Hoofdstuk 15

Logica en imperatief programmeren

15.1 INLEIDING

De logische formalismen die eerder in dit boek aan de orde zijn geweest, worden in de informatica ook gebruikt bij het redeneren *over* imperatieve programma's en hun verwerking. Daarbij fungeert de gegeven semantiek van logische talen als inspiratie voor het opzetten van een semantiek voor programmeertalen. De eerste vraag daarbij is de volgende. Tot nu toe dienden logische formalismen voor de weergave van 'statische' beweringen over situaties (modellen). Hoe kunnen we ons gezichtspunt nu zo uitbreiden dat 'dynamische' programmeertekst, met opeenvolgende instructies om situaties juist te veranderen, toch ook binnen het logische kader komt te passen?
Men zou in eerste instantie kunnen denken dat de fysische activiteit binnen de computer zelf een realistische 'interpretatie' van een programma vormt ('de betekenis van een programma is wat het doet'). Dit gezichtspunt helpt echter niet bij het *ontwerp* van programma's of het toetsen van hun *correctheid*. Vandaar dat we meer abstracte, geïdealiseerde modellen hanteren om de betekenis van programma's weer te geven. Hierbij zijn overigens nog vele keuzes mogelijk, afhankelijk van het beoogde doel.
In dit hoofdstuk geven we een semantiek voor imperatieve programma's in termen van *overgangen van geheugentoestanden* van een rekenautomaat.

15.2 PROGRAMMA'S EN TOESTANDSOVERGANGEN

Beschouw om te beginnen een of ander model $M = (D, I)$ voor een predikaatlogische taal. We denken voortaan over D als een *gegevensstructuur* waarop we kunnen rekenen. Predikaatlogische formules φ met vrije variabelen $x_1, \ldots, x_n$ kunnen vervolgens in deze situatie beschouwd worden als beweringen over momentane waarden van die variabelen, weergegeven in een *bedeling*. En aan dit laatste begrip valt nu een heel concrete computationele duiding te geven. Een bedeling was in het

voorgaande een functie die aan variabelen x, y, z, ... waarden $b(x)$, $b(y)$, $b(z)$, ... uit het domein D toekent, of in meer concrete termen, een 'vulling van de registers x, y, z, ... met gegevens'. Dat laatste is een goede omschrijving van een momentopname van een geheugentoestand in een rekenautomaat. Een rekenproces gestuurd door een programma verandert stapsgewijs waarden in registers, dat wil zeggen, de lopende bedeling.

Voorbeeld 15.1

Het eerdere STIP–programma voor vermenigvuldiging uit voorbeeld 14.3:

$(x := n\ ;\ (y := 0\ ;$ WHILE $\neg(x = 0)$ DO $((z := m\ ;$ WHILE $\neg(z = 0)$ DO $(y := Sy\ ;\ z := Pz))\ ;\ x := Px)))$

geeft de volgende overgangen te zien bij input $n = 2$ en $m = 2$ (het streepje staat hier voor een willekeurige waarde):

lopende bedeling b	$b(x)$	$b(y)$	$b(z)$
begintoestand b_0	–	–	–
b_1	2	–	–
b_2	2	0	–
b_3	2	0	2
b_4	2	1	2
b_5	2	1	1
b_6	2	2	1
b_7	2	2	0
b_8	1	2	0
b_9	1	2	2
b_{10}	1	3	2
b_{11}	1	3	1
b_{12}	1	4	1
b_{13}	1	4	0
eindtoestand b_{14}	0	4	0

De totale actie van een programma π bestaat dus uit vele achtereenvolgende 'lokale' veranderingen in de bedeling, en kan worden beschreven als: een bedeling b – de 'begintoestand' – is na een geslaagde uitvoering van π overgegaan in een andere bedeling e – de 'eindtoestand'. Anders gezegd: een programma bepaalt een overgangsrelatie tussen bedelingen.

Dit is overigens een zeer 'ruwe' beschrijving. Er wordt immers alleen naar de begin- en eindtoestand gekeken. Een rijkere semantiek ontstaat als ook andere aspecten in het spel komen, zoals de doorlopen tussenstadia ('traces'), gemaakte keuzes of gebruikte tijdspannes. Voor onze doeleinden hier volstaat echter het eenvoudigste beeld.

15.3 STIP-PROGRAMMA'S

Om de ideeën uit de vorige paragraaf concreter te kunnen uitwerken, keren we terug naar de STIP-programma's uit hoofdstuk 14, nu iets algemener omschreven.

DEFINITIE 15.1

Syntaxis van STIP

Net als in het vorige hoofdstuk, alleen nu voor predikaatlogische termen in het algemeen en niet slechts voor rekenkundige termen.

