, että ohjelma tuottaa oikean tuloksen ja pysähtyy.

3.1 Predikaattimuunnin

Dijkstran logiikassa heikoin esiehto ajatellaan tuotettavan funktiolla, jolle annetaan syötteinä ohjelmointikielellä kirjoitettu lause ja haluttu jälkiehto. Tätä esiehdon tuottavaa funktiota kutsutaan nimellä predikaattimuunnin (engl. predicate transformer).

Merkitään symbolilla States tilojen joukkoa ja kirjaimilla P, Q, R, T ja F funktioita tilojen joukolta totuusarvoille (merkitään symboleilla true ja false), ts.

States := { ohjelman mahdolliset tilat }

P: States $\rightarrow$ { true, false }.

Funktiot T ja F eroavat muista funktioista siten, että T palauttaa aina arvon true ja F aina arvon false, ts. $\forall$ s $\in$ States : T(s) = true $\wedge$ F(s) = false.

Funktioita P, Q, R, T ja F kutsutaan tilapredikaateiksi tai lyhemmin predikaateiksi. Merkitään kaikkien predikaattien joukkoa symbolilla Predicates ja symbolilla Programs Guarded Command -ohjelmointikielen ohjelmien joukkoa (Programs-joukkoon kuuluvat siis kaikki kieliopin mukaiset merkkijonot).

Predikaattimuuntimen tehtävänä on tuottaa heikoin esiehto annetun ohjelman lauseen ja sen jälkiehdon perusteella. Sille annetaan siis parametreina joku Programs-joukon alkio ja joku predikaatti ja sen tuloksena on jokin predikaatti. Kaavoissa predikaattimuunninta merkitään kirjaimilla wp (tulee sanoista weakest pre-condition). Matemaattisin merkinnöin: wp: Programs × Predicates → Predicates.

Vaihtoehtoinen luonnehdinta saadaan käsittämällä predikaatti tilojen potenssijoukon (merkitään 2^{States}) alkiona:

$$\text{wp: Programs} \times 2^{States} \rightarrow 2^{States}.$$

Koska olemme kiinnostuneita vain sellaisista esi- ja jälkiehdoista, jotka ovat voimassa tutkittavalle lauseelle kaikissa suorittimen tiloissa, predikaattimuuntimen lähtö- ja maalialueiden määrittelyssä ei tarvitse mainita tilaa erikseen.

Esimerkiksi sijoituslauseelle x := 1 ja jälkiehdolle { x ≥ y } voidaan kirjoittaa:

$$\text{wp("x := 1", } \{x \geq y\} \text{)} = \{1 \geq y\}$$

Predikaattimuuntimen tulos on saatu sijoittamalla jälkiehtoon x:n paikalle sijoituslauseen oikea puoli (siis luku 1), mikä vastaa täysin Hoaren sijoitusaksioomaa [8]. Sijoitus-aksiooman perusteella siis tiedämme, että tulos kelpaa esiehdoksi, mutta tässä vaiheessa ei ole takeita siitä, että saatu esiehto on heikoin mahdollinen. On siis määriteltävä tarkemmin, mitä ominaisuuksia heikoimmalta esiehdolta ja predikaattimuuntimelta edellytetään, ennenkuin voidaan päättää onko jokin predikaatti heikoin esiehto vai ei.

3.2 Predikaattimuuntimen ominaisuuksia

Tässä luvussa käymme läpi predikaattimuuntimen perusominaisuuksia, jotka ovat johdateltavissa annetun epäformaalin määritelmän ja predikaattilogiikan avulla. Seuraavissa luvuissa näitä ominaisuuksia käytetään predikaattimuuntimen määritelmänä; mikäli jollakin oliolla on ominaisuudet 1—4, niin hyväksymme sen "heikoimman esiehdon tuottavaksi predikaattimuuntimeksi". Ominaisuudet on mainittu jo lähteessä [4], mutta tässä esitettävät perustelut ovat lähteestä [5].

3.2.1 Ominaisuus 1 "Poissuljetun ihmeen laki"

Kaikille ohjelman tiloille pätee: $wp(S, F) = F$. (1)

Perustelu: Mikäli tämä ei olisi totta, olisi ainakin yksi tila, jolle $wp(S, F)$ on totta. Jos tässä tilassa suoritetaan S, suoritin siirtyy tilaan, jolle F on totta (ristiriita, sillä oletuksen mukaan F on epätosi kaikille tiloille).

3.2.2 Ominaisuus 2 "Predikaattimuuntimen monotonisuus"

Kaikille ohjelman tiloille pätee: Jos $P \Rightarrow Q$, niin $wp(S, P) \Rightarrow wp(S, Q)$. (2)

Perustelu: Kun tiedetään, että $wp(S, P)$ johtaa P:n voimassaoloon S:n suorituksen jälkeen ja että $P \Rightarrow Q$, niin tiedetään, että $wp(S, P)$ takaa myös Q:n toteutumisen S:n suorituksen jälkeen. Joten $wp(S, Q)$ on totta ainakin silloin kun $wp(S, P)$ on totta, siis implikaatio $wp(S, P) \Rightarrow wp(S, Q)$ pätee.

3.2.3 Ominaisuus 3 "Konjunktion distributiivisuus"

Kaikille ohjelman tiloille pätee: $[wp(S, P) \wedge wp(S, Q)] = wp(S, P \wedge Q)$. (3)

Perustelu: Osoitetaan implikaatiot molempiin suuntiin:

$\Rightarrow$: Vasemmanpuoleinen tilapredikaatti $wp(S, P) \wedge wp(S, Q)$ takaa P:n ja Q:n voimassaolon S:n suorituksen jälkeen, ts. $P \wedge Q$ on tosi kun S päättyy.

$\Leftarrow$: Käytetään ominaisuutta 2 kahden implikaation osoittamiseen:

$$[(P \wedge Q) \Rightarrow P] \Rightarrow^{(2)} [wp(S, P \wedge Q) \Rightarrow wp(S, P)]$$

$$[(P \wedge Q) \Rightarrow Q] \Rightarrow^{(2)} [wp(S, P \wedge Q) \Rightarrow wp(S, Q)]$$

Nyt siis tiedetään, että

$$[wp(S, P \wedge Q) \Rightarrow wp(S, P)] \text{ ja } [wp(S, P \wedge Q) \Rightarrow wp(S, Q)],$$

toisin sanoen implikaatio $wp(S, P \wedge Q) \Rightarrow wp(S, P) \wedge wp(S, Q)$ pätee.

3.2.4 Ominaisuus 4 "Disjunktion distributiivisuus"

Kaikille ohjelman tiloille pätee: $[wp(S, P) \vee wp(S, Q)] \Rightarrow wp(S, P \vee Q)$. (4)

Perustelu: Käytetään jälleen ominaisuutta 2 kahden implikaation osoittamiseen

$$[P \Rightarrow (P \vee Q)] \Rightarrow^{(2)} [wp(S, P) \Rightarrow wp(S, P \vee Q)]$$

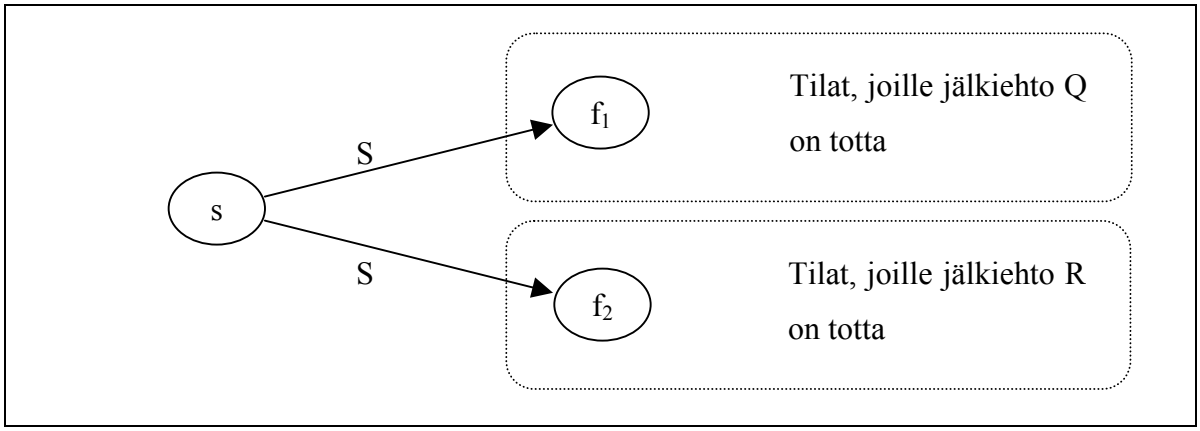
$$[Q \Rightarrow (P \vee Q)] \Rightarrow^{(2)} [wp(S, Q) \Rightarrow wp(S, P \vee Q)]$$

Siis $[wp(S, P) \Rightarrow wp(S, P \vee Q)]$ ja $[wp(S, Q) \Rightarrow wp(S, P \vee Q)]$,

ts. $[wp(S, P) \vee wp(S, Q)] \Rightarrow wp(S, P \vee Q)$.

