Poglavje 2

SKLEPANJE

Teorija je definirana z množico pravil. Pravila lahko definirajo preverjanje tipov stavka danega jezika, definirajo interpretacijo jezika, ...

Sklepanje predstavimo s sekvenco aplikacij pravil na stavku, ki je podan kot vhodni parameter. Sklepanje je torej predstavljeno s sekvencami instanc pravil.

Vse instance pravil predstavljajo interpretacijo jezika, vsa možna obnašanja programov.

Sklepanje je torej tudi interpretacija stavka predstavljena kot sekvenca instanc pravil.

Sekvenca instanc pravil lahko predstavlja: preverjanje strukture stavkov, preverjanje tipov stavkov, interpretacija stavkov, itd.

2.1 Objekti in sodbe

Sodba je *izjava* o *objektih*, ki jih opazujemo. Izraz *sodba* velikokrat zamenjamo z izrazom *izjava*.

Uporabljali bomo najrazličnejše oblike sodb:

n nat	n je naravno število
$n = n_1 + n_2$	n je vsota n_1 in n_2
a ast	a je abstraktno sintaksno drevo
$\mathcal{T}$ type	$\mathcal{T}$ je tip
$e : \mathcal{T}$	izraz e je tipa $\mathcal{T}$
$e \Downarrow v$	izraz e ima vrednost v

(2.1)

Sodba (izjava) pravi, da ima eden ali več objektov neko lastnost ali obstaja razmerje med objekti

Lastnost sodbe ali samo relacijo definirano s sodbo imenujemo *forma sodbe*. Sodba o konkretnem objektu je *instanca* forme sodbe. Formo sodbe imenujemo tudi *predikat* in objekte, ki sodelujejo *subjekti*.

Metaspremenljivko P bomo uporabili za nespecificirano obliko sodbe in meta-spremenljivke $a, b, c, \ldots$ za nespecificirane objekte.

Zdaj lahko napišemo $a\ P$, kar pomeni da ima a lastnost P.

Ko ni pomembna vsebina sodbe uporabimo meta-spremenljivko J, ki določa nespecificirano sodbo.

Namerno bomo predstavili univerzum objektov bolj splošno. Dovolimo katerikoli objekt, ki ga definiramo s končnim postopkom.

Predpostavimo, da unverzum vsebuje vse objekte, ki jih je možno kreirati kot n-terico $(a_1, a_2, \ldots, a_n)$ objektov a_i.

Predpostavimo tudi, da univerzum vsebuje neskončno *simbolov*, ki so zaprti za označevanje.

2.2 Pravila sklepanja

Pravila služijo kot osnoven način za opisovanje pomena programskih jezikov.

V zadnjih desetletjih so se "pravila" zelo različno obravnavala: od enostavnih prehodov med stanji tranzicijskega sistema, pravil operacijske semantike, ki prav tako predstavljajo prehode med različnimi stanji, do pravil LF, ki so zapisane v logiki višjega reda.

Pravila lahko obravnavamo kot *logično implikacijo*. Osnova semantike programskih jezikov je torej spet logika! Izrazna moč pravil je ključnega pomena pri opisovanju semantike jezikov.

Tukaj si bomo ogledali pravila, ki imajo zelo splošno obliko. Pravila obravnavamo kot induktivne definicije forme sodbe.

Definicija forme sodbe J je sestavljena iz množice pravil oblike:

$$\frac{a_1\ J \quad a_2\ J \ldots a_k\ J}{a\ J} \tag{2.2}$$

Izjavo nad črto imenujemo *premisa*. Izjavo pod črto imenujemo *posledica*. Če pravilo nima premis ga imenujemo *aksiom*.

Pravilo beremo kot *logično implikacijo*, kjer so premise zadosten pogoj za sklepanje na posledico pravila. Da bi pokazali $a\ J$ je zadosti pokazati $a_1\ J, \ldots$

Lahko imamo več pravil z isto posledico vsako s svojimi premisami. V primeru, da velja posledica ni nujno, da veljajo premise vseh pravil.