Interpretatie van STIP

STIP-programma's π worden nu van een betekenis voorzien middels de volgende inductieve toekenning van bijbehorende overgangsrelaties $T(\pi)$ in een model $M = (\boldsymbol{D}, I)$.

DEFINITIE 15.2

Semantiek van STIP-programma's

a $b_1\ T(x := t)\ b_2 \Leftrightarrow b_2 = b_1[x \mapsto V_{(\boldsymbol{D}I),b_1}(t)]$

b $b_1\ T(\pi_1 ; \pi_2)\ b_2 \Leftrightarrow$ er is een bedeling b zodat $b_1\ T(\pi_1)\ b$ en $b\ T(\pi_2)\ b_2$

c $b_1\ T(\text{IF } \varepsilon \text{ THEN } \pi_1 \text{ ELSE } \pi_2)\ b_2 \Leftrightarrow (\boldsymbol{D}, I), b_1 \models \varepsilon$ en $b_1\ T(\pi_1)\ b_2$, of $(\boldsymbol{D}, I), b_1 \not\models \varepsilon$ en $b_1\ T(\pi_2)\ b_2$

d $b_1\ T(\text{WHILE } \varepsilon \text{ DO } \pi)\ b_2 \Leftrightarrow$ er is een *eindige* rij bedelingen $b_1 = b^1, \ldots, b^n = b_2$ zodat voor alle $i < n$: $(\boldsymbol{D}, I), b^i \models \varepsilon$ en $b^i\ T(\pi)\ b^{i+1}$ en $(\boldsymbol{D}, I), b_2 \not\models \varepsilon$

Voorbeeld 15.2

Semantiek van STIP

Laat $\pi = (\text{IF } x < y \text{ THEN } z := y - x \text{ ELSE } z := x - y)$. Laat verder b_1 en b_2 twee bedelingen zijn waarvoor geldt: $b_1(x) = 5$; $b_1(y) = 4$; $b_2(x) = 5$; $b_2(y) = 4$; $b_2(z) = 1$. We werken op $\boldsymbol{D} = \langle \mathbb{N}, <, - \rangle$ met voor de hand liggende I. Nu geldt:

i $(\boldsymbol{D}, I), b_1 \not\models x < y$

Voor $\pi_2 = (z := x - y)$ geldt:

ii $b_2 = b_1[z \mapsto V_{(\boldsymbol{D}I),b_1}(x - y)]$

En dit betekent volgens definitie 15.2a: $b_1\ T(\pi_2)\ b_2$.

Uit i en $b_1\ T(\pi_2)\ b_2$ volgt met 15.2c: $b_1\ T(\pi)\ b_2$.

Determinisme

Een effect van deze definitie is dat STIP–programma's *deterministisch* zijn: vanuit elke bedeling is via $T(\pi)$ hoogstens één andere bedeling bereikbaar. In het algemeen kan men zo'n semantiek voor overgangsrelaties overigens ook voor *indeterministische* programma's geven.

Partiële bereikbaarheid

Wanneer het programma niet termineert, is zelfs helemaal geen andere bedeling bereikbaar. De $T(\pi)$–relatie is dan partieel.

15.4 DE HOARE–CALCULUS VOOR CORRECTHEID

De Hoare–calculus (genoemd naar een van de grondleggers, Hoare) is wellicht de bekendste toepassing van deze semantiek en heeft betrekking op zogenaamde *correctheidsbeweringen*. Dit zijn beweringen van de vorm:

$\{\varphi\}\ \pi\ \{\psi\}$

De betekenis hiervan is: als voor verwerking van π de formule φ geldt in de begintoestand in een model M, dan geldt ψ na verwerking van π in de eindtoestand. De formule φ wordt de 'preconditie' genoemd, ψ de 'postconditie'. Formeel:

DEFINITIE 15.3

Correctheidsbewering
$M \models \{\varphi\}\ \pi\ \{\psi\}$ als voor alle b_1, b_2 met $b_1\ T(\pi)\ b_2$ geldt:
als $M, b_1 \models \varphi$, dan $M, b_2 \models \psi$.

Voorbeeld 15.3

Correctheidsbeweringen
Beschouw de volgende correctheidsbeweringen:
a $\{x = 3\}\ y := x \cdot x\ \{y = 9\}$
b $\{x = z\}\ y := x \cdot x\ \{y = z \cdot z\}$
c $\{x = 1\}\ y := x \cdot x\ \{y = 2\}$
De beweringen a en b zijn juist, c niet.