3.2.4.1 Huomautus

Ominaisuus 4:n implikaatio ei "käänny" kun kyseessä on epädeterministinen lause. Annetaan tästä vastaesimerkkinä kolmitilainen automaatti (kts. seuraava kuva), jonka alkutilana on s, suoritettavana ohjelmana epädeterministinen S ja mahdollisina lopputiloina f_1 ja f_2. Edustakoon Q tilapredikaattia, joka on totta vain tilassa f_1 ja R tilapredikaattia, joka on totta vain tilassa f_2.



Kuva 1: Epädeterministisen lauseen S suorittaminen alkutilassa s johtaa joko lopputilaan f_1 tai f_2.

Mikäli implikaatio olisi voimassa myös toiseen suuntaan, ts.

$$[wp(S, Q) \vee wp(S, R)] = wp(S, Q \vee R)$$

olisi totta, niin silloin seuraava yhtäsuuruus olisi voimassa

{tilat, joille $wp(S, Q) \vee wp(S, R)$ on totta } = {tilat, joille $wp(S, R \vee Q)$ on totta}.

Kuitenkin

{tilat, joille esiehto $wp(S, Q)$ on totta} = $\varnothing$ ja

{tilat, joille esiehto $wp(S, R)$ on totta} = $\varnothing$, joten

{tilat, joille predikaatti $wp(S, Q) \vee wp(S, R)$ on totta } = $\varnothing$,

kun taas yhtälön oikea puoli on epätyhjä:

{tilat, joille esiehto $wp(S, Q \vee R)$ on totta} = { s }.

3.2.5 Ominaisuus 4'

Deterministiselle lauseelle S ja kaikille ohjelman tiloille pätee:

$$[wp(S, P) \vee wp(S, Q)] = wp(S, P \vee Q). \qquad (5)$$

Perustelu: Osoitetaan implikaatiot molempiin suuntiin:

$\Rightarrow$: Totta ominaisuus 4:n perusteella.

$\Leftarrow$: Koska S on deterministinen, jokaiselle alkutilalle on olemassa täsmälleen yksi lopputila, näin erityisesti alkutilalle, joka toteuttaa esiehdon $wp(S, P \vee Q)$. Tälle tilalle on - determinisyydestä johtuen - totta $wp(S, P)$ tai $wp(S, Q)$.

3.2.5.1 Huomautus

Viimeisen todistuksen ideana oli, että "mikäli jollekin lopputilalle predikaatti P on totta, niin sitä vastaavalle alkutilalle wp(S, P) on totta". Mikäli S olisi epädeterministinen, ts. mahdollisia lopputiloja olisi useampia, joukossa voi olla tila, jolle P ei ole totta, jolloin wp(S, P) ei olisi totta alkutilalle. Muistin virkistykseksi: wp(S, P) takaa, että "kun S suoritetaan, päädytään varmasti tilaan, jossa P on totta".

3.2.6 Kohti lopullista määritelmää

Edellä kuvatut ja perustellut ominaisuudet on siis tarkoitus ottaa predikaattimuuntimen määritelmäksi, ts. hyväksymme annetun matemaattisen olion predikaattimuuntimeksi, jos sillä voidaan osoittaa olevan olemassa kaikki luetellut ominaisuudet. Koska matemaatikon perushyve on laiskuus, haluaisimme kaikin tavoin helpottaa työtämme, mikäli joudumme osoittamaan jonkin olion predikaattimuuntimeksi.

Kahdelle ensimmäiselle ominaisuudelle emme voi mitään, sillä ne on johdateltu predikaattimuuntimen intuitiivisesta määritelmästä. Ominaisuus 4 on todistettu suoraan ominaisuudesta 2, voimme siis jättää sen määritelmän ulkopuolelle, samoiten käy ominaisuuden 3 implikaation oikealta vasemmalle ja ominaisuuden 4' implikaatiolle vasemmalta oikealle. Tuloksena saadaan seuraava määritelmä.

3.2.7 Määritelmä (Heikoimman esiehdon predikaattimuunnin)

Funktio wp: Programs $\times$ Predicates $\rightarrow$ Predicates on ohjelman S heikoimman esiehdon tuottava predikaattimuunnin, mikäli kaikissa tiloissa mielivaltaisille predikaateille P ja Q seuraavat väittämät ovat totta.

1. $wp(S, F) = F$ (o1)
2. Jos $P \Rightarrow Q$, niin $wp(S, P) \Rightarrow wp(S, Q)$ (o2)
3. $[wp(S, P) \wedge wp(S, Q)] \Rightarrow wp(S, P \wedge Q)$ (o3)

Mikäli S on deterministinen, seuraavan implikaation on lisäksi oltava totta.

4. $wp(S, P \vee Q) \Rightarrow [wp(S, P) \vee wp(S, Q)]$ (o4)

4 Ohjelmointikielen semantiikan esittäminen

Tässä luvussa mallinnamme edellä määritellyn Guarded Command -kielen semantiikkaa predikaattimuuntimien avulla. Idea on yksinkertainen: valitsemme tarkasteltavan syntaktisen rakenteen (esim. sijoituslause), keksimme ehdotelman predikaattimuuntimeksi ja testaamme, toteuttaako ehdotelma predikaattimuuntimen ominaisuudet 1—4. Mikäli vastaus on kyllä, olemme tyytyväisiä ja siirrymme seuraavaan rakenteeseen, muutoin jatkamme pohdintaa. Esitämme myös joitain esimerkkejä asioiden konkretisoimiseksi.

4.1 Puolipiste-operattorin heikoin esiehto

Guarded Command -kielessä käytetään puolipistettä lauseiden välillä erotinmerkkinä. Sen tarkoitus on siirtää ohjelman suoritus jatkumaan seuraavasta lauseesta. Haluamme, että ohjelman nykytilalla on samat ominaisuudet edellisen lauseen päätyttyä kuin seuraavan lauseen alkaessa, ts. Hoaren artikkelissa [8] esitetty yhdistämissääntö (engl. Rule of Composition)

$$\frac{\{ P \}\ S_1\ \{ R \},\ \{ R \}\ S_2\ \{ Q \}}{\{ P \}\ S_1; S_2\ \{ Q \}}$$

(kts. liite 1) on totta kaikille ohjelman tiloille.

Predikaattimuunnin tuottaa annetun ohjelmanpätkän ja jälkiehdon perusteella heikoimman esiehdon, joka johtaa ohjelman pysähtymiseen ja jälkiehdon täyttymiseen. Ohjelmanpätkä tässä tilanteessa muodostuu kahdesta peräkkäisestä lauseesta S_1 ja S_2, jotka on erotettu toisistaan puolipisteellä. Jälkiehto olkoon mielivaltainen predikaatti Q ja predikaattimuuntimen tuottama heikoin esiehto P. Pitäisi siis löytää jokin olio wp, joka toteuttaa yhtälön

$$wp("S_1; S_2", Q) = P.$$

Voimme olettaa, että lauseelle S_2 on olemassa predikaattimuunnin $wp(S_2, Q)$, joka tuottaa heikoimman esiehdon R siten, että S_2 pysähtyy ja predikaatti Q astuu voimaan, ts.

$$wp(S_2, Q) = R$$

tunnetaan. Huomataan, että R:n tulee olla voimassa lauseen S_1 suorituksen jälkeen, jotta S_2 varmasti pysähtyisi ja tuottaisi halutun tuloksen Q. Oletetaan, että lauseelle S_1 ja sen jälkiehdolle R on olemassa predikaattimuunnin $wp(S_1, R)$, jonka tulos takaa lauseen S_1 suorituksen päättymisen ja predikaatin R toteutumisen, toisin sanoen $wp(S_1, R)$ tuottaa S_1:n heikoimman esiehdon, joka takaa Q:n toteutumisen S_1:n ja S_2:n suorituksen jälkeen, eli $wp(S_1, R)$ tuottaa alussa määritellyn ehdon P. Yhdistämällä palaset

$$P = wp(S_1, R) \text{ ja } R = wp(S_2, Q)$$

saadaan kaava

$$wp(S_1, wp(S_2, Q)) = P,$$

joka on ehdokkaamme puolipisteoperaation heikoimman esiehdon tuottavaksi predikaattimuuntimeksi. Nyt on vielä tarkistettava, että tämä ehdokas kelpaa predikaattimuuntimeksi osoittamalla, että sillä on predikaattimuuntimen ominaisuudet 1—4 (ominaisuuden 4 oletukset ovat voimassa vain jos S_1 ja S_2 ovat deterministisiä).

4.1.1 Ominaisuus 1

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$wp(S_1, F) = F, \tag{6}$$

$$wp(S_2, F) = F \text{ ja} \tag{7}$$

$$wp("S_1; S_2", P) = wp(S_1, wp(S_2, P)) \text{ kaikille predikaateille P.} \tag{8}$$

Väite:

$$wp("S_1; S_2", F) = F.$$

Todistus :

$$wp("S_1; S_2", F) =^{(8)}$$

$wp(S_1, wp(S_2, F)) =^{(7)}$

$wp(S_1, F) =^{(6)}$

F.

