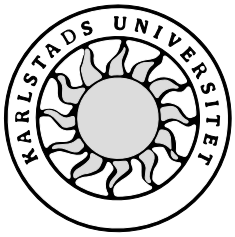


Datavetenskap

Niclas Norlin

Bevis för att försvagning av kontrakt definierar en Liskov-subtyp

---

Examensarbete, C-nivå

2003:27

Bevis för att försvagning av kontrakt definierar en Liskov-subtyp

Niclas Norlin

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Niclas Norlin

Godkänd, 2003-10-17

Handledare: Eivind Nordby

Examinator: Stefan Lindskog

Sammanfattning

Målet med den här C-uppsatsen är att få ett formellt bevis för att försvagning av kontrakt definierar en subtyp av en specifik typ. Att ”försvagning av kontrakt definierar en subtyp”, är en hypotes uppställd av Software Engineering Research Group (SERG) vid Karlstads universitet. I tidigare arbeten har Barbara Liskov presenterat substitutionsprincipen som definierar begreppet subtyp. Substitutionsprincipen anger dock endast om en typ är en subtyp av en annan typ, inte tillvägagångssättet. Krishna Kishore Dhara presenterar en egen definition av en subtyp och med hjälp av en teknik för specifikationsarv är det möjligt att specificera under vilka förutsättningar en typ är en subtyp av en annan typ. Dhara har bevisat att om kraven för specifikationsarv är uppfyllda så är också kraven för en subtyp uppfyllda. Dharas definition baseras på Liskovs ursprungliga men Dharas teknik baseras på matematik och är en komplex metod att använda. Vid varje enskilt tillfälle måste det visas att kraven är uppfyllda för specifikationsarv. SERG:s angreppssätt att varje gång visa att ett kontrakt är svagare än ett annat är enklare men det leder som sagt till en hypotes. Uppsatsens mål är att visa att SERG:s försvagning av kontrakt uppfyller Dharas villkor för specifikationsarv och erhålla ett formellt samband.

En annan viktig slutsats med uppsatsen är att SERG:s definition inte är tillräcklig. Detta följer av försöken att bevisa hypotesen. Uppsatsen innehåller därför ett extra avsnitt där det bevisas att SERG:s definition inte är tillräcklig inklusive en komplettering av definitionen.

A proof for that weakening of a contract defines a Liskov-subtype

Abstract

The purpose with this work is to achieve a formal proof for that weakening of a contract defines a subtype of a specific type. That “weakening of a contract defines a subtype” is only a hypothesis stated by the Software Engineering Research Group (SERG) at Karlstad University. Related works in this area is done by Barbara Liskov who presents the substitution principle which is the definition of a subtype. The substitution principle only states whether a type is a subtype of another type, not how to perform it from the beginning. Krishna Kishore Dhara presents his own definition of a subtype and by a technique for specification inheritance it is under certain circumstances possible to specify whether a type is a subtype of another type. Dhara has proved that if all demands are satisfied for specification inheritance then also the demands are satisfied for a subtype. Dhara’s definition is based on Liskov’s original definition but Dhara’s technique for defining a subtype is based on mathematics and his method is difficult to use. It has to be shown for every specific case that the demands are satisfied for specification inheritance. SERG:s way to show that a contract is weaker than another contract is easier but leads to a hypothesis. The purpose with the present work is to show that SERG:s way of weakening of contract satisfies Dhara’s demands of specification inheritance and to state a formal proof for this conjecture.

This work also contains another extra achievement. It has been proved that SERG:s definition has deficiencies and therefore is not entirely correct. Therefore this work contains an extra subchapter with a new definition which replaces the original one.

Innehållsförteckning

Figurförteckning

1 Inledning

Detta arbete är en C-uppsats i datavetenskap utförd i samarbete med Software Engineering Research Group (SERG) vid Karlstads universitet. Uppsatsen är genomgående abstrakt och fordrar läsarens uppmärksamhet. Inledningen börjar med en förklaring av bakgrund och syfte med uppsatsen och sedan förklaras uppsatsens mål. Därefter följer en kort presentation av tidigare arbeten samt ett avsnitt där uppsatsens struktur beskrivs för att underlätta läsningen.

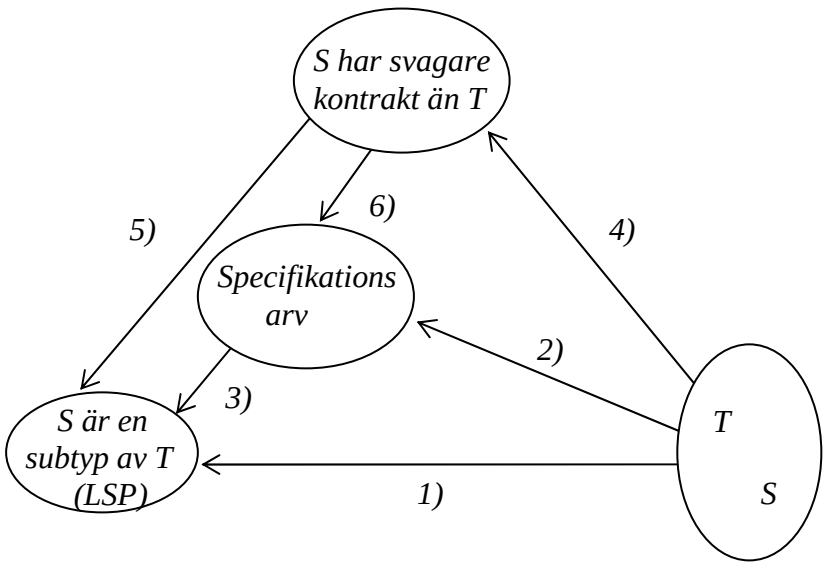
1.1 Bakgrund och syfte

SERG studerar semantiska beskrivningar i programmering. Semantiken i ett program beskriver det som programmet producerar till skillnad mot syntax som beskriver hur programmet är implementerat. Semantiska beskrivningar är viktiga vid förändringar av programvara och med förändring här menas främst underhåll, där man skiljer på fyra olika sorters underhåll: korrigerande, förbättrande, anpassande och förebyggande. Underhåll medför att delar av systemet förändras, med risk för att andra, beroende delar påverkas menligt. Om delar av systemet förändras kan de beroende delarna påverkas.

Uppsatsen ingår i en ansats att identifiera i vilka fall förändringen är menlig och i vilka fall den är menlös. I detta avseende kommer begreppet *subtyp* in som är av avgörande betydelse, eftersom subtyp innebär menlös förändring. Barbara Liskov presenterar *substitutionsprincipen* (LSP) [4] som definierar begreppet subtyp. Krishna Kishore Dhara presenterar en teknik som med hjälp av *specifikationsarv* specificerar under vilka förutsättningar en typ är en subtyp av en annan typ [3]. SERG har definierat en mer pragmatisk form som betecknas *starka* respektive *svaga kontrakt* (kontrakt förklaras utförligt längre fram) som möjliggör ett sätt att jämföra olika kontrakt. SERG:s angreppssätt är att: ”Ett kontrakt kan förändras från ett starkt kontrakt till ett svagare utan att det får konsekvenser för en klientmodul”, vilket motsvaras av definitionen av en subtyp. Att försvagning av ett kontrakt definierar en subtyp är dock en hypotes uppställd av SERG och saknar formell grund. Syftet med uppsatsen är att formellt visa att ett försvagat kontrakt definierar en subtyp av en specifik typ. Med specifik typ avses den typ som representeras av det ursprungliga kontraktet och här alltså utgör det starka kontraktet.

1.2 Uppsatsens mål

Målet med uppsatsen är att få ett formellt bevis för att försvagning av kontrakt definierar en subtyp av en specifik typ. Med försvagning av ett kontrakt menas egentligen försvagning av kontraktet för en *operation* i en typ, men för att målet med uppsatsen ska kunna förklaras lämnas vissa detaljer åt sidan i det här avsnittet, även om en del måste förklaras redan här. Dhara presenterar en egen definition av en subtyp som han kallar *Pre-Behavioral Subtyping* (PBS) [3] som förklaras längre fram. Med PBS och specifikationsarv är det möjligt att specificera under vilka förutsättningar en typ är en subtyp av en annan typ. Till hjälp för att illustrera och förklara LSP, SERG:s angreppssätt och Dharas teknik kommer genomgående Figur 1.1 att användas i uppsatsen. Detaljerna kommer att förklaras utförligt längre fram men det är fördelaktigt att presentera figuren redan nu. Oavsett angreppssätt eller teknik är målet LSP som definierar om en typ *S* är en subtyp av en typ *T* vilket vid enklare fall kan avgöras genom ett intuitivt resonemang. SERG:s angreppssätt motsvaras av definitionen av en subtyp, men handlar egentligen om operationer i *moduler*, här modulerna *S* och *T* och inte operationer i typer. I Figur 1.1 utgör *T* en klientmodul, och det är därför inte helt korrekt att en modul kan vara en subtyp av en annan modul, eller att en modul har ett kontrakt. Det är operationer i en typ eller modul som har kontrakt men resonemanget förenklas något. I fortsättningen när inte annat sägs används *S* och *T* i betydelsen av operationer för en typ eller modul. Begrepp som operation, typ och modul tillsammans med andra detaljer förklaras utförligt i kapitel 2.



Figur 1.1 LSP, SERG:s angreppssätt och Dharas teknik i förhållande till varandra

Första delen, se 4) i Figur 1.1, av SERG:s angreppssätt utgörs av försvagning av kontrakt och måste uföras varje gång. Om en operation i modul *S* har svagare kontrakt än en operation i modul *T*, se Figur 1.1, så är SERG:s hypotes att *S* är en subtyp av *T*. Men att kraven för LSP då är uppfyllda är inte bevisat och därför saknas en formell grund till SERG:s påstående. SERG:s hypotes, se 5) i Figur 1.1, är svår att direkt bevisa. Dharas teknik skiljer sig ifrån SERG:s angreppssätt.

Dharas teknik går ut på att försöka visa att specifikationen för operationen i typerna *S* och *T* uppfyller definitionen för PBS. Om definitionen för PBS är uppfylld så kan specifikationsarv användas. Det är enkelt eftersom Dhara i tidigare arbete [3] redan bevisat att specifikationsarv leder till att LSP är uppfylld, se 3) Figur 1.1, så det är inte nödvändigt att visa varje gång. Dharas teknik går därför ut på att varje gång visa att specifikationen för operationen i typerna *S* och *T* uppfyller definitionen för PBS, se 2) Figur 1.1. Problemet är att Dharas teknik utgörs av en komplex metod och är svår att använda. SERG:s angreppssätt är enklare att använda men leder inte ända fram. Ett sätt är att försöka kombinera SERG:s angreppssätt med Dharas teknik.