Zij π het programma $(y := S0\ ;$ WHILE $\neg(x = 0)$ DO $(y := y \cdot x\ ;\ x := Px))$.
Intuïtief valt niet moeilijk in te zien dat dit de functie *faculteit* berekent.
Beschouw nu de volgende correctheidsbeweringen:
d $\{\text{TRUE}\}\ \pi\ \{y = x!\}$
e $\{x = z\}\ \pi\ \{y = z!\}$
Bewering d is onjuist omdat de oude waarde van x in het rekenproces is verdwenen. Bewering e is juist.

Voor lange complexe programma's is het rekengedrag, weerspiegeld in correctheidsbeweringen, niet altijd doorzichtig, zodat systematische

controle urgent wordt. Er bestaan daarom diverse *bewijssystemen* om systematisch bij programma's behorende correctheidsbeweringen te bewijzen, die uitdrukken dat het programma onder zekere voorwaarden (preconditie) precies een gewenst resultaat (postconditie) bereikt.

We geven hier de zogenaamde *Hoare–axiomatiek* voor STIP–programma's, die op natuurlijke wijze de inductieve structuur van onze programmeertaal volgt:

DEFINITIE 15.4

Hoare–axiomatiek

H1 $\{[t/x]\varphi\}\ x := t\ \{\varphi\}$, mits t vrij voor x in φ

H2 uit $\{\varphi\}\ \pi_1\ \{\psi\}$ en $\{\psi\}\ \pi_2\ \{\chi\}$ volgt $\{\varphi\}\ (\pi_1\ ;\ \pi_2)\ \{\chi\}$

H3 uit $\{\varphi \wedge \varepsilon\}\ \pi_1\ \{\psi\}$ en $\{\varphi \wedge \neg\varepsilon\}\ \pi_2\ \{\psi\}$ volgt $\{\varphi\}$ IF εTHEN π_1 ELSE $\pi_2\ \{\psi\}$

H4 uit $\{\varphi \wedge \varepsilon\}\ \pi\{\varphi\}$ volgt $\{\varphi\}$ WHILE ε DO $\pi\{\neg\varepsilon \wedge \varphi\}$

H5 als $\varphi \rightarrow \varphi'$ en $\psi' \rightarrow \psi$ waar zijn op $M = (D, I)$ dan geldt:
uit $\{\varphi'\}\ \pi\{\psi'\}$ volgt $\{\varphi\}\ \pi\{\psi\}$

De eerste drie regels zijn tamelijk voor de hand liggend. De vierde introduceert een nieuw idee: de zogenaamde *'lus–invariant φ'*, een bewering die bij elke doorgang voor π vanuit een ε–toestand blijft opgaan. Vinden van geschikte lus–invarianten is een kunst bij het programmeren.
De vijfde regel zegt dat de preconditie in een correctheidsbewering versterkt en de postconditie verzwakt mag worden.

Voorbeeld 15.4

Een correctheidsbewijs

Beschouw het volgende programma π dat voor een willekeurig natuurlijk getal x de waarde $x + (x-1) + (x-2) + \ldots + 1$ berekent:

$y := 0\ ;\ z := x\ ;$ WHILE $\neg(z = 0)$ DO $(y := y + z\ ;\ z := z - 1)$

We willen de volgende correctheidsbewering bewijzen:

$\{\text{TRUE}\}\ \pi\{y = x \cdot (x+1)/2\}$

Om deze postconditie te bereiken nemen we voor de lus–invariant φ in H4 de formule $y + z \cdot (z+1)/2 = x \cdot (x+1)/2$. Immers, als op een bepaald moment $z = 0$ (dit is $\neg\varepsilon$) geldt, dan stopt de lus en $\varphi \wedge \neg\varepsilon$ impliceert dan $y = x \cdot (x+1)/2$, precies wat we willen.
Dat deze lus–invariant voldoet, blijkt uit de volgende twee toepassingen van H1:

$\{y + z \cdot (z+1)/2 = x \cdot (x+1)/2\}\ y := y + z\ \{y + (z-1) \cdot z/2 = x \cdot (x+1)/2\}$

$\{y + (z-1) \cdot z/2 = x \cdot (x+1)/2\}\ z := z - 1\ \{y + z \cdot (z+1)/2 = x \cdot (x+1)/2\}$

H2 geeft nu:

$\{\varphi\}\ (y := y + z\ ;\ z := z - 1)\ \{\varphi\}$

En H5 levert:

$\{\varphi \wedge \neg(z = 0)\}\ (y := y + z\ ;\ z := z - 1)\ \{\varphi\}$

Uit H4 volgt nu de WHILE–lus:

$\{\varphi\}$ (WHILE $\neg(z = 0)$ DO $y := y + z\ ;\ z := z - 1$) $\{\varphi \wedge z = 0\}$

Nu is nog twee keer H1 nodig:

$\{\text{TRUE}\}\ y := 0\ \{y = 0\}$
$\{y = 0\}\ z := x\ \{y + z \cdot (z+1)/2 = x \cdot (x+1)/2\}$

Drie keer toepassen van H2 levert nu de uiteindelijke correctheidsbewering. □

We vermelden nog enkele meer theoretische resultaten.