4.1.2 Ominaisuus 2

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$P \Rightarrow Q, \tag{9}$$

$$wp(S_2, P) \Rightarrow wp(S_2, Q) \tag{10}$$

$$wp("S_1; S_2", P) = wp(S_1, wp(S_2, P)) \text{ ja} \tag{11}$$

$$wp("S_1; S_2", Q) = wp(S_1, wp(S_2, Q)). \tag{12}$$

Väite:

$$wp("S_1; S_2", P) \Rightarrow wp("S_1; S_2", Q).$$

Todistus:

$wp("S_1; S_2", P) =^{(11)}$

$wp(S_1, wp(S_2, P)) \Rightarrow^{(o2 \,\&\, 10)}$

$wp(S_1, wp(S_2, Q)) =^{(12)}$

$wp("S_1; S_2", Q).$

4.1.3 Ominaisuus 3

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$[wp(S_1, R) \wedge wp(S_1, S)] \Rightarrow wp(S_1, R \wedge S), \tag{13}$$

$$[wp(S_2, P) \wedge wp(S_2, Q)] \Rightarrow wp(S_2, P \wedge Q), \quad (14)$$

$$wp("S_1; S_2", P) = wp(S_1, wp(S_2, P)), \quad (15)$$

$$wp("S_1; S_2", Q) = wp(S_1, wp(S_2, Q)) \text{ ja} \quad (16)$$

$$wp("S_1; S_2", P \wedge Q) = wp(S_1, wp(S_2, P \wedge Q)). \quad (17)$$

Väite:

$$[wp("S_1; S_2", P) \wedge wp("S_1; S_2", Q)] \Rightarrow wp("S_1; S_2", P \wedge Q)$$

Todistus:

$$wp("S_1; S_2", P) \wedge wp("S_1; S_2", Q) =^{(15 \,\&\, 16)}$$

$$wp(S_1, wp(S_2, P)) \wedge wp(S_1, wp(S_2, Q)) \Rightarrow^{(13)}$$

$$wp(S_1, wp(S_2, P) \wedge wp(S_2, Q)) \Rightarrow^{(o2 \,\&\, 14)}$$

$$wp(S_1, wp(S_2, P \wedge Q)) =^{(17)}$$

$$wp("S_1; S_2", P \wedge Q).$$

4.1.4 Ominaisuus 4

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$wp(S_1, R \vee S) \Rightarrow [wp(S_1, R) \vee wp(S_1, S)], \quad (18)$$

$$wp(S_2, P \vee Q) \Rightarrow [wp(S_2, P) \vee wp(S_2, Q)] \text{ ja} \quad (19)$$

$$wp("S_1; S_2", P \vee Q) = wp(S_1, wp(S_2, P \vee Q)). \quad (20)$$

Väite:

$$wp("S_1; S_2", P \vee Q) \Rightarrow [wp("S_1; S_2", P) \vee wp("S_1; S_2", Q)].$$

Todistus:

$$wp(\text{"}S_1; S_2\text{"}, P \vee Q) =^{(20)}$$

$$wp(S_1, wp(S_2, P \vee Q)) \Rightarrow^{(o2\ \&\ 19)}$$

$$wp(S_1, wp(S_2, P) \vee wp(S_2, Q)) \Rightarrow^{(o1)}$$

$$wp(S_1, wp(S_2, P)) \vee wp(S_1, wp(S_2, Q)) =^{(20)}$$

$$wp(\text{"}S_1; S_2\text{"}, P) \vee wp(\text{"}S_1; S_2\text{"}, Q).$$

4.2 Sijoituslause

Sijoituslauseen tehtävänä on vaihtaa jonkin muuttujan arvoa. Edellytämme, että kaikki se, mikä on totta sijoitettavan lausekkeen arvolle ennen sijoitusoperaation suorittamista, on voimassa muuttujan arvolle sijoituksen jälkeen. Muistamme Hoaren artikkelin [8] sijoitusaksiooman { P[E/x] } x := E { P } (kts. liite 1) ja muotoilemme sitä vastaavan ehdokkaan sijoitusoperaation predikaattimuuntimen määritelmäksi:

$$wp(\text{"x := E"}, P) = P[E/x].$$

Tarkistetaan seuraavaksi ominaisuuksien 1—4 voimassaolo.

Ominaisuus 1

$$wp(\text{"x := E"}, F) =^{(määr.)} F[E/x] = F.$$

Ominaisuus 2

$$wp(\text{"x := E"}, P) =^{(määr.)}$$

$$P[E/x] \Rightarrow^{(oletuksen\ mukaan)}$$

$$Q[E/x] =^{(määr.)}$$

$$wp(\text{"x := E"}, Q).$$

Ominaisuus 3

$$wp("x := E", P) \wedge wp("x := E", Q) =^{(määr.)}$$

$$P[E/x] \wedge Q[E/x] =$$

$$(P \wedge Q)[E/x] =^{(määr.)}$$

$$wp("x := E", P \wedge Q)$$

Ominaisuus 4

$$wp("x := E", P \vee Q) =^{(määr.)}$$

$$(P \vee Q)[E/x] =$$

$$P[E/x] \vee Q[E/x] =^{(määr.)}$$

$$wp("x := E", P) \vee wp("x := E", Q)$$

Joissain lähteissä ohjelmien selkeyttämiseksi käytetään yhtäaikaisia sijoituslauseita (engl. concurrent assignment), jotka ovat muotoa $x_1, ..., x_n := E_1, ..., E_n$, missä i:nnen muuttujan (merkitään symbolilla x_i) arvoksi tulee i:nnen lausekkeen arvo (merkitään E_i). Yhtäaikaisia sijoituksia käytettäessä on huomattava, että sama muuttujasymboli saa esiintyä vasemmalla puolella korkeintaan yhden kerran.

4.3 Skip-lause

Skip-lauseella halutaan kuvata "ei minkään tekemistä". Haluamme, että tilapredikaattien totuusarvot eivät muutu skip-lauseen suorittamisen tuloksena, ts. { P } skip { P } on totta kaikille predikaateille P kaikissa suorittimen tiloissa. Sama kirjoitettuna wp-merkinnällä:

$$wp("skip", P) = P.$$

Tarkistetaan predikaattimuuntimen ominaisuudet 1—4.

4.3.1 Ominaisuus 1

$$wp("skip", F) =^{(määr.)} F.$$

4.3.2 Ominaisuus 2

Oletus: $P \Rightarrow Q$ kaikissa ohjelman tiloissa (21)

Todistus: $wp("skip", P) =^{(määr.)} P \Rightarrow^{(21)} Q =^{(määr.)} wp("skip", Q)$.

4.3.3 Ominaisuus 3

$$wp("skip", P) \wedge wp("skip", Q) =^{(määr.)} P \wedge Q =^{(määr.)} wp("skip", P \wedge Q) .$$

4.3.4 Ominaisuus 4

$$wp("skip", P \vee Q) =^{(määr.)} P \vee Q =^{(määr.)} wp("skip", P) \vee wp("skip", Q) .$$

4.4 Abort-lause

Abort-lauseen tehtävänä on keskeyttää ohjelman suoritus, ts. se ei missään alkutilassa suoritettuna voi johtaa lopputilaan, jolle predikaatti P on totta. Tämä kirjoitettuna wp-merkinnöin antaa

$$wp("abort", P) = F,$$

mille tahansa predikaatille P. Saadulle predikaattimuuntimelle ominaisuudet 1—4 pätevät määritelmän nojalla triviaalisti.

4.5 IF-lause

Olkoon $B_1,...,B_n$ joukko suoritusehtoja ja $S_1,...,S_n$ niitä vastaavat lauseet. Rakenteen

$$\text{IF } B_1 \rightarrow S_1 \square ... \square B_n \rightarrow S_n \text{ FI}$$

heikoin esiehto jälkiehdolle R, merkitään wp(IF, R), vaatii, että vähintään yksi suoritusehdoista $B_1,...,B_n$ on tosi ja että minkä tahansa lauseen $S_1,...,S_n$ suorittaminen johtaa tilaan, jolle tilapredikaatti R on tosi, ts.

$$wp(IF, R) = (BB \wedge \forall i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, R)),$$

missä BB tarkoittaa lauseketta $\exists i \in \{1,..,n\}: B_i$, ts. vähintään yksi suoritusehto on aina tosi. Osoitetaan seuraavaksi wp(IF, R) predikaattimuuntimeksi tarkistamalla ominaisuuksien 1—3 olemassaolo (ominaisuus 4 ei päde, sillä IF-lause on epädeterministinen). Kuten puolipisteoperaattorin yhteydessä, voimme tässäkin olettaa, että IF-lauseen sisältämillä lauseilla S_i, $i \in \{1,..,n\}$, on olemassa predikaattimuuntimet.

4.5.1 Ominaisuus 1

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$\forall i \in \{1,..,n\}: wp(S_i, F) = F \text{ ja} \tag{22}$$

$$wp(IF, F) = [BB \wedge (\forall i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, F))]. \tag{23}$$

Väite:

$$wp(IF, F) = F.$$

Todistus:

$$wp(IF, F) =^{(23)}$$

$$(\exists i \in \{1,..,n\}: B_i) \wedge (\forall i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, F)) =^{(22)}$$

$$(\exists i \in \{1,..,n\}: B_i) \wedge (\forall i \in \{1,..,n\}: B_i \Rightarrow F) =^{(\text{implikaation totuustaulu})}$$

$$(\exists i \in \{1,..,n\}: B_i) \wedge (\forall i \in \{1,..,n\}: \neg B_i) =$$

$$F.$$

Toiseksi viimeisellä rivillä sanotaan, että "on olemassa B_i, joka on totta" ja "kaikki B_i:t ovat epätotta". Väite on siis muotoa "P ja $\neg$P", joka on aina epätotta.