Pravimo, da je sodba J zaprta za dano pravilo, če in samo če izjave $a_1\ J, \ldots, a_k\ J$ implicira $a\ J$. Forma sodbe je definirana z indukcijo nad pravili.

Pravila so potreben kot tudi zadosten pogoj za sklepanje na posledico na osnovi premis.

Primer 2.2.1 *Induktivna definicija sodbe a nat.*

$$\frac{}{\textit{zero nat}} \qquad \frac{a\ \textit{nat}}{\textit{succ}(a)\ \textit{nat}} \tag{2.3}$$

□

Primer 2.2.2 *Iduktivna definicija forme sodbe za drevo.*

$$\frac{}{\textit{empty tree}} \qquad \frac{a\ \textit{tree} \quad b\ \textit{tree}}{\textit{node}(a,b)\ \textit{tree}} \tag{2.4}$$

□

Primer 2.2.3 *Induktivna definicija enakosti naravnih števil.*

$$\frac{}{zero = zero\ \textit{nat}} \qquad \frac{a = b\ \textit{nat}}{\textit{succ}(a) = \textit{succ}(b)\ \textit{nat}} \tag{2.5}$$

□

Primer 2.2.4 *Podobno je definirana induktivno enakost nad drevesi.*

$$\frac{}{\textit{empty} = \textit{empty tree}} \qquad \frac{a_1 = b_1\ \textit{tree} \quad a_2 = b_2\ \textit{tree}}{\textit{node}(a_1,a_2) = \textit{node}(b_1,b_2)\ \textit{tree}} \tag{2.6}$$

□

Spremenljivke a in b imenujemo *metaspremeljivke*. Domena metaspremenljivk so objekti, ki so predmet definicije. Pravila so torej sheme, ki stojijo za neskončno mnogo konkretnih objektov.

2.3 Izpeljave

Pravilnost induktivne definicije sodbe lahko pokažemo tako, da konstruiramo *drevo izpeljave*. Drevo izpeljave dobimo s kompozicijo pravil, ki se začnejo z aksiomi in se zaključijo v sodbi.

Struktura izpeljave je drevo, ki je prikazano kot kopica s pravili naloženimi eno na drugo. Naj bo

$$\frac{a_1\ J \ldots a_k\ J}{a\ J} \tag{2.7}$$

pravilo izpelave in naj bodo $\nabla_1 \ldots \nabla_k$ izpeljave premis, potem je

$$\frac{\nabla_1 \ldots \nabla_k}{a\ J} \tag{2.8}$$

izpeljava posledice pravila $a\,J$. Na primer, izpeljava $\mathrm{succ}(\mathrm{succ}(\mathrm{succ}(\mathrm{zero})))\,\mathrm{nat}$ izgleda takole:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{zero\ nat}}}{\mathrm{succ}(\mathrm{zero})\ \mathrm{nat}}}{\mathrm{succ}(\mathrm{succ}(\mathrm{zero}))\ \mathrm{nat}}}{\mathrm{succ}(\mathrm{succ}(\mathrm{succ}(\mathrm{zero})))\ \mathrm{nat}} \tag{2.9}$$

Podobno predstavlja naslednji izraz izpeljavo $\mathrm{node}(\mathrm{node}(\mathrm{empty}, \mathrm{empty}), \mathrm{empty})\ \mathrm{tree}$:

$$\dfrac{\dfrac{\dfrac{}{\mathrm{empty\ tree}} \quad \dfrac{}{\mathrm{empty\ tree}}}{\mathrm{node}(\mathrm{empty}, \mathrm{empty})\ \mathrm{tree}} \quad \dfrac{}{\mathrm{empty\ tree}}}{\mathrm{node}(\mathrm{node}(\mathrm{empty}, \mathrm{empty}), \mathrm{empty})\ \mathrm{tree}} \tag{2.10}$$

Da bi pokazali, da je neka izjava pravilna moramo poiskati izpeljavo.

Obstajata dve glavni metodi za izpeljave izrazov: *veriženje naprej* ali *konstrukcija od spodaj navzgor* in *veriženje nazaj* oz. *konstrukcija od zgoraj navzdol*.