Correct

De Hoare–calculus is semantisch correct: alle voor een model bewijsbare correctheidsbeweringen zijn in dat model ook waar. Overigens is de overeenkomst in terminologie tussen semantisch *correct* ('sound') en *correctheids*bewering ('correct') een toevalligheid van het Nederlands.

Volledig

Omgekeerd geldt een soort volledigheidsresultaat: op voldoende 'rijke' gegevensstructuren is elke ware correctheidsbewering Hoare-bewijsbaar.

15.5 DYNAMISCHE LOGICA

Naast correctheid zijn er andere belangrijke eigenschappen van programmaverwerking, die tot nu toe buiten beeld bleven. Voorbeelden hiervan zijn 'terminatie', 'determinisme' en 'equivalentie' van programma's. Willen we ook deze verdisconteren, dan blijkt een rijker logisch systeem nodig dan de Hoare–calculus. Een goede kandidaat voor zo'n generalisering is een toepassing van de modale logica die *dynamische logica* heet. Anders dan in de Hoare–axiomatiek, kunnen we

in dynamische logica terminatie, determinisme en equivalentie van programma's beschrijven.

De dynamische logica is ontstaan vanuit de informatica. In deze logica worden mogelijke werelden geïnterpreteerd als geheugentoestanden van een computer. In zo'n toestand hebben alle variabelen een waarde, zodat deze is te vergelijken met een bedeling. Maar we gaan nu over van het *predikaatlogisch* perspectief dat we hanteerden bij de STIP-programma's naar een *modaal–logisch* perspectief.

Voorbeeld 15.5

Laat b een toestand zijn waarin $x = 0$, $y = 1$ en $z = 4$. De propositieletters p, q en r staan voor respectievelijk '$x = 0$', '$y < 0$' en '$z > 5$'. Dan geldt onder andere:

- $b \models p$
- $b \not\models p \rightarrow q$
- $b \models p \vee r$

De overgangsrelatie $T(\pi)$ tussen geheugentoestanden die door een programma π wordt bepaald, kunnen we zien als een modale *toegankelijkheidsrelatie*. Deze wordt als volgt gedefinieerd:

$b\ T(\pi)\ e \Leftrightarrow$ vanuit begintoestand b is er een succesvolle uitvoering van programma π die eindigt in eindtoestand e

De operator $\Box$ behoeft nu enige aanpassing. Immers, wat toegankelijk is hangt nu af van het programma. Daarom moet bij $\Box$ worden aangegeven over welk programma het gaat. In plaats van $\Box$ noteren we dan $[\pi]$. Dit levert:

$b \models [\pi]\varphi \Leftrightarrow$ voor elke toestand e met $b\ T(\pi)\ e$ geldt $e \models \varphi$

Ook $\Diamond$ wordt programma–afhankelijk (notatie: $\langle\pi\rangle$):

$b \models \langle\pi\rangle\varphi \Leftrightarrow$ er is een toestand e zodat $b\ T(\pi)\ e$ en $e \models \varphi$

In plaats van $b\ T(\pi)\ e$ hadden we dus ook $R_\pi be$ kunnen schrijven. Het mogelijke–wereldenmodel waarbinnen we interpreteren is hier impliciet. Dit bestaat echter eenvoudigweg uit alle mogelijke geheugentoestanden van een computer. Met behulp van de dynamische operatoren kunnen diverse beweringen over programmagedrag geformuleerd worden.

Correctheidsbewering

De *correctheidsbeweringen* die we hiervoor reeds hebben geïntroduceerd, zijn semantisch wellicht de belangrijkste beweringsvorm in de informatica. In de dynamische logica heeft een correctheidsbewering de volgende vorm:

$$\varphi \rightarrow [\pi]\psi$$

Hierin wordt uitgedrukt dat π bij een bepaalde invoerconditie φ altijd een uitvoer geeft die voldoet aan de conditie ψ.

Determinisme

Beschouw de volgende formule:

$$\langle\pi\rangle\varphi \rightarrow [\pi]\varphi$$

Deze formule interpreteren we als volgt: als er *een* toestand e is zodat $b\ T(\pi)\ e$ en $e \models \varphi$, dan geldt voor *elke* toestand e' met $b\ T(\pi)\ e'$ dat $e' \models \varphi$. Omdat φ een willekeurige formule is, drukt $\langle\pi\rangle\varphi \rightarrow [\pi]\varphi$ uit dat π *deterministisch* is.

