Логическо програмиране

Антон Зиновиев

27 май 2016 г.

Съдържание

Глава 1

Математическа логика и програмиране

1.1. Безсмисленото начало

> Каква е ползата от Вашите красиви изследвания за π? Защо да се занимаваме с такива неща при положение, че ирационалните числа не съществуват?
>
> Леополд Кронекер

В математиката винаги явно или неявно са се използвали множества, но в продължение на дълго време множествата били образувани по сравнително прост начин. Имало е множества от реални числа, множества от функции между реални числа, множества от точки и т.н. и винаги е имало ясно и разбираемо правило, определящо кои точно са елементите на дадено множество. Математиците не се увличали по измамни занимания с далечни от ума обекти като множества, които съдържат всички подмножества на множеството, което съдържа всички подмножества на множеството, което съдържа всички подмножества на множеството, което съдържа всички естествени числа.

През последната четвърт на 19-ти век обаче се появила съблазън, която разклатила здравомислието на математиците. Основна заслуга за това имал Георг Кантор, който неблагоразумно дръзнал да разработи основите на това, което днес се нарича *теория на множествата*. За

съжаление, оказало се, че заниманията с теория на множествата неизбежно изискват от математѝка да описва свойствата на обекти, които поради самия им характер е невъзможно човек да си ги представи ясно. Тогава малцина съзрели колко опасни могат да бъдат тези прелъстителни занимания, а с особена прозорливост сред тях се отличил Леополд Кронекер. Той не спирал доблестно да предупреждава всички, че Кантор е псевдоучен, шарлатанин, ренегат и развратител на младежта, но — уви — малцина се вслушали в гласа на разума. И наказанието, не се забавило...

Едно след друго в теория на множествата се появили противоречия. Така например Кантор доказал, че за всяко множество X множеството 2^X има повече елементи от X. Но какво правим когато X е множеството, което съдържа всички множества? Как така 2^X ще съдържа повече елементи от множеството, което съдържа всички множества? За да избегне това противоречие, Кантор решил, че множеството, което съдържа всички множества, е твърде голямо и затова не можело да бъде множество. Друг парадокс е свързан с т.н. ординални числа. Оказало се, че от всяко ординално число има по-голямо, а в същото време множеството от всички ординални числа трябва да бъде най-голямото ординално число. И в този случай Кантор решил, че множеството от всички ординални числа е твърде голямо и затова не можело да бъде множество. Друго известно противоречие е парадоксът на Ръсел.[*] За да бъде преодоляно то, през 1908 Цермело измислил аксиоми, според които множествата се строят на безкрайна трансфинитна редица от стъпки като всяко множество може да съдържа само елементи, които вече са били образувани на някоя предшестваща стъпка. По този начин математиката започнала да прилича на съвременна компютърна програма, която е пълна с грешки и която след всяка новооткрита грешка трябва да се оправя.[**]

Кронекер починал, но призивът за конструктивизъм в математиката бил подет от други математици, напр. Анри Поанкаре. Особено активен бил Брауер, който измислил нов вид конструктивна математика, която нарекъл *интуиционизъм*.[***] Брауер установил, че ако ис-

[*] Обичайните множества не съдържат себе си като елементи. Да наречем такива множества *нормални* и нека R е множеството, което съдържа всички нормални множества. Самото R нормално множество ли е?

[**] Все още не е открито противоречие в аксиомите на Цермело.

[***] Името „интуиционизъм“ идва от това, че според Брауер математиката трябва да включва само конструкции, които можем да си ги представим ясно. Благодарение на това математикът може да установява правилността на една конструкция посредством просто позоваване на интуицията си.

каме да разработваме математиката конструктивно, ще се наложи да използваме логика, която е различна от обичайната. Така например в интуиционизма изказването „всяко твърдение е вярно или невярно“ е лъжа и въпреки това изказването „няма твърдение, което не е вярно и не е невярно“ е истина.

С огорчение Давид Хилберт гледал как дори собствените му ученици започвали да се увличат от идеите на интуиционизма. А когато към интуиционистите се присъединил и Херман Вейл, Хилберт решил да се намеси. Той заклеймил интуиционистите като метежници, организирали идеологическа чистка, които без да разполагат с доказателство за виновност унищожавали всичко, за което решат, че буди подозрения. А конкретно за Брауер той писал: „Аз съм безкрайно удивен от факта, че дори и сред математиците се оказва, че силата на внушението на един единствен човек, без значение колко темпераментен или остроумен, е в състояние да предизвика толкова невероятни и чудати въздействия.“

През 1922 г. Хилберт обявил своята знаменита програма за „спасяване“ на математиката. Тя се състои в следното:

(1). Дефиниране на формално точен математически език, на който може да се формулират математическите твърдения. Така формулираните математически твърдения се наричат *формули*.

(2). Формулиране на точно определени правила, посредством които от вече доказани формули можем да получаваме нови доказани формули.

(3). Доказателство, че така формулираните правила са достатъчни, за да се докажат всички математически верни твърдения.

(4). Доказателство, че с така формулираните правила не може да се стига до противоречие. В това доказателство Хилберт иска да се използват само методи, които биха били убедителни дори и за Брауер.

(5). Доказателство, че когато използваме в доказателствата сложни и неконструктивни обекти, ще можем да правим по-кратки доказателства, но няма да може да се докажат неща, които не могат да се докажат и без използването на неконструктивни обекти.

(6). Намиране на метод, посредством който може да се решава дали едно математическо твърдение е вярно.

Отначало нещата потръгнали добре. Още през 1879 г. Готлоб Фреге бил дефинирал логически език, който е еквивалентен по изразителна сила на съвременната предикатна логика от първи ред и на който може да се формулира цялата математика. Освен това Фреге формулирал и

точни правила, посредством които от вече доказани формули може да се получават нови доказани правила. С това първата и втората точка от плана на Хилберт можело да се считат за изпълнени.

Някои значителни успехи били постигнати и по отношение на останалите точки от плана. Например през 1929 г. Курт Гьодел доказал, че правилата на Фреге са достатъчни, за да се докаже с тях всяка логическа истина. Много на брой и извънредно важни резултати били получени в продължение само на няколко години. Това било окуражаващо и вдъхновяващо и изглеждало, че аха-аха, още съвсем малко и програмата на Хилберт ще бъде изпълнена.

Но през 1931 г. се случило немислимото — Гьодел доказал своите знаменити теореми за непълнота. От тях следвало, че:

- ако една математическа теория включва в себе си аритметиката,[*] то е невъзможно с краен брой формални правила и аксиоми да се докажат всички верни неща в нея;
- ако една математическа теория включва в себе си аритметиката, то тогава нейната непротиворечивост не може да се докаже, използвайки само методи, които се включват в тази теория;

Това означава, че по колкото и хитър начин да формулираме аксиомите, винаги ще има верни математически твърдения, които не следват от тях. И също така означава, че ако решим да се ограничим само с онези твърдения, които следват от аксиомите, няма да може да докажем, че от избраните аксиоми не може да се получи противоречие. Малко по-късно (през 1936 г.) Чърч и Тюринг показали, че и последната точка от плана на Хилберт е неосъществима — няма алгоритмичен метод, посредством който можем да познаваме дали едно математическо твърдение е вярно или не. Оригиналният план на Хилберт се оказал неосъществим...[**]

В резултат от усилията да се изпълни програмата на Хилберт се появил един нов раздел в математиката — *математическата логика*. Този нов раздел обаче не успял да осъществи това, заради което бил създаден. Той не успял да подсигури основите на математиката,

[*] По точно, ако в нея може да се дефинира какво значи събиране и умножение на естествени числа.

[**] Всъщност през 1936 г. Герхард Генцен успял да докаже по задоволителен начин непротиворечивостта на аритметиката. Няколко десетилетия по-късно Гаиши Такеучи доказал непротиворечивостта на различни теории, които са достатъчно богати, за да обхванат по-голямата част от съвременната математика. Засега обаче не знаем дори как да подходим към проблема за непротиворечивостта на теорията на множествата на Цермело, а тя все още се счита за основа на съвременната математика.

а обикновените математици продължили да си правят математика без много-много да се интересуват какво се случва в математическата логика. Изобщо математическата логика била абстрактна математическа дисциплина, която се оказала толкова безполезна за каквото и да е, че според Мартин Давис, когато той бил студент (1944 – 1950 г.), дори тополозите* се присмивали на логиците, че витаели някъде далеч в космическото пространство.

Но това съвсем скоро щяло да се промени...

1.2. Денят не си личи по заранта

> Самата конструкция е изкуството,
> а приложението ѝ в света — зъл паразит.
>
> Лойтцен Егбертъс Ян Брауер

Въпреки че приложенията на математическата логика в останалите дялове на математиката все още са като цяло незначителни,** приложенията ѝ в областта на информатиката и компютрите са толкова много и нерядко неочаквани, че човек неволно си задава въпроса защо това е така. Наистина, и други дялове на математиката имат компютърни приложения, напр. линейната алгебра, геометрията, теория на числата, числените методи, теория на вероятностите, теория на графите и комбинаториката. На нито един от тях обаче приложенията не са толкова разнообразни и всеобхватни, колкото приложенията на математическата логика.

Традиционно математическата логика се разделя на четири поддяла. Всеки един от тях има важни „компютърни“ приложения. Тези поддялове са:

*Топологията е полезна математическа дисциплина, в която има страшно много теореми, които вероятно никога няма да получат каквото и да е практическо приложение.

**Разбира се те са незначителни само на фона на големите очаквания. Ето някои немаловажни математически приложения на математическата логика. Теория на моделите има някои приложения в алгебрата, например доказателството на основната теорема на алгебрата (това, че всяко поле има алгебрически затворено разширение) се опростява, ако се използва логическата теорема за компактност. В математическия анализ обосноваването на инфинитезималното смятане на Лайбниц с безкрайно малки величини може да се счита за едно от най-важните постижения на математиката на 20-ти век. Много от резултатите в топологията пък са тясно свързани с дълбоки резултати в теория на множествата, а основните приложения на т.н. *дескриптивна теория на множествата* са в областта функционалния анализ, ергодичната теория и операторната алгебра.

– обща логика и теория на моделите;
– теория на множествата;
– теория на изчислимостта;
– теория на доказателствата и конструктивна математика.

Обща логика и теория на моделите

Машините, които хората ползвали от праисторическо време, променяли свойствата на различни материални обекти. Например пещта на грънчаря преобразувала неопечените глинени съдове в годна за използване керамика, каруцата на търговеца променяла географските координати на натоварените материални ценности, така че стойността им на новите координати да бъде по-голяма и т.н. Устройствата, които работели с нематериални обекти, т.е. данни, били малко на брой и сравнително прости.

През 1642 г. Блез Паскал изобретил първата механична сметачна машина. Сметачните машини били първите машини, които можели да преобразуват данни, но както при всички останали машини от онова време, действията, които те извършвали, се контролирали непосредствено от човек.

През 1805 г. Жозеф Мари Жакард изобретил тъкачен стан, който можел да тъче платове с повтарящи се орнаменти. Машината можела да чете данните, описващи нужните орнаменти, от перфокарти. Това била първата машина, чиито операции не се управлявали непосредствено от човек, а зависели от програма, записана в паметта на машината.

След като тези две изобретения били комбинирани, се получили т.н. *програмируеми сметачни машини* — сметачни машини, които се управляват не непосредствено от човек, а от програма. Първият проект за програмируема сметачна машина била „Аналитичната машина“ на Чарлз Бебидж (1837 г.). Програми за тази машина са писани от Ада Лавлейс, поради което тя днес се счита за пръв програмист.

За съжаление работата на Бебидж останала малко известна и не е използвана от по-късните конструктори. Първата действително конструирана и работеща програмируема сметачна машина е „Колумбийският диференциален табулатор“, създаден почти един век по-късно (1930 г.) от фирмата Ай Би Ем. Вестниците, винаги радващи се на сензации, нарекли тази машина „суперкомпютърно устройство с умствените способности на 100 математика, дори при решаване на много сложни алгебрични задачи“.

Възможностите на програмируемите сметачни машини бързо се увеличавали и машината „Цет 3“, създадена от Конрад Цузе през 1941 г. била първото устройство, което като изключим по-малката му памет и скорост на работа, има на теория същите изчислителни възможности както и съвременните компютри.

За да получим *компютър*, оставало да се реализира само още един ключов елемент — за програмируемите сметачни машини програмите и обработваните от тях данни били две напълно различни неща. Те можели да четат програмите си, можели да четат и входните си данни, но можели да записват само данни. Това ограничение означава, че за тях не може да се пишат компилатори, защото генерираният от компилатора изпълним код представлява хем данни, хем програма. Първият компютър, при който не се прави разлика между данни и програми, е машината „Ес Ес И Си“ на Ай Би Ем (1948 г.), вторият — машината „ЕДСАК“ на Кеймбриджкия университет (1949 г.) и третият — машината „МЕСМ“ на Киевския институт по електротехнология (1950 г.).

Това, че в данните, подавани на един компютър, се съдържат и инструкциите за компютъра, означава, че имаме нужда от формален език, на който да се формулират тези инструкции. Математическата логика е разделът от математиката, който ни дава техники за дефиниране на синтаксиса и семантиката на различни формални езици и изследване на свойствата на тези езици. Формалните езици, които могат да бъдат полезни на практика, далеч не се ограничават само с различни видове езици за програмиране. Всеки формален език, който допуска ефективна обработка, обикновено се оказва или полезен, или много полезен.

* * *

Много и най-разнообразни езици са дефинирани в математическата логика, но ако трябва само един от тях да бъде наречен „Езикът на математическата логика“, то честта я има *предикатната логика от първи ред*. Този език притежава голяма изразителна сила — почти цялата съвременна математика може да бъде формулирана на него. Освен това той е и много удобен за използване — нерядко в математически статии твърденията се формулират, използвайки този език, а не на естествен език (български, английски и т.н.). А приложението на този език като език за заявки към бази данни представлява един от най-впечатляващите примери за компютърно приложение на математическата логика.

По принцип, една от пречките за ефективна обработка на предикатни формули е използването в тях на т.н. свързани променливи. Тази пречка била преодоляна още през 1948 г. от Алфред Тарски, който в желанието си да алгебризира предикатната логика, измислил *цилинд-*

ричните алгебри. Езикът на цилиндричните алгебри има същата изразителна сила като езика на предикатната логика с равенство,[*] но за разлика от него не притежава свързани променливи.

Оставало само да се види как идеите от цилиндричните алгебри можели да се реализират ефективно. Това направил Едгар Код през 1969 г., предлагайки *релационния модел за управление бази данни*. Код доказал, че релационният модел притежава същата изразителна сила, както и предикатната логика от първи ред. Освен това този модел допуска и изключително ефективна реализация — ако разполагахме с възможност за неограничена паралелизация, тогава заявките към една релационна база данни биха се изпълнявали за константно време без значение какъв е нейният размер. Съвсем скоро след публикацията на Код се появил и езикът ес кю ел, който реализира релационния модел и е най-популярен език за управление на бази данни вече повече от 40 години.[**]

* * *

Един друг важен за математическата логика език е езикът на λ-*смятането* измислено от Аланзо Чърч (1930 – 1936 г.). Също както предикатната логика е дала математическата основа на релационните бази данни, така λ-смятането е дало математическата основа на функционалното програмиране. И също както в предикатното смятане има свързани променливи, поради които то е неудобно за непосредствена реализация, така и в λ-смятането има свързани променливи. Само че докато при базите данни хората (поне засега) се мъчат и вместо езици със свързани променливи използват по-неудобни езици като ес кю ел, то при функционалното програмиране почти няма функционални езици без свързани променливи. Решението, което гарантира хем удобство, хем ефективност, е следното: ще предоставим на програмистите удобни езици със свързани променливи, но на ниско ниво ще реализираме тези езици посредством бързи виртуални машини без свързани променливи, а преводът ще се прави от компилатора напълно автоматично.

Първият и изключително остроумен начин за елиминация на свър-

[*] При цилиндричните алгебри няма функционални символи, но от гледна точка на изразителната сила това не е съществено ограничение.

[**] В последно време скоростта на компютрите при последователни изчисления лека полека започва да достига своя максимум. Все още обаче има възможност за увеличаване на паралелността. Тъй като релационните бази данни могат да използват в пълнота паралелността на компютрите, това означава, че през следващите години можем да очакваме, че бързината и възможностите на специализираните системи за управление на бази данни ще се увеличат в значително по-голяма степен, отколкото бързината на изпълнение на обичайните компютърни програми.

заните променливи от λ-смятането всъщност е измислен преди да се появи самото λ-смятане. Това е *комбинаторната логика* на Мойсей Шейнфинкел и Хаскел Къри (1924 – 1930 г.). На нейна база много покъсно е измислена т.н. SK-*машина*, която се използва за компилация напр. на езиците SASL и миранда. Освен оригиналния метод на Шейнфинкел и Къри, вече са разработени и много други методи за отстраняване на свързаните променливи. На практика всяка ефективна виртуална машина, използвана за реализация на съвременен функционален език (напр. категорната абстрактна машина, виртуалните машини с комбинатори и суперкомбинатори), е получена в резултат на теоретични логически изследвания.

* * *

Въпреки, че като цяло уменията на компютрите да измислят математически доказателства са несравнимо по-слаби от уменията на един професионален математик, има една област, в която компютрите нерядко се справят значително по-добре от хората — доказателства за коректност на сложни паралелни програми.

Изчислението на една паралелна програма протича недетерминистично и за да можем да кажем, че тя е коректна, трябва да сме сигурни, че при всички възможни изчисления програмата работи вярно. Броят на начините за протичане на изчисленията е огромен и е непосилно човек да провери ръчно всеки един от тях. Затова ако една програма е паралелна по сложен начин, програмистите няма да бъдат в състояние да се уверят, че тя е вярна. Не случайно в съвременната програмистка практика въпреки нарастващата паралелизация на съвременните компютри, съществено паралелни програми почти не се използват.

Въпреки, че отказът от паралелизация е донякъде допустим подход при програмиране, той е напълно непрактичен когато се разработват интегрални схеми. Изчислителните процеси в една интегрална схема задължително трябва да бъдат силно паралелни, защото в противен случай интегралната схема би работила твърде бавно.

Решението на проблема се състои в използването на т.н. *проверка на модели* (на англ. model checking). Първо задачата, която искаме да решим, се формулира на някакъв логически език, а след това проверяваме, че интегралната схема или програмата удовлетворява логическата формулировка. Една от най-използване за целта логики е *линейната темпорална логика*. Тази логика е удобна, защото верността при нея може да се проверява посредством сравнително бързи алгоритми, използващи крайни автомати.[*] В днешно време няма разработчик на

[*]Под „сравнително бързи“ тук се има предвид с експоненциална сложност. При

интегрални схеми, който не използва проверка на модели. Що се касае до използването на този метод за проверка на софтуер, то тук приложенията са засега по-ограничени и свързани най-вече с проверка на драйвери на физически устройства.

Трябва да отбележим, че проверката на модели е нещо напълно различно от т.н. доказателствено програмиране, за което ще спомена малко по-нататък. Тъй като далеч не всяка задача, която искаме да решим, може да се формулира на езика на линейната темпорална логика, методът проверка на модели е приложим само за един сравнително тесен клас задачи. Ако обаче той е приложим, то той със сигурност ще ни даде еднозначен отговор дали програмата е вярна или не. За разлика от проверката на модели, доказателственото програмиране е приложимо при всякакъв вид задачи, но не винаги дава отговор.

* * *

Различни видове *логики за знания* могат да се използват, за да се доказва коректност и безопасност на комуникационни протоколи, на възстановяване на база данни след аварии и др. Да разгледаме един хипотетичен пример. Хакерите Чингиз и Атила решили да атакуват уеб-сайта на Демагог. За да бъде успешна атаката им, те трябва да я започнат едновременно. Как могат да се договорят за това? Чингиз праща съобщение до Атила, в което съобщава часа на атаката и му пише, че няма да атакува, ако не получи потвърждение. Атила отговаря положително на запитването на Чингиз и също му пише, че ще участва в атаката, ако знае, че в нея ще участва и Чингиз. Ще осъществят ли атаката двамата? Не, защото Атила не знае дали Чингиз ще атакува и затова Атила няма да атакува. От друга страна Чингиз знае, че Атила не знае дали Чингиз ще атакува и затова знае, че Атила няма да атакува и значи и той няма да атакува. За да бъде преодолян този проблем, Чингиз пише до Атила, че знае, че Атила знае, че Чингиз иска да атакува. Но и след това потвърждение Чингиз все още не може да атакува, защото не знае дали съобщението му е било получено от Атила. Затова Атила пише до Чингиз, че знае, че Чингиз знае, че Атила знае, че Чингиз иска да атакува. Но тъй като в този момент той не знае дали Чингиз знае, че Атила знае, че Чингиз знае, че Атила знае, че Чингиз иска да атакува, то атаката отново е невъзможна.

Очевидно по този начин Чингиз и Атила не могат да започнат координирана атака. Има ли друг, по-смислен протокол, посредством който двамата могат да постигнат желаната договорка? Оказва се, че не. С

този вид задачи сме се примирили да считаме експоненциалните алгоритми за бързи.

използването на логики за знания може да се докаже, че колкото и да са хитри двамата, е невъзможно да се уверят, че всеки от тях знае каквото трябва да знае.

Съждителната динамична логика с номинали е един от най-мощните логически езици, за които е известен алгоритъм за проверка на верността. С тази логика може да се решават всички задачи, които могат да се решават посредством линейната темпорална логика, но за разлика от нея, тя може да се използва и като логика за знания с цел да се проверява коректността на комуникационни протоколи. Тя е открита от Соломон Паси и проф. Тинко Тинчев, а Георги Гаргов е установил, че за тази логика съществува алгоритъм, който проверява дали една логическа формула е вярна. Това е едно от онези български математически постижения, които са цитирани най-често в международните научни публикации.