Veriženje naprej začne z aksiomi in napreduje proti ciljnemu izrazu, medtem ko veriženje nazaj začne z izrazom in napreduje proti aksiomom.

Pri sklepanju naprej začnemo s prazno množico in iterativno širimo množico z uporabo pravil - posledico pravila dodamo množici v primeru, da se premise ujemajo. Proces se konča, ko je željena sodba v množici.

Če uporabljamo vsa pravila pri enem koraku postopka potem pričakujemo, da bomo našli izpeljavo, ne moremo pa zagotovo trditi, da bomo izpeljavo našli. Takšen postopek iskanja ni odločljiv (decidable).

Postopek izpeljave lahko dodaja posledice k množici, brez da bi prišel do izpeljave. Odločljivost posameznih izrazov lahko definiramo samo na osnovi poznavanja strukture jezika in splošnih lastnosti jezika.

Veriženje naprej je neusmerjeno iskanje, ker pri izvajanju koraka postopka ne upoštevamo cilj izpeljave.

Veriženje nazaj usmerjeno iskanje saj algoritem usmerjajo komponente ciljnega izraza.

Na vsaki fazi algoritma imamo seznam ciljev za katere iščemo izpeljave. Začetno vsebuje seznam samo ciljno izjavo. Vsaka faza odstani izjavo iz vrste in jo nadomesti s premisami pravil katerih posledica je dan cilj. Proces se konča, ko je vrsta prazna in smo dosegli vse cilje.

Enako kot pri veriženju naprej se nam lahko zgodi tudi pri veriženju nazaj, da ne uspemo najti izpeljave. V splošnem ne obstaja algoritmična metoda, ki bi nam povedala, da se neka izjava da izpeljati. Seznam ciljev se nam lahko širi ne da bi prišli do točke, kjer lahko rečemo, da je začetni cilj dosežen.

2.4 Indukcija pravil

Induktivno definirana sodba drži samo v primeru, da imamo kakšno izpeljavo s pravili.

Lastnosti sodb lahko dokažemo z uporabo indukacije pravil ali indukcije na izpeljavah.

Napišemo $P(J)$, če hočemo povedati, da lastnost P velja, če je sodba J izpeljiva.

Če hočemo pokazati, P velja za vse izpeljive sodbe J je zadosti, da pokažemo, da je P zaprta za pravila, ki definirajo J.

Za vsako pravilo oblike

$$\frac{J_1 \dots J_k}{J} \tag{2.11}$$

velja

$$\text{if } P(J_1) \dots P(J_k), \text{ then } P(J) \tag{2.12}$$

Konjunkcije lastnosti $P(J_1), \dots, P(J_k)$ imenujemo *induktivne hipoteze*. Dokaz implikacije same imenujemo *induktivni korak.*

Opazka 2.4.1 *Princip indukacije pravil je izraz dejstva, da je sodba J induktivno definirana z množico pravil in je najmočnejša sodba zaprta z danimi pravili.*

J je lahko izpeljiva samo zato, ker obstaja pravilo, kjer je J posledica in je vsaka premisa v pravilu tudi izpeljiva.

Po principu indukcije lahko predpostavljamo, da P velja za vse premise in potem pokažemo, da P velja tudi za J.

Če to naredimo za vsako pravilo, potem P mora veljati za vse sodbe J, ki so izpeljive.

V primeru, da pravilo nima premis potem velja P brez pogojev. □

Če je $P(J)$ zaprta za množico pravil, ki definirajo sodbo J, potem to velja tudi za $Q(J) = P(J) \wedge J$. J je zaprta tudi za pravila, ki jo definirajo.

To pomeni, da lahko v vsakem indukcijskem koraku predpostavimo, da velja J_i in $P(J_i)$ za vsako od premis pravila za izpeljavo $P(J)$ kot posledico.

Poglejmo si še notacijo. Če ima J obliko C za nek predikat C, včasih napišemo $PC(a)$, ali samo $P(a)$, namesto $P(a\ C)$ pri uporabljanju indukcije v dokazu. Za specifične lastnosti P pogosto uporabljamo ad-hoc notacijo katere pomen je razviden iz konteksta.