4.5.2 Ominaisuus 2

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$P \Rightarrow Q \tag{24}$$

$$\forall\, i \in \{1,..,n\}: wp(S_i, P) \Rightarrow wp(S_i, Q) \tag{25}$$

$$wp(IF, P) = [\, BB \wedge (\forall\, i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, P)\,)\,] \tag{26}$$

$$wp(IF, Q) = [\, BB \wedge (\forall\, i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, Q)\,)\,] \tag{27}$$

Väite:

$$wp(IF, P) \Rightarrow wp(IF, Q).$$

Todistus:

$$wp(IF, P) =^{(26)}$$

$$BB \wedge (\forall\, i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, P)\,) \Rightarrow^{(25)}$$

$$BB \wedge (\forall\, i \in \{1,..,n\}: B_i \Rightarrow wp(S_i, Q)\,) =^{(27)}$$

$$wp(IF, Q).$$

Todistuksen toisen rivin implikaatiossa on kyse seuraavanlaisesta päättelystä: "Tiedetään, että kaikilla i pätee $B_i \Rightarrow wp(S_i, P)$ ja $wp(S_i, P) \Rightarrow wp(S_i, Q)$, mistä seuraa, että kaikilla i myös $B_i \Rightarrow wp(S_i, Q)$." Päättelyn oikeellisuuden voi tarkistaa totuustaululla (totuustaulun rivejä numeroivat väitteiden B_i, $wp(S_i, P)$ ja $wp(S_i, Q)$ kombinaatiot, yhteensä siis 8 kpl, ja väittämä $[(B_i \Rightarrow wp(S_i, P)) \wedge (wp(S_i, P) \Rightarrow wp(S_i, Q))] \Rightarrow [B_i \Rightarrow wp(S_i, Q)]$ on kaikilla kombinaatioilla tosi), kuten liitteessä 2 on tehty.

4.5.3 Ominaisuus 3

Oletukset (voimassa kaikissa ohjelman tiloissa):

$$\forall\, i \in \{1,..,n\}\colon [wp(S_i, P) \wedge wp(S_i, Q)] \Rightarrow wp(S_i, P \wedge Q), \tag{28}$$

$$wp(IF, P) = [\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow wp(S_i, P)\ )\ ], \tag{29}$$

$$wp(IF, Q) = [\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow wp(S_i, Q)\ )\ ] \text{ ja} \tag{30}$$

$$wp(IF, P \wedge Q) = [\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow wp(S_i, P \wedge Q)\ )\ ]. \tag{31}$$

Väite:

$$wp(IF, P) \wedge wp(IF, Q) \Rightarrow wp(IF, P \wedge Q).$$

Todistus:

$$wp(IF, P) \wedge wp(IF, Q) =^{(29\ \&\ 30)}$$

$$[\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow wp(S_i, P)\ )\ ] \wedge$$

$$[\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow wp(S_i, Q)\ )\ ] =$$

$$[\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow (wp(S_i, P) \wedge wp(S_i, Q))\ )\ ] \Rightarrow^{(28)}$$

$$[\ BB \wedge (\forall\, i \in \{1,..,n\}\colon B_i \Rightarrow wp(S_i, P \wedge Q)\ )\ ] =^{(31)}$$

$$wp(IF, P \wedge Q).$$

4.6 DO-lause

Olkoon $B_1,...,B_n$ joukko suoritusehtoja ja $S_1,...,S_n$ niitä vastaavat lauseet. Rakenteen

$$DO\ B_1 \rightarrow S_1\ \square\ ...\ \square\ B_n \rightarrow S_n\ OD$$

jälkiehto $H_0(P)$ on muotoa $\{ \neg BB \wedge P \}$, missä BB tarkoittaa lauseketta $\exists\, i \in \{1,..,n\}: B_i$. Esiehdon tulisi johtaa $H_0(P)$:n toteutumiseen äärellisen monen iteraatiokerran jälkeen. Määritellään rekursiivisesti

$$\forall\, i > 0: H_i(P) = (wp(IF, H_{i-1}(P)) \vee H_0(P)),$$

missä IF tarkoittaa samaa <guarded command set> -esiintymää kirjoitettuna IF..FI -rakenteen sisään, ts. vähintään yksi suoritusehdoista $B_1,...,B_n$ on tosi ja minkä tahansa lauseen $S_1,...,S_n$ suorittaminen johtaa $H_{i-1}(P)$:n toteutumiseen. Nyt voimme kirjoittaa ehdokkaan DO-rakenteen heikoimmalle esiehdolle muodossa

$$wp(DO, P) = \exists\, i \geq 0 : H_i(P).$$

Seuraavaksi tarkistamme, että saatu predikaattimuunnin toteuttaa ominaisuudet 1—3. Kuten arvata saattaa, todistuksissa tarvitaan induktiota silmukan suorituskertojen maksimimäärän suhteen.

4.6.1 Ominaisuus 1

Väite: $wp(DO, F) = F$.

Väite sanoo, että ei ole olemassa alkutilaa siten, että DO-silmukan suoritus johtaisi lopputilaan, jolle tilapredikaatti F on tosi. Päättelyn

$$(wp(DO, F) = F) =^{(määr.)}$$

$$((\exists\, i \geq 0 : H_i(F)) = F) =$$

$$\forall\, i \geq 0 : (H_i(F) = F)$$

nojalla väite voidaan kirjoittaa muodossa $\forall\, i \geq 0 : (H_i(F) = F)$, joka voidaan osoittaa oikeaksi induktiolla muuttujan i suhteen.

1° Väite pätee, kun $i = 0$, sillä $H_0(F) =^{(määr.)} (\neg BB \wedge F) = F$.

2° Induktio-oletus: Väite pätee, kun $i = K-1$, missä $K \geq 1$, ts. $H_{K-1}(F) = F$.

3° Induktioaskel: Väite pätee, kun i = K, missä K ≥ 1, sillä

$$H_K(F) =^{(määr.)}$$

$$wp(IF, H_{K-1}(F)) \vee H_0(F) =^{(induktio\text{-}oletus\ \&\ 1°)}$$

$$wp(IF, F) \vee F =^{(IF\text{-}lauseen\ predikaattimuuntimen\ määritelmä)}$$

$$F \vee F =$$

$$F$$

on totta.

4.6.2 Ominaisuus 2

Väite: Jos kaikissa tiloissa pätee $P \Rightarrow Q$, niin myös $wp(DO, P) \Rightarrow wp(DO, Q)$ on totta kaikissa tiloissa.

Implikaation määritelmän nojalla tiedämme, että väite pätee triviaalisti, mikäli implikaatio $P \Rightarrow Q$ ei ole totta. Oletetaan siis implikaation

$$P \Rightarrow Q \tag{32}$$

olevan totta kaikissa tiloissa. Väitteen oikea puoli

$$wp(DO, P) \Rightarrow wp(DO, Q)$$

voidaan DO-lauseen predikaattimuuntimen määritelmän nojalla kirjoittaa muodossa

$$(\exists\, i \geq 0 : H_i(P)) \Rightarrow (\exists\, j \geq 0 : H_j(Q)). \tag{33}$$

Implikaation määritelmän nojalla väite pätee, mikäli kaavan (33) vasen puoli ei ole totta. Riittää siis tarkastella kaavan (33) voimassaoloa silloin kun löytyy $i \geq 0$ siten, että $H_i(P)$ on totta.

Nyt pitäisi löytää sellainen $j \geq 0$, jolla $H_j(Q)$ on totta. Kokeillaan valintaa $j = i$. Väite pätee, mikäli voidaan osoittaa implikaation $H_i(P) \Rightarrow H_i(Q)$ olevan totta kaikille $i \geq 0$, ts.

$$\forall\, i \geq 0 : (H_i(P) \Rightarrow H_i(Q)) \qquad (34)$$

pätee. Osoitetaan kaava (34) todeksi induktiolla muuttujan i suhteen.

1° Väite pätee, kun $i = 0$, sillä

$$H_0(P) \Rightarrow^{(määr.)}$$

$$(\neg BB \wedge P) \Rightarrow^{(34)}$$

$$(\neg BB \wedge Q) =^{(määr.)}$$

$$H_0(Q).$$

2° Induktio-oletus: Väite pätee, kun $i = K-1$, missä $K \geq 1$, ts.

$$H_{K-1}(P) \Rightarrow H_{K-1}(Q).$$

3° Induktioaskel: Väite pätee, kun $i = K$, missä $K \geq 1$, sillä

$$H_K(P) =^{(määr.)}$$

$$[\ wp(IF, H_{K-1}(P)) \vee H_0(P)\ ] \Rightarrow^{(1° \,\&\, 2° \,\&\, \text{IF-lauseen predikaattimuuntimen monotonisuus})}$$

$$[\ wp(IF, H_{K-1}(Q)) \vee H_0(Q)\ ] =^{(määr.)}$$

$$H_K(Q)$$

on totta.