Използвайки съждителната динамична логика с номинали, Борислав Ризов е разработил система, с помощта на която филолози могат да откриват думи, притежаващи определени семантични свойства. Например те могат да зададат на системата следния въпрос: „намери ми дума, която в българския език е синоним на думата 'обитавам', но ако преведем тази дума и думата 'обитавам' на английски, ще получим думи, които не са синоними“.[*]

* * *

През последните години бурно развитие имат различни логики, описващи взаимодействията между различни геометрични обекти, в които базовото геометрично понятие не е „точка в пространството“ а „област (регион) в пространството“. Такива логики могат да се използват при разработката на „умни“ устройства, които могат да се придвижват в пространството без да бъдат управлявани от човек. Тези логики се използват също и при разработката на „изкуствения интелект“ на някои видове компютърни игри. Няколко български математици са работили върху „безточковите“ геометрии.[**] Например Владислав Ненчев е изобретил безточкови геометрични логики, в които могат да се изказват сложни твърдения от типа „докато обект A се допира до B или е в контакт с C, A ще се намира изцяло в региона D“. Освен това Ненчев е намерил и ефективен алгоритъм, посредством който е възможно да се проверява верността на твърдения, изказани в тези логики.

[*] Достъп до системата може да се получи на адрес `http://dcl.bas.bg/bulnet/`.

[**] Сред тях са Николай Белухов, проф. Димитър Вакарелов, проф. Георги Димов, Татяна Иванова, Владислав Ненчев, проф. Тинко Тинчев.

Теория на множествата

Компютърните програми се пишат на езици с точен синтаксис и общо взето точна семантика. Независимо по колко сложен начин са свързани по между си компонентите на една програма, този начин е описан по най-прецизен начин в текста на всяка една компютърна програма. А този програмен текст не се появява от нищото, ами преди изобщо да сме в състояние да започнем да го пишем, е нужно да отделим не малко време за мислене. И колкото по-сложна е една програма и по колкото по-сложен начин са свързани по между си компонентите ѝ, толкова по-сложна и решаващо важна е тази първа стъпка от създаването на програмата — обмислянето на нейната архитектура, на потоците от данни в нея, на взаимодействието на програмата с външния свят, на взаимодействието на един програмен компонент с друг, на структурите данни и алгоритмите, които ще се използват, на методите за откриване на грешки, на възможностите за бъдещо развитие на програмата. Ако по време на това предварително обмисляне допуснем грешка, последствията са сериозни и нерядко непоправими. Програмата трябва или да се пренапише, или да се остави с дефекти, описани в документацията ѝ.

По време на предварителното обмисляне на една програма се налага да работим с много по-абстрактни понятия, отколкото когато дойде време да я пишем. Начинът, по който разсъждаваме, е същият, който използваме тогава, когато измисляме нова математическа теория. А всеки работещ математик е познал от собствен опит, че математическата интуиция може да заблуждава. След като измисли нещо ново, математикът задължително го проверява — дава прецизни дефиниции на понятията и формулира разсъжденията, които дотогава са съществували само в неясен вид в ума му. Ако всичко излезе както трябва — хубаво, но нерядко се оказва, че след точна формулировка нещата излизат не точно такива, каквито си ги е мислил математикът.

Същото се случва и при предварителното обмисляне на една програма. И колкото по-рано открием допуснатите грешки, толкова по-добре. Само че докато математикът е трениран да използва точен математически език и точни методи за разсъждение, то проектантът на една програма за съжаление най-често е самоук и не знае как провери предварителните си идеи. На него не му остава нищо друго, освен да пристъпи към написването на програмата и попътно да си доизяснява нещата и да ги поправя, ако все още не е твърде късно.

В университетските математически курсове на студентите се преподават доказателства. На пръв поглед това може да се стори ненужно — защо например е нужно в курса по математически анализ да ни се

преподава доказателството на правилото на Гийом Франсоа дьо Лопитал? Самото правило се помни лесно и се използва лесно, а хиляди математици преди нас са чели доказателството му и са се уверили, че то е вярно. Защо трябва и ние да учим това доказателство, нима се съмняваме във верността му? Е, някога математиката се е преподавала без доказателства, а просто като съвкупност от проверени практически правила, но за щастие от тогава сме се поучили, че така не бива. В областта на програмирането обаче, още не е дошло времето да си извлечем аналогична поука.

В древността е имало велики архитекти, проектирали сгради, на които и днес се възхищаваме, но не е имало наука архитектура. За да бъдеш добър архитект се е искало огромно количество талант, изкуство, усет, опитност, вярна интуиция. Чак когато натрупаният опит бил оформен във вид на систематизирано знание, станало възможно и по-обикновени хора да станат архитекти или строителни инженери и да проектират сгради, които няма да паднат преди да са построени. Аналогична е ситуацията и със софтуерната архитектура и софтуерното инженерство. За съжаление обаче опитите за систематизиране на знанията в тези области са сравнително малко, така че добрите софтуерни архитекти и софтуерни инженери са такива не защото са завършили специалност софтуерно инженерство в някой университет.

Съвременната ситуация с обучението по информатика не е добра — в много университети студентите изобщо не се обучават как да проектират правилни компютърни програми. Езиците за спецификация, ако изобщо се преподават, се преподават сравнително повърхностно, а най-лошото е, че студентите така и не разбират кога и как трябва да се използват тези езици на практика. Немалко специалисти например препоръчват в книгите си използването на формални спецификации като вид споразумение между клиентите и програмистите, а няма съмнение, че те точно това преподават и на студентите си. Въпреки, че в някои случаи това наистина е полезно, най-често формалната спецификация от една страна не отговаря на истинските желания на клиента, а от друга — забранява на програмистите да реализират варианти, които са хем по-лесни за програмиране, хем по-полезни за клиента.

В други случаи се създава погрешното впечатление, че е най-добре най-напред да се специфицира цялата програма и едва след това да се пристъпи към писане на програмния код. В действителност ако програмата е достатъчно сложна, това е наистина абсурден начин за програмиране. То е все едно учените и инженерите, участващи в проект за изпращане на хора до Луната, директно да се опитват да създадат чертежите на необходимата ракета и космическия кораб, както и плана

на полета, без преди това да са осъществили някоя по-проста част от целия проект. Полетът до Луната би бил невъзможен, ако преди това разработчиците не бяха усвоили излизането със скафандър в открития космос, маневрирането в околоземното и окололунното пространство, скачването на два апарата в околоземното и окололунното пространство и т.н. По същия начин трябва да се постъпва и при програмиране. Колкото и внимателно да сме подготвили спецификацията на програмата, неминуемо реалното писане на код ще ни поднесе изненади. Затова след като някоя част от програмата ни се изясни и сме готови със спецификацията ѝ, най-добре е да пристъпим към програмирането ѝ, макар останалата част от програмата все още да остава неясна.

За съжаление доброто желание не винаги е достатъчно... Много е лесно ако използването на език за спецификации бъде наложено силово от ръководството на една фирма, а екипът няма нужния опит, вместо този език да стане мощно средство за подпомагане на програмистите, той да се превърне в инструмент за мъчение.* Наистина има какво да се сбърка — кои свойства на програмата е важно да бъдат включени в предварителната спецификация и за кои ще бъде вредно това да се прави, до каква степен да проявяваме гъвкавост и да модифицираме предварителната спецификация, ако след като започне писането на програмата установим, че нещо е било по-добре да се направи по друг начин и т.н. И всички тези проблеми са при положение, че екипът вече е усвоил използването на формален език за спецификации, което само по себе си е предизвикателство.

* * *

Вече споменахме, че използването на формална спецификация може да помогне на проектантите да открият фундаментални грешки и недостатъци в конструкцията на програмата много преди да започне нейното писане. Тази полза е налице дори тогава, когато проектантът е един човек и прави спецификацията само за себе си. И наистина, формалното описание на нужните свойства отнема много по-малко време, отколкото самото написване на програмата, а откритите благодарение на него грешки се поправят много по-бързо. И когато наистина дойде време да пишем програмния текст, структурата на програмата ни е ясна и затова програмираме по-бързо и допускаме по-малко грешки.

От използването на формални спецификации произтича и една друга, по-незабележима, но съвсем не маловажна полза. Каквато и дейност да извършва човек, той неволно винаги се опитва да я свърши по въз-

*Един познат програмист ми каза точно това — езиците за спецификация са инструмент за измъчване на програмистите и заблуждаване на клиентите.

можно най-лесния начин. Затова когато програмистите пишат програмния текст, те разсъждават локално — как да свършат задачата по най-простия и елегантен начин без да мислят дали създадените по този начин библиотеки от класове или функции са прости за използване и с ясно и разбираемо действие. Когато обаче се пише спецификация, човек пак в желанието да си спести писането, не мисли колко трудна е реализацията, а само за това колко просто са формулирани желаните свойства на програмните компоненти. В резултат на това взаимодействието на програмните компоненти става възможно най-опростено и лесно за разбиране, а програмните грешки най-често са локални и лесни за оправяне. Опростеното взаимодействие на програмните компоненти помага и при последващата поддръжка на програмния код. Когато модифицират вече написаната програма, програмистите, дори да не са участвали в първоначалното ѝ написване, се чувстват много по-уверени, че действията им няма да доведат до непредвидени грешки.

Фирми, които са опитвали да специфицират вече написани програми, са установили, че това е безсмислена задача, защото изведнъж става нужно да опишат точно всичката онази невидима сложност във вече написаната програмата. [1, стр. 16] Сложност, която със сигурност е затруднила поправянето на програмните грешки и със сигурност ще затрудни последващата поддръжка.

* * *

Къде във всичко това присъства теория на множествата? Е, оказва се, че всички езици за формална спецификация по един или друг начин се свеждат до използване на езика на теория на множествата. Тази незаменимост на езика на теория на множествата не е изненадваща — щом като този език е от една страна необходим в съвременната математика, а от друга страна напълно достатъчен, за да изкажем на него цялата съвременна математика, без нито едно изключение, защо да не очакваме същото да важи и за програмирането?*

По-старите езици за спецификация (напр. зед) умишлено подчертават връзката си с теория на множествата. Използването на тези езици естествено е невъзможно от програмисти, които не са получили нужната математическа подготовка. В по-ново време обаче стана ясно, че вместо непосредствено за множества, можем да използваме терминология, по-разбираема за програмистите. Например вместо да казваме, че x е елемент на множеството A, можем да казваме, че x е обект от

*Дълго време изглеждаше, че конструктивната математика задължително изисква използването на не-множествен език. След създаването на конструктивната теория на множествата стана ясно, че това не е така.

класа A; вместо да казваме, че предикатът $p(x, y)$ е истина за x и y, можем да казваме, че обектът x има поле с име p, чиято стойност е y; вместо да използваме квантори, може да *скулемизираме* и т.н. Всичко това обаче в никакъв случай не е заместител на теория на множествата, а просто начин да се даде на програмистите без математическа подготовка език, равносилен на езика на теория на множествата. При използването на такъв език е важно да се знае, че „класовете“, за които се говори в спецификацията, изобщо не е нужно да съответстват на реални класове в програмата, „полетата“, които притежават обектите, не е нужно да съществуват и в истинската програма и т.н.

Теория на изчислимостта

Когато на света се появили компютрите, оказало се, че важна част от теорията на това, какво и как може да се смята с тях, вече съществувала. Днес този дял от математическата логика се нарича „*теория на изчислимостта*“.*

В доказателството на своите теореми за непълнота Гьодел дефинирал понятието „*примитивнорекурсивна функция*“. Малко по-късно, през 1934 г., Гьодел дефинирал и понятието „*общорекурсивна функция*“, което се оказало еквивалентно на съвременното понятие „изчислима функция“.** По онова време обаче Гьодел все още не допускал, че понятието, което той дефинирал, обхващало всички функции на естествени числа, за които може да се счита, че са изчислими.

Тезата, че интуитивното понятие „изчислима функция“ отговаря на даден, точно дефиниран клас от математически функции, е изказана за пръв път през 1936 г. от трима математици — Аланзо Чърч, Алън Тюринг и Емил Пост. Всеки от тях дал различна дефиниция на това какво значи изчислима функция, но тези дефиниции се оказали еквивалентни както по между си, така и на по-ранната дефиниция на Гьодел. Според Чърч изчислимите функции са точно функциите, които могат да се дефинират с т.н. λ-термове. Това твърдение днес е известно като „*тезис на Чърч*“, а създаденото от Чърч *λ-смятане* служи за математическа основа на почти всички съвременни функционални езици. Тюринг пък дефинирал това, което днес е известно като „машина на Тюринг“, а твърдението, че изчилимите функции са точно функциите, които може да се пресмятат с машина на Тюринг, е известно като

*Дълго време този дял се наричал „теория на рекурсията“, а изчислимите функции — рекурсивни функции.

**Според Гьодел идеята за тези функции му е била подсказана от Ербран.

„*тезис на Тюринг*“.* Дефиницията на Пост за изчислима функция е удивително сходна с дефиницията на Тюринг, макар да е получена напълно независимо. Простотата на тези две дефиниции ги прави много удобни за математически изследвания.

По-късно се появили и други, различни от тях дефиниции на „изчислима функция“, но и те се оказали еквивалентни по между си. Сред тях по-важни са каноничните системи на Пост (1943), нормалните алгоритми на Марков (1948) и различните видове регистрови машини (1954 – 1967). Каноничните системи на Пост и нормалните алгоритми на Марков стоят в основата на символните пресмятания в съвременната компютърна алгебра, както и на някои езици за програмиране, напр. рефал. Регистровите машини пък и особено машините на Мински съчетават теоретичните удобства, които имат машините на Тюринг, с обичайния стил на програмиране, който се използва при императивните езици за програмиране.

В някои случаи теорията на различните изчислителни модели дава идеи за нови стилове за програмиране (напр. идеята на функционалното програмиране е дошла от λ-смятането). В други случаи тя дава идеи за ефективно използване на виртуални машини. Методи от математическата логика се използват също и при разработката на компилатори и особено на оптимизиращи компилатори. Все по-голямо значение добиват системите за статичен анализ на програмен код. Такива системи например автоматично могат да откриват дали някоя променлива се използва без да е инициализирана.

* * *

В теория на изчислимостта се разработват различни методи за дефиниране на семантиката на различните езици за програмиране. Наличието на точна семантика на един език е много важно при реализацията му, защото в противен случай комбинираното използване на различни свойства на езика, всяко от които само по себе си изглежда ясно, се оказва, че води до програми, за които не е ясно какво точно трябва да правят, и затова компилаторът работи неправилно.

Различните изчислителни модели и семантиките на езиците за програмиране могат да подскажат на създателите на нови езици за програмиране как да дефинират езиците така, че те да бъдат ясни и с проста семантика (в противен случай става много трудно да се пишат верни програми на тези езици). Например в областта на обектноориентираното програмиране използването на сходна терминология от различните

*Тъй като дефинициите на Чърч и Тюринг са еквивалентни, от математическа гледна точка тезисът на Чърч и тезисът на Тюринг казват едно и също нещо.

езици за програмиране създава илюзията, че всички те реализират на практика едно и също нещо. В действителност на семантиката на различните обектноориентирани езици показва, че те могат да се разделят на две групи. В едната група спадат езици, чиято математическа теория се базира на безтипови обекти. В тази група попадат например езиците смолтоук и обджектив си.* При множественото наследяване при тези езици най-много един от родителските класове може да бъде истински, докато останалите могат да предлагат само интерфейс, но не и конкретна реализация. Втората група обектноориентирани езици са тези, чиято математическа теория се базира на някоя теория на типовете. Към тази група езици спада айфел. При множественото наследяване при тези езици често се допуска всеки един от родителските класове да предлага конкретна реализация. А има и езици като си++, при чието създаване явно не се е обръщало достатъчно внимание на това каква точно трябва да бъде тяхната семантика. Може би затова множественото наследяване при си++ е трудно за използване и всъщност никой не го използва.**

В днешно време има много езици, които изобщо не притежават оператор goto, но това, че операторът goto наистина не е нужен, става ясно от една теорема на Корадо Бьом и Джузепе Якопини от 1966 г.

Важни резултати в областта на семантиките на езиците за програмиране са получени в България от проф. Иван Сосков. По-горе се спомена, че съществуват различни дефиниции на понятието изчислима функция и всички те се оказват еквивалентни. Това е така обаче само когато говорим за функции върху естествени числа. Затова един естествен въпрос е какво се случва ако вместо функции върху естествени числа използваме функции, дефинирани за произволен абстрактен тип данни. Оказва се, че в този случай различните математически модели за изчислимост не са еквивалентни. Така например Сосков е установил, че: 1. за всяка императивна програма можем да намерим еквивалентна на нея функционална програма, но с функционални програми понякога може да се правят и неща, които не могат да се направят с императивни [26] и 2. за всяка функционална програма можем да намерим еквивалентна на нея логическа програма, но с логически програми

*Към тази група езици спада и джава, въпреки че езикът създава илюзията за статична типизация.

**Понякога хора, повлияни от си++, изказват тезата, че множественото наследяване е едва ли не безполезно и опасно, когато повече от един от родителските класове предлага реализация. Всъщност езикът айфел нагледно показва точно обратното — ако езикът е базиран на хубава математическа теория, такова наследяване не само е напълно безопасно и полезно, но също така и много приятно за използване.

понякога може да се правят и неща, които не могат да се направят с функционални. [17]

* * *

Един от важните проблеми в съвременната информатика е свързан с удобното и безопасно използване на паралелни изчисления. Този проблем има голямо практическо значение, защото компютрите стават все по-паралелни. За решаването му се разчита на създаването на прост математически модел на паралелните изчисления, но за съжаление такъв все още не е измислен. Вредата от това е не само теоретична — отсъствието на хубава математическа теория води до това, че паралелното програмиране с право се счита за опасно и на практика почти винаги води до трудни за откриване и оправяне програмни грешки. Два съществуващи модела на паралелните изчисления се дават напр. от π-*смятането* и *джойн-смятането*. Макар те да са по-сложни, отколкото ни се иска, езиците за програмиране, в които средствата за паралелизъм са базирани на някое от тези две смятания, са по-удобни за писане на правилни паралелни програми, отколкото езиците, при които паралелизмът не е базиран на никаква теория.[*]

Скоро след появата на компютрите станало ясно, че има разлика между теоретично изчислима функция и функция, която е изчислима на практика. Това мотивирало бързото развитие на изключително важната теория за сложността на изчисленията. За съжаление в тази област има много задачи, които се оказват изключително трудни и все още остават нерешени. Най-знаменитата и все още не решена задача е въпросът дали „P=NP“.[**] Тук P обозначава функциите, които грубо казано може да се пресмятат за полиномиално време на последователен компютър, а NP — функциите, които може да се пресмятат за полиномиално време на неограничено паралелен компютър. Този проблем е важен по две причини. От една страна класът P съдържа функциите, за които можем да считаме, че се смятат за разумно кратко време на компютър,[***] а от друга — има много на брой изключително важни

[*] Изводът, който можем да си направим, е следният: програмирането с лоша математика е по-добро от програмирането с никаква математика.

[**] Обявена е награда от 1 000 000 долара за този, който пръв намери вярно решение на този проблем.

[***] Става въпрос за компютър със стандартна архитектура. Квантовите компютри могат да решават ефективно някои задачи, които изискват неограничена паралелизация. В частност повечето от използваните в момента алгоритми за асиметрична криптография ще престанат да бъдат сигурни, ако разполагаме с квантови компютри с няколко хиляди кубита памет. Предполага се обаче, че не всяка NP задача може да се решава ефективно от квантов компютър, и има алгоритми за асиметрична криптография, които са използваеми с нормален компютър и за които се счита,

за практиката задачи, които принадлежат към класа NP. Ако имаме щастието да се окаже, че P=NP, то това означава, че ще разполагаме с ефективен алгоритъм за решаване на тези важни задачи. От друга страна, ако имаме нещастието да се окаже, че P=NP, то това гарантирано ще направи неизползваеми всички използвани на практика криптографски алгоритми.

Освен с директно измерване на времето за изчисление и използваната памет, един алтернативен подход за измерване на сложността на една изчислима функция е това колко сложна е нейната дефиниция (на пръв поглед не се вижда защо има връзка между сложността на дефиницията на една функция и времето за нейното пресмятане, но такава има). Функциите, които притежават проста дефиниция, се изучават в теорията на субрекурсивните функции. Субрекурсивните функции описват по-добре отколкото изчислимите функции това, какво на практика може да се пресмята с компютър. Затова е важно да бъде изследвана не само субрекурсивната изчислимост между естествени числа, но също и субрекурсивната изчислимост между други полезни математически обекти. Субрекурсивната изчислимост на реални числа все още е сравнително малко изследвана област, като повечето от съществуващите резултати са получени в България от Иван Георгиев и проф. Димитър Скордев.

Когато в математиката бъде разработена теорията на определен вид, в някакъв смисъл подобни обекти, е естествено тази подобност да бъде формализирана, след което систематично да бъдат потърсени други обекти, притежаващи така формулираните свойства. Например в алгебрата след като се открият общите свойства на различните числови полета, получаваме аксиомите на абстрактното понятие поле и установяваме, че има полезни нечислови полета. В областта на теория на изчислимостта същото нещо е направено за пръв път от проф. Димитър Скордев [25, 16]. Той е дефинирал аксиоматично понятието „*комбинаторно пространство*“ и е показал че някои от основните резултати в теория на изчислимостта могат да бъдат изведени като следствие от аксиомите на комбинаторните пространства. Самите комбинаторни пространства пък се оказва, че имат много интересни модели, които ни дават неочаквани нови видове изчислимости (много от тях все още очакват някой да им разработи теорията в пълнота). След комбинаторните пространства Любомир Иванов [8] е дефинирал понятието „*оперативно пространство*“, а важни резултати в тази област са получени

че биха останали сигурни дори и да разполагаме с квантови компютри с голяма памет.