Om det går att bevisa att när *S* har svagare kontrakt än *T*, så är också kravet för Dharas specifikationsarv uppfyllt, som i sin tur betyder att LSP är uppfylld. Målet med uppsatsen är att visa just det sambandet, se 6) i Figur 1.1, alltså att SERG:s försvagning av kontrakt uppfyller kraven för Dharas specifikationsarv, och få ett formellt bevis.

1.3 Tidigare arbeten

Barbara Liskov presenterar substitutionsprincipen (LSP) [4] som definierar begreppet subtyp som andra författare hänvisar till för att vidareutveckla begreppet. I Liskovs senare arbete [5], vilket baseras på hennes tidigare där LSP presenteras, försöker Liskov med sin definition av *subtypsrelation* (SR) att ställa mer formella krav på en definition av en subtyp.

Krishna Kishore Dharas egen definition Pre-Behavioral Subtyping (PBS) [3] baseras på Liskovs SR men PBS skiljer sig från SR vad gäller krav på specifikation. PBS baseras på beteendet av datatyper och kan användas vid specifikation och verifikation av programvara. Dharas teknik med PBS, som uteslutande baseras på specifikationsarv, specificerar under

vilka omständigheter en typ är en subtyp av en annan typ [3]. Tekniken baseras på matematik och är en komplex metod att använda.

Luca Cardelli använder i ett arbete även han begreppet subtyp i sin definition av begreppet *structural subtyping* [2]. Även om structural subtyping inte har någon direkt betydelse för den här uppsatsen kan ändå en jämförelse mellan definitioner av en subtyp vara till hjälp. Vid en jämförelse med Cardellis definition av structural subtyping åskådliggörs att Dharas definition av en subtyp är baserad både på syntax och semantik för typer. Något om varför just Dharas definition är mer lämplig än Cardellis i detta arbete förklaras längre fram i avsnitt 4.1 i samband med Dharas teknik.

1.4 Uppsatsens struktur

För att underlätta läsningen av uppsatsen följer här en sammanfattning av strukturen samt motivering av syfte med respektive kapitel. En förutsättning för att tillgodogöra sig innehållet i den här uppsatsen är att läsaren är väl förtrogen med den terminilogi som förklaras i kapitel 2. Avsnitten i kapitel 2 kan därför verka övertydliga i vissa avseenden för en läsare som redan är bekant med delar av terminologin. I avsnitt 2.6 diskuteras en del om specialisering, tid och gränssnitt som ger information om bakgrund och egentligt skäl för uppsatsen.

Underlaget för den här uppsatsen presenteras i kapitel 3 och i samband vid presentation av olika begrepp följer direkt en utförlig förklaring. För att ytterligare förklara, bland annat vissa skillnader mellan olika begrepp i underlaget, finns också exempel med tillämpningar samt ett resonemang i avsnitt 3.3. I avsnitt 3.4 diskuteras speciellt Liskovs och Dharas definitioner av subtyp, samt förbereder läsaren inför en redogörelse av Dharas teknik i kapitel 4. En stor del av uppsatsens arbete bestod i att reda ut om Dhara faktiskt visade något överhuvudtaget eller om han endast försökte att göra det. Dharas artikel har därför noggrant studerats men det är inte nödvändigt att här presentera en utförlig analys av artikeln. Därför redogörs det kortfattat för Dharas teknik och syftet med artikeln. För den läsare som vill sätta sig in Dharas definitioner finns de i bilagorna B.2 och B.3 för senare läsning.

I kapitel 5 är läsaren mer bekant med underlaget och målet med uppsatsen kan beskrivas i detalj. Här beskrivs igen de olika alternativ som diskuterades i samband med Figur 1.1. Här utreds och förklaras hur LSP, SERG:s angreppssätt och Dharas teknik förhåller sig till

varandra genom matematiska relationer. Med hjälp av logik (diskret matematik) kan SERG:s hypotes bevisas. Sist i kapitel 5 sker också en nödvändig komplettering av SERG:s definition. Redovisning av bevis ges i kapitel 6 och uppsatsen avslutas med slutsatser och diskussion.

2 Terminologi

Här presenteras och förklaras en del termer och begrepp som är vanligt förkommande och av betydelse i uppsatsen. I avsnitt 2.1 och 2.2 förklaras en del grundläggande termer som beskrivs kortfattat. Komplicerade begrepp som kräver en utförlig genomgång förklaras var för sig i avsnitt 2.3-2.5. Förståelse av begrepp är en förutsättning inför vidare läsning. För den läsare som önskar en ingående förklaring av symboler senare i uppsatsen hänvisas till bilaga A. I sista avsnittet diskuteras specialisering, tid och gränssnitt som ger läsaren något mer information om bakgrund och egentligt skäl för uppsatsen.

2.1 Operation, metod och specifikation

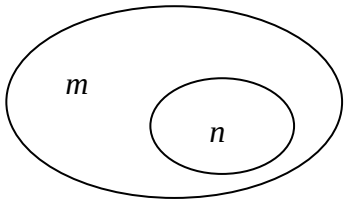
För att inte blanda ihop begrepp längre fram är det är viktigt att förstå skillnaden mellan olika termer. Två termer som är viktiga att hålla isär är *operation* och *metod*. En operation är en *specifikation* av en metod. En specifikation anger signaturen av en metod som består av namn och antal parametrar (argument) samt respektive parameters typ. En operation kan ha en eller flera metoder. En metod är enligt UML[1] en implementation av en operation.

2.2 Modul, typ och klass

En *modul* är en samling data och operationer som hör ihop logiskt. Om en modul är en matematisk beskrivning av data och operationer så utgör en klass själva programmet. En modul i sin tur definierar en *typ* och i den här uppsatsen är en *typ* en definitionsmängd tillsammans med operationer för den definitionsmängden. Det här tydliggör skillnaden mellan en typ och modul och varför det inte är helt korrekt att en modul ibland kallas för en subtyp till en annan modul som det diskuterades om i samband med SERG:s angreppssätt i avsnitt 1.2.

[1] UML – Unified Modeling Language

En modul *m* är en modul (här i meningen subtyp) av en annan modul *n* endast om varje metod som tillhör modul *n* även tillhör *m*, se Figur 2.1.



Figur 2.1 En modul m är en modul av en annnan modul n endast om varje metod som tillhör modul n även tillhör m

Vidare bör läsaren förstå innebörden av vad som sker med en modul när en av dess operationer förändras. När en operation förändras medför det att dess modul definieras om. Eftersom en modul definierar en typ så definieras även en ny typ och det är därför som det är intressant att diskutera subtyper. Däremot kan en operations metod förändras utan att varken operationen förändras eller modulen måste definieras om. För övrigt kan en modul också befinna sig i olika tillstånd. Egentligen oändligt många tillstånd eftersom en modul kan ha olika metoder. Ett tillstånd kännetecknas som en fas i implementationen.

2.3 Förvillkor och eftervillkor

Ett speciellt viktigt begrepp i den här uppsatsen är villkor. De två villkor som används är *för-* och *eftervillkor*, precondition respektive postcondition, varav här oftast används förkortningarna *pre* respektive *post*. En metod har två villkor och med ett förvillkor menas det villkor som måste vara uppfyllt före anrop till metoden. Ett eftervillkor är det villkor som skall vara uppfyllt efter exekveringen av metoden. En metod kan till exempel kräva som förvillkor att en variabel *x* ska vara större än eller lika med ett. En klient måste då uppfylla: *pre* $x \geq 1$. Under förutsättning att klienten uppfyller förvillkoret, garanterar metoden i sin tur att eftervillkoret $x > 10$ kommer att uppfyllas. Eftervillkoret för metoden är då: *post* $x > 10$.

För att hålla isär förvillkor från eftervillkor, villkor för en metod från en annnan metod, är det nödvändigt att använda index. Vanligast i uppsatsen är att *pre* för *T*, skrivs som pre_T^m, där *m*

syftar på metod. Här kan *T* betyda en typ eller modul. På motsvarande sätt för eftervillkor och andra typer. När metoden inte är intressant eller om metoderna är lika utelämnas ibland *m*.

2.4 Starka respektive svaga villkor och relation

När ett villkor betraktas i förhållande till ett annat villkor beskrivs den här relationen i termer av *starka* respektive *svaga* villkor. Till exempel kan två förvillkor jämföras. Antag två förvillkor, där om det ena förvillkoret är uppfyllt så är även det andra det, fast inte tvärtom. Den här relationen anger då att det första förvillkoret är starkare än det andra. Det andra förvillkoret sägs vara svagare än det första.

En relation mellan två villkor i denna uppsats anges ofta som en matematisk implikation. För relationen $pre_T^m \Rightarrow pre_S^m$ betyder detta, att om *pre* för typen *T* är uppfyllt, så implicerar (medför) det även att *pre* för *S* är uppfyllt. Den här relationen kan också ses som en jämförelse mellan de två villkoren. Här är det första förvillkoret är starkare än det andra. På samma sätt är förstås nu också det andra svagare än det första.

Värt att notera är att i den här uppsatsen skrivs förvillkor som pre_T^m där *m* syftar på metod. I Dharas relationer däremot syftar av någon anledning *m* inte på metod utan på operation. Objekt i Dharas relationer anges som $pre_T^m(self') \Rightarrow pre_S^m(self\wedge)$, där förvillkoret har ett objekt *self'*, och eftervillkoret ett objekt $self\wedge$. Ett objekt kan till exempel vara en vektor.

2.5 Kontrakt och starka respektive svaga kontrakt

En viktig term i forskningen är *kontrakt* som beskriver vad en eller flera parter i en relation kan förvänta sig av varandra. Kontrakt introducerades av Bertrand Meyer [6], och är i programmering en beskrivning av en överenskommelse mellan två parter, här en klient och en leverantör. En klient anropar en metod och ska före anrop uppfylla ett förvillkor. Endast om förvillkoret är uppfyllt garanterar metoden, här leverantören, att leverera något och därmed uppfylla ett eftervillkor. Klienten och leverantören är bundna till varandra av ett kontrakt.

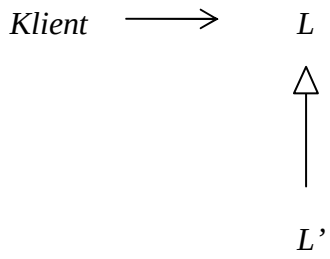
Starka respektive svaga kontrakt har som tidigare nämnts definierats av SERG. Eftersom ett kontrakt består av villkor, kan därför liksom enskilda villkor, kontrakt betraktas i relation till

varandra. På liknande sätt som för villkor beskrivs denna relation i termer av starka respektive svaga kontrakt. Mer om kontrakt i avsnitt 3.3.