Terminatie

Dit laat op zich nog open dat in sommige toestanden in het geheel geen succesvolle berekening van π mogelijk is. Wanneer dit wel zo is, noemen we dit *terminatie* en dat is ook met een dynamische formule te beschrijven:

$$\langle\pi\rangle\text{TRUE}$$

Hierin staat TRUE voor de 'ware' bewering, bijvoorbeeld 1 = 1. Als $\langle\pi\rangle$TRUE waar is in zekere toestand b, dan betekent dit dat er ten minste een toestand e is waarin we terechtkomen als π wordt uitgevoerd. Maar dan is π kennelijk gestopt.

Equivalentie

We kunnen ook in een modale formule uitdrukken dat twee programma's π_1 en π_2 *equivalent* zijn:

$$[\pi_1]\varphi \leftrightarrow [\pi_2]\varphi$$

Nog complexere uitspraken zijn ook voorstelbaar, met gestapelde modaliteiten, zoals bijvoorbeeld:

$$p \rightarrow [\text{WHILE } p \text{ DO } \pi_1]\langle\pi_2\rangle p$$

Betekenis: 'als in een toestand p geldt, dan is na elke verdere toestand waarin $\neg p$ is ontstaan door herhalen van π_1, via uitvoeren van π_2 weer een p–toestand te bereiken'. Overigens is dit natuurlijk ook een – iets concretere – correctheidsbewering.

Om al dit soort beweringen te kunnen bewijzen is weer een calculus nodig waarvan de regels op een natuurlijke manier aansluiten bij de aanwezige programmastructuur. Het voordeel van de dynamische logica is nu dat we ons daarbij kunnen richten op bekende bewijssystemen uit de modale logica.

Reguliere programma's

Ter wille van de elegantie is het dan echter wel nodig over te schakelen op een iets ruimere klasse programmaconstructies dan tot nu toe in STIP is gebruikt. Daartoe definiëren we inductief de zogenaamde *reguliere programma's*:

DEFINITIE 15.5

Reguliere programma's

Basisprogramma's

a $a, b, c, \ldots$ 'atomaire acties'

b $\varphi?$ 'test of een bewering φ waar is'

Programmaopbouw

Als π_1, π_2, π reguliere programma's zijn, dan ook:

c $(\pi_1 ; \pi_2)$ 'opvolging'

d $(\pi_1 \cup \pi_2)$ 'keuze'

e π^* 'iteratie'

Het programma $\varphi?$ test of φ waar is. Zo ja, dan wordt doorgegaan met de volgende opdracht: zo nee, dan wordt afgebroken.
Het programma $(\pi_1 \cup \pi_2)$ maakt een willekeurige keuze tussen π_1 en π_2 en voert het gekozen programma uit.
Het programma π^* kiest een of ander eindig natuurlijk getal n en voert vervolgens n keer π uit.

Formules van de dynamische logica

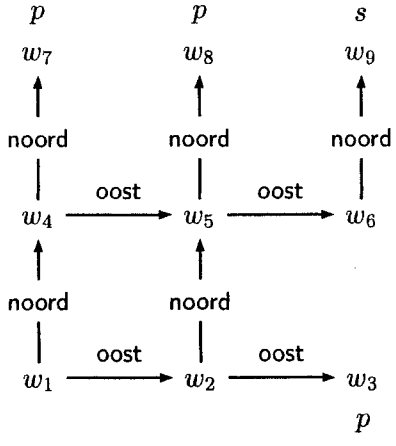
De klasse der *formules* van onze dynamische logica ontstaat nu net als in hoofdstuk 13. Er wordt begonnen met atomaire proposities $p, q, r, \ldots$ en vervolgens opgebouwd via de propositionele connectieven $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ alsmede de programmamodaliteiten $[\pi]$ en $\langle\pi\rangle$ voor elk regulier programma π.

Vervlechting van formules en programma's

Merk op dat dit tot interactie leidt tussen de (logische) 'beweringen' en de (programma) 'instructies' in het systeem: modale proposities worden gevormd met behulp van programma's, maar ook omgekeerd vormen proposities programma's via de test op het basisprogramma. Zo'n interactie komt in meer geavanceerde programmeerformalismen wel vaker voor.