и от Йордан Зашев. Всички тези резултати са дали началото на това, което днес се нарича аксиоматична теория на изчислимостта.*

Теория на доказателствата и конструктивна математика

Вече бе споменато, че ако искаме да разработваме математиката конструктивно, ще се наложи да разсъждаваме, използвайки алтернативна логика, която е различна от класическата. Тази логика се нарича *интуиционистка логика*, а законите ѝ били формулирани от Аренд Хейтинг през 1930 г.**

В класическата математика когато твърдим, че нещо съществува, е все едно дали разполагаме с метод, посредством който можем да намерим съществуващото нещо. В конструктивната математика това не е така. Ако един конструктивист докаже, че съществува обект с определено свойство, то значи той разполага с метод, посредством който може да бъде намерен този обект. Ако пък бъде доказано твърдение от вида „за всяко x е вярно $x \leqq 0$ или $x \geqq 0$“, от тук автоматично ще следва, че има ефективен метод, с помощта на който за всяко x можем да познаваме дали е вярно $x \leqq 0$ или $x \geqq 0$.

Според конструктивистите доказателството на едно твърдение A представлява точно определена конструкция, която реализира A. Например доказателството на твърдение от вида „ако A, то B“ ни дава конкретен метод, с помощта на който ако разполагаме с конкретно доказателство на A, ще получим конкретно доказателство на B. С други думи можем да си мислим, че доказателството на „ако A, то B“ е функция, на която ако ѝ дадем като аргумент доказателство на A, ще ни върне като стойност доказателство на B. Доказателството пък на твърдение от вида „A и B“ представлява просто комбинацията от конкретно доказателство на A и конкретно доказателство на B.***

Казаното до тук може би подсказва, че би трябвало да има някаква връзка между интуиционистката логика и програмирането. Дали няма

*Това засега може би е единственият дял в математиката, който е създаден в България. В България той по-често се нарича „алгебрична теория на рекурсията“.

**Това, че е възможно формулираме конструктивните разсъждения, използвайки сравнително прости формални правила и аксиоми, е било изненада за създателя на интуиционизма Брауер. Отначало той считал, че не някакви формални правила и аксиоми, а единствено ясната математическа интуиция може да ни даде правилна основа на математиката. Затова той се е противопоставял на опитите за формализация на интуиционизма, а изследванията на Хейтинг наричал „безрезултатни упражнения“.

***Този начин за интерпретация на логическите връзки е известен като интерпретация на Брауер-Хейтинг-Колмогоров.

да се окаже, че всяка интуиционистка логическа формула дефинира някаква изчислителна задача, а доказателството на тази интуиционистка формула представлява програма, която решава тази изчислителна задача? Оказва се, че това е точно така и това е забелязано от Хаскел Къри и Уилям Алвин Хауърд. Може да считаме, че всяка интуиционистка формула представлява някакъв тип данни, а всяко доказателство на формулата е програмен обект (напр. функция), притежаващ съответния тип. Вярно е и обратното — може да си мислим, че типовете данни са интуиционистки формули от определен вид, а всяка компютърна програма представлява някакво интуиционистко доказателство. Наборът обекти, дефинирани в една компютърна библиотека, може да бъде считан за набор от аксиоми на интуиционистка математическа теория, а всяка програма, която е написана, използвайки единствено обектите от предоставената библиотека, представлява интуиционистка теорема, доказана използвайки само дадените аксиоми.

Това удивително съответствие между програмиране и математика е известно като „*изоморфизъм на Къри – Хауърд*“. То може да бъде изказано не само неформално (както направихме тук), но също и като точно математическо твърдение. Изоморфизмът на Къри – Хауърд показва, че програмирането и правенето на математика представляват не просто две подобни дейности, а напълно идентични дейности.

Ползата от този изоморфизъм е не само философска, защото той прояснява какво всъщност представляват типовете данни в езиците за програмиране. Създателите на почти всички типизирани функционални езици за програмиране умишлено разчитат на този изоморфизъм, за да си гарантират, че системата от типове ще бъде хем проста за използване, хем пълна. Напълно заслужено на името на Хаскел Къри е наречен един от най-популярните в момента езици за функционално програмиране — хаскел.

Този изоморфизъм има и други практически приложения. Например той подсказва как да разширим типовете данни в един език за програмиране по такъв начин, че спецификацията на това какво прави една функция да бъде част от нейния тип. И също както при използването на обикновени типове компилаторите могат статично да проверяват дали няма нарушения на типовете, така и при използването на тези разширени типове се оказва, че това е възможно. Това позволява компилаторът не само да компилира програмата, но и да удостоверява, че тя е вярна и не съдържа нито една програмна грешка. Въпреки, че засега всички съществуващи системи от този тип са трудни за използване, вече има оптимизиращ компилатор на си, за който знаем, че няма програмни грешки, защото е програмиран, използвайки система

с интуиционистки типове.

В България изследвания свързани с изоморфизма на Къри – Хауърд са правени от Трифон Трифонов.

* * *

От напълно различен вид е връзката между математика и програмиране при логическото програмиране. При логическото програмиране логическите формули се интерпретират не като типове данни (както е при функционалното програмиране), а като компютърни програми. Процесът на търсене на доказателство на логическата формула пък представлява изпълнението тази програма.

Освен като теоретична основа на логическото програмиране, компютърното търсене на доказателства е полезно и само по себе си. Наистина, съществуващите в момента алгоритми за автоматично доказателство на теореми не могат да се сравняват по ефективност с уменията на един професионален математик. Всеки, който се е занимавал професионално с програмиране, обаче знае, че през повечето време на програмистите им се налага да пишат напълно тривиален и безинтересен от математическа гледна точка програмен код. Не може ли „отговорността“ за този тривиален програмен код да бъде поверена на компютрите, а за хората да останат само по-интересните и интелектуално предизвикателни задачи?

На този въпрос може да бъде отговорено двояко. От една страна — да. Съществуват системи, които се справят много добре с генерирането на програмен код. Но на практика — не. Причината е тази, че да обясним на компютъра какво точно искаме да направи дадена програма може да бъде не по-малко досадно, отколкото просто да седнем и да си напишем сами програмата. Въпреки това изследванията за намиране на удобна за използване система за автоматично генериране на код продължават.*

Съвсем по-друг начин стоят нещата при доказателственото програмиране, т.е. тогава, когато се пишат програми, за които искаме да сме сигурни, че не съдържат грешки. До голяма степен именно благодарение на успехите в областта на автоматичното доказателство на теореми доказателственото програмиране буквално през последните няколко години от теоретично академично упражнение се превърна в нещо, което е напълно използваемо на практика. Фирмите, които се занимават с писането на програми без грешки, са установили, че дори и при много големи програми обикновено е достатъчно в екипа да има само един

*Когато такава система бъде измислена, навсякъде ще се търсят логици, а програмистите ще останат без хляб.

математик логик, който да установява коректността на онези места в програмата, чиято коректност не е могла да се докаже автоматично от компютър [1, стр. 376]. Интересно е, че системите за доказателствено програмиране могат да формулират задачите, които не са успели сами да докажат, по такъв начин, че не е нужно математикът логик да умее да програмира и всъщност не е нужно дори да знае за какво служи програмата, от която е възникнала поставената задача.

1.3. Логическо програмиране

Създаването на пролог

През 1958 Франция била разкъсвана от политически и икономически проблеми. Остров Корсика бил завлядян от алжирски военни и имало реална опасност от държавен преврат. В обществото нямало единство, политическата власт била слаба. Тогава в политиката се намесил генерал Шарл дьо Гол, герой от Втората световна война. Генерал Дьо Гол решил, че за да се решат проблемите, френското общество трябвало да бъде по-обединено, а единственият начин това да стане, е да го поведе под патриотични лозунги. Затова той заявил: „Нашата страна пред лицето на другите страни трябва да се стреми към велики цели и да не се прекланя пред никого, защото в противен случай може да се окаже в смъртна опасност“. Франция променила коренно политиката си. Въпреки ненавистта си към комунизма, Дьо Гол изкарал Франия от НАТО и дори започнал политическа заигравка със СССР. Използвайки противопоставянето между тогавашните суперсили САЩ и СССР, Дьо Гол успял да увеличи значително международната значимост на Франция и правото ѝ да води независима политика.

През 1967 пред стохиляден митинг в Монреал Шарл дьо Гол се провикнал: „Да живее свободен Квебек!“. Отношенията между Франция и Квебек достигнали ниво подобно на това, което имат независимите държави. Франция отзовала своя „Генерален консул в град Квебек“ и вместо него назначила „Генерален консул пред правителството на Квебек“. Започнали взаимни визити на френски и квебекски политици. Френските политици редовно нарушавали дипломатическите правила като посещавали Квебек без изобщо да се появят преди това в канадската столица Отава. Разширило се научното сътрудничество между Франция и Квебек. Френски учени се премествали да работят и преподават в Квебек, а квебекски — във Франция.

През 1967 г. в Монреал, Квебек, се преместил и младият френски учен Ален Колмеро. Тук той създал Кю-системите — един от първи-

те формализми за обработка на естествени езици. Колмеро бил научен ръководител на квебекчанина Жан Трудел, който работил в областта на автоматичното доказателство на теореми. През 1970 г. Колмеро поканил в Монреал и току-що дипломиралите се французи Робер Пазеро и Филип Русел. Там те се запознали с компютърната обработка на естествени езици.

През 1970 г. Дьо Гол починал и година по-късно групата около Колмеро заедно с квебекчанина Пазеро се върнала във Франция. Тук те създали система, която можела да води диалог на естествен език, като сама правела логически изводи. Езиковата част била разработена от Колмеро и Пазеро, а за автоматичното правене на логически изводи отговаряли Трудел и Русел. Още през същата година системата била достатъчно развита, за да води следния диалог:

Потребител:

Котките убиват мишки.

Том е котка, която не обича мишки, които ядат сирене.

Джери е мишка, която яде сирене.

Макс не е мишка.

Какво прави Том?

Компютър:

Том не обича мишки, които ядат сирене.

Том убива мишки.

Потребител:

Кой е котка?

Компютър:

Том.

Потребител:

Какво яде Джери?

Компютър:

Сирене.

Потребител:

Кой не обича мишките, които ядат сирене?

Компютър:

Том.

Потребител:

Какво яде Том?

Компютър:

> Това, което ядат котките, които не харесват мишки, които ядат сирене.

През това време Русел и Трудел продължили да експериментират с различни методи за автоматично правене на логически изводи. През 1971 Трудел се запознал с току-що измислената от Робърт Ковалски и Доналд Кунър SL-резолюция. Ковалски и Кунър работели в Единбург, но Трудел убедил останалите от групата да поканят Ковалски за кратко посещение в Марсилия, за да им разкаже за метода.

Марсилската група около Колмеро се състояла от хора практици. Те не пропускали случая да се запознаят с нови теоретични резултати, които биха могли да им свършат работа, но собствената им научна дейност не била свързана със създаване на нова математика. От друга страна, Ковалски бил теоретик. Когато прилагаме една теория на практика, винаги се налага да решаваме проблеми, които са от практическо естество и не представляват теоретичен интерес. На математиците теоретици, какъвто бил и Ковалски, не им е безинтересно как се прилагат на практика откритите от тях неща, но много често на тях не им харесва сами да решават практическите проблеми свързани с приложението на теорията. Когато обаче се срещнат добри практици с добри теоретици, винаги всеки научава нещо ново и полезно за себе си. Марсилската група се запознала по-добре със свойствата на SL-резолюцията, а това се оказало решаващо важно за измислянето на пролог. Ковалски пък научил неща за практическото използване на резолюцията, които не знаел по-рано, създал теоретичните основи на логическото програмиране и в крайна сметка точно с това се и прочул. За SL-резолюцията, от която тръгнало всичко, днес малцина си спомнят, а още по-малко са тези, които знаят какво всъщност представлява тя.

През 1972 г. Ковалски бил поканен да посети Марсилия за втори път, този път за по-дълго. По това време групата вече имала по-ясна интуиция как да си представя автоматичното доказване с SL-резолюция като изчислителна процедура. Те знаели как да аксиоматизират прости неща като събиране на числа, конкатенация на списъци, обръщане на списък и т.н. по такъв начин, че SL-резолюцията да намира нужния резултат ефективно.

След заминаването на Ковалски, Колмеро измислил как да елиминира Кю-системите. Всъщност, използвайки открития от Колмеро начин, и днес на пролог можем да анализираме много лесно произволен безконтекстен език. По този начин за пръв път започнала да изглежда възможна идеята, цялата система, която разработвали в Марсилия, да

се реализира, използвайки логика.

Взето било едно много важно решение: оказало се, че за да се „изпрограмират“ нужните неща, било достатъчно да се използват само хорнови дизюнкти. С цел по-добра ефективност и по-ясна процедурна семантика, те взели още едно важно решение: позволили само резолвенти с първия литерал на хорновите дизюнкти. По този начин от чисто практически съображения те открили това, което днес се нарича SLD-резолюция. Те предполагали, че SLD-резолюцията е непълна, но считали, че това е приемлива цена заради по-добрата ефективност. Така в края на 1972 г. се появила първата реализация на езика пролог. На следващата година (1973) Ковалски доказал, че SLD-резолюцията всъщност не е непълна [9], а през 1974 заедно с Мартен ван Емден дефинирал и семантика на логическите програми с неподвижна точка [20].*

Името „пролог“ било измислено от съпругата на Филип Русел и е съкращение от „PROgrammation en LOGique“ (програмиране посредством логика).

Приложенията на пролог

Езикът пролог впечатлява с това по какъв елегантен начин съчетава в себе си мощ и простота. Кой друг език не съдържа нито една запазена дума и при това остава удобен за използване, дори ако го лишим от всички вградени в него предикати, функции и т.н.? На мнозина от първите ентусиасти на пролог им се е струвало, че е само въпрос на време, кога пролог ще стана най-популярният език за програмиране. Някои считали, че други езици за програмиране изобщо не било нужно да се учат.

Разбира се, вече знаем, че тези представи се оказали много далеч от истината. Езикът пролог изобщо не успял да стане толкова популярен, колкото очаквали. Въпреки това има някои видове приложения, при които пролог се е доказал като един от най-удобните съществуващи езици за програмиране.

* * *

Първата група приложения на пролог са свързани с основния тип данни на езика — термовете. Ако ни се налага да извършваме сложни манипулации на дървесни структури данни, то си струва да обмислим, дали да не програмираме на пролог. Компилаторите, и особено оптимизиращите компилатори, са програми, на които им се налага манипулират такива дървесни структури. Първата реализация на езика

*Ван Емден е автор на термина „SLD-резолюция“.

ерланг била написана на пролог. Също и при езика СПАРК важна част от реализацията, която се грижи за коректността на компилираната програма, е написана на пролог.

* * *

Втората група от приложения на пролог е свързана с това, че всяка програма на пролог представлява съвкупност от правила, а в някои случаи е най-удобно да опишем действията на приложението, което искаме да реализираме, именно посредством съвкупност от правила. Ако решим да програмираме една такава програма на традиционен език, бихме получили огромно количество вложени един в друг условни оператори. Логиката на такава програма е трудна за проследаване, а грешките — невъзможни за оправяне.

На пролог е удобно да се реализират т.н. бизнес правила. Така например корпорацията Ериксон с такава система е вземала решения, свързани с продажбите на продукти, а банки и застрахователни компании са оценявали заявленията за предоставяне на кредити и очакваните печалби.

На пролог е написана една от най-популярните система за автоматизация на документи (DealBuilder). Тя може автоматично да генерира текстове на договори и други юридически документи, използвайки сложен набор от правила.

На пролог е написана система, която помага на потребителите да настроят мобилните си телефони. Тя се използва на уеб сайтовете на много мобилни оператори. Ясно е, че не е лесно да се съчетаят в прост алгоритъм специфичните изисквания на всеки един телефон със специфичните изисквания на отделните услуги, предоставяни от оператора.

Около една трета от резервациите на самолетни билети по света се извършва посредством система, написана на пролог.

В Windows NT 3.1 системата, която решава кои драйвери е най-подходящо да се заредят от ядрото, е написана на пролог.

* * *

Третата група от приложения са тези, при които програмата трябва да реагира по сложен начин на огромно количество взаимнозависими фактори.

По данни на Централното разузнавателно управление на САЩ, на пролог е написан софтуерът на руската космическа совалка „Буран“, която за пръв път извършила кацане на летище в напълно автоматичен режим (безпилотно и без радиоуправление).

На космонавтите на Международната космическа станция често им се налага да извършват сложни дейности. При тяхното извършване те трябва да си подсказват с инструкции от таблет, което отвлича внима-

нието им и губи време. Американската космическа агенция НАСА обаче е реализирала на пролог система за гласово управление на браузър. Космонавтът казва на глас каква информация иска от компютъра, а компютърът също му отговаря със синтезиран глас.

Понякога на пролог е удобно да се пишат системи за извличане на ценна информация от огромно количество неструктурирани данни (data mining). Например Американската агенция за електронно разузнаване използва пролог, за да анализира социалните мрежи и да открива хора, за които има вероятност да се превърнат в терористи.

Много от съществуващите експертни системи са написани на пролог. Някои примери за експертни системи, написани на пролог са:

- експертна система, която анализира условията, при които се отглеждат свинете в свинефермите, и предлага промени за начина на хранене на свинете, температурата, влажността и вентилацията на въздуха, използваните породи и т.н. В някои случаи тази експертна система е увеличавала печалбата до пет пъти;
- на пролог са написани няколко експертни системи, които анализират околната среда. Те се използват за прогнозиране на времето, за прогнозиране на предстоящи глобални изменения в климата, за прогнозиране на замърсеността на въздуха в градовете и др.;
- експертна система, която предлага методи за снабдяване на населението с питейна вода в случай на стихийни бедствия, производствени аварии или война;
- самолетостроителната фирма Боинг използва пролог за програма, която снабдява работниците с необходими инструкции. Тези инструкции се получават от различни източници — обикновени файлове, бази данни, обектноориентирани бази данни, други експертни системи. Достъпът до тези източници обикновено е отдалечен (по мрежата), но всичко става невидимо за потребителя. Един от разработчиците казва: „Работата по написването на машина за заявки на пролог бе минимална в сравнение с лисп. А да я напишем на някой обикновен език би било немислимо, защото това на практика би означавало да измислим отново пролог“;
- „изкуственият интелект“ на някои компютърни игри използва пролог.

Логическо програмиране с ограничения

. . .
. . .
. . . *constraint (logic) programming*
. . .
. . .

Други разновидности логическо програмиране

Най-строгата дефиниция на това какво означава „логическо програмиране“ е следната:

Дефиниция. *Логическото програмиране* е метод за описание на компютърни алгоритми, при който:

- компютърната програма представлява логическа задача, формулирана на някакъв формален логически език;
- изпълнението на програмата от компютъра представлява търсене на доказателство на поставената логическа задача.

Например при езика пролог програмата представлява задача от следния вид: ако Γ е множество от хорнови клаузи, а φ е конюнкция от атомарни формули, да се докаже, че от Γ следва изпълнимостта на φ. Методът, който компютърът използва, за да реши тази задача, е SLD-резолюцията.

Ако вместо обикновени хорнови клаузи, използваме друг логически език, получаваме други езици за логическо програмиране.

* * *

Езикът мъркюри е на пръв поглед тривиално разширение на пролог — той се получава просто като добавим към езика типове данни и т.н. режими на предикатите. От логическа гледна точка добавянето на типове не дава на езика по-голяма изразителна сила. Когато обаче става въпрос за програмиране, тази проста добавка има следните две важни следствия. Първо, езикът мъркюри позволява да се пишат изключително бързи програми. Второ, за разлика от пролог, при мъркюри конюнкцията е напълно комутативна, а ако програмата не се зацикля, тогава и дизюнкцията става комутативна и съответствието между програмата и желаната логическа семантика е много по-пълно. Въпреки, че при мъркюри няма нелогически предикати като предиката за отсичане на пролог, на мъркюри може да се пишат далеч по-бързи програми, отколкото на пролог.

* * *

Езикът ламбда-пролог добавя към пролог следните неща:

– Типове данни.

– Явни квантори. Доказателството на $\exists \mathtt{x}\, \mathtt{p}(\mathtt{x})$ се прави по същия начин, както и на пролог (където кванторът $\exists$ се подразбира). Доказателството на $\forall \mathtt{x}\, \mathtt{p}(\mathtt{x})$ се прави посредством *скулемизация* — компютърът добавя нов символ за константа `c`, доказва $\mathtt{p}(\mathtt{c})$ и след това „забравя“ за символа `c`.

– Импликация в целите. Доказателството на $\varphi \Rightarrow \psi$ се прави по следния начин — компютърът добавя към базата знания φ, доказва ψ и след това премахва φ. При програмиране тази възможност е удобно да се използва, за да се намали броят на аргументите на един предикат, както и за модулно програмиране.

– Термове със свързани променливи. Тъй като не е известен логически език или език за програмиране, при който формулите или програмите да не могат да се представят по естествен начин посредством термове със свърани променливи, то ламбда-пролог е вероятно най-удобният програмен език за обработка на формални езици.

Добавките, които ламбда-пролог прави към пролог не са „случайни приумици“, а се базират на интересна логическа теория.

* * *

Има няколко известни езика, които макар и да не отговарят на строгата дефиниция за „логическо програмиране“, дадена по-горе, все пак са силно повлияни от пролог и логическото програмиране.