4.6.3 Ominaisuus 3

Väite: Kaikissa tiloissa pätee $wp(DO, P) \wedge wp(DO, Q) \Rightarrow wp(DO, P \wedge Q)$.

Todistus sivutetaan.

5 WP-laskennan oikeellisuus ja täydellisyys

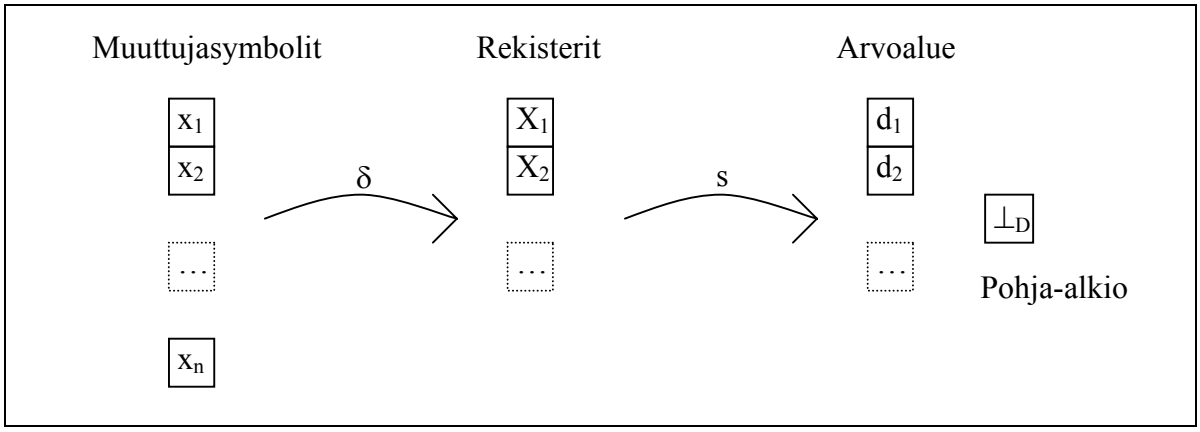
WP-laskenta tarjoaa keinon mallintaa ohjelmien semanttisia ominaisuuksia, esimerkiksi millä tavalla suorittimen tila muuttuu ohjelman ajon aikana ja mitkä ovat vähimmäisedellytykset ohjelman onnistuneelle suoritukselle. Tämän luvun tehtävänä on tutkia, onko edellä kehitelty matemaattinen koneisto edes teoriassa käyttökelpoista, ts. pitääkö kaikki WP-laskennalla päätelty paikkansa ja riittääkö WP-laskenta kuvaamaan kaikkien ohjelmien käyttäytymistä. Nämä kaksi kysymystä tunnetaan myös nimellä oikeellisuus- ja täydellisyyskysymykset ja niitä ovat Hoare-tyyppisille järjestelmille tutkineet mm. Cook 1978 [3], Clarke 1979 [2] ja Apt 1988 [1]. Seuraavaksi määritellään joukko käsitteitä, joiden avulla WP-laskennan oikeellisuus- ja täydellisyyskysymyksiä kyetään käsittelemään lähteen [3] mukaisesti.

Tulkintamalli tarkoittaa samaa kuin edellä käytetty termi "suoritin". Mallilla on ääretön joukko rekisterejä, merkitään X_1, X_2, ..., joiden arvokombinaatiot määräävät mallin kulloisenkin tilan. Malli vaihtaa tilaansa ohjelman suorituksen aikana sivuilla 29 ja 30 esitettävän Comp-funktion mukaisesti. Tulkinta I on kolmikko <D, P, F>, missä D on arvojoukko (esim. Scottin alue [7]), P = { P_1, P_2, … } ovat predikaattikielen L_A predikaattisymboleja tulkitsevia D:n predikaatteja ja F = { f_1, f_2, … } ovat L_A:n funktiosymboleja tulkitsevia D:n funktioita.

Olkoon L_E kieli, jolla kuvataan arvojoukon alkiot (esim. luonnolliset luvut), tarvittavat laskutoimitukset sekä vertailuoperaatiot (esim. +, ·, =, <) ja L_A kielen L_E laajennuskieli[2], joka sisältää edellisten lisäksi myös loogiset operaattorit ($\wedge$, $\vee$, $\neg$) ja kvanttorit ($\forall$ ja $\exists$). L_E-kieltä käytetään laskulausekkeiden kuvaamiseen (luvun 2 BNF-kuvauksen <addition>-produktio) Guarded Command –kielisissä ohjelmissa ja L_A-kieltä kuvaamaan tilapredikaatteja. Muuttujasijoitus (engl. variable assignment, kääntäjätekniikassa register

[2] Termi "laajennuskieli" ei tarkoita samaa kuin predikaattilaskennan laajennuskieli (= sama kieli laajemmalla indeksijoukolla), joka on määritelty mm. lähteessä [10, s. 66].

allocation) δ on osittainen injektio kielen L_A muuttujasymboleilta mallin rekistereille ja tila s on osittainen funktio rekistereiltä arvojoukolle D (kts. kuva seuraava kuva).



Kuva 2: Kielen L_A muuttujasymbolit, mallin rekisterit, arvoalue D sekä funktiot s ja δ.

Lausekkeen $E \in L_E$ arvo tilassa s muuttujasijoituksella δ, merkitään E(s, δ), on

$$E(s, \delta) = E[\ s(\delta(x_1)), \ldots, s(\delta(x_n)) / x_1, \ldots, x_n\ ]$$

ja vastaavasti tilapredikaatin $P \in L_A$ totuusarvo tilassa s muuttujasijoituksella δ on

$$P(s, \delta) = P[\ s(\delta(x_1)), \ldots, s(\delta(x_n)) / x_1, \ldots, x_n\ ].$$

Kolmikko { P } S { Q } on tosi mallissa M, joss kaikissa tiloissa mielivaltaiselle δ pätee

$$P(s, \delta) \Rightarrow Q(\ Out(S, s, \delta), \delta\ ),$$

missä Out(S, s, δ) on se tila, johon päädytään suoritettaessa S alkutilasta s muuttujasijoituksella δ. Kolmikko on validi, joss se on tosi kaikissa malleissa. Vastaavasti päättelysääntö

$$\frac{\alpha_1, \ldots, \alpha_n}{\beta}$$

on validi, joss kaikissa malleissa, joissa $\alpha_1, \ldots, \alpha_n$ ovat tosia, myös β on tosi.

Lopuksi määrittelemme funktion, joka tuottaa syötteinä annettujen ohjelman, alkutilan ja muuttujasijoituksen avulla tilasekvenssin, joka kertoo, mitä tiloja malli käy läpi Guarded Command –kielellä kirjoitetun ohjelman suorituksen aikana. Annamme funktiolle nimeksi Comp (kuten Cookin artikkelissa [3]).

Comp on funktio ohjelmointikielen lauseiden, tilojen ja muuttujasijoitusten tulojoukolta tiloista koostuvien sekvenssien joukolle. Seuraavan määrityksen jokainen rivi koostuu ohjelmointikielen lausetyypistä, nuolesta ja lausetyypin mukaisen lauseen suorituksen tuloksena saatavasta tilasekvenssistä.

Suoritettavaa ohjelmointikielen lausetta merkitään isolla S-kirjaimella, alkutilaa pienellä s-kirjaimella, muuttujasijoitusta δ-symbolilla (delta) sekä muuttujia pienillä ja rekisterejä isoilla X-kirjaimilla. Merkintä $<s_1, s_2, ...>$ tarkoittaa tilojen s_1, s_2, ... muodostamaa sekvenssiä ja hattumerkkiä (^) käytetään ilmaisemaan sekvenssien konkatenaatiota. Out(S, s, δ) tarkoittaa sekvenssin Comp(S, s, δ) viimeistä alkiota ja $<\infty>$ tarkoittaa päättymätöntä sekvenssiä. Out(S, s, δ) on määritelty vain jos S ei jää ikuiseen silmukaan kun laskenta aloitetaan tilasta s muuttujasijoituksella δ.

Comp(S, s, δ) =

$$x := E \rightarrow <s'> \text{, missä } s'(X_i) = \begin{cases} s(X_i), \text{ jos } \delta(x) \neq X_i \\ E(s, \delta), \text{ jos } \delta(x) = X_i \end{cases}$$

$$S_1; S_2 \rightarrow \text{Comp}(S_1, s, \delta)\text{^Comp}(S_2, \text{Out}(S_1, s, \delta), \delta)$$

$$\text{IF} \rightarrow \begin{cases} \text{Comp}(S_i, s, \delta) & \text{, jollekin i, jolle } B_i(s, \delta) \text{ totta} \\ <\infty> \text{ (päättymätön sekvenssi)} & \text{, mikäli kaikki suoritusehdot ovat epätosia} \end{cases}$$

$$\text{DO} \rightarrow \begin{cases} \text{Comp}(S_i, s, \delta)\text{^Comp}(\text{DO}, \text{Out}(S_i, s, \delta), \delta) & \text{, jollekin i, jolle } B_i(s, \delta) \text{ totta} \\ <s> & \text{, muulloin} \end{cases}$$

skip → <s>

abort → <∞> (päättymätön sekvenssi)

5.1.1 Huomioita Comp-funktion määritelmästä

Yhdestä skip-lauseesta koostuvaa ohjelmaa ajaessaan järjestelmä suorittaa tilasiirtymän alkutilasta itseensä, ts. se läpikäy kaksi tilaa. Vastaavaan määrittelyyn ovat päätyneet myös Cook [3, s. 74] ja Clarke [2, s. 132] "begin end"-rakenteensa suhteen. (Toinen mahdollisuus - jonka valinta ei vaikuttaisi tuleviin laskuihin lainkaan - on tyhjän sekvenssin valitseminen kuvaamaan skip-lauseen suoritusta.)