Zdaj lahko začnemo z *uporabo indukije* za dokazovanje lastnosti sodb. Poglejmo si nekaj primerov.

Primer 2.4.1 *Pri specializaciji na pravila 2.3 princip indukcije pravi, da je za veljavnost $P(a\ \textit{nat})$ vedno ko a nat zadosti pokazati:*

1. $P(\textit{zero nat})$.
2. $P(\textit{succ}(a)\ \textit{nat})$, *pri predpostavki* $P(a\ \textit{nat})$.

To je soroden princip matematični indukciji in poseben primer indukcije pravil.

Primer 2.4.2 *Podobno velja tudi pri indukciji po pravilih 2.4 velja, da če želimo pokazati $P(a\ \textit{tree})$ vedno ko a tree, je zadosti pokazati:*

1. $P(\textit{empty tree})$.
2. $P(\textit{node}(a1; a2)\ \textit{tree})$, *če predpostavimo* $P(a_1\ \textit{tree})$ *in* $P(a_2\ \textit{tree})$.

To imenujemo princip indukcije nad drevesi, ki je tudi primer indukcije pravil.

Poglejmo si primer dokaza refleksivnosti enakosti naravnih števil z indukcijo nad pravili 2.3.

Lema 2.4.1 *Če a nat, potem $a = a$ nat.*

Dokaz. Uporabili bomo indukcijo na osnovi pravil 2.5.

Najprej vidimo na osnovi pravila 2.5a, da velja zero $=$ zero nat.

Predpostavimo zdaj, da velja $a = a$ nat. Pravilo 2.5b pravi, da potem velja tudi succ$(a) =$ succ(a) nat. □

Poglejmo si še en primer uporabe indukcije pravil. Pokazali bomo, da je predhodnik naravnega števila tudi naravno število. Dokaz je enostaven, želimo pa pokazati kako je lastnost izpeljana iz začetnih principov.

Lema 2.4.2 *Če succ(a) nat, potem a nat.*

Dokaz. Poglejmo najprej osnovo. V primeru, da je succ(zero) naravno število, potem je zero naravno število po aksiomu 2.3a.

Predpostavimo zdaj, da imamo neko poljubno naravno število succ(a). To naravno število je lahko nastalo samo z uporabo pravila 2.3b, kar pomeni, da velja a nat.

Pokažimo zdaj, da je operacija naslednik injektivna.

Lema 2.4.3 *Če succ$(a_1) =$ succ(a_2) nat, potem $a_1 = a_2$ nat.*

Dokaz. Trditev je koristno prepisati tako, da bomo lažje uporabljali indukcijo pravil.

Pokazali bomo da če velja $b_1 = b_2\ nat$ in če velja da b_1 je succ(a_1) in b_2 je succ(a_2), potem velja $a_1 = a_2$ nat. Uporabili bomo indukcijo po pravilih 2.5:

Poglejmo najprej primer, ko je $b_1 =$ succ(zero) in $b_2 =$ succ(zero). Pravilo 2.5a pravi, da velja zero $=$ zero nat.

Če predpostavimo premiso $b_1 = b_2$ nat, potem pokažemo da: če succ(b'_1) je succ(a_1) in succ(b'_2) je succ(a_2), potem predpostavimo, da $b'_1 = b'_2$ nat in po drugem pravilu 2.5 velja tudi succ$(b'_1) =$ succ(b'_2) nat. Torej velja $b'_1 = a_1$ in $b'_2 = a_2$ in tudi $a_1 = a_2$ nat, ker $b'_1 = b'_2$. □

2.4.1 Iterativne in simultane indukcijske definicije

Induktivne definicije so pogosto podane *iterativno*: ena induktivna definicija je definirana skupaj z drugo.