2.6 Specialisering, tid och gränssnitt

Skälet till att en leverantör (metod) ändras eller en ny tillkommer i en programvara är flera. Vid specialisering kan det tillkomma en ny leverantör och vid underhåll förändras en leverantör över tid. Förändring av programvara kan som det beskrivs i inledningen vara av korrigerande, förbättrande, anpassande eller förebyggande art. Genom att använda kontrakt mellan en klient och en leverantör finns en väldefinierad relation som är en förutsättning för att underlätta vid förändring. Tre olika perspektiv kan identifieras och här följer en beskrivning för respektive perspektiv.

Första perspektivet beskriver specialisering. Vid specialisering eller omdefinition av operationer tillämpas arv. För en relation mellan en klient och en leverantör *L* där *L* ska specialiseras, uppkommer en arvsrelation mellan *L*, och en annan leverantör *L'*. Det här perspektivet kan illustreras med hjälp av bilden i Figur 2.2 där den tomma pilen representerar ett arv.

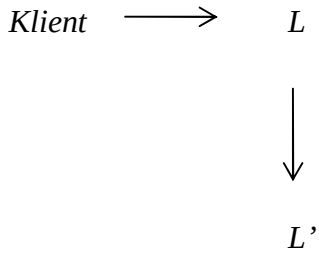


Figur 2.2 Vid specialisering tillämpas arv

Om *L* har ett kontrakt med en klient så måste även *L'* uppfylla kontraktet ut mot klienten. I det här perspektivet substitueras inte någon leverantör. Inte heller utvecklas en leverantör över tid. Det är däremot syftet för det som nästa perspektiv behandlar.

Det andra perspektivet beskriver det som inträffar över tid som vid förändring eller underhåll av en leverantör. Förändring innebär i det här perspektivet att en leverantör vidareutvecklas över tid genom att leverantörens olika delar underhålls. Avsikten med det här perspektivet är

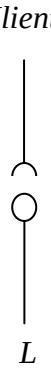
alltså inte att hela leverantören ska substitueras mot en annan leverantör på en gång. Det skulle ju också innebära en förändring. Det här perspektivet behandlar istället förändring över tid som en kontinuerlig process och det är precis det som är huvudsyftet med underhåll. Det finns olika former av underhåll, men det är inte relevant i det här sammanhanget och behöver därför inte tas upp här. Ett tidsperspektiv illustreras i Figur 2.3.



Figur 2.3 Vid tidsperspektiv tillämpas förändring eller underhåll

Vänstra sidan i bilden representerar som i föregående bild klienten. Högra sidan representerar leverantören *L* som vid varje förändring blir leverantören L'. Innan varje underhåll måste det undersökas att klienten inte kommer att märka någon skillnad efteråt. Klienten ska inte märka att *L* har förändrats till L'.

Det tredje och sista perspektivet behandlar när en klient använder en leverantör. Det som då sker är att specifikationerna jämförs. Ett kontrakt mellan en klient och en leverantör kan beskrivas utifrån ett gränssnittsperspektiv, se Figur 2.4 där halvcirkeln och cirkeln illustrerar två gränssnitt. Ett gränssnitt är en specifikation av design för en del av systemet.



Figur 2.4 Förväntat och levererat gränssnitt

Med gränssnitt ges en bra möjlighet till en tydlig uppdelning av specifikationerna. Om en leverantör utför mer än nödvändigt saknar betydelse för klienten eftersom det viktiga är att leverantören endast uppfyller klientens krav. På motsvarande sätt kan en klient utnyttja mindre än det som en leverantör kan uppfylla. Klientens gränssnitt betraktas som ett ”förväntat gränssnitt” och symboliseras av halvcirkeln i bilden. Leverantörens gränssnitt betraktas som ett ”levererat gränssnitt” och symboliseras av cirkeln i samma bild.

3 Underlag för arbetet

Meyer [6] och Liskov [4] är två forskare, vars arbeten denna uppsats vilar på och inför kommande kapitel är det nödvändigt att ha förståelse för två grundläggande begrepp som de har introducerat. I första avsnittet behandlas kontrakt som beskrivs utifrån Meyers *Assertion Redeclaration Rule* (ARR) där kontrakt används i samband med arv av klasser. Förutom ARR behandlas i andra avsnittet Liskovs substitutionsprincip (LSP) som till en del påminner om ARR, men har en annan utgångspunkt. I avsnitt 3.3 resoneras det runt ARR, LSP och även SERG:s angreppssätt tillsammans med tillämpningar. Med hjälp av detta avsnitt förtydligas det på ett mer praktiskt sätt vad de egentligen innebär. Slutligen diskuteras en jämförelse mellan Liskovs SR och Dharas definition, för att reda ut och förklara varför det är Dharas definition som är intressant och användbar för den här uppsatsen.

3.1 Ett omdefinierat kontrakt - Assertion Redeclaration Rule (ARR)

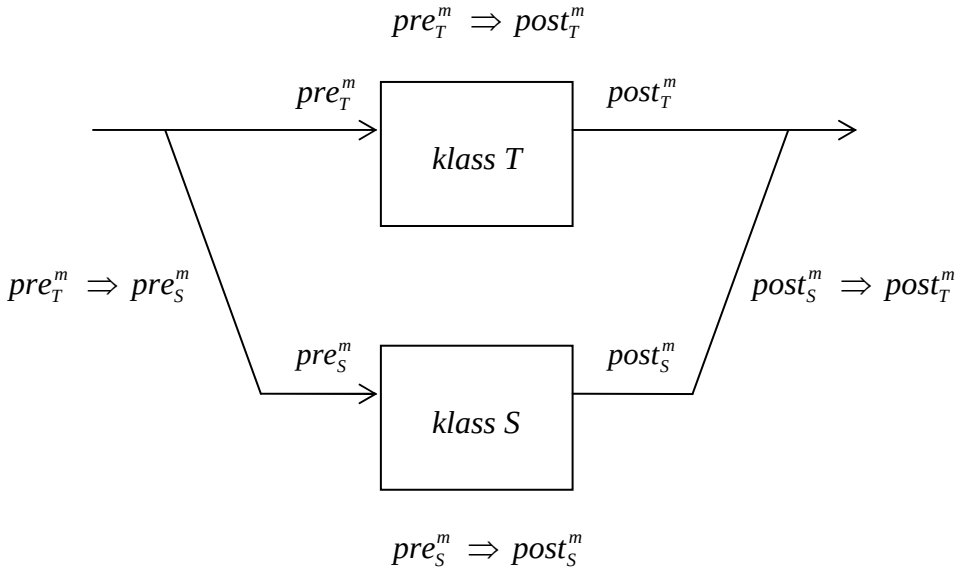
Meyer ger en bra beskrivning [6] av ett kontrakt, där *m* är en metod: ”If you promise to call *m* with *pre* satisfied then I, in return, promise to deliver a final state in which *post* is satisfied”. Med ett kontrakt menas att omm[2] en klient anropar en metod enligt gällande förvillkor, garanterar leverantören att uppfylla anropet med sitt eftervillkor. ARR kan uttryckas som:

> *ARR är en samling regler i form av relationer, som anger hur metoder i en klass måste förhålla sig till metoder i en annan klass, för att klassarv ska kunna tillämpas.*

ARR beskriver under vilka förutsättningar en klass kan ärva av en annan klass så att en klient inte påverkas.

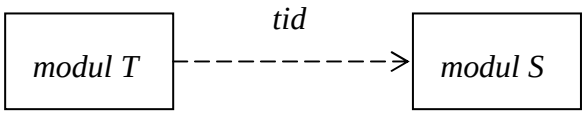
[2] omm – om och endast om

Ett omdefinierat kontrakt skiljer sig ifrån originalet om till exempel förvillkoret är svagare än eller lika med det ursprungliga förvillkoret och eftervillkoret är starkare än eller lika med det ursprungliga eftervillkoret. Med detta anges vilka förutsättningar som ska vara uppfyllda för att klass *S* ska kunna ärva av klass *T*. Meyers ARR anger på detta sätt två relationer för de villkor som måste vara uppfyllda för att en klass *S* ska få ärva från en annan klass *T*. Det innebär också att metoden m_S omdefinierar metoden m_T. ARR anger att förvillkoret för en metod m_S i klass *S* ska vara svagare än eller lika med det ursprungliga förvillkoret för metoden m_T i klass *T*. Eftervillkoret för *S* ska vara starkare än eller lika med det ursprungliga eftervillkoret för *T*. I Figur 3.1 som är baserad på en bild från en OH-presentation [1], visas en bild som illustrerar de här två relationerna ytterst på vänster och höger sida.



Figur 3.1 ARR anger förutsättningarna för att en klass S ska kunna ärva av en klass T så att en klient inte påverkas

Relationerna ytterst på vänster och höger sida gäller även var för sig. På detta sätt anger ARR med de här två relationerna det kontrakt som måste vara uppfyllt för att en klass *S* ska kunna ärva så att en klient inte påverkas. Observera att bilden ovan är något missvisande på grund av att den visar båda klasserna samtidigt. Klass *S* ärver av klass *T* och de två klasserna existerar alltså inte samtidigt. Det är viktigt att tänka på att arv sker över tid och detta illustreras i Figur 3.2. Meyer använder inte begreppet modul utan enbart klass men för att inte använda för många begrepp används i fortsättningen modul.



Figur 3.2 En modul vidareutvecklas över tid

Vid en analys av ARR avslöjas dock att relationen mellan eftervillkor är onödigt restriktiv och det här kommer att illustreras med hjälp av två exempel. I första exemplet, se Figur 3.3, representeras en modul *T* av $f(x) = \sqrt{x}$. Under förutsättning att förvillkoret för metod m_T i modul *T* i Figur 3.3 är uppfyllt kan modulen beräkna ett positivt värde för $f(x)$ inom en viss felmarginal ε. En modul *S* ska ärva av *T* och i villkoren för *S* inkluderas därför villkoren för modul *T* före S:s egna villkor. Se Figur 3.3 för villkor.

Modul T:

pre $x \geq 0$

post $\left|result^2 - x\right| < \varepsilon \wedge result \geq 0$

Modul S:

pre $x \geq 0 \vee x < 0$

post $\left(\left|result^2 - x\right| < \varepsilon \wedge result \geq 0\right) \wedge \left(x < 0 \Rightarrow result = -1\right)$

Figur 3.3 Villkoren för modul T inkluderas i respektive villkor för modul S

Observera att i förvillkoret för modul *S* vid arv, så gäller det egna villkoret *eller* det för modul *T*. Det betyder i det här första exemplet att modul *S* alltid kan anropas oberoende av värde på *x* och här gäller relationen $pre_T \Rightarrow pre_S$. Förvillkoret för *S* anger här att *x* kan vara större än eller lika med noll, *eller* så kan *x* vara mindre än noll. Här är pre_T starkare än pre_S, eftersom om pre_T är uppfyllt så är också pre_S uppfyllt. Däremot gäller inte relationen $post_S \Rightarrow post_T$. För eftervillkoret så gäller att det ”egna” eftervillkoret för modul *S* ska gälla

och det för modul *T*. Om $post_S$ är giltigt i exemplet så behöver alltså inte det innebära att också $post_T$ alltid är det i exemplet. Det betyder att klienten i det här första exemplet kan komma att märka skillnad vilket i sin tur enligt ARR innebär att modul *S* inte kan ärva av modul *T*.