Dijkstran [4, s. 454] mukaan DO-lause, jolla ei ole lainkaan tosia suoritusehtoja, on "semanttisesti ekvivalentti" (engl. semantically equivalent) skip-lauseen kanssa. Tästä syystä Comp tuottaa alkutilan itsensä (kuten skip-lauseen tapauksessa) mikäli kaikki suoritusehdot ovat epätosia. Edelleen lähteet [3] ja [2] ovat samaa mieltä.

Kappaleen 4.4 mukaan abort-lauseen tarkoitus on ohjelman suorituksen keskeyttäminen tai kuten Dijkstra [5, s. 26] asian ilmaisee: *"When evoked, the mechanism named 'abort' will therefore fail to reach a final state: its attempted activation is interpreted as a symptom of failure."* Ensimmäiseksi sanasta "keskeytys" tulee mieleen tilasekvenssin päättäminen siihen tilaan, jossa oltiin ennen kuin abort-lauseen suoritus alkoi. Tällöin kuitenkin tilasekvenssillä olisi yksikäsitteinen lopputila ja Dijkstran mainitsema ehto "*fail to reach a final state*" ei tule täytetyksi. Vastaavalla tavalla pääteltynä mikään äärellinen sekvenssi ei käy abort-lauseen suoritusta kuvaavaksi toiminnaksi. Ainoaksi vahtoehdoksi jää siis äärettömän pituinen sekvenssi.

Mielenkiintoinen ero Cookin [3] ja Clarken [2] artikkeleissa löytyy "if-then-else"-lauseen käsittelyssä. Cookin määritelmän mukaan Comp palauttaa if-lausetta suorittaessaan sekvenssin, jonka ensimmäisenä alkiona on alkutila, ts. ohjelmat

"skip; S_1" ja "if E then S_1 else S_2 fi"

tuottavat täsmälleen samat tilasekvenssit, mikäli E(s, δ) on alkutilassa s tosi. Clarken mukaan alkutilaa ei toisteta, mikä edellisten esimerkkiohjelmien perusteella tuntuu järkevämmältä. Kysymys lieneekin painovirheestä artikkelissa [3].

5.2 Oikeellisuus

Tässä kappaleessa vastataan kysymykseen: “Onko kaikki WP-laskennan aksioomilla ja päättelysäännöillä päätelty totta?”. Ideana on tarkastella kysymystä eri mittaisille (ts. kuinka monta kertaa aksioomia ja päättelysääntöjä sovelletaan) päättelyille samaan tapaan kuin lähteessä [10] on propositiologiikan päättelyjonon pituuden suhteen tehty. Induktiivisen todistuksen alkuoletuksena tarkistetaan, että aksioomat ovat valideja, jonka jälkeen käydään läpi IF- ja DO-lauseen sekä puolipisteoperaattorin päättelysäännöt olettaen induktiivisesti, että niissä esiintyville komponenttilauseille S_1, …, S_n on olemassa validit päättelysäännöt. Tarkastelua ei ole kirjoitettu skip- ja abort-lauseiden päättelysääntöjen suhteen.

5.2.1 Sijoitusaksiooman validius

On siis osoitettava, että kaava { P[E/x] } x:= E { P } on totta kaikilla malleilla, mikä määritelmän mukaan tarkoittaa implikaation

$$P[E/x_i](s, \delta) \Rightarrow P(Out(\text{“}x:=E\text{”}, s, \delta), \delta)$$

voimassaoloa mielivaltaisille x ja δ. (Alaindeksi i on lisätty asioiden selkeyttämiseksi jatkossa.)

Olkoon X_k sijoituksen δ muuttujaan x_i liittämä rekisteri. Tällöin Comp-funktion määritelmän nojalla lopputila s’ = Out(“x:=E”, s, δ) liittää rekisteriin X_k lausekkeen E(s, δ) arvon. Predikaatin P totuusarvoksi lopputilassa s’ saadaan

$$P(s', \delta) \Leftrightarrow^{(\text{määr.})}$$

$$P[s'(\delta(x_1)), \ldots, s'(\delta(x_i)), \ldots, s'(\delta(x_n)) / x_1, \ldots, x_i, \ldots, x_n] \Leftrightarrow^{(\delta \text{ injektio})}$$

$$P[s'(\delta(x_1)), \ldots, s'(X_k), \ldots, s'(\delta(x_n)) / x_1, \ldots, x_i, \ldots, x_n] \Leftrightarrow$$

$$P[s'(\delta(x_1)), \ldots, E(s, \delta), \ldots, s'(\delta(x_n)) / x_1, \ldots, x_i, \ldots, x_n].$$

Koska Comp-funktion määritelmän nojalla $s(\delta(x_j)) = s'(\delta(x_j))$ pätee kaikille $j \in \{1, \ldots n\}$, missä $j \neq i$, niin viimeinen rivi voidaan kirjoittaa muodossa

$$P[s(\delta(x_1)), \ldots, E(s, \delta), \ldots, s(\delta(x_n)) / x_1, \ldots, x_i, \ldots, x_n],$$

joka edelleen voidaan yhtäpitävästi kirjoittaa muotoon

$$P[E(s, \delta)/x_i][s(\delta(x_1)), \ldots, s(\delta(x_i)), \ldots, s(\delta(x_n)) / x_1, \ldots, x_i, \ldots, x_n],$$

mikä tarkoittaa predikaatin $P[E/x_i](s, \delta)$ totuusarvoa. Näin on todistettu ekvivalenssi

$$P(s', \delta) \Leftrightarrow P[E/x_i](s, \delta)$$

ja erityisesti siis implikaatio $P[E/x_i](s, \delta) \Rightarrow P(s', \delta)$ pätee.

5.2.2 Yhdistyssäännön validius

Tehtävänä on osoittaa kaavan $\{ wp(\text{“}S_1; S_2\text{”}, P) \} S_1; S_2 \{ P \}$ validius, ts. implikaation

$$wp(\text{“}S_1; S_2\text{”}, P)(s, \delta) \Rightarrow P(Out(\text{“}S_1; S_2\text{”}, s, \delta), \delta)$$

voimassaolo mielivaltaiselle alkutilalle s ja muuttujasijoitukselle δ.

Puolipisteoperaattorin predikaattimuuntimen määritelmän mukaan pätee

$$wp(\text{”}S_1; S_2\text{”}, P) = wp(S_1, wp(S_2, P)), \quad (35)$$

ja induktio-oletuksen nojalla implikaatioiden

$$wp(S_1, wp(S_2, P))(s, \delta) \Rightarrow wp(S_2, P)(s', \delta) \text{ ja} \quad (36)$$

$$wp(S_2, P)(s', \delta) \Rightarrow P(s'', \delta), \quad (37)$$

missä

$$s' = Out(S_1, s, \delta) \text{ ja}$$

$$s'' = Out(S_2, Out(S_1, s, \delta), \delta), \quad (38)$$

voidaan olettaa olevan voimassa.

Nyt nähdään, että väite pätee, sillä

$$wp(\text{”}S_1; S_2\text{”}, P)(s, \delta) =^{(35)}$$

$$wp(S_1, wp(S_2, P))(s, \delta) \Rightarrow^{(36)}$$

$$wp(S_2, P)(s', \delta) \Rightarrow^{(37)}$$

$$P(s'', \delta) =^{(38)}$$

$$P(Out(S_2, Out(S_1, s, \delta), \delta), \delta) =^{(\text{Comp-funktion määr.})}$$

$$P(Out(\text{“}S_1; S_2\text{”}, s, \delta), \delta)$$

on voimassa.

5.2.3 IF-lauseen päättelysäännön validius

On osoitettava kaava $wp(IF, P)(s, \delta) \Rightarrow P(Out(IF, s, \delta), \delta)$ validiksi. IF-lauseen predikaattimuuntimen määritelmän mukaan pätee

$$wp(IF, P) = (\exists\, i \in \{1, \ldots, n\} : B_i) \wedge (\forall\, i \in \{1, \ldots, n\} : B_i \Rightarrow wp(S_i, P)),$$

ja induktio-oletuksen nojalla kaavojen

$$\exists\, i \in \{1, \ldots, n\} : B_i(s, \delta), \tag{39}$$

$$\forall\, i \in \{1, \ldots, n\} : B_i(s, \delta) \Rightarrow wp(S_i, P)(s, \delta) \text{ ja} \tag{40}$$

$$\forall\, i \in \{1, \ldots, n\} : wp(S_i, P)\,(s, \delta) \Rightarrow P(Out(S_i, s, \delta), \delta) \tag{41}$$

voidaan olettaa olevan tosia. Kaavan (39) voimassaolosta ja Comp-funktion määritelmästä seuraa, että

$$Out(IF, s, \delta) = Out(S_j, s, \delta) \tag{42}$$

on totta jollekin $j \in \{1, \ldots, n\}$, jota vastaava suoritusehto B_j on tosi (ts. ääretön sekvenssi eli ohjelman keskeytys ei tule kysymykseen).