Primer 2.4.3 *Naslednji definiciji opišeta sodbo a list, ki pravi da je a seznam naravnih števil.*

$$\frac{}{nil\ \textit{list}} \tag{2.13}$$

$$\frac{a\ \textit{nat} \quad b\ \textit{list}}{\textit{cons}(a, b)\ \textit{list}} \tag{2.14}$$

Drugo pravilo vsebuje kot premiso sodbo a nat, ki je definirana hkrati. □

Simultana indukcijska definicija množice sodb $J_1, J_2, \ldots, J_n$ je opisana z množico med sabo povezanih pravil. Pravila definirajo vse sodbe hkrati. Vsako pravilo lahko referencira katerokoli sodbo, ki se definira.

Primer 2.4.4 *Poglejmo si naslednja pravila, ki hkrati (simultano) induktivno definirajo sodbe a even, ki pravi, da je a sodo naravno število, in sodba a odd, ki pravi, da je a liho naravno število:*

$$\frac{}{\textit{zero even}} \tag{2.15}$$

$$\frac{a\ \textit{odd}}{\textit{succ}(a)\ \textit{even}} \tag{2.16}$$

$$\frac{a \textit{ even}}{\textit{succ}(a) \textit{ odd}} \tag{2.17}$$

Primer 2.4.5 *Poglejmo si še en primer. Princip indukcije za ta pravila pravi, da bi pokazali $P(a$ even$)$ vedno ko velja a even in $P(a$ odd$)$ vedno ko velja a odd, je zadosti, da bi pokazali naslednje:*

1. *P (zero even);*
2. *če $P(a$ odd$)$, potem $P($succ(a) even$)$;*
3. *če $P(a$ even$)$, potem $P($succ(a) odd$)$.*

Ko prepišemo izraza z $P_{\textit{even}}(a)$ in $P_{\textit{odd}}(a)$, so pogoji naslednji:

1. *$P_{\textit{even}}($zero$)$;*
2. *če $P_{\textit{odd}}(a)$, potem $P_{\textit{even}}($succ$(a))$.*
3. *če $P_{\textit{even}}(a)$, potem $P_{\textit{odd}}($succ$(a))$;*

2.4.2 Definicija funkcij s pravili

Pogosto uporabimo induktivne definicije za opis grafa, ki predstavlja funkcijo. Graf predstavlja razmerje med vhodi in izhodi s pomočjo premis in posledic pravil.

En način za definicijo seštevanja naravnih števil je induktivna definicija sodbe sum(a, b, c), katere pomen je $c = a + b$.

$$\frac{b \text{ nat}}{\text{sum}(zero, b, b)} \tag{2.18}$$

$$\frac{\text{sum}(a, b, c)}{\text{sum}(\text{succ}(a), b, \text{succ}(c))} \tag{2.19}$$

Pokazati želimo, da je c enolično določen z a in b.

Izrek 2.4.1 *Za vsak a nat in b nat obstaja natančno en c nat, kjer velja $c = a + b$.*

Dokaz. Veljati mora, da za vsak a nat in b nat obstaja natančno ena vrednost c nat, tako da sum(a, b, c). Dokaz se sestoji iz dveh delov:

1. (obstoj) Če a nat in b nat potem obstaja c nat tako, da velja sum(a, b, c).
2. (enoličnost) Če a nat in b nat potem obstaja natančno en c nat tako, da velja sum(a, b, c).

Pokažimo najprej obstoj. Naj bo $P(a \text{ nat})$ izjava: če b nat potem obstaja c nat tako, da sum(a, b, c). Dokažemo, da če velja a nat potem velja $P(a \text{ nat})$ z indukcijo po pravilih 2.18. Imamo dva primera.

Pokažimo najprej $P(\text{zero nat})$. Če predpostavimo b nat in če vzamemo da je c enak b potem dobimo sum$(zero, b, c)$ po pravilu 2.18a.

Predpostavimo zdaj, da $P(a \text{ nat})$, in pokažimo, da velja $P(\text{succ}(a) \text{ nat})$. Ker $P(a \text{ nat})$ velja in velja tudi b nat potem po predpostavki obstaja c tako da sum(a, b, c). Če zdaj apliciramo pravilo 2.18b potem velja sum$(\text{succ}(a), b, \text{succ}(c))$, torej velja tudi $P(\text{succ}(a) \text{ nat})$.