* * *

Езикът ерланг се използва за писане на разпределени, паралелни програми, които са устойчиви на аварии и работят без да се спират. Много от най-натоварените уеб-сайтове по света използват ерланг за част от услугите, които предоставят. Първата версия на ерланг е реализирана на пролог, а синтаксисът на ерланг е силно повлиян от синтаксиса на пролог.

* * *

Тъй като езикът ес кю ел не използва свързани променливи, а алгебрични операции, той е лесен за реализация, но неудобен за използване. Най-перспективният език със свързани променливи за заявки към бази данни е дейталог. Една заявка на дейталог на пръв поглед прилича на програма на пролог. Нека например в базата данни имаме таблица $\mathtt{оценка}(x, y)$, която казва, че студентът x има оценка y по логическо

програмиране. И нека още имаме таблица `група`(x, y), която казва, че студентът x е от група y. Питаме се в кои административни групи има студенти, получили слаба оценка по логическо програмиране. На дейталог това може да бъде направено така:

```
проблем(Група) :- оценка(Студент,2), група(Студент,Група).
?- проблем(Група).
```

Същата заявка на ес кю ел включва следните операции:

- нека t_1 е таблицата, която се получава от `проблем` като отделим само редовете, в които оценката е слаба;
- нека t_2 е таблицата, която се получава от t_1 като изтрием стълба с оценките;
- нека t_3 е таблицата, която се получава като слеем таблица `група` с таблица t_2, приравнявайки стълбовете със студентите;
- нека t_4 е таблицата, която се получава от t_3 като изтрием стълба със студентите;
- изкарай като отговор таблица t_4.

* * *

Счита се, че не съществува алгоритъм, който винаги успява да решава ефективно задачи, за които е доказано, че са NP-пълни. Едно сравнително скорошно неочаквано откритие е това, че има алгоритми, които макар и не винаги, но в много случаи успяват да решават NP-пълни задачи. Тъй като има много важни за практиката задачи, които са NP-пълни, това откритие има не само теоретична, но и голяма приложна стойност. Реализирани са системи за answer set programming, които в повечето случаи използват езици, които много приличат на пролог.

*
* * *
*

За подобряването на тази глава помогнаха д-р Стефан Вътев, Стефан Герджиков, Соломон Паси, проф. Димитър Скордев, Александра Соскова, проф. Тинко Тинчев и други. Нито един от тях не може по никакъв начин да се счита отговорен за допуснатите грешки. Също така, ни най-малко не може да се счита, че те споделят изцяло или дори част от изказаните в този текст нематематически тези.

Глава 2

Синтаксис

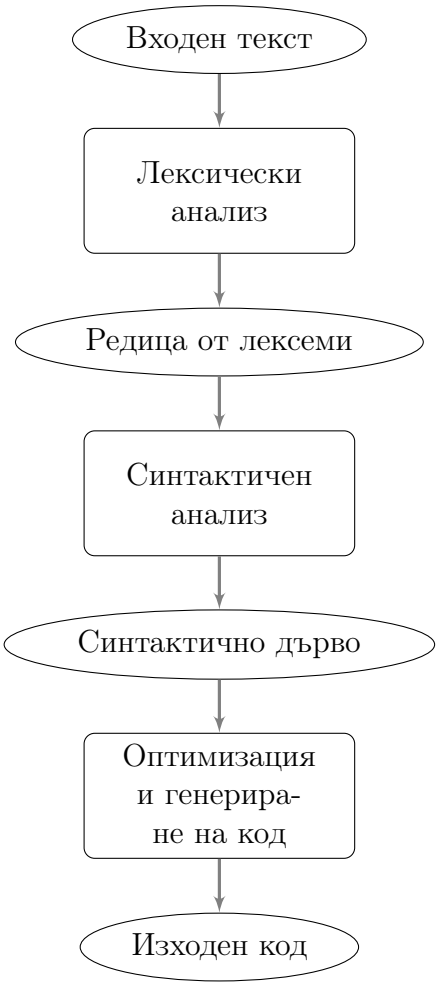
> Онези, които предпочитат небрежния изказ, могат само да се благодарят, ако създателите на реклами използват сръчно речника и синтаксиса, за да заблуждават слабите им умове.
>
> Дороти Сеърз [15]

2.1. Синтактични дървета

Транслаторът е *програма*, която преобразува програма от един формален компютърен език (т.н. *входен език*) на друг формален компютърен език (т.н. *изходен език*). Когато входният език е от високо ниво (напр. ада, си, джава), а изходният — от ниско (машинен език или байт-код), транслаторът се нарича *компилатор*. Когато пък входният език е от ниско ниво, а изходният — машинен език, тогава транслаторът се нарича *асемблер*. Има и транслатори, които превеждат от един език от високо ниво на друг език от високо ниво. Такива транслатори най-често се наричат *конвертори*.

Обикновено транслацията се извършва на няколко стъпки, както това е показано във фигура 1.

Първата стъпка от процеса на транслацията се извършва от т.н. лексически анализатор. На входа му постъпва програмният текст под формата на редица от символи (напр. илюстрирания във фигура 2). След



Фиг. 1. Обичаен строеж на транслаторите

```
function НОД(M, N: Natural) return Natural is
begin
   if N = 0 then
      return M;
   else
      return НОД(N, M mod N);
   end if;
end НОД;
```

Фиг. 2. Примерен програмен текст на ада

function нод (m , n : natural) return natural is begin if n = 0 then return m ; else return нод (n , m mod n) ; end if ; end нод ;

Фиг. 3. Редица от лексеми за програмата на ада

като бъде обработена от лексическия анализатор, програмата се превръща в редица от лексеми. Примерна редица от лексеми, отговаряща на програмата от фиг. 2, е показана във фигура 3.* Както може да се забележи, всяка лексема представлява или дума (напр. идентификатора `Natural` и служебната дума `return`), или литерал (число, низ, и др.), или разделител, или оператор, или коментар. Като основна техника за реализиране на лексическия анализатор се използват крайни автомати. Благодарение на това алгоритмите, използвани от лексическия анализатор са много бързи.**

След като от лексическия анализ получим редица от лексеми, преминаваме към втората стъпка от транслацията — синтактичния анализ. Целта на синтактичния анализатор е да се получи т.н. синтактично дърво. Като основна техника за реализиране на синтактичния анализатор се използват безконтекстни граматики от специален вид. ***

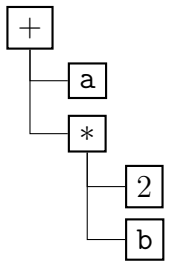
Смисълът на синтактичното дърво е да изрази ясно структурата на израза във вид, удобен за последваща комютърна обработка. Да

*В езика ада не се прави разлика между малки и главни букви. Например „Natural“, „natural“ и „NATURAL“ са един и същи идентификатор. Това е причината, поради която в лексемите от фигура 3 са използвани само малки букви.

**По принцип цялата работа на лексическия анализатор може да се свърши и от синтактичния, така че ако не се интересувахме от бързината на компилатора, то изобщо не би имало нужда от лексически анализатор.

***Алгоритмите, които могат да се използват за синтактичен разбор на произволна безконтекстна граматика са неефективни. Например алгоритъмът на Кок–Янгър–Касами за разбор на безконтекстна граматика, приведена в нормална форма на Чомски, който обикновено се учи в курсовете по дискретна математика, дискретни структури и езици, автомати и изчислимост във ФМИ, е с кубична сложност. Безконтекстните граматики на повечето езици за програмиране обаче имат специални свойства, които позволяват разборът да се извършва за линейно време (макар че дори и в този случай синтактичният анализ е значително по-бавен от лексическия).

Както е известно, крайните автомати разпознават по-тесен клас езици от безконтекстните граматики. Поради това не е възможно да елиминираме нуждата от безконтекстни граматики и да разчитаме изцяло на бързите алгоритми от лексическия анализатор. Форт е един извънредно рядък пример за език от високо ниво, който обаче има регулярна граматика и поради това може да се анализира без да се използват безконтекстни граматики.



Фиг. 4. Синтактично дърво на `a` + 2 ∗ `b`

разгледаме например аритметичния израз „`a` + 2 ∗ `b`“. Не е лесно да се напише директно компютърна програма, която пресмята стойността на такъв израз. Например с какво трябва да съберем `a` — с 2 или 2 ∗ `b`? Само от израза не можем да отговорим на този въпрос, защото от него не личи какъв е приоритетът на различните операции. Нека разгледаме обаче съответното синтактично дърво от фигура 4 — това дърво не оставя никакви съмнения, че `a` се събира с 2 ∗ `b`, а рекурсивният му вид прави алгоритмичното пресмятане на аритметичния израз проста задача.

Във фигура 5 е показано примерно синтактично дърво за програмата от фигура 2.

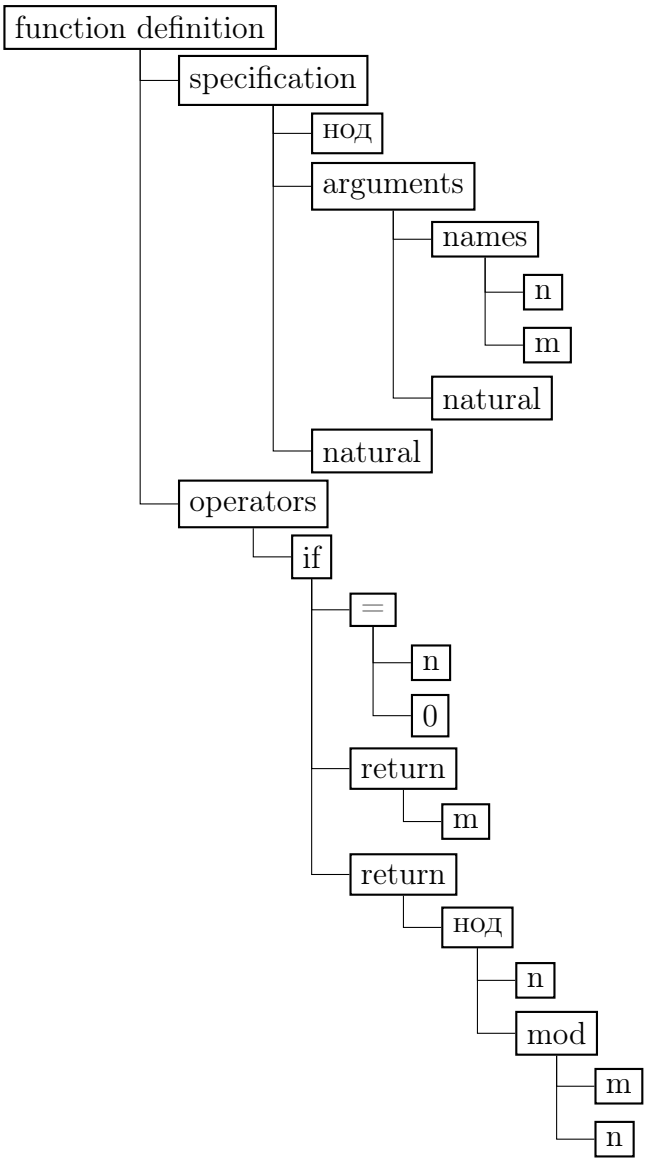
***Задача* 1:** Напишете синтактични дървета за следните изрази на езика си: `x+(y+z)` и `(x+y)+z`. Обърнете внимание, че в си операцията `+` е лявоасоциативна, така че изразите `(x+y)+z` и `x+y+z` имат едно и също синтактично дърво. Същото синтактично дърво имат и изразите `((x+y))+z` и `(x)+y+z`.

***Задача* 2:** Напишете синтактични дървета за следните изрази на езика си: `x+(++y)`, `(x++)+y` и `x+++y`.

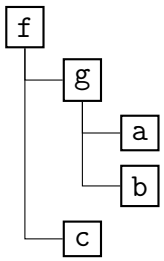
2.2. Термове

Математиците са открили синтактичните дървета много преди да се появят компютрите и хората да започнат да мислят как компютърните програми могат да превеждат едни компютърни програми в други компютърни програми. Само че вместо за синтактични дървета, математиците говорят за *термове*.*

*Гьоте е казал, че математиците са като французите — ти им кажеш едно нещо, а те си го превеждат на собствения език и в крайна сметка се получава нещо съвсем различно. Например когато един дискретен математик говори за „букви“, той всъщ-



Фиг. 5. Примерно синтактично дърво за програмата от фиг. 2



Фиг. 6. Синтактично дърво за терма $\mathtt{f}(\mathtt{g}(\mathtt{a},\mathtt{b}),\mathtt{c})$

Термовете са универсален тип данни. Ако един език за програмиране притежава тип данни „терм“, то той няма нужда от никакви други типове, защото всеки друг дип данни ще може да бъде изразен посредством термове. За това специално свойство на термовете си има специална математическа и по-точно алгебрична причина — оказва се, че алгебрата на термовете е инициален обект в категорията на всички алгебри (единствен с точност до изоморфизъм). В този курс няма да обясняваме какво точно означава горното, но като компенсация ще докажем теоремата на Ербран, от която също ще се види, че термовете в някакъв смисъл са универсални обекти.

Тъй като, записано на хартия, изображението на дърветата заема много място и не винаги е удобно те да бъдат рисувани по средата на изречението, вместо във вид на дървета, математиците изобразяват термовете посредством изрази от специален вид. Ако коренът на едно синтактично дърво е надписан със символа `f`, то неговият терм ще има вида $\mathtt{f}(X_1, X_2, \ldots, X_n)$, където $X_1, X_2, \ldots, X_n$ са термовете, отговарящи на поддърветата, които са непосредствени наследници на корена. Например на синтактичното дърво от фигура 6 отговаря терма $\mathtt{f}(\mathtt{g}(\mathtt{a},\mathtt{b}),\mathtt{c})$, на синтактичното дърво от фигура 4 отговаря терма $+(\mathtt{a},*(2,\mathtt{b}))$, а на голямото синтактично дърво от фигура 5 отговаря следният терм (който очевидно няма как да поберем само на един ред):

```
function_definition(specification(нод,
                                  arguments(names(n,m),natural),
                                  natural),
                    operators(if(=(n,0),
                                 return(n),
                                 return(нод(n,mod(m,n)))
```

ност си мисли за произволни символи. Когато същият математик говори за „думи“, той всъщност си мисли за низове и затова не се плаши от филологически абсурди като „празната дума“.

За удобство, вместо да пишем термове като $+(\mathtt{a}, +(\mathtt{b}, \mathtt{c}))$ и $+(+(\mathtt{a}, \mathtt{b}), \mathtt{c})$, ние ще си позволяваме „волността“ да използваме много по-четливите записи $\mathtt{a} + (\mathtt{b} + \mathtt{c})$ и $(\mathtt{a} + \mathtt{b}) + \mathtt{c}$.*

2.3. Термовете в математиката

> Фразите на един формален език всъщност не представляват редици от символи, а абстрактни обекти, означавани посредством редици от символи, също както естествените числа са абстрактни обекти, означавани посредством редици от цифри. Следователно да дефинираме семантиките посредством функции върху редици от символи е също толкова заобиколно, колкото ако дефинираме аритметичните функции върху редици от цифри.
>
> Джон Рейнолдс [14]

Преди да видим каква трябва да бъде точната математическа дефиниция на *терм*, нека обърнем внимание, че математическите обекти винаги представляват идеализации, които притежават само онези свойства, които са нужни на математиците. Например точките в геометрията имат точно определено местоположение, но нямат нито дължина, нито ширина, нито височина, нито ориентация. Разбира се, в реалния свят няма обекти без размер. Въпреки това, има много ситуации, при които се интересуваме единствено от местоположението на дадени обекти, но не и от техните размери или ориентация. В тези случаи е удобно да считаме, че тези обекти са точки.

Нека забележим, че е погрешно да считаме, че размерите на обектите, отъждествявани с точки, са пренебрежимо малки. Например в определени ситуации един астроном може да счита, че планетите и звездите са точки в пространството. А в други ситуации не можем да приемем за точкови дори частици като протоните и неутроните.** Това, че считаме дадени обекти за точки, няма никакво отношение към размерите на тези обекти, а означава само едно — че не се интересуваме от техните размери и ориентация, а само от местоположението им.

*Но няма да забравяме, че математиците са като французите, така че когато пишем $\mathtt{a} + (\mathtt{b} + \mathtt{c})$ ние всъщност ще имаме предвид терма $+(\mathtt{a}, +(\mathtt{b}, \mathtt{c}))$.

**Например при експерименти с дълбочинно нееластично разсейване се проявяват трите кварка, от които са образувани тези частици.

Всъщност оказва се, че геометрията може да съществува и без в нея да се използват точки.* При това се оказва, че в безточковите геометрии можем да правим съвсем същите неща, които и в обикновените геометрии. При безточковите геометрии обаче се работи по-сложно, защото когато искаме да обозначаваме позиции в пространството не можем за целта да използваме просто точки, а се налага да правим това по по-заобиколен начин.**

Нека сега видим кои са ценните математически свойства на термовете, т.е. кои са свойствата, които е удобно да бъдат взети предвид при тяхната дефиниция.

Преди всичко да забележим, че надписвахме възлите на синтактичните дървета с най-разнообразни текстове — например `f`, `c`, `return`, `function_definition`, $+$, $*$. В нито един случай вътрешната структура на тези надписи не бе от значение. Например не е от значение, че надписът `function_definition` се състои от двете думи `function` и `definition`. Също така без никакво значение е и това, че надписът `return` се състои от шест символа, а не от седем. Затова в математическата дефиниция на *терм* се приема, че всеки един от тези надписи се състои от един единствен символ. В частност ще считаме, че надписите `function_definition` и `return` са съставени от един символ.*** Интересно е, че тази математическа абстракция се оказва полезна не само на теория, но и на практика, защото съответства на начина, по който

*Теорията на безточковите геометрии е разработвана и от много български математици — Николай Белухов, Димитър Вакарелов, Георги Димов, Татяна Иванова, Владислав Ненчев, Тинко Тинчев.

**Квантовата механика ни кара да се запитаме има ли всъщност точки в пространството и не е ли всъщност геометрията на реалния свят безточкова. Засега обаче никой не е разработил безточкова геометрия, която да съответства на света квантовата механика. Всички разработени до момента безточкови геометрии са по един или друг начин еквивалентни на обикновената евклидова геометрия. Регионите, с които се работи в тези геометрии, имат точно определени граници, докато обектите в квантовата механика не притежават точно определени размери, координати и скорост. Тази неопределеност не се дължи на ограничените ни познания, а е фундаментална характеристика на света. Съгласно една теорема на Джон Стюард Бел, ако допълним квантовата механика с допълнителни параметри, които да направят нещата в нея точно определени, то получената теория или няма да дава същите предсказания за поведението на света, както съществуващата квантова механика, или ще трябва да допуска предаване на информация със скорост по-бърза от скоростта на светлината. Това е така, дори ако предположим, че тези допълнителни параметри са „скрити“ и недостъпни за нашето познание.

***Разбира се, че в действителност `return` не се състои от един символ, а от шест. Това обаче за нашите цели ще бъде без значение. Също и планетите не са точки, но понякога астрономът може да си мисли, че са.

се правят транслаторите. Да забележим, че лексическият анализатор превръща израза `return` в *лексема*, след което синтактичният анализатор работи с тази лексема все едно, че тя представлява един единствен символ.

Да разгледаме отново терма $\mathtt{f}(\mathtt{g}(\mathtt{a},\mathtt{b}),\mathtt{c})$; неговото дърво бе изобразено във фигура 6. В този терм надписите $\mathtt{f}$ и $\mathtt{g}$, които наподобяват извиквания на функции с аргументи, отговарят на възли в синтактичното дърво, които не са листа. Математиците наричат такива символи *функционални символи*. Символите пък $\mathtt{a}$, $\mathtt{b}$ и $\mathtt{c}$, които в синтактичното дърво са листа, се наричат *символи за константи*, *индивидни константи* или просто *константи*.

Забележка: Наименованието „функционален символ“ е избрано не само защото участията на функционалните символи в термовете наподобяват извиквания на функции с аргументи. Всъщност връзката между функционални символи и функции е много по-дълбока. По-нататък ще дефинираме понятието *структура* и тогава ще стане ясно, че може да считаме, че функционалните символи са означения на функции (макар самите те да не са функции). Също и символите за константи са означения на константи. Например символът π може да бъде означение за числото $3,141\,592\,653\,589\,793\ldots$, но все пак π не е число, а символ и по-точно буква от гръцката азбука.

За краткост често вместо термина „символ за константа“ се използва по-краткото „константа“. Това е възможно, тъй като в контекста на логическото програмиране и математическата логика истински константи почти няма и значи винаги когато кажем, че нещо е „константа“, е ясно, че всъщност искаме да кажем, че то е „символ за константа“. В това отношение функционалните символи са онеправдани, защото в логическото програмиране се използват функции и затова не можем вместо „функционален символ“ да казваме накратко „функция“.*

В много програмни езици имената на функции, методи, процедури и т.н. могат да се претоварват. Това означава, че на две напълно различни функции могат да се дадат едни и същи имена, стига начинът, по който функциите се извикват, да ни позволява да различаваме коя точно функция се има предвид. Например на пролог едноименни предикати с различен брой аргументи се считат за напълно различни предикати. На си++ не само броят на аргументите на една функция

*На изпита по логическо програмиране се счита за груба грешка, ако за нещо, което е функционален символ, се каже че е функция.

или метод е от значение, но също и типовете на аргументите. А при ада дори типът на стойността е от значение.