Nyt nähdään, että oletusten nojalla

$$wp(IF, P)(s, \delta) \Rightarrow^{(40)}$$

$$(\forall i \in \{1, \ldots, n\} : B_i \Rightarrow wp(S_i, P)(s, \delta)) \Rightarrow^{(41)}$$

$$(\forall i \in \{1, \ldots, n\} : B_i \Rightarrow P(Out(S_i, s, \delta), \delta))$$

pätee. Oletuksen 39 perusteella tiedämme, että tasan yksi lauseista tulee suoritettua. Merkitään tätä lausetta S_j:llä. Suoritusehdon B_j on oltava tosi ja edellisen päättelyn viimeisen rivin perusteella predikaatti P on tosi myös S_j:n suorituksen jälkeen, ts. kaavan $P(Out(S_j, s, \delta), \delta)$ tiedetään olevan totta. Tuloksen (42) nojalla myös $P(Out(IF, s, \delta), \delta)$ on totta ja kaavan $wp(IF, P)(s, \delta) \Rightarrow P(Out(IF, s, \delta), \delta)$ validius on todistettu.

5.2.4 DO-lauseen päättelysäännön validius

On osoitettava implikaation

$$wp(DO, P)(s, \delta) \Rightarrow P(Out(DO, s, \delta), \delta)$$

voimassaolo. Kun vasen puoli on epätosi, implikaatio on triviaalisti tosi. Kun vasen puoli on tosi, DO-lauseen predikaattimuuntimen määritelmän (kts. sivu 24) nojalla $\exists i \geq 0 : H_i(P)(s, \delta)$ on totta. Osoitetaan väite oikeaksi induktiolla suorituskertojen maksimilukumäärän (kaavan symboli i) suhteen.

1° Kun suorituskertoja on nolla kappaletta, niin DO-lauseen predikaattimuuntimen määritelmän nojalla

$$H_0(P)(s, \delta) = (\neg BB \wedge P)(s, \delta) = \neg BB(s, \delta) \wedge P(s, \delta)$$

Comp-funktion määritelmän mukaan kaavan $\neg BB(s, \delta)$ voimassaolosta seuraa, että $Out(DO, s, \delta) = s$, joten edellisen päättelyn nojalla

$$P(s, \delta) = P(Out(DO, s, \delta), \delta),$$

eli väite pitää paikkansa kun suorituskertoja on nolla kappaletta.

2° Induktio-oletus: Väite pätee K-1 suorituskerralla, missä $K \geq 1$, ts.

$$H_{K-1}(P)(Out(S_i, s, \delta), \delta) \Rightarrow P(Out(DO, s, \delta), \delta).$$

Huomattavaa on, että $H_{K-1}(P)$ evaluoidaan tilassa, johon on päädytty jonkin lauseen S_i suorituksen tuloksena. (Tämä on tilanne aina kun silmukka suoritetaan vähintään yhden kerran.)

3° Induktioaskel: Induktio-oletuksen nojalla väite pätee myös kun suorituskertoja on $K \geq 1$ kappaletta, sillä

$$H_K(P)(s, \delta) =^{\text{(DO-lauseen pred. muun. määr.)}}$$

$$(wp(IF, H_{K-1}(P)) \vee H_0(P))(s, \delta) =^{\text{(ryhmittely)}}$$

$$wp(IF, H_{K-1}(P))(s, \delta) \vee H_0(P)(s, \delta),$$

missä

$$wp(IF, H_{K-1}(P))(s, \delta) =^{\text{(IF-lause oletetaan validiksi)}}$$

$$H_{K-1}(P)(Out(IF, s, \delta), \delta) \Rightarrow^{\text{(Comp-funktion määritelmä)}}$$

$$H_{K-1}(P)(Out(S_i, s, \delta), \delta) \Rightarrow^{\text{(induktio-oletuksen nojalla)}}$$

$$P(Out(DO, s, \delta), \delta)$$

ja $$H_0(P)(s, \delta) \Rightarrow^{\text{(alkuaskeleen nojalla)}}$$

$$P(Out(DO, s, \delta), \delta).$$

Nyt siis tiedetään, että olipa silmukan suorituskertoja kuinka monta tahansa, silmukan päättyessä predikaatti P on tosi ja DO-lauseen päättelysääntö on siis validi.

5.3 Täydellisyys

Tässä kappaleessa pyritään vastaamaan kysymykseen: ”Onko jokainen WP-laskennan tosi kaava todistuva?”. Kaavalla tarkoitetaan tässä kolmikkoa { wp(S, P) } S { P }, johon päädytään rinnastamalla WP-laskenta Hoaren [8] logiikkaan (kuten esim. Dijkstra [4, s. 455] tekee) ja soveltamalla Cookin [3, s. 78] käsitettä ”Hoaren logiikan kaava”, jolla tarkoitetaan esiehdosta, ohjelmasta ja jälkiehdosta koostuvaa kolmikkoa. Edelleen Cookin [3] määritelmien mukaan kaavan { wp(S, P) } S { P } totuus tarkoittaa implikaation

$$wp(S, P)(s, \delta) \Rightarrow P(\ Out(S, s, \delta), \delta\ )$$

voimassaoloa kaikille tiloille s ja muuttujasijoituksille δ ja todistuvuus sitä, että kolmikko on perusteltavissa WP-laskennan aksioomia ja päättelysääntöjä hyväksikäyttäen.

Myöntävä vastaus täydellisyyskysymykseen saataisiin osoittamalla, että jokaiselle kolmikolle { wp(S, P) } S { P } löytyy todistus sillä ehdolla, että kolmikko tiedetään todeksi, ts. implikaatio $wp(S, P)(s, \delta) \Rightarrow P(\ Out(S, s, \delta), \delta\ )$ pätee kaikille tiloille s ja muuttujasijoituksille δ. (Mikäli todistus löytyisi myös epätodelle kolmikolle, WP-laskenta ei olisi ristiriidatonta ja edellisen kappaleen tulos ei pitäisi paikkaansa.)

Huomataan, että lähes haluttu tulos saavutetaan tekemällä kielelle L_A ns. ekspressiivisyysoletus kielen L_E ja tulkinnan I suhteen. Ongelmana on, että oletus koskee osittaista oikeellisuutta, joten sen perusteella voimme päätellä vain heikoimman liberaalin esiehdon (= esiehto, joka takaa jälkiehdon toteutumisen mikäli ohjelma pysähtyy, kts. esim. [5, s. 21]) löytymisen jokaiselle ohjelmalle ja jälkiehdolle. Kysymys totaalisen WP-laskennan täydellisyydestä jää siis auki.

5.3.1 Liberaalien esiehtojen täydellisyys

Clarke määrittelee artikkelissaan [2, s. 132] tilapredikaattien kuvaamiseen käytettävän kielen L_A ekspressiivisyyden seuraavasti:

Kieli L_A on ekspressiivinen suhteessa kieleen L_E ja tulkintaan I joss jokaiselle ohjelmalle S, jälkiehdolle Q ja ympäristölle e löytyy predikaatti $WP(S, e, Q) \in L_A$ siten, että $WP(S, e, Q)$

ilmaisee heikoimman esiehdon, joka takaa predikaatin Q voimassaolon ohjelman S pysähtyessä.

Clarken määritelmässä esiintyvä ympäristö e liittyy aliohjelmien käsittelyyn ja voimme siis jättää sen omien tarkastelujemme ulkopuolelle. Huomioitavaa on, että Clarken määritelmä koskee osittaista oikeellisuutta - WP(S, e, Q) voi johtaa myös ikuiseen silmukkaan. Esimerkkinä ekspressiivisyysehdon toteuttavasta asetelmasta Clarke mainitsee tilanteen, jossa L_A ja L_E ovat ensimmäisen asteen kieliä ja arvoalue D on äärellinen. Muita esimerkkejä aiheesta löytyy mm. lähteistä [1] ja [3].

Oletamme siis jatkossa, että kielestä L_A löytyy kaikille ohjelmille S ja jälkiehdoille P predikaatti wp(S, P), joka toteutuessaan alkutilassa takaa jälkiehdon P voimassaolon lopputilassa, mikäli laskenta pysähtyy. Tarkastellaan seuraavaksi muutamaa WP-laskennan päättelysääntöä olettaen annetun kaavan olevan totta tarkasteltavan päättelysäännön nojalla ja tutkien, löytyykö kielestä L_A ekspressiivisyysoletuksen nojalla predikaatit, joilla kaava voidaan todistaa. Mikäli predikaatit löytyvät, sanotaan, että kaava on todistuva. Tarkastelua ei tarvitse suorittaa sijoitusaksiooman sekä skip- ja abort-lauseiden päättelysääntöjen suhteen, sillä ne ovat aksioomina triviaalisti todistuvia.