Pokažimo se unikatnost vrednosti c. Dokažemo, da če velja sum(a, b, c_1) in sum(a, b, c_2), potem velja $c_1 = c_2$ nat po indukciji na osnovi pravil 2.18.

Poglejmo najprej idukcijsko osnovo. Naj bo $a =$ zero in $c_1 = b$. Pravilo 2.18a pravi, da potem velja sum(zero, b, b) torej $c_1 = c_2 = b$.

In še indukcijski korak. Predpostavimo, da unikatnost vrednosti velja za vsoto a in b: sum(a, b, c), kjer je c enoličen. Z uporabo pravila 2.18a velja isto tudi za naslednika a: vsota succ(a) in b je succ(c) oz. sum$(\text{succ}(a), b, \text{succ}(c))$, kjer je succ$(c)$ enoličen. □

2.5 Hipotetične sodbe

Kategorična sodba je brezpogojna izjava o nekem objektu univerzuma.

Hipotetična sodba je sestavljena na osnovi ene ali več hipotez ali predpostavk, ki izpeljejo zaključek.

Ogledali si bomo dve vrsti hipotetičnega sklepanja: izpeljevanje in dopustno sklepanje.

2.5.1 Izpeljivost

Ena forma hipotetičnega sklepanja izraža izrazljivost posledic iz danih predpostavk ali hipotez. Takšno sklepanje ima naslednjo obliko.

$$A_1, \ldots, A_n \vdash A. \tag{2.20}$$

Izjave $A_1, \ldots, A_n$ so *hipoteze* in A predstavlja *posledico*.

Pomen zgornje izjave: lahko izpeljemo A s kompozicijo pravil začevši z danimi predpostavkami kot začasnimi aksiomi. Z drugimi besedami, dokaz za pravilnost hipotetične sodbe je izpeljava zaključka s pomočjo instanc pravil iz danih predpostavk.

Primer 2.5.1 *Hipotetična sodba*

$$a\ \mathit{nat} \vdash \mathit{succ}(\mathit{succ}(a))\ \mathit{nat} \tag{2.21}$$

Premiso a nat uporabimo kot aksiom iz katerega izpeljemo posledico succ(succ(a)) nat na osnovi pravil 2.3. Tole je izpeljava posledice.

$$\dfrac{\dfrac{\dfrac{a\ \mathit{nat}}{\mathit{succ}(a)\ \mathit{nat}}}{\mathit{succ}(\mathit{succ}(a))\ \mathit{nat}}}{\mathit{succ}(\mathit{succ}(\mathit{succ}(a)))\ \mathit{nat}} \tag{2.22}$$

Zanimivo je, da je izpeljava popolnoma neodvisna od začetnega objekta a. Lahko bi vzeli namesto a nek poljuben objekt, npr. smet, in izpeljali naslednjo izjavo.

$$smet\ \mathit{nat} \vdash \mathit{succ}(\mathit{succ}(smet))\ \mathit{nat} \tag{2.23}$$

Sodbo $smet$ nat vzamemo kot aksiom iz katerega izpeljemo stavek z uporabo pravil za definicijo naravnih števil. □

Poglejmo si nekaj *strukturnih lastnosti* izpeljav.

Refleksivnost. Vsaka sodba je posledica same sebe: $A \vdash A$. Posledica je upravičena sama s seboj kot aksiom.

Omejevanje. Če $\Gamma \vdash A$ potem velja tudi $\Gamma, A' \vdash A$. Izpeljava A uporablja pravila iz Γ in se ne spremeni ob dodajanju novih sodb.

Tranzitivnost. Če $\Gamma, A' \vdash A$ in $\Gamma' \vdash A'$ potem $\Gamma, \Gamma' \vdash A$. Če dodamo dokaze za neko hipotezo premisi potem ta sodba (hipoteza) ni več potrebna med premisami.