I det andra exemplet, se Figur 3.4, representeras en modul *T* av samma metod: $f(x) = \sqrt{x}$. I det här andra exemplet är förvillkoret lika som i första exemplet men inte eftervillkoret. Förvillkoret för modul *T* är här implicit i eftervillkoret för *T*, se *Post* för *Modul T* i Figur 3.4. Det innebär att när eftervillkoret för modul *T*, på motsvarande sätt som i första exemplet, inkluderas i eftervillkoret för modul *S*, så kommer klienten inte att märka någon skillnad. Observera att det är möjligt att klienten kanske inte skulle ha märkt någon skillnad i första exemplet heller.

Modul T:

pre $x \geq 0$

post $x \geq 0 \Rightarrow \left(\left|result^2 - x\right| < \varepsilon \wedge result \geq 0\right)$

Modul S:

pre $x \geq 0 \vee x < 0$

post $\left[x \geq 0 \Rightarrow \left(\left|result^2 - x\right| < \varepsilon \wedge result \geq 0\right)\right] \wedge \left[x < 0 \Rightarrow result = -1\right]$

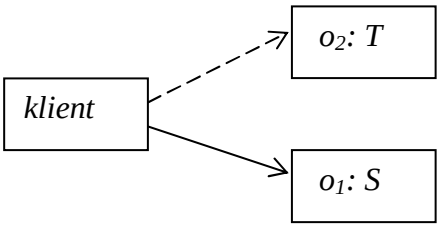
Figur 3.4 Förvillkoret för modul T inkluderas implicit i eftervillkoret för modul T

Kontraktet i *S* för eftervillkoret har här relationen $(pre_T \Rightarrow post_T) \wedge (pre_S \Rightarrow post_S)$. Utan att närmare diskutera detaljer räcker det med att konstatera att fastän relationen $post_S \Rightarrow post_T$ fortfarande inte gäller kommer klienten nu *inte* att märka någon skillnad. Detta visar att relationen för eftervillkor är onödigt restriktiv vilket också betyder att ARR är för restriktivt. Vidare innebär $pre_T \Rightarrow pre_S$ att kontraktet är försvagat men innan begreppet: ”... försvagning av ett kontrakt definierar en subtyp ...”, som är syftet med denna uppsats, tydligare kan förklaras, behöver läsaren kännedom om Liskovs substitutionsprincip och om vad en subtyp är.

3.2 Liskovs substitutionsprincip (LSP)

Liskovs substitutionsprincip (LSP) definierar begreppet subtyp [4]. Med subtyp avses en typ som substituerar en annan typ. Innebörden av substituera är att typen ersätts eller byts ut, men i den här uppsatsen används enbart substituera för enkelhetens skull. För att en typ *S* skall vara av en subtyp skall *S* uppfylla en specifik definition. I samband med begreppet subtyp används också automatiskt begreppet supertyp. En subtyp *S* är alltid en subtyp av någon typ *T*. Typen *T* kallas för supertyp och en subtyp *S* kan ha flera supertyper T_1, T_2, ..., T_j. En subtyp måste även ha en simuleringsfunktion. Kortfattat kan redan nu sägas att en simuleringsfunktion möjliggör ett sätt att avbilda ett objekt som tillhör en subtyp till ett objekt som tillhör en supertyp.

Förutom subtypens operationer som är *ärvda* från en supertyp, där subtypen kan tillhandahålla alla operationer som supertypen innehar, kan en subtyp även ha egna extra operationer. LSP anger huruvida en typ *S* kan substituera en annan typ *T*. Det innebär att om ett objekt, o_1, som tillhör en typ *S*, kan substituera ett objekt, o_2, som tillhör en annan typ *T*, så att en klient inte märker någon skillnad, då är *S* en subtyp av *T*, se Figur 3.5 .



o_1 substituerar o_2

Figur 3.5 Liskovs substitutionsprincip

Formellt säger[3]. LSP att;

> *”om, för alla objekt o_1 som tillhör en typ S, det existerar ett objekt o_2 för en typ T, där o_1 kan substituera o_2 utan att för alla klienter uttryckta i termer av T märker någon skillnad, då är S en subtyp av T (T är då en supertyp)”*

[3] översatt från eng citat

Observera här att Meyer tidigare inte alls använder begreppet subtyp utan enbart behandlar klassarv. Med införandet av begreppet subtyp är Liskov istället inriktad på att substituera en modul. Viktigt att notera är att LSP inte på något sätt anger *hur* man kan avgöra om klienten kan komma att märka någon skillnad. LSP anger endast att om klienten *inte* gör det så är *S* en subtyp till *T*. För att med LSP kunna avgöra detta krävs därför ett intuitivt resonemang. Med ARR jämförs istället villkor på ett mer formellt sätt.

3.3 Ett resonemang runt ARR, LSP och SERG

För att ytterligare förklara begreppen i rubriken underlättar det med ett resonemang och några exempel. Oavsett vilket av dessa tre begrepp som används så är frågan densamma. Oavsett om en modul ska ärva eller substitueras så handlar det om huruvida en klient kommer eller *kan* komma att märka någon skillnad efteråt. En klient är i detta fall en modul *K* som använder en annan modul *T*. Antag som exempel att *T* i något avseende behöver förbättras. Vid förbättringen kommer *T* att förbättras till modulen *S*. Efteråt vill vi helt enkelt veta om *K* kan använda *S* på samma sätt som *K* tidigare använde *T*. Med att ”på samma sätt” *K* efteråt kan använda *S* menas att *K* inte behöver anpassas. Det kanske är uppenbart men värt att påpeka. Därför behövs ett svar på frågan om *K* kommer eller kan komma att märka skillnad. Alla tre ovanstående begrepp kan på ett mer eller mindre bra sätt svara på den här frågan.

Som en tillämpning av begreppen används ett exempel för en modul *T* med *f(x)*. Modul *T* ska förbättras och vidareutvecklas eller substitueras med modul *S*, se Figur 3.6. Vad *f(x)* utför i det här exemplet är inte så viktigt men låt *x* vara positionen i en kö. Funktionen kan till exempel returnera information om ett objekt på position *x*. För att ett korrekt resultat ska returneras från *f(x)* krävs vid anrop till *T* att villkoret $x \geq 2$ är uppfyllt. I modul *T* kan alltså inte *f(x)* hantera position ett vilket begränsar modulens användbarhet. Därför ska *T* substitueras med *S* där villkoret även tillåter den första positionen. I det här exemplet betraktas enbart förvillkoren så för enkelhetens skull antas att eftervillkoren är lika. Om det saknas ett objekt på position *x* så returneras sant annars falskt.

Modul T:

f(x)

pre $x \geq 2$ *och x är ett heltal*

Modul S:

f(x)

pos $x \geq 1$ *och x är ett heltal*

Figur 3.6 En modul S ska ärva av modul T, eller T ska substitueras med S

Först behandlas exemplet utifrån den inledande frågan enligt Meyers ARR. ARR beskriver under vilka förutsättningar en modul (klass) kan ärva så att en klient inte påverkas. Observera att ARR inte innehåller någon sats som: ”om det här är uppfyllt så gäller detta” som LSP gör. ARR anger enbart vad som *måste* gälla för att en klient inte ska påverkas. ARR behandlar inte heller typer utan helt och hållet klasser och huruvida en klass *S* kan ärva från en klass *T*. För att *S* ska kunna ärva från *T*, så att *K* inte påverkas, måste enligt ARR en specifik relation gälla mellan villkoren för de två klasserna. För förvillkoren måste relationen $pre_T \Rightarrow pre_S$ gälla. Relationen anger att villkoret för *S* måste vara lika med eller svagare än för *T*. För exemplet i Figur 3.6 måste det därför verifieras att $(x \geq 2) \Rightarrow (x \geq 1)$. Denna implikation gäller och därför är kravet uppfyllt. Om *x* är större eller lika med två medför detta givetvis att *x* även är större än ett. På motsvarande sätt måste enligt ARR också relationen $post_S \Rightarrow post_T$ gälla. Eftersom eftervillkoren i det här exemplet är lika så är även detta krav uppfyllt.

För samma fråga som tidigare behandlas nu exemplet enligt LSP. LSP behandlar modulerna utifrån typer. En typ kan representera en klass men även något annat. LSP anger huruvida en typ *S* kan substituera en annan typ *T*. LSP innehåller som tidigare nämnts inga relationer för olika villkor som ARR gör utan baseras på ett intuitivt resonemang. Ett intuitivt resonemang kring villkoren för att verifiera att villkoren uppfyller definitionen för LSP. Det innebär att om ett objekt som tillhör en typ *S*, kan substituera ett objekt som tillhör en annan typ *T*, så att en klient inte märker någon skillnad, då är *S* en subtyp av *T*. För LSP är frågan enbart om *S* är en subtyp av *T* eller inte. I det här exemplet kan även simuleringsfunktionen bortses ifrån.

Eftersom *f(x)* för de två typerna har samma deklaration[4] så behövs inte här någon avbildning från ett objekt till ett annat. Eftersom *K* även i fortsättningen ska uppfylla förvillkoret inses här lätt att *K* inte kommer att märka någon skillnad. Enligt LSP kan därför sägas att *S* i exemplet utgör en subtyp av *T*. Om däremot förvillkoret för *S* ändras till $x \geq 3$ kan *K* förr eller senare märka skillnad. Ett intuitivt resonemang är dock inte något formellt bevis utan endast ett resonemang.

Det är här syftet med SERG:s angreppssätt kommer in. SERG:s mål är nämligen att slippa ett intuitivt resonemang. SERG har därför definierat en form som betecknas svaga respektive starka kontrakt och anger att ett kontrakt kan vara svagare eller starkare än ett annat. SERG:s begrepp är att:

> *”Ett kontrakt kan förändras från ett starkt kontrakt till ett svagare utan att det får konsekvenser för en klientmodul”*

vilket motsvaras av definitionen av en subtyp (LSP). SERG:s begrepp består av två delar och är en kombination av en generaliserad ARR och LSP. Att ARR är generaliserad beror på att ARR är för restriktivt och därför inte direkt kan användas för SERG:s syfte (i avsnitt 3.1 visades det att eftervillkoret i ARR är för restriktivt). Första delen av SERG:s definition, se 5) i Figur 1.1, består därför av en generaliserad ARR och andra delen av definitionen motsvaras av LSP. Att andra delen i SERG:s definition skulle vara uppfylld är dock en hypotes. Det finns inget formellt bevis som säger att försvagning av ett kontrakt definierar en subtyp.