Voorbeeld 15.6

We illustreren de uitdrukkingskracht van de dynamische logica aan het 'schateiland'–voorbeeld uit hoofdstuk 13 over modale logica. Om te beginnen maken we de situatie nog iets gevaarlijker dan deze al was. In voorbeeld 13.4 kon je de piraten nog bevechten en daarna de schat bereiken. Nu daarentegen, betekent piraten tegenkomen de onverbiddelijke dood. Vanuit de werelden met piraten zijn daarom geen andere meer bereikbaar. De situatie is dus als volgt:



De atomen zijn p, voor 'aanwezigheid van piraten', en s, voor 'locatie van de schat', net als in het eerdere voorbeeld. De atomaire acties zijn n, voor 'naar het noorden gaan', en o, voor 'naar het oosten gaan'. Deze worden uiteraard geïnterpreteerd als de voor de hand liggende overgangen $T(n) = \mathit{noord}$ en $T(o) = \mathit{oost}$ in de figuur. Dit is dus anders dan hiervoor, waar er slechts van een enkele toegankelijkheidsrelatie sprake was.

Op dit model geldt bijvoorbeeld dat $[o\,;o\,;o]\neg\mathrm{TRUE}$. Drie keer naar het oosten lopen kan nooit. Of met andere woorden: dit programma termineert niet. Tevens geldt dat $[n\,;o]\varphi \leftrightarrow [o;n]\varphi$. Of je nu eerst noord en dan oost gaat, of andersom, maakt niet uit. Met andere woorden: deze programma's zijn hier *equivalent*. Of je onderweg piraten tegenkomt is hierbij niet van belang: de bewering zegt dat *als* je eerst naar het noorden en daarna naar het oosten had kunnen gaan, *dan* had het ook andersom gekund. De atomaire acties zijn deterministisch: $\langle o\rangle\varphi \rightarrow [o]\varphi$ en tevens $\langle n\rangle\varphi \rightarrow [n]\varphi$. Als je piraten tegenkomt ben je dood (en kun je dus niet verder): $[p?\,;(n \cup o)]\neg\mathrm{TRUE}$. Vanuit het startpunt kan de schat bereikt worden: $w_1 \models \langle(n \cup o)^*\rangle s$. Dit lukt namelijk via het pad w_1, w_4, w_5

w_6, w_9, waarmee de sequentie $n\,;\,o\,;\,o\,;\,n$ correspondeert. Dit is een concretisering van $(n \cup o)^4$ waarbij de eerste keuze n is, de tweede o, enzovoorts. Hiermee is het dus een mogelijke uitvoering van $(n \cup o)^*$ (namelijk: 'kies vier keer'). Vanuit een wereld waar piraten zijn, is het niet mogelijk de schat te bereiken: $p \rightarrow \neg\langle(n \cup o)^*\rangle s$.

Deze voorbeelden dienen ook ter illustratie van de algemene logisch-dynamische semantiek, waarvan de op STIP toegespitste inperking nu volgt.

Inbedding van STIP in reguliere programma's

Om STIP–programma's te laten aansluiten bij de dynamische logica eisen we bovendien:

a atomaire acties zijn toekenningen $x := t$;

b voorwaarden in testprogramma's zijn kwantorvrije Boolese condities.

Hiermee gaan we over van bijvoorbeeld een propositieletter p die staat voor '$x = 0$' naar de voorwaarde $x = 0$. Dit komt tot uitdrukking in de semantiek.

DEFINITIE 15.6

Semantiek van reguliere programma's

De beoogde semantiek van reguliere programma's luidt dan als volgt, in modellen $M = (\boldsymbol{D}, I)$:

$$b_1\ T(x := t)\ b_2 \quad \Leftrightarrow \quad b_2 = b_1[x \mapsto V_{(\boldsymbol{D}I),b_1}(t)]$$
$$b\ T(\varepsilon?)\ b \quad \Leftrightarrow \quad M, b \vDash \varepsilon$$

De overgangsrelatie T voor de drie overige programmaoperaties kan inductief gegeven worden met behulp van standaardoperaties op binaire relaties:

$$T(\pi_1\,;\,\pi_2) = T(\pi_1) \circ T(\pi_2) \qquad \text{'compositie'}$$
$$T(\pi_1 \cup \pi_2) = T(\pi_1) \cup T(\pi_2) \qquad \text{'vereniging'}$$
$$T(\pi^*) = T(\pi)^* \qquad \text{'transitieve en reflexieve afsluiting'}$$

Hierbij staat $T(\pi)^*$ voor $\bigcup_{n\geq 0} T(\pi)^n$, waarbij $T(\pi)^n = T(\pi^n)$ en π^n staat voor $(\pi\,;\,\ldots\,;\,\pi)$ (n maal herhaald). Het programma π^0 komt overeen met TRUE?, het testprogramma dat altijd slaagt.

Indeterminisme

Merk op dat T–relaties voor reguliere programma's niet functioneel hoeven zijn. Vanuit een gegeven toestand kunnen ze meerdere mogelijke voortzettingen hebben. Reguliere programma's modelleren

daarmee bepaalde aspecten van 'indeterminisme'.