Да забележим обаче, че претоварването, макар понякога да е удобно когато даваме имена, всъщност не добавя никакви съществено нови възможности към езика за програмиране. Ако в даден език за програмиране претоварването е забранено, просто можем да даваме различни имена на различните функции, методи и т.н. И при термовете ситуацията е аналогична. Няма проблем да позволяваме един функционален символ да се използва с различен брой аргументи, но това няма да добави никакви съществено нови възможности. Това е така, защото вместо да използваме едновременно термове $\mathtt{f}(\mathtt{a},\mathtt{b})$ и $\mathtt{f}(\mathtt{a},\mathtt{b},\mathtt{c})$, винаги можем вместо това да използваме напр. $\mathtt{f}_2(\mathtt{a},\mathtt{b})$ и $\mathtt{f}_3(\mathtt{a},\mathtt{b},\mathtt{c})$, т.е. при различен брой аргументи да използваме различни функционални символи (в случая $\mathtt{f}_2$ и $\mathtt{f}_3$). И тъй като това леко ще облекчи математическото използване на термовете, ние ще постъпим точно така — ще приемем, че всеки функционален символ има фиксирана *арност*, която казва с колко аргумента той се използва. Впрочем и при програмирането претоварването не винаги е желателно и затова някои езици за програмиране умишлено го забраняват. Например в подмножеството на ада, което се използва за писане на програми без грешки,[*] претоварването е забранено.

Някои математици считат, че символите за константи всъщност са специален вид функционални символи, а именно — функционални символи с арност 0, т.е. с нула аргументи. При други математици пък функционалните символи задължително имат арност поне 1 и символите за константи не се считат за функционални.

За константите (или функционалните символи с арност 0) казваме, че са *нулместни* или *нуларни*. За функционалните символи с арност 1 казваме, че са *едноместни* или *унарни*. За функционалните символи с арност 2 казваме, че са *двуместни* или *бинарни*. За функционалните символи с арност 3 казваме, че са *триместни* или *тернарни*.

[*] Става въпрос за SPARK. Да — има технологии, позволяващи писането на програми (почти) без грешки. В някои случаи писането на програми без грешки просто е необходимост. Например и най-малката грешка в компютърна програма, управляваща пътнически самолет, може да се окаже фатална. На много места има метро, в което влаковете не се управляват от човек, а автоматично от компютър — тук също не искаме управляващата програма да има грешки. Или да си представим програма, управляваща медицински скенер, където една излишна нула може да облъчи пациента с фатална доза радиация. Една грешка в програма за електронен подпис може да вкара невинен човек в затвора. Компютъризацията в обществото непрекъснато расте, а с това се увеличава и броят на приложенията, при които е нужно програмите да бъдат без грешки.

За различни нужди ще се оказва полезно освен функционални символи и символи за константи да можем да включваме в термовете и специални служебни символи, които ще наричаме променливи. Например ако $\mathtt{x}$ е променлива, $\mathtt{c}$ е символ за константа, а $\mathtt{f}$—двуместен функционален символ, то $\mathtt{f}(\mathtt{c},\mathtt{x})$ е терм, съдържащ променливата $\mathtt{x}$. В някои отношения термовете, съдържащи променливи са „втора категория“ термове.

2.1. Уговорка. За да не се налага винаги изрично да се уточнява кой символ е константа, кой функционален и кой променлива, математиците използват следните уговорки:

- за променливи се използват буквите $\mathtt{x},\mathtt{y},\mathtt{z},\mathtt{t},\mathtt{u},\mathtt{v},\mathtt{w}$;
- буквите $\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e}$ са символи за константи;
- за функционални (не нулместни) символи се използват буквите $\mathtt{f},\mathtt{g},\mathtt{h}$, а при нужда още и $\mathtt{j},\mathtt{k},\mathtt{l}$.

Ако трябват повече символи, се използват индекси, примове и т.н., напр. $\mathtt{x}$, $\mathtt{x}'$, $\mathtt{x}_3$ и $\mathtt{x}''_2$ са променливи.

Благодарение на тази уговорка, когато видим някой терм, например $\mathtt{f}(\mathtt{g}(\mathtt{c}),\mathtt{x},\mathtt{b})$, е ясно, че в него $\mathtt{f}$ е триместен функционален символ, $\mathtt{g}$ е едноместен функционален символ, $\mathtt{c}$ и $\mathtt{b}$ са символи за константи и $\mathtt{x}$ е променлива.

***Задача* 3:** Открийте функционалните символи, константите и променливите в терма $\mathtt{f}(\mathtt{f}(\mathtt{c},\mathtt{a}),\mathtt{f}(\mathtt{g}(\mathtt{g}(\mathtt{x})),\mathtt{h}(\mathtt{y})))$. За всеки от функционалните символи определете неговата арност.

***Задача* 4:** Кои от следните изрази не са термове и защо: $\mathtt{f}(\mathtt{c})$, $\mathtt{c}(\mathtt{f})$, $\mathtt{c}$, $\mathtt{f}$, $\mathtt{x}(\mathtt{c})$, $\mathtt{f}(\mathtt{f}(\mathtt{x}))$, $\mathtt{f}(\mathtt{f}(\mathtt{f}(\mathtt{c})))$, $\mathtt{f}(\mathtt{f}(\mathtt{f}(\mathtt{c},\mathtt{c})))$, $\mathtt{g}(\mathtt{g}(\mathtt{x},\mathtt{x}),\mathtt{g}(\mathtt{x},\mathtt{y}))$.

***Задача* 5:** Съществуват ли термове:

- Които съдържат скоби, но не съдържат запетаи?
- Които съдържат запетаи, но не съдържат скоби?
- В които броят на левите скоби е по-голям от броя на десните скоби?

***Задача* 6:** Напишете терм, който съдържа 2 скоби и 6 запетаи. Напишете терм, който съдържа 6 скоби и 2 запетаи.

***Задача* 7:** Колко скоби се съдържат в терма $\mathtt{c}+\mathtt{x}*\mathtt{a}$?*

* *Упътване:* спомнете си, че математиците са като французите.

2.4. Сигнатури

В много случаи това, което вече бе казано за употребата на термове в математиката и особено уговорка 2.1, се оказва достатъчно. Понякога обаче е нужно да имаме по-голяма свобода когато определяме кои са функционалните символи и константите. Например, ако искаме термовете да моделират аритметични изрази на си, е подходящо да поискаме да имаме функционален символ `%`. От друга страна, при пролог символът `%` се използва за слагане на коментари и няма смисъл той да бъде функционален символ при моделиране на изрази на пролог.

За да излезем от това затруднение, ще дефинираме понятието сигнатура или език. Сигнатурата/езикът е математически обект, който ще определя кои точно са функционалните символи, символите за константите и др. По такъв начин става възможно да дефинираме различни сигнатури/езици, например една, $\mathfrak{sig}_1$, за изрази на си и друга, $\mathfrak{sig}_2$, за изрази на пролог. Когато искаме термът τ да бъде терм, отговарящ на изразите на си, ще казваме, че τ е терм при сигнатура/език $\mathfrak{sig}_1$ и когато искаме термът τ да бъде терм, отговарящ на изразите на пролог, ще казваме, че τ е терм при сигнатура/език $\mathfrak{sig}_2$.

2.2. ***Забележка*:** Терминът сигнатура се използва в математическата логика, алгебрата, теория на езиците за програмиране и информатиката. Понякога в математическата логика и алгебрата вместо за сигнатура се говори за „език“. В универсалната алгебра сигнатурите понякога се наричат „типове“. В теория на моделите пък се използва терминът „речник“.

Ще дефинираме сигнатурата да бъде функция. Аргументът на тази функция ще бъде символ, за който искаме да разберем дали е променлива, константа, и т.н., а стойността ѝ ще ни казва какъв точно е символът, даден като аргумент. Например ако $\mathfrak{sig}(\xi) = \texttt{пром}$, то значи символът ξ е променлива от сигнатурата $\mathfrak{sig}$.

2.3. Дефиниция. а) За символите лява и дясна скоба, запетая, &, $\vee$, $\neg$, $\Rightarrow$ и $\Leftrightarrow$ ще казваме, че са *запазени*.

б) Функцията $\mathfrak{sig}$ е *сигнатура* или *език*, ако дефиниционната ѝ област включва само символи, които не са запазени и има безброй много символи `x`, за които $\mathfrak{sig}(\texttt{x}) = \texttt{пром}$.

в) Символът `x` е *променлива* от сигнатурата $\mathfrak{sig}$, ако $\mathfrak{sig}(\texttt{x}) = \texttt{пром}$.

г) Символът `c` е *символ за константа* (или нулместен функционален символ) от сигнатурата $\mathfrak{sig}$, ако $\mathfrak{sig}(\texttt{c}) = \texttt{конст}$.

д) Нека $n \geqq 1$. Символът $\mathtt{f}$ е n-местен *функционален символ* от сигнатурата $\mathfrak{sig}$, ако $\mathfrak{sig}(\mathtt{f}) = (\mathtt{функсим}, n)$.

Да забележим, че когато символът $\mathtt{f}$ е функционален символ, не е достатъчно за сигнатурата да ни каже само това, защото трябва от някъде да разберем и каква е арността на $\mathtt{f}$. Затова когато $\mathtt{f}$ е функционален символ, ще поискаме $\mathfrak{sig}(\mathtt{f})$ да бъде наредена двойка, чийто пръв елемент ще бъде думата $\mathtt{функсим}$, а вторият елемент ще ни дава арността на $\mathtt{f}$.

Ограничението сигнатурата да не е дефинирана за запазените символи се дължи на това, че ще използваме тези символи за специални цели и няма да бъде удобно да ги използваме и като функционални символи, константи или променливи.

Да забележим, че условието в дефиниция 2.3, според което има безброй много символи $\mathtt{x}$, за които $\mathfrak{sig}(\mathtt{x}) = \mathtt{пром}$, ни гарантира, че винаги съществуват безброй много променливи. В доказателството на някои твърдения се налага да се използват изречения от следния вид: „Нека $\mathtt{x}$ е произволна променлива, която не се среща в терма τ“. Такива доказателства не би биха били верни, ако се окаже, че термът τ съдържа всички възможни променливи. За да си спестим този проблем, е удобно да поискаме да съществуват безброй много променливи. Така, тъй като в един терм могат да се съдържат само краен брой променливи, със сигурност ще има променливи, които не се съдържат в τ.

Вече сме готови да дадем и дефиницията на терм. Тя ще бъде индуктивна. Това означава, че ако за даден израз успеем да докажем, че е терм, използвайки в доказателството единствено трите правила от дефиниция 2.4, то тогава този израз е терм. Ако пък за даден израз е невъзможно да се докаже, че е терм, използвайки единствено тези три правила, то тогава той не е терм.

2.4. Дефиниция. Нека $\mathfrak{sig}$ е сигнатура. Понятието *терм* при сигнатура $\mathfrak{sig}$ се дефинира индуктивно посредством следните правила:

а) Ако $\mathtt{x}$ е променлива от $\mathfrak{sig}$, то $\mathtt{x}$ е терм при сигнатура $\mathfrak{sig}$.

б) Ако $\mathtt{c}$ е символ за константа от $\mathfrak{sig}$, то $\mathtt{c}$ е терм при сигнатура $\mathfrak{sig}$.

в) Ако $\mathtt{f}$ е n-местен функционален символ от $\mathfrak{sig}$ и $\tau_1, \tau_2, \ldots, \tau_n$ са термове при сигнатура $\mathfrak{sig}$, то изразът $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ е терм при сигнатура $\mathfrak{sig}$.

***Задача* 8:** Нека сигнатурата $\mathfrak{sig}$ е такава, че $\mathtt{x}$ е променлива, $\mathtt{c}$ е символ за константа, а $\mathtt{f}$ — двуместен функционален символ. Използвайки правилата от дефиниция 2.4, докажете, че $\mathtt{f}(\mathtt{c}, \mathtt{f}(\mathtt{c}, \mathtt{x}))$ е терм при сигнатура $\mathfrak{sig}$.

***Задача 9:** Да допуснем, че дефиницията позволяваше специалните символи да се използват като константи и функционални символи. Нека лявата скоба е символ за константа, дясната — едноместен функционален символ, а запетаята — двуместен функционален символ. Открийте функционалните символи и символите за константи в „термовете“, оградени с правоъгълници:

`(` `)(()` `)()(())` `,((,()` `,()((),,((,())`

Един любопитен факт с дълго доказателство е това, че подобни изрази винаги притежават еднозначен прочит — никога не възниква ситуация, при която някоя скоба е възможно да се изтълкува хем като истинска скоба, хем като функционален символ. Това отчасти се използва в езика пролог, в който символът запетая се използва хем като разделител на аргументите на функционалните символи, хем като двуместен функционален символ.

Винаги, когато даваме индуктивна дефиниция на дадено понятие, е в сила съответен принцип за индукция. Сравнете внимателно трите правила от горната дефиниция с трите условия в следния принцип за индукция:

2.5. Принцип за индукция по построението на терм. *Нека* $\mathfrak{sig}$ *е сигнатура и* $\mathfrak{p}$ *е някакво свойство, за което е вярно, че:*

а) ако $\mathtt{x}$ *е променлива от* $\mathfrak{sig}$*, то* $\mathtt{x}$ *притежава свойството* $\mathfrak{p}$*;*

б) ако $\mathtt{c}$ *е символ за константа от* $\mathfrak{sig}$*, то* $\mathtt{c}$ *притежава свойството* $\mathfrak{p}$*;*

в) ако $\mathtt{f}$ *е* n*-местен функционален символ от* $\mathfrak{sig}$ *и* $\tau_1, \tau_2, \ldots, \tau_n$ *са термове при сигнатура* $\mathfrak{sig}$*, които притежават свойството* $\mathfrak{p}$*, то също и* $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ *притежава свойството* $\mathfrak{p}$*.*

В такъв случай всички термове при сигнатура $\mathfrak{sig}$ *притежават свойството* $\mathfrak{p}$*.*

Доказателство. Нека τ е произволен терм при сигнатура $\mathfrak{sig}$.

Тъй като τ е терм, то това може да бъде доказано, използвайки единствено трите правила от дефиниция 2.4. В едно такова доказателство ще се изказват твърдения, че определени изрази са термове при сигнатура $\mathfrak{sig}$. Нека $\sigma_1, \sigma_2, \ldots, \sigma_m$ са всички изрази, за които в това доказателство се казва, че са термове, подредени според срещането им в доказателството. Ще докажем, че термовете $\sigma_1, \sigma_2, \ldots, \sigma_m$ притежават свойството $\mathfrak{p}$. Тъй като в това доказателство със сигурност ще е изказано и твърдението, че τ е терм (т.е. $\tau = \sigma_i$ за някое i), то по този начин ще получим исканото.

Да изберем $i \in \{1, 2, \ldots, m\}$. Ще докажем, че σ_i притежава свойството $\mathfrak{p}$ с пълна математическа индукция по i. Индукционното предположение ще казва, че термовете $\sigma_1, \sigma_2, \ldots, \sigma_{i-1}$ притежават свойството $\mathfrak{p}$.

Тъй като в разглежданото доказателство се позволява да се използват единствено трите правила от дефиниция 2.4, то значи фактът, че σ_i е терм при $\mathfrak{sig}$ е доказан с някое от тези три правила.

Ако този факт е доказан с правило 2.4 а), то значи σ_i е променлива от $\mathfrak{sig}$. В такъв случай от правило 2.5 а) получаваме, че σ_i притежава свойството $\mathfrak{p}$.

Ако този факт е доказан с правило 2.4 б), то значи σ_i е символ за константа от $\mathfrak{sig}$. В такъв случай от правило 2.5 б) получаваме, че σ_i притежава свойството $\mathfrak{p}$.

Остава възможността това да е било доказано с правило 2.4 в). В този случай $\sigma_i = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$, където $\mathtt{f}$ е n-местен функционален символ от $\mathfrak{sig}$, а $\tau_1, \tau_2, \ldots, \tau_n$ са изрази, за които по-рано в разглежданото доказателство вече е било доказано, че са термове при сигнатура $\mathfrak{sig}$ (с други думи всеки от термовете $\tau_1, \tau_2, \ldots, \tau_n$ е равен на някой от термовете $\sigma_1, \sigma_2, \ldots, \sigma_{i-1}$). Съгласно индукционното предположение $\sigma_1, \sigma_2, \ldots, \sigma_{i-1}$ притежават свойството $\mathfrak{p}$, а значи и $\tau_1, \tau_2, \ldots, \tau_n$ притежават свойството $\mathfrak{p}$ и от правило 2.5 в) получаваме, че σ_i притежава свойството $\mathfrak{p}$. ∎

Забележка: Много е важно този принцип за индукция да не се учи наизуст, а човек сам да може да съобразява какво гласи той, помнейки единствено каква е дефиницията на терм. Доказателството му трябва да се разбере, но също не е нужно да се учи.

2.6. Пример. Ще използваме индуктивния принцип за термове, за да докажем, че всички термове съдържат равен брой леви и десни скоби.

Доказателство. а) Ако `x` е променлива, то `x` съдържа равен брой леви и десни скоби (нула), защото променливите не са запазени символи, а скобите са и значи `x` не е скоба.

б) Ако `c` е символ за константа, то `c` съдържа равен брой леви и десни скоби (нула), защото символите за константи не са запазени символи, а скобите са и значи `c` не е скоба.

в) Нека `f` е n-местен функционален символ, да допуснем, че $\tau_1, \tau_2, \ldots, \tau_n$ са термове с равен брой леви и десни скоби.* Нека k_i е броят на левите скоби в τ_i (същото число е равно и на броя на десните скоби). Тъй като `f` не е запазен символ, то `f` не е скоба. Следователно броят на левите

*Това допускане са нарича *индукционно предположение*.

скоби в терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ е $1 + k_1 + k_2 + \cdots + k_n$ — една отворена скоба непосредствено след символа `f` плюс скобите в $\tau_1, \tau_2, \ldots, \tau_n$. Броят на десните скоби е същият — затворената скоба в края на терма плюс скобите в $\tau_1, \tau_2, \ldots, \tau_n$. ■

***Задача* 10:** Нека сигнатурата $\mathfrak{sig}$ е такава, че в нея няма нито един символ за константа. Използвайки индуктивния принцип за термове, докажете, че всеки терм при сигнатура $\mathfrak{sig}$ съдържа поне една променлива.

***Задача* 11:** Нека сигнатурата $\mathfrak{sig}$ е такава, че всички функционални символи от $\mathfrak{sig}$ имат арност 2 (или 0). Използвайки индуктивния принцип за термове, докажете, че във всички термове при сигнатура $\mathfrak{sig}$ броят на запетаите е равен на броя на десните скоби.

***Задача* 12:** Да разгледаме следната индуктивна дефиниция на понятието „буртант“:

1. 5 е буртант;
2. 8 е буртант;
3. Ако n и m са буртанти, то nm^2 е буртант.

Какъв е индуктивният принцип за буртанти? Използвайки го, докажете, че ако n е буртант, то $n + 1$ е естествено число, което се дели на 3 без остатък.

2.5. Термовете в пролог

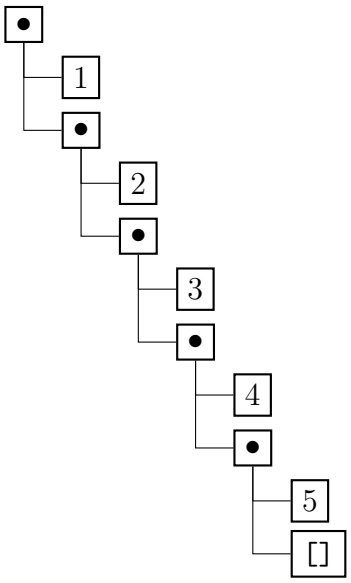
Синтаксис

2.7. Езикът пролог използва синтаксис, подобен на математическия, със следните по-важни различия:

Първо, функционалните символи и символите за константи на пролог могат да се претоварват. Така например $\mathtt{f}(\mathtt{f}(\mathtt{f}, \mathtt{f}))$ е правилен терм. В него първият символ `f` е едноместен, вторият — двуместен, а третият и четвъртият — символи за константи. Въпреки, че са означени по един и същ начин, пролог интерпретира тези символи като напълно различни един от друг.*

Второ, пролог не използва математическата уговорка, съгласно която напр. `b` трябва да бъде символ за константа, а `x` — променлива.

*За избягване на недоразумения, в документацията на пролог арността се обозначава с наклонена черта /. Например `f/2` е двуместният функционален символ `f`, а `c/0` — символът за константа `c`.



Фиг. 7. Синтактично дърво за списъка `[1,2,3,4,5]`

Вместо това, пролог използва следното правило: ако името започва с главна буква, то тогава то е променлива, а ако започва с малка буква, то е символ за константа или функционален символ.* Например `x(C)` е правилно построен терм, в който `x` е едноместен функционален символ, а `C` е променлива.

Трето, както във всеки нормален език за програмиране, имената в пролог могат да бъдат многобуквени. Например `abc(abc,Abc)` е терм, в който първото `abc` е двуместният функционален символ `abc`, второто `abc` е символът за константа `abc`, а `Abc` е променлива.

Списъци

Списъците са една от най-важните структури данни в програмирането. Ако представим списъка `[1,2,3,4,5]` във вид на свързан списък, в паметта на компютъра ще се образува структура, която донякъде наподобява синтактичното дърво, показано във фигура 7. Като се вземе предвид това, че термовете са универсален тип данни, фактът, че можем да използваме синтактични дървета (т.е. термове), за да представяме списъци, никак не е изненадващ.

Следователно ако искаме да използваме списъка `[1,2,3,4,5]` на

*Броят на аргументите показва дали е символ за константа или функционален символ.