Första delen av SERG:s definition[5] [1], och som utgör en generaliserad ARR, säger att ett omdefinierat kontrakt är svagare än originalet om:

- Förvillkoret är svagare än eller lika med förvillkoret för originalet
- Eftervillkoret är starkare än eller lika med eftervillkoret för originalvillkoret i domänen för originalets förvillkor

Figur 3.7 Första delen av SERG:s begrepp

[4] samma signatur och returtyp
[5] översatt från eng

Den viktiga skillnaden här mot ARR är tillägget i eftervillkoret. Genom att lägga till att eftervillkoret bara behöver gälla inom domänen för originalets förvillkor undviks problemet med att ARR är för restriktivt. Andra delen av SERG:s definitionen, se 4) i Figur 1.1, och som motsvaras av LSP, säger att när en leverantörsmodul förändras så att något av dess kontrakt försvagas, leder det till att dess klientmodul inte påverkas, vilket i sin tur medför att LSP är uppfyllt, se Figur 3.8.

När ett kontrakt försvagas:

- Klienten till en modul påverkas inte

vilket medför att:

- LSP är uppfylld

Figur 3.8 Andra delen av SERG:s begrepp

”... att dess klientmodul inte påverkas, vilket ...”, är en omskrivning av definitionen för LSP. Det är den här delen som inte är formellt bevisad och påståendet i utsagan utgörs av: ”när en leverantörsmodul förändras så att något av dess kontrakt är försvagas, leder detta till ...”.

3.4 Liskovs kontra Dharas definition av subtyp

Fastän även Liskov har en definition av en subtypsrelation (SR) [5] som Dhara baserar sin egen definition på, är Dharas mer lämplig för ändamålet. SR anger vad som ska gälla för att en typ ska få vara en subtyp. SR anger däremot inte hur en typ ska definieras för att typen ska kunna bli en subtyp (precis som LSP). Till skillnad från Liskov visar Dhara att om vissa krav (relationer mellan olika villkor som i ARR fastän mer invecklat) av hans definition är uppfyllda, så leder det med hjälp av specifikationsarv till definitionen av en subtyp. I klartext betyder det att Dharas definition egentligen anger *hur* en subtyp ska definieras. Milt uttryckt framgår inte detta klart i Dharas definition. Men det är delen för relationerna mellan olika villkor, som åstadkommer detta och det utgör därför den viktigaste delen i Dharas definition. Det är också den här delen som helt saknar motsvarighet i Liskovs formella definition, och hennes definition innehåller egentligen inte alls något liknande. Det är därför som hennes definition av en subtyp inte alls kan användas för att uppnå det mål som är uppställt i den här uppsatsen. För att visa att SERG:s krav uppfyller Dharas krav och erhålla ett formellt

samband är det därför en förutsättning att utgå ifrån Dharas definition istället för Liskovs. Det är tveksamt om en studie av SR underlättar en djupare förståelse av Dharas definition av PBS i nästa kapitel. Men eftersom PBS baseras på SR så ges en kort förklaring till Liskovs SR [5] i bilaga B.1. Bilagan är endast tänkt att användas som en senare fördjupning för den intresserade läsaren.

4 Dharas teknik

Dhara presenterar en modulär specifikationsteknik som automatiskt ”tvingar” en subtyp till en beteendemässig subtyp. Genom att för- och eftervillkor måste, ”tvingas”, att stå i en viss relation till varandra, uppfyller villkoren automatiskt Dharas krav för en beteendemässig subtyp. Med beteendemässig subtyp innebär det att en subtyp har samma beteendemässiga uppförande som dess supertyp (så att en klientmodul i sin tur inte påverkas). Före en kortare redogörelse av Dharas teknik som kallas för specifikationsarv, och definitionen av PBS, inleds första delen av kapitlet med en beskrivning av modulär specifikationsteknik där även begreppet simuleringsfunktion förklaras.

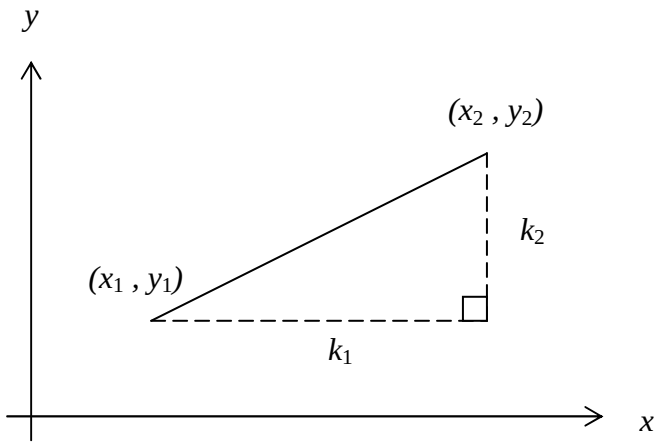
4.1 Modulär specifikationsteknik och simulering

Teknik för specifikation och verifikation som ej kräver ny specifikation och verifikation när extra objekt läggs till kallas för modulär. Behavioral Subtyping (i fortsättningen BS eller beteendemässig subtyp) baseras på beteendet av datatyper och kan användas vid modulär specifikationsteknik, där en subtyp automatiskt kan tvingas till en beteendemässig subtyp. Dharas sätt att lösa problemet med specifikation är att kräva att BS tillåter ett sätt att simulera specifikationen till en specifikation för en annan typ. En förutsättning för att en simulering skall vara möjlig, är förstås att det existerar någon form av relation mellan de objekt där simuleringen skall ske mellan. Med hjälp av simulering kan då en typ bete sig som en annan typ. Följande exempel förklarar ingående vad en simuleringsfunktion är.

Ett objekt i en typ *T* med argumenten, $[k_1, k_2]$, betecknar kateterna för triangeln i Figur 4.1. Med katet avses horisontell eller vertikal längd mellan de två punkterna med koordinaterna (x_1, y_1) och (x_2, y_2). En metod *m* kan med information om kateternas värden beräkna

distansen mellan de två punkterna i Figur 4.1. Däremot vet inte metoden punkternas placering.

En typ *S* införs med ett nytt objekt, $[x_1, x_2, y_1, y_2]$, där argumenten istället betecknar de två punkterna. Det nya objektet tillhandahåller information om punkternas placering men även indirekt information om kateterna. För att *m* korrekt skall kunna hantera ett objekt från *S* sker en simulering mellan objekten, enligt $c_{S\to T}[x_1, x_2, y_1, y_2] == [x_2 - x_1, y_2 - y_1]$, som är en simuleringsfunktion från *S* till *T*, där $[x_2 - x_1, y_2 - y_1] = [k_1, k_2]$.



Figur 4.1 Koordinaterna (x_1, y_1) och (x_2, y_2) simuleras till kateterna k_1 och k_2

Genom att specificera en simuleringsfunktion $c_{S\to T}$, kan objekt som tillhör en typ *S* avbildas så att det beter sig som objekt som tillhör typ *T*. Eftersom ett objekt som tillhör *S* nu beter sig som ett objekt som tillhör *T*, kan en klient som anropar *m* inte påverkas, och då är enligt LSP *S* en subtyp av *T*.

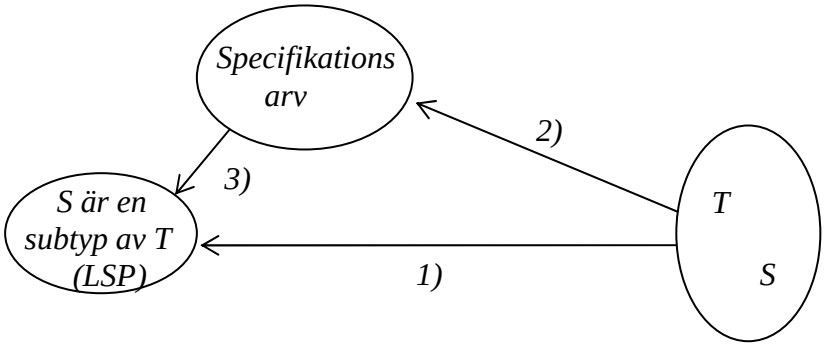
Därför, om alla typer i en programvara är en beteendemässig subtyp av en typ, kan teknik för simulering av objekt som tillhör en typ användas vid modulär verifikation. Om subtypen är en beteendemässig subtyp krävs inte heller någon ny verifikation för metoden. Även om följande definition av Cardelli inte är helt relevant i sammanhanget, kan Dharas jämförelse mellan Cardellis definition för structural subtyping och BS ge läsaren ytterligare förståelse för vad BS egentligen är. BS är baserad både på syntax och semantik för typer. Definition av Cardelli; ”Subtyp är en relation mellan typer, liknande relationen mellan mängder. Om *S* är en subtyp till *T*, så är ett godtyckligt objekt som tillhör typen *S* också ett objekt som tillhör typen *T*.” [2].

Alltså, ett objekt som tillhör *S* har alla egenskaperna som ett objekt som tillhör *T*. För att bättre förstå vad BS är beskriver Dhara skillnaden mellan en strukturell subtyp och BS: ”Syntaktiska villkor som för strukturell subtyp, garanterar att ett uttryck för en subtyp kan ersätta ett uttryck för en supertyp utan typfel.” och ”Semantiska villkor garanterar att objekt som tillhör en subtyp beter sig som objekt som tillhör en supertyp så att inte ett oförutsett beteende uppstår.”.

4.2 Specifikationsarv och Pre-Behavioral Subtyping (PBS)

Dharas artikel [3] är komplex och det är till en början inte speciellt tydligt vad Dhara egentligen visar. Efter en utförlig analys har dock meningen med artikeln avslöjats och Dharas definition av PBS finns i bilaga B.2. Dharas definition av PBS är inte en komplett definition av en subtyp. Det beror på att Dhara har två definitioner av en subtyp. De två definitionerna är *Strong-* respektive *Weak* Behavioral Subtyping (SBS, WBS) och finns i bilaga B.3. PBS utgör den delen av definitionerna för SBS och WBS som är lika. SBS och WBS används beroende på syfte. Vid användande av WBS kan inte en klient använda extra operationer som tillhör WBS och därför spelar det ingen roll om operationer för subtypen inte uppfyller klientens krav. Däremot måste subtypens operationer som också supertypen har uppfylla klientens krav. Annars kommer eller kan klienten märka skillnad. För SBS ställs strängare krav och där får inte några operationer bryta mot supertypens specifikation.