Dat hier echt van een generalisering sprake is ten opzichte van STIP, zegt de volgende bewering:

BEWERING 15.1

Elk STIP–programma kan uitgedrukt worden als een regulier programma met dezelfde overgangsrelatie.

Bewijs

We bewijzen dit met inductie naar de opbouw van STIP–programma's.
Atomaire toekenningen
Voor de atomaire toekenningen is de bewering per definitie het geval.
Opvolging
Opvolging komt bij STIP– en bij reguliere programma's voor.
Voorwaardelijke keuze
IF ε THEN π_1 ELSE $\pi_2 = (\varepsilon? ; \pi_1) \cup (\neg\varepsilon? ; \pi_2)$
Terwijl–lus
WHILE ε DO $\pi = ((\varepsilon? ; \pi)^* ; \neg\varepsilon?)$ □

Wat hier wordt bewerkstelligd is overigens loutere gelijkheid van overgangsrelaties. Er blijven namelijk intuïtieve verschillen tussen onze manieren van begrijpen hoe de programma's aan weerszijden van de gelijktekens functioneren. Zo lijkt kiezen voor het verkeerde alternatief rechts in het geval van de voorwaardelijke keuze, te leiden tot afbreken van de verwerking: anders dan aan de linkerkant, waar van keuze geen sprake is. Maar dergelijke verschillen vallen weg zolang we slechts op uiteindelijke succesvolle toestandsovergangen letten.

DEFINITIE 15.7

Axiomatiek voor de dynamische logica
We presenteren nu een axiomatiek voor de propositionele dynamische logica:

Propositielogische axioma's

a Elke instantie van een propositionele tautologie is een axioma.
b Modus Ponens: uit φ en $\varphi \rightarrow \psi$ volgt ψ.

Axioma's voor minimale modale logica

c Distributie
Π1 $[\pi](\varphi \rightarrow \psi) \rightarrow ([\pi]\varphi \rightarrow [\pi]\psi)$
d Dwang: indien φ bewijsbaar is, dan ook $[\pi]\varphi$.

Logica voor programma's

e Uitleg van toekenningen: $[x := t]\varphi \leftrightarrow [t/x]\varphi$, mits t vrij is voor x in φ.
f Decompositieaxioma's voor test, sequentie en keuze:
Π2 $[\varepsilon?]\varphi \leftrightarrow (\varepsilon \rightarrow \varphi)$
Π3 $[\pi_1 ; \pi_2]\varphi \leftrightarrow [\pi_1][\pi_2]\varphi$
Π4 $[\pi_1 \cup \pi_2]\varphi \leftrightarrow [\pi_1]\varphi \wedge [\pi_2]\varphi$
g Inductieaxioma's voor iteratie:

Π5 $[\pi^*]\varphi \leftrightarrow (\varphi \wedge [\pi][\pi^*]\varphi)$
Π6 $(\varphi \wedge [\pi^*](\varphi \rightarrow [\pi]\varphi)) \rightarrow [\pi^*]\varphi$

We geven nog een illustratie van de werking van deze calculus:

BEWERING 15.2

Alle principes van de Hoare–calculus zijn in propositionele dynamische logica afleidbaar.

Als voorbeeld leiden we de regel van voorwaardelijke keuze H3 af. We gebruiken de vertaling van STIP– naar reguliere programma's:

a $(\varphi \wedge \varepsilon) \rightarrow [\pi_1]\psi$ (aanname in H3)
b $\varphi \rightarrow (\varepsilon \rightarrow [\pi_1]\psi)$ a + propositielogica
c $\varphi \rightarrow [\varepsilon?][\pi_1]\psi$ b + Π2
d $\varphi \rightarrow [\varepsilon? ; \pi_1]\psi$ c + Π3

Op analoge wijze vinden we uit de andere aanname van H3 een propositioneel–dynamische afleiding van:
e $\varphi \rightarrow [\neg\varepsilon? ; \pi_2]\psi$
We combineren dan d en e propositioneel tot:
f $\varphi \rightarrow ([\varepsilon? ; \pi_1]\psi \wedge [\neg\varepsilon? ; \pi_2]\psi)$
Dit leidt met axioma Π4 tot de gewenste conclusie van H3:
g $\varphi \rightarrow [(\varepsilon? ; \pi_1) \cup (\neg\varepsilon? ; \pi_2)]\psi$
Immers, g geeft precies de dynamische versie van de correctheids-bewering voor de IF...THEN...ELSE–opdracht.

Reguliere programma's en dynamische logica worden in toenemende mate in de informatica gebruikt, ook voor andere dan numerieke toepassingen.