Strukturne lastnosti se dajo izraziti kot pravila izpeljave, ki jih imenujemo *strukturna pravila izpeljave*.

$$\frac{}{\Gamma, A \vdash A} \tag{2.24}$$

$$\frac{\Gamma \vdash A}{\Gamma, A' \vdash A} \tag{2.25}$$

$$\frac{\Gamma \vdash A' \quad \Gamma, A' \vdash A}{\Gamma \vdash A} \tag{2.26}$$

Izpeljivost je relativno močen pogoj, ki je stabilen za razširjanje množice pravil, ki definirajo sodbo. Če je pravilo izpeljivo iz danih pravil potem je izpeljivo tudi iz razširjene množice. Dodatna pravila torej ne spremenijo obstoječih izpeljav.

Te lastnosti nima forma sklepanja, ki jo bomo ogledali v naslednji temi.

Teorija domen predstavlja več o urejenosti in strukturnih lastnostih prostora sodb. Predstavljena bo v enem od naslednjih poglavij.

2.5.2 Dopustno sklepanje

Druga oblika hipotetičnega sklepanja je *dopustno sklepanje*.

$$J \models K \tag{2.27}$$

Sodba K je izpeljiva na osnovi danih pravil vedno, ko je sodba J izpeljiva z isto množice pravil.

Dopustno sklepanje je enostavno pogojna izjava, ki pravi, da če je sodba J izpeljiva s pravili, potem je tudi sodba K.

Kot pri sklepanju z izpeljavo lahko tudi tukaj iteriramo dopustno sodbo tako, da zapišemo $J_1, \ldots, J_n \models J$, kar pomeni, če je J_1 izpeljiva,..., J_n izpeljiva, potem je tudi J izpeljiva.

Pravimo, da je naslednje pravilo dopustno, če in samo če $J_1, \ldots, J_n \models J$.

$$\frac{J_1, \ldots, J_n}{J} \tag{2.28}$$

Primer 2.5.2 *Za poljuben a nat velja naslednja dopustna izpeljava.*

$$\textit{succ}(a)\ \textit{nat} \models a\ \textit{nat} \tag{2.29}$$

Izjava je veljavna glede na pravila (2.3). To lahko pokažemo z indukcijo. Če velja succ(a) nat potem je moralo biti pri izpeljavi succ(a) nat uporabljeno drugo pravilo (2.3b). Toda potem mora veljati tudi a nat, ker je premisa pravila. □

Dopustno sklepanje je striktno šibkejše od izpeljevanja. Če velja $J_1, \ldots, J_n \vdash J$ potem velja tudi $J_1, \ldots, J_n \models J$. Obratno sploh ni nujno, da velja.

Poglejmo dokaz v levo stran. Če predpostavimo $J_1, \ldots, J_n \vdash J$, potem so $J_1, \ldots, J_n$ izpeljivi s pravili zapisano $\vdash J_1, \ldots, \vdash J_n$. Če upoštevamo pravila oslabitve in tranzitivnost, dobimo $\vdash J$, kar pomeni, da je J izpeljiv iz originalnih pravil.

Po drugi strani velja, da succ(a) nat $\nvdash a$ nat: ne obstaja način po katerem bi s pravili izpeljali a nat iz succ(a) nat.

Definicija dopustnega sklepanja pogojuje enake *strukturne lastnosti* kot hipotetično sklepanje. Veljajo naslednje lastnosti: refleksivnost, omejevanje in tranzitivnost.

Za razliko od hipotetičnega sklepanja, dopustno sklepanje je občutljivo na dodajanje pravil. Recimo, da smo razširili pravila (2.3) z naslednjim pravilom:

$$\frac{}{\text{succ}(smet)\ \text{nat}} \tag{2.30}$$

S to razširitvijo pravil ne velja več succ(a) nat $\models a$ nat, ker se novo dodano pravilo ujema z succ(a) nat v posledici, ne obstaja pa premisa, ki katere pravilnost bi dopustila a nat.

2.6 Opombe

Poglavje vsebuje prevode izbranih sekcij iz učbenika *Practical Foundations for Programming Languages* [5] avtorja Roberta Harperja.