Bilden nedan, Figur 4.2, är Dharas del i Figur 1.1. Dhara har bevisat i sin artikel att specifikationsarv 3) medför att LSP är uppfylld. Utifrån modulerna *T* och *S* anger PBS den specifikation 2) som krävs av för- och eftervillkor för att specifikationsarv ska kunna tillämpas. De andra kraven är inte relevanta här men finns i definitionen av PBS.



Figur 4.2 LSP och Dharas begrepp i förhållande till varandra

Specifikationen av för- och eftervillkor finns i definitionen av PBS och i bilaga C, men visas också i Figur 4.3. Numreringen hör till relationerna i bilaga C och står därför inom parentes. Specifikationen i Figur 4.3 utgörs enbart av relationer och är svåra att förstå direkt.

- *Precondition rule*

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow pre_S^m\left(self\wedge, c_{\vec{V}\to\vec{U}}\left(\vec{y}\wedge\right)\right) \qquad ([C.5])$$

- *Postcondition rule*

$$\left(pre_S^m\left(self\wedge, \vec{y}\wedge\right)\Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\Rightarrow \qquad ([C.6])$$

$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)$$

Figur 4.3 Dharas specifikation för för- och eftervillkor

Specifikationen för förvillkoret anges i *Precondition rule* och eftervillkoret anges i *Postcondition rule*. Det som är nytt i relationerna är argumentvektorn $\vec{y}$ med respektive beteckning $\vec{y}\wedge$ eller $\vec{y}'$. En argumentvektor innehåller ett eller flera argument och $\vec{y}\wedge$ anger värdet av argumenten för förvillkor och $\vec{y}'$ anger värdet av argumenten för eftervillkor.

5 Hur man visar att försvagning av kontrakt definierar en subtyp

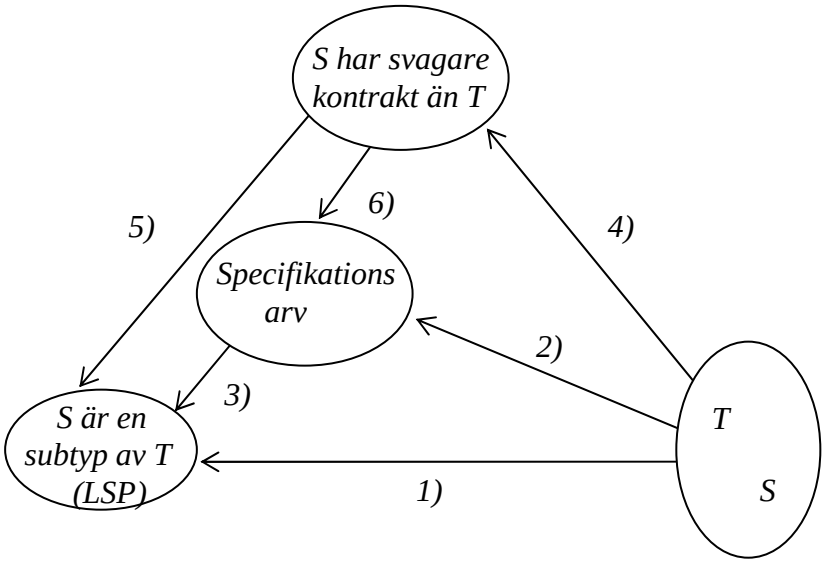
I avsnitt 3.3 resonerades det runt begrepp och definitioner för ARR, LSP och SERG och med föregående kapitel som beskriver Dharas teknik finns nu alla bitar på plats för att fokusera på problemet i detalj. I avsnitt 5.1 och 5.2 utreds och förklaras inbördes förhållanden mellan LSP, SERG:s angreppssätt och Dharas teknik. I avsnitt 5.2 också sker en övergång från SERG:s definition i textform till relationer i symbolisk form. Detta eftersom alla bevis i kapitel 6 nästan helt består av logiska operationer för relationer.

I avsnitt 5.3 och 5.4 kommer även att diskuteras en brist i SERG:s definition. Första delen av SERG:s definition är nämligen inte tillräcklig och definitionen kan därför inte leda till en

subtyp. Sist i det här kapitlet följer därför en redogörelse till varför definitionen inte är tillräcklig samt en komplettering av SERG:s definition så att den blir tillräcklig. Ett bevis för detta följer i avsnitt 6.3.

5.1 Möjliga alternativ

Syftet med uppsatsen är alltså att visa att om kontraktet för en typ *S* är svagare än för en typ *T*, då är *S* en subtyp till *T*. Istället för att visa det varje gång på mer eller mindre olika bra sätt är syftet att formellt visa att svagare kontrakt definierar subtyper. Kraven beskrivs olika av SERG respektive Dhara där Dhara försöker avgöra detta på ett formaliserat sätt. Dharas sätt fungerar men är besvärligt eftersom han använder en komplex metod för att visa detta vid varje enskilt tillfälle. Med hjälp av illustrationen i Figur 5.1 (identisk med Figur 1.1) kan de möjliga alternativen för att bevisa SERG:s hypotesen studeras och utredas.



Figur 5.1 LSP, SERG:s angreppssätt och Dharas teknik i förhållande till varandra

Utgångspunkten är den högra bubblan som innehåller två moduler *S* och *T*. Målet oavsett alternativ i bilden är den vänstra ellipsen. Nämligen att *LSP* är uppfylld. En möjlighet är genom ett intuitivt resonemang 1) som kan leda till att *S* är en subtyp av *T*. Alltså ett liknande resonemang som för LSP i exemplet i avsnitt 3.3. En annan möjlighet att komma fram till *LSP* är att använda Dharas komplexa metod och utnyttja beviset för specifikationsarv i 3).

SERG:s angreppssätt illustreras av 4) och 5). Första delen av SERG:s definition, se Figur 3.7, motsvarar direkt 4), och andra delen av SERG:s definition motsvarar 5), se Figur 3.8. Om man kan visa att ett kontrakt är svagare än ett annat, alltså uppfylla första delen av SERG:s definition, som här innebär att *S* har ett svagare kontrakt än *T*, har den översta bubblan nåtts.

Om det är visat att *S* har svagare kontrakt än *T* återstår sedan två alternativ. Att direkt visa den andra delen av SERG:s definition, 5). Det är troligen inte möjligt att bevisa den här vägen direkt, men det lämnas till vidare arbeten. Mer troligt är att det är nödvändigt att uppfylla Dharas villkor som via specifikationsarv leder till LSP. Problemet är att Dharas metod inte är hela vägen utan endast 3). Om det går att bevisa att när *S* har svagare kontrakt än *T*, så är också kravet för Dharas specifikationsarv uppfyllt, som i sin tur betyder att LSP är uppfylld. Målet med uppsatsen är att visa just det sambandet, se 6) i Figur 5.1, alltså att SERG:s försvagning av kontrakt uppfyller kraven för Dharas specifikationsarv, och få ett formellt bevis. Hur ska det här bevisas?

Eftersom många relationer mellan olika begrepp i den här uppsatsen är komplexa är det kanske inte möjligt att diskutera sig fram till ett bevis. Ett enklare sätt är att översätta definitioner till relationer som symboliserar dem. Med hjälp av diskret matematik kan då problemet lösas på enklare sätt. Diskret matematik innehåller de logiska operationer som är applicerbara på problemet. I nästa avsnitt sker en övergång från textform för de olika definitionerna till relationer i symbolisk form.

5.2 Bevisföring

För att nå målet, alltså *Specifikationsarv* i Figur 5.1 måste det bevisas att om *S har svagare kontrakt än T* är uppfyllt så följer 6). Första delen av SERG:s definition symboliseras av relationerna [C.3] och [C.4] i bilaga C men visas också i Figur 5.2. Kortfattat kan sägas att de här två relationerna precis beskriver det som texten säger i SERG:s definition i Figur 3.7.

Relationen i *Precondition rule* symboliserar att: *Förvillkoret är svagare än eller lika med förvillkoret för originalet* i SERG:s definition. Den andra relationen, *Postcondition rule, symboliserar* att: *Eftervillkoret är starkare än eller lika med eftervillkoret för originalvillkoret i domänen för originalets förvillkor* i SERG:s definition.

- *Precondition rule*

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right) \quad \text{([C.3])}$$

- *Postcondition rule*

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right) \Rightarrow \right. \quad \text{([C.4])}$$
$$\left. post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)$$

Figur 5.2 Första delen av SERG:s definition i symbolisk form

Det som ska bevisas är att [C.3] och [C.4] är ekvivalent med, eller åtminstone ska implicera Dharas specifikation för eftervillkor i regeln [C.6], se Figur 5.3.

- *Postcondition rule*

$$\left(pre_S^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right) \Rightarrow \quad \text{([C.6])}$$
$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)$$

Figur 5.3 Dharas specifikation för eftervillkor

Matematiskt innebär det att ([C.3] $\wedge$ [C.4]) $\Rightarrow$ [C.6]. Innan presentation av beviset krävs dock en komplettering av SERG:s definition.

5.3 SERG:s definition av ett försvagat kontrakt är inte tillräckligt för att leda till en subtyp

Hypotesen som skall bevisas i föregående avsnitt är inte möjlig att bevisa med nuvarande definition. Anledningen är att SERG:s definition inte är tillräcklig. Beviset att så är fallet följer i avsnitt 6.3. Komplettering av definitionen sker i avsnitt 5.4. Där ges också en ny symbolisk definition.

5.4 Komplettering av SERG:s definition

Första delen av SERG:s definition säger att ett omdefinierat kontrakt är svagare än originalet om två villkor är uppfyllda. Det bevisas i avsnitt 6.3 att de två villkor som finns i definitionen inte är tillräckliga och måste därför kompletteras med relationen [C.2], se bilaga C eller Figur 5.4.

- *Rule for T*

$$pre_T^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_T^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right) \qquad ([C.1])$$

- *Rule for S*

$$pre_S^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right) \qquad ([C.2])$$

Figur 5.4 Allmän definition av ett kontrakt

Relationen i [C.2] betyder att: *Förvillkoret för den nya typen (S) leder till eftervillkoret för densamma*. Reglerna för [C.1] och [C.2] kan sägas vara det allmänna kontraktet för en operation. Anledningen till att de här båda relationerna inte finns med i SERG:s definition av kan vara att de är underförstådda i sammanhanget eftersom det egentligen är självklart att de måste gälla. Men det är egentligen inte så självklart eftersom [C.1] inte behöver vara med. Det betyder att [C.1] finns med implicit i SERG:s definition. Det gör däremot inte [C.2] och måste därför vara med för sig. I Figur 5.5 står relationen [C.2] i textform och kompletterar första delen i SERG:s ursprungliga definition.