пролог, вместо него можем да използваме терма

```
•(1,•(2,•(3,•(4,•(5,[])))))
```

В този терм `[]` е символ за константа, представящ списъка с нула елементи (т.н. *празен списък*),* а • е двуместен функционален символ като смисълът на $\bullet(A, X)$ е списък, чийто пръв елемент е A, а X е списък от всички елементи без първия.

Първият елемент на даден списък се нарича *глава* на списъка, а списъкът от всички елементи без първия — *опашка*. Например 1 е глава на списъка `•(1,•(2,•(3,•(4,•(5,[])))))`, а `•(2,•(3,•(4,•(5,[]))))` — опашка.

Използването на този синтаксис за работа със списъци е възможно, но неудобно. Затова в пролог е предвидено следното улеснение: вместо да пишем списъци като `•(1,•(2,•(3,•(4,•(5,[])))))`, можем да използваме далеч по-четливия запис `[1,2,3,4,5]`. Този начин за записване на списъци обаче не е нищо повече от синтактична захар, тъй като не добавя никакви нови възможности към езика, а само прави програмите по-кратки и по-четливи, улеснявайки по този начин живота на програмистите.

Вместо да пишем $\bullet(\alpha, \chi)$, пролог позволява да използваме записа $[\alpha|\chi]$. Тук α е главата, т.е. първият елемент на $[\alpha|\chi]$, а χ — опашката. Освен това можем да пишем и изрази като $[\alpha_1, \alpha_2, \alpha_3|\chi]$ вместо $\bullet(\alpha_1, \bullet(\alpha_2, \bullet(\alpha_3, \chi)))$. Тук α_1 е първият елемент на $[\alpha_1, \alpha_2, \alpha_3|\chi]$, α_2 — вторият елемент, α_3 — третият, а χ е списъкът от всички елементи от четвъртия нататък.**

Ще казваме, че даден запис на спъсък на пролог е *ненормален*, ако в него скобата `[` се среща непосредствено след вертикална черта `|`. В противен случай записът е *нормален*. Винаги има начин да трансформираме един ненормален запис на списък в нормален. Например списъкът `[1|[2|[]]]` е равен на `[1,2]`, а списъкът `[[1,2]|[3,4]]` е равен на `[[1,2],3,4]`. Нарисувайте синтактични дървете, за да се убедите, че това е така.***

*Тук се вижда разликата между „константа“ и „символ за константа“. Празният списък е константа (в случая нещо, което може би се намира някъде в паметта на компютъра), а `[]` е символ (т.е. означение) за тази константа.

**Ако списъкът съдържа само трите елемента α_1, α_2 и α_3, то χ е празният списък.

***Нормалният запис на списъците е много по-четлив, така че никога няма смисъл да записваме списъците по ненормален начин. Затова, когато на писмения изпит някоя програма на пролог съдържа ненормални списъци, проверяващите започват да стават подозрителни.

***Задача* 13:** Уверете се, че термът

```
•(•(•(1,[]),•(2,[])),•(•(3,•(4,[])),•(5,•(6,[]))))
```

е същият като списъка с ненормален запис

```
[[[1|[]]|[2|[]]]|[[3|[4|[]]]|[5|[6|[]]]]]
```

Нарисувайте синтактично дърво за този списък. Как можем да го запишем по нормален начин? Забележете колко по-четлив е нормалният запис.

Оператори

> Операторите се използват с цел да направят програмите по-четливи. Хората са доста гъвкави и могат да се приспособят към странни среди — например те могат да свикнат да четат програми на лисп и фортран. Въпреки това, ние считаме, че синтаксисът е важен; силата на добрата нотация е добре известна в математиката. Съществена част от хубавия синтаксис на пролог е възможността да се описват и използват оператори.
>
> Лион Стерлинг и Ехуд Шапиро [18]

Да си припомним, че когато математиците напишат терм във вида $3/4+(2-4*5)$, те всъщност имат предвид терма $+(/(3,4),-(2,*(4,5)))$. И пролог прави същото — изразът `3/4+(2-4*5)` ще бъде изтълкуван така, сякаш вместо това сме записали `+(/(3,4),-(2,*(4,5)))`.*

Нещо интересно обаче, което отличава пролог от почти всички останали езици за програмиране, е това, че пролог позволява на програмиста да дефинира и много други оператори. Използвайки тази възможност, на пролог могат да се дефинират езици с доста сложен синтаксис. Например в приложение А е дадена пълната реализация на един императивен език за програмиране, наречен ИМПЕРАТОР, като това е така направено, че ИМПЕРАТОР да стане част от езика пролог. Ето как например може да дефинираме предикат, за пресмятането на факториела на дадено число:

```
% по дадено естествено число N записва в F неговия факториел
факториел(N,F) :- император
```

*Личи си, че пролог е измислен във Франция, нали?

```
f := 1;
for i in 2..N loop
   f := f * i;
F <== f.
```

Дефинирането на нови оператори на пролог става посредством вградения предикат `op`. Например след изпълнението на*

```
:- op(700,xf, [е_животно]).
:- op(700,xfx,[има]).
:- op(600,fy, [бащата_на]).
```

пролог ще интерпретира изразите

```
бащата_на лиско
лиско е_животно
бащата_на лиско е_животно
бащата_на бащата_на лиско има козина
```

все едно, че сме написали термовете

```
бащата_на(лиско)
е_животно(лиско)
е_животно(бащата_на(лиско))
има(бащата_на(бащата_на(лиско)), козина)
```

Тук числото, което даваме като пръв аргумент на `op`, е приоритетът на оператора като колкото по-малко е числото, толкова по-голям е приоритетът. Да забележим, че ако на оператора `бащата_на` дадем приоритет 800 вместо 600, то пролог ще интерпретира третия от горните изрази като

```
бащата_на(е_животно(лиско))
```

което, разбира се, е правилен терм, но лишен от смисъл.

Вторият аргумент на `op` определя типа на оператора. Съществуват следните типове:

fx Едноместни префиксни оператори, в които аргументът не може да има същият приоритет. Например ако дефинираме оператор `вали` от тип `fx`, то изразът `вали дъжд` ще се интерпретира като `вали(дъжд)`, а `вали вали дъжд` ще бъде невалиден израз.

*Не всяка реализация на пролог позволява директното използване на кирилица във функционалните символи, символите за константи и променливите. СУИ-пролог е една от най-популярните реализации на пролог и тя позволява това. Вижте `http://www.swi-prolog.org`. Кирилица може да се използва и при строубери пролог. Вижте `http://dobrev.com`.

оператор	*тип*	*приоритет*
унарни + и -	fy	200
^	xfy	200
* и /	yfx	400
бинарни + и -	yfx	500
= и <	xfx	700
,	xfy	1000
;	xfy	1100
:- и ?-	xfx	1200

Таблица 1. Приоритет и тип на някои от предефинираните оператори на пролог

fy Едноместни префиксни оператори, в които аргументът може да има същият приоритет. Например `бащата_на бащата_на лиско`.

xf и yf Аналогично на `fx` и `fy`, но за постфиксни оператори. Например `лиско е_животно`.

yfx Лявоасоциативни инфиксни оператори. Например изразът `2+3+4` се интерпретира като `(2+3)+4`.

xfy Дясноасоциативни инфиксни оператори. Например изразът `2^3^4` се интерпретира като `2^(3^4)`.

xfx Двуместни инфиксни оператори, при които нито отляво, нито отдясно се позволяват оператори със същия приоритет. Например изразите 2=3=4 и 2<3<4 са недопустими, защото операторите = и < са от тип `xfx`.

При определяне на приоритета на операторите е добре да се имат предвид приоритетите на операторите, които в пролог са предефинирани. По-важните от тях са дадени в таблица 1.

Операторът запетая в пролог е особен, защото се използва хем като инфиксен оператор (като смисълът му е конюнкция), хем като разделител между аргументите. Например ако искате да кажете, че не е вярно, че „5=5 и 3=4“ и напишете израза `not(5=5,3=4)`, пролог ще се оплаче, че не е ясен смисълът на `not`, когато се използва с два аргумента. Проблемът се решава с една двойка допълнителни скоби: `not((5=5,3=4))`.

***Задача* 14:** В пролог `not` е обикновен функционален символ, а не оператор. В следствие на това трябва да пишете `not(5=4)` вместо `not 5=4` и `not((5=5,3=4))` вместо `not (5=5,3=4)`. Обаче, ако дефинирате `not` ка-

то оператор, изразите `not 5=4` и `not (5=5,3=4)` ще станат коректни.* Как можете да направите това?

***Задача* 15:** Дефинирайте на пролог оператори `и`, `или`, `не`, благодарение на които ще можем да програмираме на български например така:

```
ремонт(Кола) :-
   не добри_спирачки(Кола) или
   шумна(Кола) или пуши(Кола) или
   външна_повреда(Кола,Повреда) и не драскотина(Повреда) или
   течове(Кола, Количество) и Количество > 194.8.
```

2.6. Термовете във функционалните езици

Функционалните езици образуват две големи семейства — семейството на диалектите на лисп, към което спада напр. скийм, и семейството на статичнотипизираните функционални езици като хаскел, о-камъл, скала и кложър.

Също както при пролог в общи линии всички неща са термове, така и на лисп всички неща са списъци. За списъците се използва следният синтаксис: $(\alpha_1\ \alpha_2\ \ldots\ \alpha_n)$. С други думи забелязват се следните разлики спрямо синтаксиса, използван от пролог: списъците се заграждат в кръгли, а не в квадратни скоби и между елементите не се поставят запетаи, а интервали.

Въпреки, че лисп няма директна поддръжка за термове, оказва се, че никак не е трудно да използваме списъци, за да записваме термове — това става като вместо терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ използваме списъка $(\mathtt{f}\ \tau_1\ \tau_2\ \ldots\ \tau_n)$. С други думи първият елемент на списъка показва кой е функционалният символ, а следващите елементи са аргументите. Например термът $\mathtt{f}(\mathtt{g}(\mathtt{a}, \mathtt{b}), \mathtt{c})$ може да бъде записан на лисп така:

```
(f (g a b) c)
```

По-нататък ще видим, че променливите — както в математиката, така и в програмирането — биват два вида: свободни и свързани. Всички променливи, за които ставаше дума досега, бяха свободни променливи. В частност променливите, които се съдържат в термове на пролог, са свободни променливи. Термовете във функционалните езици обаче, и в частност при лисп и диалектите му, не могат да съдържат свободни променливи.

*Забележете интервала между `not` и отворената скоба.

За сметка на това обаче диалектите на лисп позволяват да се използват т.н. ламбда-термове, които съдържат свързани променливи. Например на скийм

```
(lambda (f g) (lambda (x) (f (g x))
```

е ламбда-терм, съдържащ трите свързани променливи `f`, `g` и `x`. Този ламбда-терм представя функцията, която взема като аргументи две функции `f` и `g` и връща като стойност тяхната композиция (което, разбира се, също е функция). Математиците[*] записват този ламбда-терм така: $\lambda fg.\lambda x.f(gx)$.

На пролог свързани променливи няма.[**]

* * *

При другата голяма група от функционални езици — тази на статичнотипизираните езици като хаскел, о-камъл, скала и кложър — в следствие на строгата типизация не е много удобно да се хитрува като се използват списъци вместо термове. Поради това тези езици имат по-непосредствена поддръжката за термове, отколкото лисп.

В много от статичнотипизираните функционални езици за термовете се използва същият синтаксис, както и при лисп, с тази разлика, че не е задължително да пишем най-външните кръгли скоби (въпреки това в примерите по-долу най-външните скоби са слагани). Например `(f` α_1 α_2 `...`α_n`)` е термът с функционален символ `f` и аргументи $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Да допуснем, че искаме да разполагаме с функционални символи двуместен `Point`, двуместен `Rectangle`, едноместен `Circle` и триместен `Positioned_shape`.

Смисълът на `(Point` x y`)` е точка с координати (x, y).

Смисълът на `(Rectangle` a b`)` е формата на правоъгълник със страни a и b като позицията на правоъгълника не е фиксирана.

Смисълът на `(Circle` r`)` е формата на окръжност с радиус r като позицията на окръжността не е фиксирана.

Смисълът на `(Positioned_shape` A φ σ`)` е формата σ (която може да бъде правоъгълник или окръжност) поставена в точката A и завъртяна на ъгъл φ.[***]

Например

```
(Positioned_shape (Point 1 5) 30 (Rectangle 3 2))
```

[*] И по-точно Аланзо Чърч през 1932 г.

[**] Свързани променливи има например в езика ламбда-пролог.

[***] Разбира се, да въртим окръжност няма смисъл. Обаче в един по-реалистичен пример освен правоъгълника със сигурност би имало и много други форми, които не са ротационно инвариантни.

представлява правоъгълник със страни 3 и 2, поставен в точка с координати $(1, 5)$ и завъртян на ъгъл 30.*

Преди да можем да дефинираме тези функционални символи, трябва да определим какви типове използват те. Забелязваме, че се използват общо четири типа:

`Float` Т.е. реално число. Този тип почти винаги е вграден в езиците и не е нужно да го дефинираме.

`Point_type` Стойността на функционалния символ `Point` и първият аргумент на `Positioned_shape` има този тип.

`Shape` Стойността на `Rectangle` и `Circle` има този тип.

`Graph_object` Това е типът на стойността на `Positioned_shape`.

След като сме решили кои са функционалните символи и какви са техните типове, можем да ги дефинираме формално. Забележете, че дефиницията на функционалните символи е част от дефиницията на типовете.

Дефиницията на хаскел изглежда така:

```
data Point_type = Point Float Float
data Shape =
   | Rectangle Float Float
   | Circle Float
data Graph_object = Positioned_shape Point_type Float Shape
```

Същата дефиницията на о-камъл изглежда така (на о-камъл типовете се пишат с малки букви):

```
type point_type = Point of float * float
type shape =
   Rectangle of float * float
   | Circle of float
type graph_object =
   Positioned_shape of point_type * float * shape
```

*Каквото и да значи това. Много самолети и ракети са падали заради това, че една част от програмата използва радиани, а друга градуси или едната метри, а другата футове.

Глава 3

Семантика

> Семантиките са странен вид приложна математика;
> тя се интересува от прозорливи дефиниции вместо от трудни теореми.
>
> Джон Рейнолдс (според [19, стр. 3])

3.1. Бази знания

Първите компютърни програми използват прости структури данни, които са заимствани директно от линейната алгебра — числа, вектори и матрици. Първият компилатор на ФОРТРАН от 1957 г. вероятно е и първата компютърна програма, използваща сложни структури данни. Но макар и сложни, тези структури данни все още нямат собствен живот — програмата ги създава, използва ги, но след като програмата приключи работата си, тези структури данни изчезват.

През 60-те години на XX век се появяват първите *бази данни*. При тях данните за пръв път получават независимо съществуване от това на използващите ги програми. След като програмата си свърши работата, базата данни продължава съществуването си, очакваща последващо стартиране на програмата. Всъщност често една база данни може да се използва не от една, а от няколко независими една от друга програми.

Едно характерно свойство на базите данни е тяхната статичност — въпреки, че програмите могат да изтриват, добавят и изменят същест-

вуващите данни, ако някоя програма не извърши някоя от тези операции, базата данни ще си стои без изменения. С други думи макар тук данните да са придобили независимо съществуване от това на програмите, всички изменения в данните се извършват под непосредствения контрол на използващите ги програми.

През 70-те на XX век във връзка с нуждите на т.н. *експертни системи* се появяват първите *бази знания*. Най-съществената разлика между базите данни и базите знания е това, че при последните има възможност да се извеждат нови знания, които не са били част от първоначалната база. Например съпоставяйки сигналите, получени от датчиците за температура и задименост, една база знания може сама направи извода, че в самолетния двигател вероятно има пожар. Нещо повече — една такава база знания може сама да инициира определени действия, като например спиране на притока на гориво към горящия двигател.*

Един от най-важните практически проблеми, свързани с базите знания, е това какво компютърно представяне на знанията да се използва. То трябва да притежава следните свойства:

– да позволява ефективен извод на нови знания;

– да позволява представянето на всички необходими знания;

– да бъде удобен за използване от хора.

За съжаление е невъзможно да бъде удовлетворено всяко едно от тези свойства. Колкото по-разнообразни видове знания могат да бъдат компютърно представени, толкова по-неефективно става извеждането на нови знания от вече наличните. Ако пък ограничим видовете знания, които могат да бъдат представени, тогава значително ще ограничим приложимостта на разработената система.

Нуждата представянето на знанията да бъде удобно за използване не само от компютри, но и от хора още повече усложнява нещата. Когато използваме естествен език, например български, ние приемаме много неща за подразбиращи се. Когато разговарят, хората добавят към чутото определен контекст, който не е директно изказан и всъщност говорещият очаква от слушателя да направи това. Например ако някой турист е бил в Охрид и като се върне каже „Белите кучета в Охрид имаха рунтави опашки“, всеки нормален човек** ще си направи

*А за нас остава да се надяваме, че тази база знания умее да преценява по-добре кога да спре притока на гориво към двигателите, отколкото правят това някои текстообработващи програми, които автоматично превръщат малките букви в главни.

**Разбира се, някои математици, логици, програмисти и други подобни субекти не се включват в категорията „всички нормални хора“.

извода, че не всички кучета в Охрид са имали рунтави опашки, въпреки че този извод е логически неправилен! Компютрите не могат сами да добавят контекст към съжденията и затова знанията в базите знания трябва да се представят изчерпателно, без подразбиращи се неща. За съжаление поради начина, по който мислим, на хората ни е трудно да формулираме знанията си изчерпателно.*

3.2. Правила

Възможността да се извеждат нови знания, от вече наличните, е едно от най-важните свойства на базите знания. Затова в много случаи се оказва от полза, ако изберем такова представяне на знанията, че автоматичното извеждане на нови знания да бъде максимално опростено. Именно такова представяне ни дават т.н. *правила*.

Ето някои примерни правила за експертна система, подпомагаща водач на мотоциклет:**

1. Ако мотоциклетът при движение се тресе или придърпва, то да се заменят свещите.
2. Ако мотоциклетът при движение се тресе или придърпва и свещите са заменени, да се провери дали има свободен път за горивото от карбуратора до двигателя.
3. Ако мотоциклетът при движение се тресе или придърпва, свещите са заменени и няма запушване или теч на гориво между карбуратора и двигателя, да се провери въздуховода до карбуратора.
4. Ако мотоциклетът при движение се тресе или придърпва, свещите са заменени, няма запушване или теч на гориво между карбуратора и двигателя и въздуховодът до карбуратора е наред, да се отиде при техник.
5. Ако е натиснат съединителя и той не задейства, да се регулира пружината на съединителя.
6. Ако пружината на съединителя е регулирана, но при натискане той не задейства, то да се потегли на втора скорост, след което

*Всъщност не е възможно човек да се научи да разсъждава логически правилно. Дори ако той е професор по математическа логика. Това се вижда нагледно например от статията [21] на Юдковски, вж. `https://intelligence.org/files/CognitiveBiases.pdf`. Но въпреки тези общочовешки ограничения, на изпита по логическо програмиране студентите трябва да могат разсъждават логически правилно.

**Тези правила са адаптация на правила от [13].

едновременно се натиска съединителя, дава се газ и се натиска спирачката.

7. Ако при натискане на съединителя, той не задейства, пружината му е регулирана и опитът да се освободи с едновременно прилагане на спирачка и газ е неуспешен, то да се отиде при техник.

Виждаме, че правилата приличат на логически твърдения от следния вид:

$$\text{АКО } \varphi_1 \text{ И } \varphi_2 \text{ И } \ldots \text{ И } \varphi_n, \text{ ТО } \psi$$

Тук $\varphi_1, \varphi_2, \ldots, \varphi_n$ се наричат *предпоставки* или *антецеденти* на правилото, а ψ — негово *заключение*, *консеквент*. В математическата логика се използват два основни начина за формален запис на правила — вертикален и хоризонтален. Вертикалният запис изглежда така:

$$\frac{\varphi_1 \quad \varphi_2 \quad \ldots \quad \varphi_n}{\psi}$$

Хоризонталният запис изглежда така:

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

като често вместо символа $\vdash$ се използват различни други символи, например $\Rightarrow$, $-\!:$, $\to$, $\mapsto$, $\rightarrowtail$.

Когато дадено правило няма предпоставки

$$\frac{}{\psi} \qquad \text{или} \qquad \vdash \psi$$

такова правило ни казва, че ψ е винаги вярно. В математическата логика правилата без предпоставки се наричат *аксиоми*, а в логическото програмиране — *факти*.

3.3. Квазимногообразия и алгебрично програмиране

Аксиоми на равенството

Когато се окаже възможно аксиомите на даден клас от структури да бъдат записани във вид на правила и при това предпоставките и заключенията на тези правила представляват равенства между алгебрични изрази, тогава казваме, че този клас от структури образува *квазимногообразие*.

Равенството е рефлексивно, симетрично и транзитивно. Това може да бъде описано посредством следните правила:

$$\vdash \mathtt{x} = \mathtt{x} \tag{1}$$

$$\mathtt{x} = \mathtt{y} \vdash \mathtt{y} = \mathtt{x} \tag{2}$$

$$\mathtt{x} = \mathtt{y},\ \mathtt{y} = \mathtt{z} \vdash \mathtt{x} = \mathtt{z} \tag{3}$$

В тези правила $\mathtt{x}$, $\mathtt{y}$ и $\mathtt{z}$ са променливи като се подразбира, че на мястото на тези променливи можем да слагаме произволни стойности

Оказва се, че за да опишем напълно свойствата на равенството, е достатъчно към тези три правила да добавим правила, които понякога се наричат *аксиоми за заместването*. За всяка алгебрична операция има по едно такова правило. Например ако имаме две бинарни операции, означени с „$+$“ и „$.$“, и една унарна операция, означена с „$-$“, тогава аксиомите за заместване изглеждат така:

$$\mathtt{x} = \mathtt{x}',\ \mathtt{y} = \mathtt{y}' \vdash \mathtt{x} + \mathtt{y} = \mathtt{x}' + \mathtt{y}' \tag{4}$$

$$\mathtt{x} = \mathtt{x}',\ \mathtt{y} = \mathtt{y}' \vdash \mathtt{x}.\mathtt{y} = \mathtt{x}'.\mathtt{y}' \tag{5}$$

$$\mathtt{x} = \mathtt{x}' \vdash -\mathtt{x} = -\mathtt{x}' \tag{6}$$

3.1. В съвкупност аксиомите, описани посредством правила (1)–(6), се наричат *аксиоми на равенството*.