5.3.2 Yhdistämissääntö

Olkoon kaava

$$\{ wp(\text{”}S_1; S_2\text{”}, P) \} S_1; S_2 \{ P \}$$

tosi. Ekspressiivisyysoletuksen nojalla kielestä L_A löytyy predikaatit $wp(S_2, P)$ ja $wp(S_1, wp(S_2, P))$, joita vastaavat kolmikot

$$\{ wp(S_2, P) \} S_2 \{ P \} \text{ ja } \{ wp(S_1, wp(S_2, P)) \} S_1 \{ wp(S_2, P) \}$$

voidaan olettaa todistuviksi ja edelleen kolmikko

$$\{ wp(S_1, wp(S_2, P)) \} S_1; S_2 \{ P \}$$

voidaan siis yhdistämissääntöä soveltamalla todistaa. Puolipisteoperaattorin predikaattimuuntimen määritelmän nojalla pätee

$$wp(\text{”}S_1; S_2\text{”}, P) = wp(S_1, wp(S_2, P))$$

ja asia on selvä yhdistämissäännön osalta.

5.3.3 IF-lauseen päättelysääntö

Olkoon kaava

$$\{ wp(IF, P) \} \text{ IF } B_1 \rightarrow S_1 \square \ldots \square B_n \rightarrow S_n \text{ FI } \{ P \} \qquad (43)$$

tosi. Tällöin ekspressiivisyysoletuksen nojalla jälkiehdolle { P } ja jokaiselle lauseelle S_i löytyy kielestä L_A heikoimman esiehdon ilmoittava tilapredikaatti $wp(S_i, P)$, joten kolmikot $\{wp(S_i, P)\}\ S_i\ \{P\}$ voidaan olettaa todistuviksi. Suoritusehdoista $B_1, \ldots, B_n$ muodostetun disjunktiolausekkeen $B_1 \vee \ldots \vee B_n$ voidaan olettaa kuuluvan kieleen L_A (sillä suoritusehdot on kuvattu kielellä L_E) kuten myös implikaatioiden $B_i \Rightarrow wp(S_i, P)$ kaikille $i \in \{1, \ldots, n\}$. Nyt nähdään, että IF-lauseen predikaattimuuntimen oletukset

$$(B_1 \vee \ldots \vee B_n) \text{ ja } (B_1 \Rightarrow wp(S_1, P)) \wedge \ldots \wedge (B_n \Rightarrow wp(S_n, P))$$

voidaan kirjoittaa kielellä L_A, ja kaava (43) voidaan todistaa soveltamalla IF-lauseen päättelysääntöä.

5.3.4 DO-lauseen päättelysääntö

Olkoon kaava

$$\{ wp(DO, P) \} \text{ DO } \{ P \}$$

tosi. Kun sovelletaan samanlaista päättelyä kuin IF-lauseen yhteydessä, huomataan, että DO-lauseen predikaattimuuntimen oletukset kuuluvat kieleen L_A ja kaava on todistuva.

Todistuksesta tulee huomattavasti hankalampi, mikäli invarianttiehto on keksittävä kuten esimerkiksi lähteessä [1, s. 438 ja 439].

6 Yhteenveto

Tutkielmassa määriteltiin Guarded Command -ohjelmointikielen kielioppi ja tyyppijärjestelmä sekä kuvattiin kielen suorittimen matemaattiset ominaisuudet. Lisäksi kehiteltiin määritelmä heikoimman esiehdon tuottavalle predikaattimuuntimelle.

Luvussa 4 kuvattiin Guarded Command -kielen semantiikka predikaattimuuntimien avulla samaan tapaan kuin esimerkiksi lähteessä [5, luku 4]. Tässä tutkielmassa on lisäksi osoitettu, että kuvaus todella toteuttaa luvussa 3 annetun predikaattimuuntimen määritelmän.

Tutkielman päätulos liittyy WP-laskennan teoreettiseen käyttökelpoisuuteen: esitettyjen päättelyjen perustella totaalinen WP-laskenta on ristiriidatonta (oikeellisuus) ja liberaali WP-laskenta lisäksi riittävää (täydellisyys) Cookin formalismin [3] mielessä. Tämä on mielenkiintoinen tulos kun muistetaan, että lähteet [1], [3] ja [2] tutkivat asiaa vain deterministisen järjestelmien osalta, joita Dijkstran [5, s. 19] mukaan voidaan pitää epädeterministisen järjestelmien erikoistapauksina. Oletus predikaattien kuvaukseen käytetyn kielen ekspressiivisyydestä riitti täydellisyyden osoittamiseen, mutta se tuskin on välttämätön ehto; tätä on tutkittu mm. lähteessä [1].

Kysymys ekspressiivisyyden voimassaolosta totaaliselle WP-laskennalle jää siis tässä tutkielmassa auki ja tarjoaa siten hedelmällisen aiheen jatkotutkimukselle. Tulosten perusteella on myös vaikea sanoa, onko WP-laskennasta käytännön ohjelmistonkehityksessä todellista hyötyä.

Lähteet

[1] Apt, Krzysztof R.,"Ten Years of Hoare's Logic: A Survey - Part 1", ACM Transactions on Programming Languages and Systems, Volume 3, Number 4, October 1981.

[2] Clarke, E. M., "Programming Language Constructs for Which It Is Impossible To Obtain Good Hoare Axiom Systems", Journal of the ACM, Volume 26, Number 1, January 1979.

[3] Cook, Stephen, "Soundness and Completeness of an Axiom System for Program Verification", SIAM Journal on Computing, Volume 7, Number 1, February 1978.

[4] Dijkstra, E. W., "Guarded Commands, Nondeterminacy and Formal Derivation of Programs", Communications of the ACM, August 1975.

[5] Dijkstra, E. W., "A Dicipline of Programming", ISBN 0-13-215871-X, Prentice-Hall, Englewood Cliffs, 1976.

[6] Floyd, Robert W., "Assigning Meanings to Programs", Proceedings of the Symposium on Applied Mathematics, Volume 19, American Mathematical Society, Providence, R.I., 1967.

[7] Gunter C. A., Scott, D. S., "Semantic Domains", kappale 12 kirjassa "Handbook of Theoretical Computer Science", osa B "Formal Models and Semantics", toim. Jan van Leeuwen, ISBN 0 444 88074 7, Elsevier science publishers B. V., 1990.

[8] Hoare, C.A.R., "An Axiomatic Basis of Computer Programming", Communications of the ACM, October 1969.

[9] Kaldewaij, Anne, "Programming: The Derivation of Algorithms", ISBN 0-13-204108-1, Prentice-Hall, 1990.

[10] Kurittu, Lassi, "Johdatus logiikkaan", luentomoniste 47, Jyväskylän yliopisto, Matematiikan laitos, 2000.

[11] Naur, Peter (toim.), "Report on the Algorithmic Language ALGOL 60", Communications of the ACM 3(5), 1960.

Liitteet

Liite 1. Hoaren artikkelin aksioomat ja päättelysäännöt

Hoare [8] määrittelee ohjelmien oikeellisuustodistuksissa käytettävät päättelysäännöt kielelle, joka koostuu

- muotoa "x:=e" olevista sijoituslauseista, missä x on muuttujasymboli ja e lauseke,
- muotoa "while e do S" olevista silmukoista, missä e on silmukan suoritusehto ja S toistettava lause ja
- edellisten lausetyyppien mukaisista lauseista muodostettuja, puolipisteillä toisistaan erotettuja listoja.

Sijoitusaksiooma

Hoare [8, s. 577]: $\vdash P_0 \{ x := f \} P$,

missä x on muuttujan nimi (engl. variable identifier), f on lauseke (engl. expression) ja P_0 saadaan predikaatista P korvaamalla kaikki x:n esiintymät f:llä.

Seuraussäännöt

Hoare [8, s. 578]:

Jos $\vdash P \{Q\} R$ ja $\vdash R \supset S$ niin $\vdash P \{Q\} S$.

Jos $\vdash P \{Q\} R$ ja $\vdash S \supset P$ niin $\vdash S \{Q\} R$.

Yhdistyssääntö

Hoare [8, s. 578]: Jos $\vdash P \{Q_1\} R_1$ ja $\vdash R_1 \{Q_2\} R$ niin $\vdash P \{(Q_1; Q_2)\} R$.

Toistosääntö

Hoare [8, s. 578]: Jos $\vdash P \wedge B \{S\} P$ niin $\vdash P \{\text{while } B \text{ do } S\} \neg B \wedge P$.

Liite 2. Todistuksen 4.5.2 totuustaulu

Merkitään B_i = A, $wp(S_i, P)$ = B ja $wp(S_i, Q)$ = C ja lisäksi tosi = T ja epätosi = E. Kaava

$$[(A \Rightarrow B) \wedge (B \Rightarrow C)] \Rightarrow [A \Rightarrow C]]$$

on validi totuustaulun

A	B	C	$A \Rightarrow B$	$B \Rightarrow C$	$(A \Rightarrow B) \wedge (B \Rightarrow C)$	$A \Rightarrow C$	$[(A \Rightarrow B) \wedge (B \Rightarrow C)] \Rightarrow [A \Rightarrow C]$
E	E	E	T	T	T	T	T
E	E	T	T	T	T	T	T
E	T	E	T	E	E	T	T
E	T	T	T	T	T	T	T
T	E	E	E	T	E	E	T
T	E	T	E	T	E	T	T
T	T	E	T	E	E	E	T
T	T	T	T	T	T	T	T

nojalla.