15.6 OPGAVEN

15.1 Druk met een correctheidsbewering op de natuurlijke getallen uit wat het volgende STIP–programma π doet:

$(((u := 0$; WHILE $u \cdot u < x$ DO $u := Su)$; $(z := 0$; WHILE $z + z < y$ DO $z := Sz))$; IF $u > z$ THEN $s := u$ ELSE $s := z)$

15.2 a Beschouw een nieuwe STIP–programmaconstructie $\pi_1 \cup \pi_2$.
Geef een semantische definitie van $\pi_1 \cup \pi_2$.
b Geef een Hoare–regel voor correctheidsbeweringen over $\pi_1 \cup \pi_2$.

* c Beschouw een nieuwe STIP–programmaconstructie $\pi_1 \cap \pi_2$. Hoe zou u deze constructie interpreteren? Probeer ook voor deze constructie een Hoare–regel te vinden.

15.3 Geef een afleiding van H2 in dynamische logica (zie bewering 15.2).

* 15.4 De actie van het STIP–programma $x := y + 1$ kan in de predikaatlogica worden weergegeven door een formule waarin naast x en y variabelen x' en y' voorkomen die de waarde van x respectievelijk y geven na afloop van het programma: $y' = y \wedge x' = y + 1$.
Noem een dergelijke bij een programma horende formule een *transitie-predikaat*.
Laat zien hoe voor elk STIP–programma π waarin de variabelen $x_1, ..., x_n$ voorkomen, een transitiepredikaat kan worden gemaakt met variabelen $x_1, ..., x_n, x_1', ..., x_1'$.

* 15.5 Bewijs dat het faculteitsprogramma π uit voorbeeld 15.3 correct is.
Hint: begin met $\{x! = z!\}\ y := S0\ \{y \cdot x! = z!\}$ en gebruik de postconditie hiervan als preconditie bij de WHILE–lus.

* 15.6 Toon aan dat de WHILE–regel correct is (in de zin van correctheid zoals bij propositie– en predikaatlogica).

kan uiterst ondoorzichtig zijn door de vele GOTO–spronginstructies. In de lijn van de ontwikkeling binnen de informatica naar meer overzichtelijke vormen van 'gestructureerd programmeren' introduceren we nu een eenvoudige hogere *programmeertaal*. In de nu volgende taal STIP ('geSTructureerd Imperatief Programmeren') zijn programma's inductief opgebouwd, net als formules in onze logische formalismen. Dit dwingt ons om gestructureerde programma's te schrijven. Overigens is STIP in geen enkel opzicht bedoeld als realistische programmeertaal. Het is een kapstok voor voorbeelden van gestructureerd programmeren.

Definitie van STIP

We nemen een predikaatlogische taal voor de rekenkunde, met individuele constante 0 en eenplaatsige functieletters S en P, 'onmiddellijke opvolger' respectievelijk 'onmiddellijke voorganger' (met stop in 0), waarmee we termen t als gebruikelijk kunnen vormen. Als predikaat nemen we alleen de identiteit.

DEFINITIE 14.3

Syntaxis van STIP voor de rekenkunde

H1 Als x een variabele is en t een term, dan is de toekenning $x := t$ een STIP–programma.

Als π_1 en π_2 STIP–programma's zijn, dan ook:

H2 $(\pi_1 ; \pi_2)$ ('opvolging')

H3 IF ε THEN π_1 ELSE π_2 ('voorwaardelijke keuze')

H4 WHILE ε DO π_1 ('terwijl–lus')

Boolese uitdrukking

De conditie ε staat voor een kwantorvrije predikaatlogische formule (in de gekozen taal) die in een toestand kan worden geëvalueerd tot een waarheidswaarde. Zo'n conditie heet een 'Boolese uitdrukking'.

Non–terminatie

Programma's termineren niet noodzakelijk. De WHILE–constructie kan oneindig doorlopen wanneer aan de conditie ε telkens wordt voldaan.

Interpretatie

Deze programma's moeten als volgt geïnterpreteerd worden. De variabelen worden opgevat als namen voor registers. De basisprogramma's zijn toekenningen van waarden van termen t aan variabelen. Een bedeling voor alle variabelen is dus een momentane geheugentoestand, dat wil zeggen een volledige beschrijving van alle registerinhouden. In het volgende hoofdstuk zullen we deze interpretatie uitvoeriger formuleren.

Voorbeeld 14.3

Vermenigvuldigen met STIP

Het volgende STIP–programma berekent voor getallen n en m hun product $n \cdot m$ in het y–register:

$(x := n ; (y := 0 ;$ WHILE $\neg(x = 0)$ DO $((z := m ;$ WHILE $\neg(z = 0)$ DO $(y := Sy ;$
$z := Pz)) ; x := Px)))$