Моноиди

Да видим как можем да използваме правила, за да опишем аксиомите на две примерни квазимногообразия. Като начало да започнем с нещо по-просто.

Структура с една асоциативна операция (означена по-долу с „$.$“), която е асоциативна и има неутрален елемент (означен по-долу с 1), се нарича *моноид*. С други думи, ако в някоя структура са изпълнени аксиомите за равенството, както и следните аксиоми, то тя е моноид:

$$\vdash \mathtt{x}.(\mathtt{y}.\mathtt{z}) = (\mathtt{x}.\mathtt{y}).\mathtt{z}$$

$$\vdash \mathtt{x}.1 = \mathtt{x}$$

$$\vdash 1.\mathtt{x} = \mathtt{x}$$

Пример. Някои примери за моноиди са:

а) Положителните цели числа с операцията умножение. Неутрален елемент е числото 1.

б) Думите от дадена азбука Σ с операцията конкатенация. Неутрален елемент е празната дума ε. Този моноид се нарича *свободен моноид* над Σ.

в) Всички езици от думи над дадена азбука с операцията конкатенация на езици. Неутрален елемент е $\{\varepsilon\}$.

г) Всички функции, изобразяващи елементите на дадено множество (например естествените числа) в същото множество, с операцията композиция на функции. Неутрален елемент е функцията идентитет.*

Асоциативни операции (а значи и моноиди) се появяват често в задачите, възникващи при програмиране. Благодарение на това много алгоритми могат да се разпаралелят по лесен и безопасен начин. Например ако е необходимо да пресметнем произведението

$$\prod_{i=1}^{8000} k_i$$

на компютър с осемядрен процесор, ние можем да извършим това по следния начин: на първото ядро пресмятаме $\prod i = 1^{1000} k_i$, на второто — $\prod_{i=1001}^{2000} k_i$, на третото — $\prod_{i=2001}^{3000} k_i$ и т.н.** Защо не можем да използваме този метод когато операцията не е асоциативна?

Полупръстени

Полупръстените образуват по-сложно квазимногообразие. Полупръстенът е структура с две операции „+“ и „.“, които са асоциативни

*Да си припомним, че ако $f, g\colon M \to M$ са две функции, то тяхната композиция $f \circ g$ е функцията, за която

$$(f \circ g)(\mu) = f(g(\mu))$$

за всяко $\mu \in M$. Функцията идентитет id е функцията, за която

$$\mathrm{id}(\mu) = \mu$$

за всяко $\mu \in M$.

**Този метод няма да подобри крайното бързодействие, ако пресмятането на произведението може да се извърши бързо и на едно ядро. Ако има вероятност пресмятането на различните частични произведения да отнема различно по продължителност време, трябва да се обмисли използвато по-сложен метод за разпаралеляване, защото иначе може се случи седем от ядрата да си свършат бързо работата, а цялата трудна работа да се падне само на осмото ядро.

и имат неутрални елементи съответно 0 и 1. Операцията „+“ е комутативна. Освен това е в сила дистрибутивен закон за „+“ и „.“, а 0 действа като нулев елемент по отношение на операцията „.“. Ето как всичко това може да бъде изразено с правила:

$$\vdash \mathtt{x} + (\mathtt{y} + \mathtt{z}) = (\mathtt{x} + \mathtt{y}) + \mathtt{z}$$
$$\vdash \mathtt{x} + \mathtt{y} = \mathtt{y} + \mathtt{x}$$
$$\vdash 0 + \mathtt{x} = \mathtt{x}$$
$$\vdash \mathtt{x}.(\mathtt{y}.\mathtt{z}) = (\mathtt{x}.\mathtt{y}).\mathtt{z}$$
$$\vdash 1.\mathtt{x} = \mathtt{x}$$
$$\vdash \mathtt{x}.1 = \mathtt{x}$$
$$\vdash \mathtt{x}.(\mathtt{y} + \mathtt{z}) = (\mathtt{x}.\mathtt{y}) + (\mathtt{x}.\mathtt{z})$$
$$\vdash (\mathtt{y} + \mathtt{z}).\mathtt{x} = (\mathtt{y}.\mathtt{x}) + (\mathtt{z}.\mathtt{x})$$
$$\vdash 0.\mathtt{x} = 0$$
$$\vdash \mathtt{x}.0 = 0$$

Пример. Някои примери за полупръстени са:

а) Естествените числа с операции събиране и умножение.

б) Езиците над дадена азбука като операцията „+“ е обединението на езици, операцията „.“ е конкатенацията на езици, 0 е празният език, а 1 е $\{\varepsilon\}$.

в) Многозначните изчислими функции. Операцията „+“ е обединението на две многозначни функции, операцията „.“ е композицията на многозначни функции, 0 е функцията която никога не връща резултат, а 1 е функцията идентитет. Този полупръстен се използва в теория на изчислимостта и може да се използва за моделиране на алгоритмични изчисления, при които паралелните процеси не комуникират по между си. [10]*

г) Реалните числа заедно с $+\infty$, където дефинираме $\mathtt{x} + \mathtt{y}$ да бъде по-малкото от $\mathtt{x}$ и $\mathtt{y}$, а $\mathtt{x}.\mathtt{y}$ е сборът на $\mathtt{x}$ и $\mathtt{y}$. Неутрален елемент за „+“ е $+\infty$ и неутрален елемент за „.“ е 0. Този полупръстен

*За съжаление все още не е открит и може би не съществува хубав математически формализъм, който може да се използва за моделиране на комуникиращи паралелни процеси. Такива процеси може да се моделират например посредством *π-смятането* и *джойн-смятането*, но и двете имат много лоши алгебрични свойства.

се нарича *тропически полупръстен*.* Той има различни приложения, едно от които е свързано с анализ на дискретни системи, т.е. система, чието поведение не е „гладко“ и затова не може да се опише посредством диференциални уравнения. Примери за дискретни системи са транспортните мрежи, комуникационните и компютърните мрежи, централните процесори на компютрите, производствените цехове в заводите и др.

Накрая да забележим, че от всички написани до момента аксиоми единствено при аксиомите на равенството правилата имаха предпоставки. Когато това се случи, казваме, че съответните понятия са не просто квазимногообразие, а *многообразие*.** Например моноидите образуват многообразие и полупръстените също образуват многообразие.

Моноидите и полупръстените представляват примери за т.н. едносортни структури. При едносортните структури всички обекти са от един и същи тип — в случая това са елементите на моноида или полупръстена. Има обаче и многосортни структури. Например линейните пространства са пример за двусортна структира, в която има два вида обекти — скалари и вектори. Например една от аксиомите за линейно пространство правилото

$$\vdash (\mathtt{a}.\mathtt{b}).\mathtt{x} = \mathtt{a}.(\mathtt{b}.\mathtt{x})$$

в което стойностите на `a` и `b` са всевъзможните скалари в структурата (напр. реални числа), а стойностите на `x` — всевъзможните вектори в структурата.

Алгебрично програмиране

Има една група езици за формална спецификация и програмиране, при които дефинираме различните типове данни аксиоматично — посредством задаване на свойствата, удовлетворявани от обектите от

*Приложната математика обикновено използва математически обекти, които са били дефинирани от математици теоретици, които изобщо не са се интересували, че с измислените от тях понятия в бъдеще ще може да се решават практически задачи. Тропическият полупръстен е рядък пример за интересен математически обект, който е бил откриван и преоткриван много пъти от математици приложници и едва в последствие е заинтересувал и математиците теоретици.

**На английски *variety*. В алгебричната геометрия има алгебрични многообразия (algebraic varieties), които са различни от многообразията, за които говорим тук. Освен това в диференциалната геометрия на английски съществува терминът *manifold*, който няма нищо общо с variety, но на български също се превежда като „многообразие“.

съответните типове. Тъй като аксиомите, описващи свойствата на типовете, се задават посредством правила, то от тук следва, че алгебричното програмиране се основава на използването на многосортни квазимногообразия. Мауде* е един от популярните езици за алгебрично програмиране. Ето как изглежда на мауде описанието на квазимногообразието на моноидите:

```
fth MONOID is
   sort Elt .
   op 1 : → Elt .
   op _*_ : Elt Elt → Elt [assoc id: 1] .
endfth
```

Тук дефинираме *сорт* (т.е. тип данни) `Elt` и две операции `1` и `*`. Операцията `1` е просто елемент на моноида, а операцията `*` е двуместна, асоциативна и притежаваща `1` като неутрален (единичен) елемент.

Полупръстените могат да бъдат дефинирани на мауде по следния начин:

```
fth SEMIRING is
   sort Elt .
   op 0 : → Elt .
   op _+_ : Elt Elt → Elt [assoc comm id: 0] .
   op 1 : → Elt .
   op _*_ : Elt Elt → Elt [assoc id: 1] .
   vars x y z : Elt .
   eq x * (y + z) = (x * y) + (x * z) [nonexec] .
   eq (x + y) * z = (x * z) + (y * z) [nonexec] .
endfth
```

Тук елементът 0 е описан като неутрален за асоциативната и комутативната операция +, елементът 1 като неутрален за асоциативната *, а дистрибутивните закони са описани с явно зададени аксиоми. Атрибутът `nonexec` казва на мауде, че не е нужно да обръща внимание на аксиомите за дистрибутивност докато пресмята стойността на операциите + и *.

От математическа гледна точка езиците за алгебрично програмиране се основават на съществуването на т.н. *инициални структури*, които са единствени с точност до изоморфизъм. Съществуването на инициална структура за всяко многообразие е доказано от Герет Биркхоф [4], а за квазимногообразията — от Анатолий Иванович Малцев [22, 23, стр. 271].

*`http://maude.cs.uiuc.edu`

Естествено възниква въпросът каква част от структурите, които бихме искали да използваме при програмирането, могат да се опишат като инициални структури на някое квазимногообразие. Йоханес Бергстра и Джон Тъкер са доказали [2, 3], че ако е възможно операциите на една структура да се пресмятат алгоритмично,* то тогава тази структура може да се опише аксиоматично не само като инициална структура на квазимногообразие, но също и чрез многообразие, стига да добавим към структурата краен брой нови операции.

Въпреки, че езиците за алгебрично програмиране наистина могат да се използват за програмиране, тяхното предназначение е друго — като езици за формална спецификация и описание на семантиката на езици за програмиране. Има различни езици за спецификация, но едно ценно свойство, което отличава езиците за алгебрично програмиране от останалите езици за спецификация, е това, че при алгебричните езици самата спецификация ни дава изпълним код. В някои случаи скоростта на изпълнение на такава програма се оказва изненадващо бърза. Например ако запишем на мауде денотационната семантика на езика скийм, то ще получим интерпретатор на скийм, чиято скорост достига до 75% от скоростта на обикновен интепретатор на скийм. И въпреки че ако оставим на мауде да изпълни формалната спецификация на виртуалната машина на джава, ще получим сравнително бавен интерпретатор, да правим това изобщо не е безсмислено. Макар и бавен, този интерпретатор със сигурност отговаря на формалната си спецификация (т.е. не съдържа програмни грешки) и затова може да се използва за безопасно изпълнение на програми, които са потенциално злонамерени (вируси и др.).

3.4. Атомарни формули

Предпоставките и заключението на правилата могат да бъдат най-разнообразни. Понякога те дори не са математически по своя характер. Например предпоставката на едно правило може чете данни от жироскопите в ръката на робот, заключението на друго правило може да бъде инструкция активираща стъпковия електродвигател на ръката. Няма как да предвидим всички възможни видове предпоставки и заключения, които биха се оказали полезни. А това означава, че не сме в състояние да дадем по такъв начин точната математическа дефиниция на това що е правило, че наистина да обхванем всички полезни за практиката видове правила.

*Такива структури се наричат *изчислими* или *рекурсивни*.

От друга страна обаче, за да можем да разсъждаваме правилно за правилата и да изградим тяхната математическа теория, ние имаме нужда от точна математическа дефиниция на това понятие. А това означава, че ще трябва да направим следния компромис: ще се ограничим с правила, чиито предпоставки и заключение имат определен вид, но ще се стремим това да бъде направено по такъв начин, че от една страна да имаме точна дефиниция, а от друга ограничението, което ще наложим, да не е съществено за повечето приложения. Ще наречем тези правила от специален вид *клаузи*.

Да си припомним правило (4) от предния раздел:

$$\mathtt{x} = \mathtt{x}',\ \mathtt{y} = \mathtt{y}' \vdash \mathtt{x} + \mathtt{y} = \mathtt{x}' + \mathtt{y}'$$

Както всяко правило, и то съдържа няколко предпоставки (в случая две) и едно заключение. Предпоставките и заключението в това правило представляват равенства между някакви изрази. Спокойно може да си мислим, че тези изрази са термове. Например, заключението на горното правило има представлява равенство между термовете $\mathtt{x} + \mathtt{y}$ и $\mathtt{x}' + \mathtt{y}'$. В тези термове $\mathtt{x}$, $\mathtt{x}'$, $\mathtt{y}$ и $\mathtt{y}'$ са променливи, а „+“ е двуместен функционален символ.

Ако разгледаме останалите правила от предния раздел, ще забележим, че всъщност при всички тях предпоставките и заключенията представляват равенства между термове. Ще бъде грешка обаче, ако поискаме предпоставките и заключенията на клаузите да бъдат винаги равенства. Ето пример за правило, при което това очевидно не е така и което бихме искали да се включва в нашата дефиниция на това що е клауза:

$$\mathtt{x} < \mathtt{y} \vdash \mathtt{x} + \mathtt{z} < \mathtt{y} + \mathtt{z}$$

За да имаме възможност да използваме по-разнообразни предпоставки и заключения в правилата, ще дефинираме понятието атомарна формула.*

3.2. Дефиниция. Нека е дадена сигнатура $\mathfrak{sig}$.

а) Символът $\mathtt{p}$ е n-местен *предикатен символ* от сигнатурата $\mathfrak{sig}$, ако $n \geqq 1$ и $\mathfrak{sig}(\mathtt{p}) = (\mathtt{предсим}, n)$.

б) Ако $\mathtt{p}$ е n-местен предикатен символ от $\mathfrak{sig}$ и $\tau_1, \tau_2, \ldots, \tau_n$ са термове при сигнатура $\mathfrak{sig}$, то изразът $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е *атомарна формула* при сигнатура $\mathfrak{sig}$.

*Засега нямаме възможност да обясним защо това понятие се нарича така. Това ще направим когато дадем дефиницията на „формула“.

в) *Клаузата* е наредена двойка (Γ, ψ), където Γ е редица от атомарни формули (може и празна), а ψ е атомарна формула.* Клаузата $((\varphi_1, \varphi_2, \ldots, \varphi_n), \psi)$ ще записваме така:

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

или така

$$\psi :- \varphi_1, \varphi_2, \ldots, \varphi_n.$$

Когато $n = 0$ ще пишем „$\vdash \psi$“, но вместо „$\psi :-$.“ ще пишем „ψ.“.

Също както при термовете за удобство се уговорихме да пишем например $\mathtt{a}+\mathtt{b}$ вместо $+(\mathtt{a}, \mathtt{b})$, така и при атомарните формули за удобство ще пишем например $\mathtt{a} = \mathtt{b}$ и $\mathtt{a} < \mathtt{b}$ вместо $=(\mathtt{a}, \mathtt{b})$ и $<(\mathtt{a}, \mathtt{b})$. Тук $=$ и $<$ са двуместни предикатни символи. Не бива обаче да забравяме, че правим това само за собствено удобство, и когато пишем напр. $\mathtt{a} < \mathtt{b}$, всъщност ще имаме предвид атомарната формула $<(\mathtt{a}, \mathtt{b})$.

3.3. Пример. Да забележим, че във всяка от структурите от предния раздел — моноиди и полупръстени — атомарните формули добиват конкретен и ясен смисъл, но този смисъл е различен в различните структури. Да разгледаме например атомарната формула

$$\mathtt{x} + 0 = \mathtt{x}$$

а) В структурата от пример 3.1 а) функционалният символ „+“ се интерпретира като събиране на естествени числа, а символът за константа 0 е просто числото нула. Затова в тази структура горната атомарна формула казва, че ако съберем едно естествено число с нула ще получим същото число.

б) В структурата от пример 3.1 б) функционалният символ „+“ се интерпретира като обединение на езици, а 0 е празният език, т.е. $\varnothing$. Затова в тази структура горната атомарна формула ни казва, че ако обединим някой език с празния език ще получим същия език.

в) В структурата от пример 3.1 в) функционалният символ „+“ се интерпретира като обединение на многозначни функции, а 0 е функцията, която не дава никакъв резултат. Затова в тази структура горната атомарна формула ни казва, че обединението на някоя

*В математическата литература терминът *клауза* се използва за различни понятия със сходно предназначение. Често клаузите, които дефинирахме тук, се наричат *хорнови клаузи*.

многозначна функция с функцията, която никога не дава резултат е същата функция. От гледна точна на теория на изчислимостта това означава, че ако пуснем някой процес $\mathtt{x}$ паралелно с процес, който се зацикля и нищо не прави, това е все едно да пуснем единствено процесът $\mathtt{x}$ (очевидно това предполага математическата идеализация, че разполагаме с безкрайни ресурси и затова не се интересуваме, че зациклящият се процес може да забави изчисленията).

г) В структурата от пример 3.1 г) (тропическия полупръстен) функционалният символ „+“ се интерпретира като операцията „минимум“, а 0 се интерпретира като $+\infty$. Затова в тази структура горната атомарна формула ни казва, че при произволна стойност на $\mathtt{x}$ по-малкият елемент на множеството $\{\mathtt{x}, +\infty\}$ е равен на $\mathtt{x}$.

За да дефинираме понятието атомарна формула, се наложи в 3.2 а) да въведем нов тип символи — предикатни символи. Що се касае конкретно до дефиницията на атомарна формула, просто сравнение на 3.2 б) с 2.4 в) показва, че една атомарна формула $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ прилича по всичко на терм с единствената разлика, че $\mathtt{p}$ не е функционален, а предикатен символ.

Тъй като атомарните формули приличат на термове, може да възникне въпросът защо изобщо имаме нужда от това понятие. Не може ли да считаме символите $=$ и $<$ за функционални, при което $\mathtt{a} = \mathtt{b}$ и $\mathtt{a} < \mathtt{b}$ ще станат термове и няма да имаме нужда от атомарни формули?

Отговорът на този въпрос е следният: ако не се интересуваме от „смисъла“ на термовете, а само от външния им вид, тогава наистина можем да минем и без атомарни формули. Следващият пример обаче показва, че смисълът на термовете и атомарните формули е много различен — за всяка конкретна структура стойността на термовете е елемент на структурата, в която оценяваме стойността на терма, докато стойността на една атомарна формула без значение каква е структурата винаги е твърдение, което в някои случаи може да бъде вярно, а в други невярно. Ненапразно сме дали така дефинициите, че термовете да могат да бъдат аргументи както на функционални, така и на предикатни символи, а атомарните формули да не могат да бъдат аргументи нито на едните, нито на другите.

3.4. Пример. Да разгледаме следните три израза, в които 0 е символ за константа, „+“ е двуместен функционален символ и $=$ е двуместен

предикатен символ:

$$0 + (0 + 0)$$
$$0 = (0 + 0)$$
$$0 + (0 = 0)$$

Първият от тези изрази е терм. Например ако интерпретираме 0 като числото нула, а „+“ като събиране на числа, тогава стойността на този терм е нула. Ако пък интерпретираме 0 като $+\infty$, а „+“ като операцията „минимум“, тогава стойността е $+\infty$. Но въпреки, че стойностите в двата случая са различни видове обекти (напр. първият е число, а вторият не е), и в двата случая стойността на терма е елемент на структурата, спрямо която интерпретираме символите в терма.

Вторият от тези изрази е атомарна формула. Например ако отново интерпретираме 0 като числото нула, а „+“ като събиране на числа, тогава атомарната формула ни казва, че нула е равно на нула, което е истина. Ако пък интерпретираме 0 като числото едно, а „+“ отново като събиране на числа, тогава същата атомарна формула ще ни казва, че едно е равно на две, което е лъжа. Други възможности за една атомарна формула няма — без значение каква е структурата, в която интерпретираме, не може тя хем да не е вярна, хем да не е невярна.

Третият от горните изрази не е нито терм, нито атомарна формула и е безсмисен.*

3.5. Уговорка (продължение на уговорка 2.1). За да не се налага да уточняваме винаги каква е сигнатурата, за предикатни символи в математиката обикновено се използват буквите `p`, `q`, `r`. Например `p`, $\mathtt{p}_1$ и $\mathtt{p}''_2$ са предикатни символи.

***Задача* 16:** Кои от следните изрази са атомарни формули и кои от тях не съдържат променливи: `c`, `f(c)`, `p(c)`, `p(f(f(f(c))))`, `p(f(x))`, `f(p(x))`, `p(p(x))`, `f(f(x))`?

3.6. *Забележка*: В пролог не се прави разлика между символи за константи, функционални символи и предикатни символи, а атомарните формули са просто вид термове. *Клауза* означава приблизително същото, каквото тук дефинирахме. Клаузите с поне една предпоставка се наричат *правила*, а клаузите без предпоставки — *факти*.