- *Förvillkoret för den nya typen leder till eftervillkoret för densamma*
- Förvillkoret är svagare än eller lika med förvillkoret för originalet
- Eftervillkoret är starkare än eller lika med eftervillkoret för originalvillkoret i domänen för originalets förvillkor

Figur 5.5 Första delen av SERG:s definition efter komplettering

Så det som ska bevisas är egentligen att: ([C.2] $\wedge$ [C.3] $\wedge$ [C.4]) $\Rightarrow$ [C.6].

6 Bevis

Följande tre avsnitt innehåller tre bevis. De första två bevisen är de som bevisar att SERG:s krav (efter komplettering) uppfyller Dharas krav. Det tredje beviset bevisar att SERG:s ursprungliga definition inte är tillräcklig.

6.1 Bevis av förvillkor

För att visa att Dharas specifikation impliceras av svaga kontrakt måste vi visa att det gäller relationerna för både förvillkoren och eftervillkoren. SERG:s första del av definitionen för ett kontrakt utgörs av [C.3] i Figur 6.1 och det är den här regeln som ska uppfylla Dharas regel för förvillkor.

- Precondition rule

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right) \qquad \text{([C.3])}$$

Figur 6.1 Precondition rule i första delen av SERG:s definition i symbolisk form

I klartext betyder regeln i Figur 6.1 att ”Förvillkoret är svagare än eller lika med förvillkoret för originalet” och egentligen uttrycks den här regeln av SERG utan simuleringsfunktion, alltså som $pre_T^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right)$ vilket inte verkar helt korrekt. Men för förvillkoret spelar faktiskt inte simuleringsfunktionen någon roll. I Dharas regel för förvillkoret [C.5] (identisk med [C.3]) simuleras ett objekt med simuleringsfunktionen, här objektet self, som tillhör subtypen *S* till ett objekt som tillhör supertypen *T*. Med simuleringsfunktionen simuleras värden från objekt som tillhör subtypen till giltiga värden för objekt som tillhör supertypen. Dock sker ingen simulering mellan typerna. Eftersom en klient kommunicerar med en leverantör genom separata gränssnitt där det gemensamma gränssnittet utgör ett kontrakt kan en klient inte märka någon skillnad på en leverantör. I det tidigare exemplet för en simuleringsfunktion, sker en simulering mellan objekt enligt $c_{S\to T}[x_1, x_2, y_1, y_2] == [x_2 - x_1, y_2 - y_1]$, där kateterna ges av $[x_2 - x_1, y_2 - y_1] = [k_1, k_2]$. Klienten är helt ovetande om en leverantör använder sig av en simuleringsfunktion eller inte. Klienten ser endast två argument (kateterna) och kan inte observera om argumenten är

differenser av koordinater. Det är därför möjligt att från klientens perspektiv bortse från simuleringsfunktionen och fortfarande ha ett giltigt kontrakt.

6.2 Bevis av eftervillkor efter komplettering av SERG:s definition

För eftervillkoret är det inte nödvändigt att visa separat hänsyn för simuleringsfunktionen eftersom den ingår. Beviset finns förutom nedan även i bilaga D. Formellt påstås att: [C.2] och [C.3] och [C.4] implicerar:

$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow\ post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$ i [C.6].

Alltså

[C.2] och [C.3] och [C.4]:

$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right)\right)\ \wedge$$
$$\left(pre_S^m\left(self\wedge, \vec{y}\wedge\right)\Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\ \wedge$$
$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$

(Law of Syllogism) $\Rightarrow$

$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\ \wedge$$
$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$
$$\Leftrightarrow$$
$$\left(\neg\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\vee\ post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\ \wedge$$
$$\left(\neg\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\vee\ \left(\ post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$

(Distributive law) $\Leftrightarrow$

$$\neg\, pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \vee \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\wedge\right.$$
$$\left.\left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$

(Modus Ponens) $\Rightarrow$

$$\neg\, pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \vee post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$$
$$\Leftrightarrow\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$$

vilket är ekvivalent med högerledet i [C.6].

Figur 6.2 Bevis av eftervillkor

6.3 Bevis av att SERG:s definition inte är tillräcklig

Beviset följer av en stor mängd formler i ett av författarens kollegieblock. Med det menas inte att beviset består av alla dessa formler. Formlerna symboliserar istället alla de lönlösa försök att bevisa hypotesen utifrån SERG:s ursprungliga definition. Utan kompletteringen av definitionen i avsnitt 5.4 är det nämligen inte möjligt. Men att bevisa att så är fallet, följer egentligen vid ett försök till bevis utifrån SERG:s ursprungliga definition, något som efteråt är helt uppenbart. Det kan därför verka märkligt att det krävdes så många försök, men det visar faktiskt bara hur komplexa relationerna är mellan olika begrepp i den här uppsatsen. Till exempel så är ju den ursprungliga definitionen tillräcklig för att bevisa förvillkoret.

Formellt påstods från början att: [C.3] och [C.4] (här utan [C.2]) implicerar $pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$ i [C.6]. På samma sätt som i föregående kapitel fast nu med endast två startvillkor, se Figur 6.3:

[C.3] och [C.4]:

$$\left(pre_T^m\left(c_{S\to T}(self\wedge),\vec{y}\wedge\right)\Rightarrow pre_S^m\left(self\wedge,\vec{y}\wedge\right)\right)\wedge$$
$$\left(pre_T^m\left(c_{S\to T}(self\wedge),\vec{y}\wedge\right)\Rightarrow\left(post_S^m\left(self\wedge,self',\vec{y}\wedge,\vec{y}',result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge),c_{S\to T}(self'),\vec{y}\wedge,\vec{y}',c_{U_r\to V_r}(result')\right)\right)\right)$$

(Law of Syllogism) $\Rightarrow$

$$\left(\neg\, pre_T^m\left(c_{S\to T}(self\wedge),\vec{y}\wedge\right)\vee pre_S^m\left(self\wedge,\vec{y}\wedge\right)\right)\wedge$$
$$\left(\neg\, pre_T^m\left(c_{S\to T}(self\wedge),\vec{y}\wedge\right)\vee\left(post_S^m\left(self\wedge,self',\vec{y}\wedge,\vec{y}',result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge),c_{S\to T}(self'),\vec{y}\wedge,\vec{y}',c_{U_r\to V_r}(result')\right)\right)\right)$$

(Distributive law) $\Leftrightarrow$

$$pre_T^m\left(c_{S\to T}(self\wedge),\vec{y}\wedge\right)\Rightarrow$$
$$\left(pre_S^m\left(self\wedge,\vec{y}\wedge\right)\wedge\left(post_S^m\left(self\wedge,self',\vec{y}\wedge,\vec{y}',result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge),c_{S\to T}(self'),\vec{y}\wedge,\vec{y}',c_{U_r\to V_r}(result')\right)\right)\right)$$

Figur 6.3 Bevis av att SERG:s definition inte är tillräcklig

Om ovanstående uttryck ska kunna uttryckas som [C.6], måste gälla att de tre sista villkoren (i Distributive law) är ekvivalenta med eller åtminstone implicerar högerledet i [C.6]. Detta följer direkt av Modus Ponens och den enda möjligheten för att uppnå detta är att någon av relationerna $pre_S^m \Leftrightarrow post_S^m$, $pre_S^m \Rightarrow post_S^m$ eller $pre_S^m \Leftarrow post_S^m$ gäller. Sista relationen kan direkt uteslutas och även den första eftersom en sådan relation troligen inte åstadkommer något och därför är meningslös. Därmed återstår bara den andra relationen som enda alternativ. Det bevisar därför att den här relationen är nödvändig och eftersom den är identisk med [C.2] är det också bevisat att SERG:s definition kräver [C.2] som komplettering för att uppfylla [C.6].

7 Slutsats, diskussion och vidare arbeten

Målet med uppsatsen har nåtts. Det har visats att SERG:s krav för specifikation uppfyller Dharas krav för specifikationsarv. Ett formellt bevis finns nu för den tidigare hypotesen. Uppsatsen har också lett till en extra slutsats. Det har bevisats att SERG:s ursprungliga definition inte är tillräcklig. SERG:s ursprungliga definition har därför kompletterats.

Något som lätt förbryllar läsaren är termerna svaga respektive starka. Från början härrör termerna från Meyers ARR. En läsare vill ”gärna” byta plats på betydelsen av termerna. Anledningen är att ett svagt kontrakt gärna betraktas som mer generellt än ett starkt. Det betyder att ett svagt kontrakt måste göras starkare vid till exempel en substitution. Det finns fördelar med nuvarande betydelse men det rekommenderas att termerna svaga respektive starka kontrakt ändras för att inte orsaka ytterligare förvirring.

Det är troligen inte möjligt att bevisa att svagare kontrakt leder till att LSP är uppfylld utan att använda sig av Dharas specifikationsarv. Det är dock inte bevisat och lämnas därför till vidare arbeten.

Referenser

[1] Blom, Martin, Brunström, Anna och Nordby, Eivind J., *”On the Relation Between Design Contracts and Errors”* – A Software Development Strategy, ECBS 2002, Lund, Computer Science, Karlstad University.

[2] Cardelli, Luca, *“Typeful Programming”*, In Neuhold, E. J., Paul, M., Eds, Formal Description of Programming Concepts, Springer-Verlag, New York, 1991.

[3] Dhara, Krishna Kishore, och Leavens, Gary T., *”Forcing Behavioral Subtyping Trough Specification Inheritance”*, Department of Computer Science, Iowa State University, USA, IEEE 1996.

[4] Liskov, Barbara H., *”Data Abstraction and Hierarchy”*, OOPSLA ’87 Addendum to the Proceedings, pp 17--34, October 1987

[5] Liskov, Barbara H., Wing, Jeannette M., *”A behavioral notion of suptyping”*, ACM Transactions on Programming Languages and Systems, 16(6):1811-1841, November 1994.

[6] Meyer, Bertrand, *“Object-oriented Software Construction”*, Prentice Hall international series in computer science, University Press, Cambridge, 1997.

A Notationer

$x \setminus s$	värdet av x i tillståndet s
$x\hat{}$	värdet av x i förvillkoret, eller x_{pre}
x'	värdet av x i eftervillkoret, eller x_{post}
m	metod m (funktion, rutin)
pre_T^m	förvillkor för metoden m för typen T
$post_T^m$	eftervillkor för metoden m för typen T
$c_{S \to T}$	simuleringsfunktion, här från subtypen S till supertypen T, samma som $A : S \to T$
y	argument (element)
$\vec{y}$	argumentvektor, $\vec{y} = (y_0, y_1, y_2, ..., y_n)$
$\vec{U}, \vec{V}$	typvektor, tex $\vec{U} = (float, Int, float, ..., classname)$, samma som α och β
$\vec{y} : \vec{U}$	argument y_i i $\vec{y}$ av respektive typ i $\vec{U}$, tex $\vec{y} : \vec{U} = (y_0 : float, y_1 : Int, y_2 : float, ..., y_n : classname)$
$\lvert\vec{U}\rvert$	mängden av distinkta typer i $\vec{U}$, $\vec{U} = (float, Int, classname)$

B Liskovs och Dharas definitioner av subtyp

B.1 Liskovs definition av en subtyp

Dharas definition av en subtyp utgår från Liskovs definition av subtypsrelation [5] (se nästa sida) där Dharas definition är en modifikation av Liskovs definition. Liskovs SR, baseras på relationen mellan två typer, σ och τ , där varje typ är specificerad att bevara respektive invariant, I_σ och I_τ, och uppfylla villkoren C_σ och C_σ. En subtyp ärver de egenskaper som innehas av en supertyp, och subtypen kan därför använda alla supertypens metoder, men subtypen kan också ha egna metoder. En kort beskrivning [5] av SR ges av Liskov och här ges en mer utförlig. Först förklaras att ”abstraction function”, $A: S \to T$ och ”renaming map”, $R: M \to N$, tillsammans motsvarar Dharas simuleringsfunktion $c_{S \to T}$. I ”method’s rule” och “constraint rule” där *x* är ett objekt av typen σ , tillhör objektet *x* i x_{pre} och i x_{post} subtypen *S*. Därför kan inte de villkor som gäller för typen τ i supertypen *T* appliceras direkt på objektet *x*. Simuleringsfunktionen avbildar därför objektet *x* för att passa till de villkor som gäller för typen τ , och med funktionen $R: M \to N$ avbildas värden i subtypen till supertypen. Under förutsättning att det existerar en simuleringsfunktion, här $A: S \to T$, anges i 1) i definitionen att simuleringsfunktionen bevarar supertypens invarianter för alla giltiga värden i subtypen, då subtypens värde genom simuleringsfunktionen ges ett giltigt värde i supertypen. Simuleringsfunktionen behöver ej heller vara en surjektion, vilket betyder att för varje objekt i supertypen behöver det inte existera ett motsvarande objekt i subtypen, men för varje objekt i subtypen kan det existera flera motsvarande objekt i supertypen. I 2) i definitionen anges villkor vid arv för de egenskaper när en subtyp ärver av en supertyp, där en metod *m* i subtypen skall uppträda på motsvarande sätt som metoden *m* i supertypen. Regeln för kontravarians avser argument, där en metod *m* måste ha lika antal argument i både *S* och *T*, och där för varje *i*, typen α_i av typvektorn α , är en subtyp till argumentet β_i av typvektorn β . Regeln för kovarians avser resultat, där en metod *m* returnerar ett resultat i både *S* och *T*, eller inte alls. Om ett resultat returneras, så är resultattypen för metoden *m* i *T*, en typ i typvektorn α , och resultattypen för metoden *m* i *S*, en typ i typvektorn β . Därför anger $\beta < \alpha$ att β är en subtyp av α . ”Exception rule”

(undantag) anger att objekt som förväntas returneras av en metod i en subtyp utgör en delmängd av en supertyps, eftersom en supertyp inte kan hantera ett oförutsett objekt. Undantag som tillhör subtypen kan endast vara sådana undantag som tillhör supertypen. Regeln för eftervillkor visar att eftervillkoret för en metod *m* i en subtyp är starkare än eftervillkoret för en metod *m* i supertypen. Det betyder att ett eftervillkor i supertypen alltid impliceras av motsvarande eftervillkor i subtypen.

Definition of the Subtype Relation (SR), <: $\sigma = \langle O_\sigma, S, M \rangle$ is a subtype of $\tau = \langle O_\tau, T, N \rangle$ if there exists an abstraction function, $A : S \to T$, and a renaming map, $R : M \to N$, such that:

1) The abstraction function respects invariants:

   - *Invariant rule*: $\forall\, s : S \,.\, I_\sigma(s) \Rightarrow I_\tau(A(s))$

   A may be partial, need not to be onto, but can be many-to-one.

2) Subtype methods preserve the supertype method's behavior. If m_τ of τ is the corresponding renamed method m_σ of σ, the following rules must hold:

   - Signature rule:

     - *Contravariance of arguments*: m_τ and m_σ have the same number of arguments. If the list of argument types of m_τ is α_i, and that of m_σ is β_i, then $\forall\, i \,.\, \alpha_i < \beta_i$.

     - *Covariance of result:* Either both m_τ and m_σ have a result or neither has. If there is a result, let m_τ's result type be α and m_σ's be is β. Then $\beta < \alpha$.

     - *Exception rule:* The exceptions signaled by m_σ are contained in the set of exceptions signaled by m_τ.

   - Method's rule. $\forall\, x : \sigma$:

     - *Pre-condition rule:* $m_\tau.pre\lfloor A(x_{pre})/\, x_{pre} \rfloor \Rightarrow m_\sigma.pre$.

     - *Post-condition rule:* $m_\sigma.post \Rightarrow m_\tau.post\lfloor A(x_{pre})/\, x_{pre},\, A(x_{post})/\, x_{post} \rfloor$.

3) Subtype constraints ensure supertype constraints.

   - *Constraint rule:* For all computations, *c*, and all states ρ and ψ in *c* such that ρ precedes ψ, $\forall\, x : \sigma :\, C_\sigma \Rightarrow C_\tau \lfloor A(x_\rho)/\, x_\rho,\, A(x_\psi)/\, x_\psi \rfloor$

B.2 Dharas definition av en subtyp

Dharas definition [3]:

Definition of Pre-Behavioral Subtyping (PBS): *S* is a pre-behavioral subtype of *T* with respect to a binary relation $\leq$ on types if an only if the following properties are satisfied:

Syntactic: For every method *m* in *T* there exists a method *m* in *S* such that the following hold:

Contravariance of arguments: If the types of the additional arguments of *m* in *S* and *T* are $\vec{U}$ and $\vec{V}$ respectively, then $\left|\vec{U}\right| = \left|\vec{V}\right| \wedge \left(\forall\, i \,.\, V_i \leq U_i\right)$.

Covariance of result: If the result types of *m* in *S* and *T* are U_r and V_r respectively, then $U_r \leq V_r$.

Exception rule: For each exception *e*: E_S of *m* in *S*, *m* in *T* has an exception *e*: E_T, and $E_S \leq E_T$.

Semantic: There exists a family of simulation functions, $\left\langle c_{X \to Y} : X \leq Y \right\rangle$, such that the following hold:

Invariant rule: For all values v_S of *S*, $I_S(v_S) \Rightarrow I_T(c_{S \to T}(v_S))$

Method's rule: For all common methods *m*, if the types of the additional arguments types of *m* in *S* and *T* are $\vec{U}$ and $\vec{V}$ respectively, and the result types of *m* in *S* and *T* are U_r and V_r respectively, then for all objects *self*: *S*, $\vec{y} : \vec{V}$, and result: U_r

- *Precondition rule.*

$$pre_T^m\left(c_{S \to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow pre_S^m\left(self\wedge, c_{\vec{V} \to \vec{U}}\left(\vec{y}\wedge\right)\right)$$

- *Postcondition rule.*

$$\left(pre_S^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right) \Rightarrow$$
$$\left(pre_T^m\left(c_{S \to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow post_T^m\left(c_{S \to T}(self\wedge), c_{S \to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r \to V_r}(result')\right)\right)$$

B.3 Strong och weak behavioral subtyping

Dharas definitioner [3]:

Definition of Strong Behavioral Subtyping (SBS), $\leq_S$: *S* is a strong behavioral subtype of *T* if *S* is a pre-behavioral subtype of *T* with respect to $\leq_S$, and the following constraint rule is satisfied:

Semantic:

Constraint rule: For all valid computations *c*, for all states pre and post in *c* such that pre precedes post, and for all objects self: *S*,

$$C_S(self \setminus pre, self \setminus post) \Rightarrow C_T(c_{S\to T}(self \setminus pre), c_{S\to T}(self \setminus post))$$

Definition of Weak Behavioral Subtyping (WBS), $\leq_W$: *S* is a weak behavioral subtype of *T* if *S* is a pre-behavioral subtype of *T* with respect to $\leq_W$, and the following constraint rule is satisfied:

Semantic:

Constraint rule: For all valid computations *c*, which do not invoke extra methods of *S*, for all states pre and post in *c* such that pre precedes post, and for all objects self: *S*,

$$C_S(self \setminus pre, self \setminus post) \Rightarrow C_T(c_{S\to T}(self \setminus pre), c_{S\to T}(self \setminus post))$$

C Sammanställning och definition av kontrakt

Sammanställningen består av tre olika kontrakt och ges i [C.1]-[C.6].

Allmän definition av ett kontrakt

- *Rule for T*

$$pre_T^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_T^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right) \quad [C.1]$$

- *Rule for S*

$$pre_S^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right) \quad [C.2]$$

Definition av ett kontrakt där *S* har ett svagare kontrakt än *T* [1]

- *Precondition rule*

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right) \quad [C.3]$$

- *Postcondition rule*

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right) \Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right) \quad [C.4]$$

Definition of Pre-Behavioral Subtyping (PBS) i [3], delen för *Method's rule* med lika typer:

- *Precondition rule*

$$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right) \quad [C.5]$$

- *Postcondition rule*

$$\left(pre_S^m\left(self\wedge, \vec{y}\wedge\right) \Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right) \Rightarrow \left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right) \Rightarrow post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right) \quad [C.6]$$

D Bevis av eftervillkor

Formellt påstås att: [C.2] och [C.3] och [C.4] implicerar

$pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow\ post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$ i [C.6].

[C.2] och [C.3] och [C.4]:

$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow pre_S^m\left(self\wedge, \vec{y}\wedge\right)\right)\ \wedge$$
$$\left(pre_S^m\left(self\wedge, \vec{y}\wedge\right)\Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\ \wedge$$
$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$

(Law of Syllogism) $\Rightarrow$

$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\ \wedge$$
$$\left(pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$
$\Leftrightarrow$
$$\left(\neg\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\vee\ post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\right)\ \wedge$$
$$\left(\neg\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\vee\ \left(\ post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\right.\right.$$
$$\left.\left.post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$

(Distributive law) $\Leftrightarrow$

$$\neg\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\ \vee\ \left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\wedge\right.$$
$$\left.\left(post_S^m\left(self\wedge, self', \vec{y}\wedge, \vec{y}', result'\right)\Rightarrow\ post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)\right)\right)$$

(Modus Ponens) $\Rightarrow$

$$\neg\ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\ \vee\ post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$$
$$\Leftrightarrow\ \ pre_T^m\left(c_{S\to T}(self\wedge), \vec{y}\wedge\right)\Rightarrow\ post_T^m\left(c_{S\to T}(self\wedge), c_{S\to T}(self'), \vec{y}\wedge, \vec{y}', c_{U_r\to V_r}(result')\right)$$

vilket är ekvivalent med högerledet i [C.6].