*В някои езици за програмиране, напр. си, този израз не е безсмислен (по-точно изразът `0+(0==0)`), но това е така само защото в тези езици равенството не е логическа, а аритметична операция, чиято стойност е числото едно или нула, а не истина или лъжа.

3.5. Структури

В предния раздел видяхме, че термовете и атомарните формули могат да имат различни стойности в различните структури. А в предния раздел разгледахме и редица конкретни примери за структури — моноиди и полупръстени. Все още обаче не сме дали точна математическа дефиниция нито на това що е структура, нито на това как да пресмятаме стойността на терм или атомарна формула в структура.

Да разгледаме следната клауза

$$\mathtt{x} \leqq \mathtt{y} \vdash \mathtt{x.z} \leqq \mathtt{y.z} \tag{7}$$

и да видим каква е нейната стойност в някои конкретни структури.

1. Ако структурата включва реалните числа и символите „$\leqq$“ и „.“ са интерпретирани по обичайния начин като „по-малко или равно“ и умножение, тогава тази клауза не е вярна. Например при $\mathtt{x} = 2$, $\mathtt{y} = 3$ и $\mathtt{z} = -1$ получаваме $2 \leqq 3 \vdash -2 \leqq -3$, което е лъжа.
2. Ако пък вместо за реалните числа структурата е за естествените числа, тази клауза става вярна, защото стойността на $\mathtt{z}$ не може да е отрицателно число и значи при умножение неравенството не може да си смени посоката.
3. Обаче никъде не е казано, че символите „$\leqq$“ и „.“ трябва да се интерпретират по обичайния начин. Ако структурата е като в 2, но символът „$\leqq$“ се интерпретира не като „по-малко или равно“, а като „по-малко“, тогава клаузата отново става невярна, защото при $\mathtt{z} = 0$ заключението ѝ ще бъде $0 < 0$.
4. Но дори да интерпретираме „$\leqq$“ като „строго по-малко“, ако освен това интерпретираме „.“ не като умножение, а като събиране, тогава клаузата отново става вярна.

Разгледаната клауза показват какви свойства трябва да има понятието „структура“. *Първо,* структурата трябва да определя какъв е смисълът на предикатните символи (напр. „$\leqq$“). *Второ,* структурата трябва да определя какъв е смисълът на функционалните символи (напр. „.“). *Трето,* въпреки, че в разгледаната клауза нямаше символи за константи, структурата трябва да определя смисълът и на тези символи (напр. 0 и 1). Да забележим още, че структурата не трябва да дава **конкретни** стойности на променливите (в случая $\mathtt{x}$, $\mathtt{y}$ и $\mathtt{z}$). Вместо това структурата трябва да определя какви са всички **възможни** стойности на променливите, напр. реалните числа, естествените числа и т.н.

3.7. Означение. а) Множеството от всички възможни стойности на променливите в дадена структура $\mathbf{M}$ се означава с $|\mathbf{M}|$ и се нарича *универсум*[*] на структурата.

б) Смисълът или *интерпретацията* на символите за константи, функционалните символи и предикатните символи в дадена структура $\mathbf{M}$ ще означаваме с горен индекс $^{\mathbf{M}}$. Например

- $\mathtt{c}^{\mathbf{M}}$ е интерпретацията на символа за константа $\mathtt{c}$ в структурата $\mathbf{M}$,
- $\mathtt{f}^{\mathbf{M}}$ е интерпретацията на функционалния символ $\mathtt{f}$ в структурата $\mathbf{M}$ и
- $\mathtt{p}^{\mathbf{M}}$ е интерпретацията на предикатния символ $\mathtt{p}$ в структурата $\mathbf{M}$.

Забележка: Някои математици използват за структурите удебелени латински букви — $\mathbf{A}$, $\mathbf{B}$, $\mathbf{M}$, $\mathbf{N}$, други калиграфски букви — $\mathcal{A}$, $\mathcal{B}$, $\mathcal{M}$, $\mathcal{N}$, трети готически — $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{M}$, $\mathfrak{N}$. Универсумът на структурите се означава или с вертикални черти — $|\mathbf{A}|$, $|\mathbf{B}|$, $|\mathbf{M}|$, $|\mathbf{N}|$, или посредством съответните обикновени латински букви — A, B, M, N.

По отношение на това какъв точно може да бъде универсумът на една структура и как трябва да се интерпретират различните символи в нея, да обърнем внимание на следното.

Универсумът е множество. Не се налага да ограничаваме по какъвто и да е начин какво точно съдържа това множесто — елементите му могат да бъдат числа, функции, други множества и т.н. Единственото ограничение, което се налага да приемем, е това универсумът да бъде непразно множество. Причината, поради която налагаме това ограничение, е тази, че когато универсумът е празното множество, се появяват някои странности, които неужно ще усложняват формулировките на някои твърдения и техните доказателства. И тъй като структурите с празен универсум очевидно не могат да бъдат кой знае колко полезни, си струва да си спестим тези усложнения като забраним структурите с празен универсум.

Интерпретацията на символите за константи трябва да бъде елемент на универсума за структурата. Например ако универсумът на структурата $\mathbf{M}$ е множеството на реалните числа (т.е. $|\mathbf{M}| = \mathbb{R}$) и 1 е символ за константа, тогава $1^{\mathbf{M}} \in \mathbb{R}$. Ако пък универсумът е множеството на естествените числа, тогава $1^{\mathbf{M}} \in \mathbb{N}$. И т.н.

[*]То се нарича още и *носител*, *основа*, *област* или *домейн* на структурата. Едва ли в математиката има друга област, където имената и означенията на основните понятия са толкова малко стандартизирани, колкото в математическата логика...

Интерпретацията на един n-местен функционален символ трябва да бъде функция с n аргумента. В по-горния пример това се вижда от интерпретацията на двуместния функционален символ „+“. Както аргументите, така и стойността на функцията са елементи на универсума на структурата.

Интерпретацията на един n-местен предикатен символ също трябва да бъде функция с n аргумента. В по-горния пример това се вижда от интерпретацията на двуместния предикатен символ „$\leqq$“. Пак от по-горния пример се вижда и кое отличава функционалните от предикатните символи — функциите, интерпретиращи функционални символи (напр. $+^{\mathbf{M}}$), връщат стойности от универсума на структурата, докато функциите, интерпретиращи предикатни символи връщат като стойност някое твърдение. Например без значение коя е структурата и как точно в нея се интерпретира предикатният символ „$\leqq$“ и без значение каква е стойността на променливите $\mathtt{x}$ и $\mathtt{y}$, стойността на $\mathtt{x} \leqq \mathtt{y}$ не е елемент на универсума, а конкретно твърдение, за което понякога можем да знаем, че е вярно, в други случаи можем да знаем, че е невярно, а в трети случаи може и все още да не е известно дали то е вярно, или не.

3.8. Дефиниция. Нека X е произволно множество.

а) n-местна *функция* в X означава n-местна функция с аргументи от X и стойност в X.

б) n-местен *предикат* в X означава n-местна функция с аргументи от X, която връща като стойност някое твърдение.

в) n-местна *релация* в X е подмножество на $X^n = \underbrace{X \times X \times \cdots \times X}_{n \text{ пъти}}$.

3.9. *Забележка*: Току що дадената дефиниция прояснява произхода на имената на различните видове символи — символите за константи се интерпретират като конкретни елементи на универсума, т.е. като константи в универсума, функционалните символи се интерпретират като функции в универсума и предикатните символи се интерпретират като предикати в универсума.

Много логици използват за интерпретацията на предикатните символи не n-местни предикати в универсума, а n-местни релации в универсума. Двете понятия обаче са взаимнозаменяеми.

Нека p е произволен n-местен предикат в X. В такъв случай съответната му n-местна релация е множеството

$$R \stackrel{\text{def}}{=} \{\, (x_1, x_2, \ldots, x_n) \mid p(x_1, x_2, \ldots, x_n) \text{ е истина} \,\}$$

И обратно, ако ни е дадена n-местна релация R в X, то на нея ѝ съответства предикатът

$$p(x_1, x_2, \ldots, x_n) \overset{\text{def}}{\longleftrightarrow} (x_1, x_2, \ldots, x_n) \in R$$

Вече сме готови да дадем точната дефиниция на структура.

3.10. Дефиниция. Нека $\mathfrak{sig}$ е сигнатура. Наредената двойка $\mathbf{M} = (\Omega, \mathtt{i})$ е *структура* за $\mathfrak{sig}$, ако Ω е непразно множество, наречено *универсум* на структурата, а $\mathtt{i}$ е функция, наречена *интерпретация*, която притежава изброените по-долу свойства а), б) и в). Множеството Ω се означава с $|\mathbf{M}|$ и за произволен символ $\mathtt{s}$ вместо $\mathtt{i}(\mathtt{s})$ ще пишем $\mathtt{s}^{\mathbf{M}}$.

а) Ако $\mathtt{c}$ е символ за константа, то $\mathtt{c}^{\mathbf{M}} = \mathtt{i}(\mathtt{c})$ е елемент на $|\mathbf{M}|$.

б) Ако $\mathtt{f}$ е n-местен функционален символ, то $\mathtt{f}^{\mathbf{M}} = \mathtt{i}(\mathtt{f})$ е n-местна функция в $|\mathbf{M}|$.

в) Ако $\mathtt{p}$ е n-местен предикатен символ, то $\mathtt{p}^{\mathbf{M}} = \mathtt{i}(\mathtt{p})$ е n-местен предикат в $|\mathbf{M}|$.

Интерпретация на символите за константи, функционалните символи и предикатните символи в структура:

$$\mathtt{c}^{\mathbf{M}} \in |\mathbf{M}|$$
$$\mathtt{f}^{\mathbf{M}} \colon |\mathbf{M}|^n \to |\mathbf{M}|$$
$$\mathtt{p}^{\mathbf{M}} \colon |\mathbf{M}|^n \to \text{твърдение}$$

Когато сигнатурата е такава, че единствен предикатен символ в нея е символът за равенство и структурата е такава, че символът за равенство в нея се интерпретира като равенство, а не като нещо друго, тогава такава структура се нарича *алгебрична структура* или просто *алгебра*. Причината за тази терминология е това, че повечето от структурите, които се използват в алгебрата, са точно такива.*

3.11. Пример. Магмите, полугрупите, моноидите, групите, квазигрупите, полупръстените, пръстените, пръстените на Ли, полетата, алгебрите на Клини, полурешетките, решетките и булевите алгебри са само някои от многото примери за алгебрични структури. Например пръстенът на целите числа е структура, чийто универсум е множеството на целите числа, има два символа за константи 0 и 1, които се интерпретират

*Понякога структурите в алгебрата са снабдени с някаква наредба. Строго погледнато, такива структури не са алгебрични.

стандартно като числото нула и числото едно, един едноместен функционален символ „−“, който се интерпретира като операцията „смяна на знака“, два двуместни функционални символа „+“ и „.“, които се интерпретират като операциите събиране и умножение и един предикатен символ =, който е двуместен и се интерпретира като равенство.

3.12. Пример. Графите са пример за структури, които не са алгебрични. Един *граф* $\mathbf{G}$ може да се мисли като структура, чийто универсум е множеството от върховете на графа, символи за константи и функционални символи няма, а единственият предикатен символ е $\mathtt{p}$, който е двуместен и $\mathtt{p}^{\mathbf{G}}(v_1, v_2)$ казва, че има ребро от върха v_1 до върха v_2.

3.13. Пример. Нека сигнатурата $\mathfrak{sig}$ е такава, че в нея има единствен символ за константа $\mathtt{c}$, единствен функционален символ $\mathtt{f}$, който е двуместен, и единствен предикатен символ $\mathtt{p}$, който също е двуместен. Нека структурата $\mathbf{M}$ за $\mathfrak{sig}$ е с универсум множеството на реалните числа, $\mathtt{c}^{\mathbf{M}} = 3$, $\mathtt{f}^{\mathbf{M}}(x, y) = x + y$ и $\mathtt{p}^{\mathbf{M}}(x, y) \longleftrightarrow x < y$. В такъв случай клаузата

$$\mathtt{p}(\mathtt{c}, \mathtt{x}) \vdash \mathtt{p}(\mathtt{c}, \mathtt{f}(\mathtt{x}, \mathtt{x}))$$

казва, че ако едно реално число x е по-голямо от 3, то $x + x$ също е по-голямо от 3.

***Задача* 17:** Какво казват в структурата от пример 3.13 следните клаузи:

$$\mathtt{p}(\mathtt{x}, \mathtt{y}), \mathtt{p}(\mathtt{y}, \mathtt{z}) \vdash \mathtt{p}(\mathtt{x}, \mathtt{z}) \tag{8}$$

$$\mathtt{p}(\mathtt{x}, \mathtt{y}) \vdash \mathtt{p}(\mathtt{f}(\mathtt{x}, \mathtt{z}), \mathtt{f}(\mathtt{y}, \mathtt{z})) \tag{9}$$

$$\mathtt{p}(\mathtt{c}, \mathtt{c}) \vdash \mathtt{p}(\mathtt{x}, \mathtt{x}) \tag{10}$$

***Задача* 18:** В пример 3.12 видяхме, че графите може да се мислят като специален вид структури. Напишете клауза, която е вярна в един граф тогава и само тогава, когато

1. графът е неориентиран;
2. графът е пълен

3.6. Оценки

В предния раздел дадохме дефиницията на структура, но все още не сме дали точна дефиниция за това какво значи една клауза да бъде вярна в структура.*

*В пример 3.13 разчитахме повече на интуицията си и здравия смисъл, отколкото на някаква точна математическа дефиниция.

Преди да можем да оценяваме клаузите в структура, е нужно да можем да оценяваме атомарните формули в структура. А преди да можем да оценяваме атомарните формули в структура, е нужно да се научим да оценяваме термовете в структура. Термовете обаче не могат да бъдат оценявани в структура, защото е невъзможно да кажем каква е стойността на един терм, съдържащ променливи, ако не знаем какви са конкретните стойности на тези променливи. Например за различни стойности на променливата $\mathtt{x}$ стойността на терма $\mathtt{x}+\mathtt{x}$ най-вероятно е различна. Поради това ще въведем понятието оценка.

3.14. Дефиниция. Нека $\mathbf{M}$ е структура за сигнатурата $\mathfrak{sig}$. *Оценка* в $\mathbf{M}$ е функция v, която съпоставя на всяка променлива от $\mathfrak{sig}$ елемент на универсума на $\mathbf{M}$.

***Задача* 19:** Докажете, че за произволна структура $\mathbf{M}$ съществува поне една оценка в $\mathbf{M}$.

Когато ни е дадена не само структура, но и оценка, не е трудно да дефинираме стойността на терм:

3.15. Дефиниция. Нека $\mathbf{M}$ е структура за сигнатурата $\mathfrak{sig}$ и v е оценка в $\mathbf{M}$. *Стойността* на терм в структурата $\mathbf{M}$ при оценка v се дефинира индуктивно:

- Ако $\mathtt{x}$ е променлива от $\mathfrak{sig}$, то стойността на терма $\mathtt{x}$ е $v(\mathtt{x})$.
- Ако $\mathtt{c}$ е символ за константа от $\mathfrak{sig}$, то стойността на терма $\mathtt{c}$ е $\mathtt{c}^{\mathbf{M}}$.
- Ако $\mathtt{f}$ е n-местен функционален символ от $\mathfrak{sig}$ и $\tau_1, \tau_2, \ldots, \tau_n$ са термове от $\mathfrak{sig}$, то стойността на терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ е $\mathtt{f}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)$, където μ_i е стойността на τ_i в $\mathbf{M}$ при оценка v.

Аналогично можем да дефинираме и стойността на атомарна формула:

3.16. Дефиниция. Нека $\mathbf{M}$ е структура за сигнатурата $\mathfrak{sig}$ и v е оценка в $\mathbf{M}$. Ако $\mathtt{p}$ е n-местен предикатен символ от $\mathfrak{sig}$ и $\tau_1, \tau_2, \ldots, \tau_n$ са термове от $\mathfrak{sig}$, то *стойността* на атомарната формула $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е $\mathtt{p}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)$, където μ_i е стойността на τ_i в $\mathbf{M}$ при оценка v.

3.17. Означение. В алгебрата стойностите на терм τ и формула φ в структура $\mathbf{M}$ при оценка v се означават съответно с $\tau^{\mathbf{M}}[v]$ и $\varphi^{\mathbf{M}}[v]$. За съжаление, въпреки че това понятие е може би по-фундаментално за математическата логика, отколкото за алгебрата, различните логици го означават по различен начин. Тук понякога ще означаваме стойността

при оценка v с $\bar{v}(\tau)$ и $\bar{v}(\varphi)$ без да отбелязваме изрично коя е структурата. Това няма да води до недоразумения, тъй като не може за някоя функция да кажем, че е оценка, без да сме уточнили за коя структура е тази оценка.

3.18. ***Забележка*:** Ако временно означим стойността в структура $\mathbf{M}$ при оценка v на произволен терм τ с $\overline{\tau}$ и на формула φ с $\overline{\varphi}$, тогава предните дефиниции може да се преформулират по следния начин:

- $\overline{\mathtt{x}} = v(\mathtt{x})$ за всяка променлива $\mathtt{x}$;
- $\overline{\mathtt{c}} = \mathtt{c}^{\mathbf{M}}$ за всеки символ за константа $\mathtt{c}$;
- $\overline{\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)} = \mathtt{f}^{\mathbf{M}}(\overline{\tau_1}, \overline{\tau_2}, \ldots, \overline{\tau_n})$ за всеки n-местен функционален символ $\mathtt{f}$ и термове $\tau_1, \tau_2, \ldots, \tau_n$.
- $\overline{\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)} \longleftrightarrow \mathtt{p}^{\mathbf{M}}(\overline{\tau_1}, \overline{\tau_2}, \ldots, \overline{\tau_n})$ за всеки n-местен предикатен символ $\mathtt{p}$ и термове $\tau_1, \tau_2, \ldots, \tau_n$.

Това показва, че пресмятането на терм или атомарна формула представлява вкарване на хоризонталните черти навътре в подизразите като при това заменяме всеки символ за константа $\mathtt{c}$ с $\mathtt{c}^{\mathbf{M}}$, всеки функционален символ $\mathtt{f}$ с $\mathtt{f}^{\mathbf{M}}$, всеки предикатен символ $\mathtt{p}$ с $\mathtt{p}^{\mathbf{M}}$ и всяка променлива $\mathtt{x}$ с $v(\mathtt{x})$. Например

$$\begin{aligned}\overline{\mathtt{p}(\mathtt{f}(\mathtt{x},\mathtt{c}),\mathtt{y})} &\longleftrightarrow \mathtt{p}^{\mathbf{M}}(\overline{\mathtt{f}(\mathtt{x},\mathtt{c})},\overline{\mathtt{y}})\\ &\longleftrightarrow \mathtt{p}^{\mathbf{M}}(\mathtt{f}^{\mathbf{M}}(\overline{\mathtt{x}},\overline{\mathtt{c}}),v(\mathtt{y}))\\ &\longleftrightarrow \mathtt{p}^{\mathbf{M}}(\mathtt{f}^{\mathbf{M}}(v(\mathtt{x}),\mathtt{c}^{\mathbf{M}}),v(\mathtt{y}))\end{aligned}$$

3.19. Пример. Нека сигнатурата $\mathfrak{sig}$ е такава, че в нея има единствен символ за константа $\mathtt{c}$, единствен функционален символ $\mathtt{f}$, който е двуместен, и единствен предикатен символ $\mathtt{p}$, който също е двуместен. Нека структурата $\mathbf{M}$ за $\mathfrak{sig}$ е с универсум множеството на реалните числа, $\mathtt{c}^{\mathbf{M}} = 3$, $\mathtt{f}^{\mathbf{M}}(x, y) = x + y$ и $\mathtt{p}^{\mathbf{M}}(x, y) \longleftrightarrow x < y$. Нека оценката v е такава, че $v(\mathtt{x}) = 35$ и $v(\mathtt{y}) = 13$. Тогава стойността в структурата $\mathbf{M}$ при оценка v на формулата

$$\mathtt{p}(\mathtt{f}(\mathtt{x},\mathtt{c}),\mathtt{y})$$

е

$$\begin{aligned}\mathtt{p}^{\mathbf{M}}(\mathtt{f}^{\mathbf{M}}(v(\mathtt{x}),\mathtt{c}^{\mathbf{M}}),v(\mathtt{y})) &\longleftrightarrow \mathtt{p}^{\mathbf{M}}(\mathtt{f}^{\mathbf{M}}(35,3),13)\\ &\longleftrightarrow \mathtt{p}^{\mathbf{M}}(35+3,13)\\ &\longleftrightarrow 35+3<13 \longleftrightarrow \text{лъжа}\end{aligned}$$

3.20. Дефиниция. а) Когато стойността на някоя атомарна формула при дадени структура $\mathbf{M}$ и оценка v е вярно твърдение, казваме, че атомарната формула е *вярна* в структурата $\mathbf{M}$ при оценка v. Използваме следното означение:

$$\mathbf{M} \vDash \varphi[v]$$

б) Когато стойността на някоя атомарна формула при дадени структура $\mathbf{M}$ и оценка v е невярно твърдение, казваме, че атомарната формула е *невярна* в структурата $\mathbf{M}$ при оценка v. Използваме следното означение:

$$\mathbf{M} \nvDash \varphi[v]$$

***Задача* 20:** Докажете, че формулата φ е невярна в структурата $\mathbf{M}$ при оценка v тогава и само тогава, когато φ не е вярна в структурата $\mathbf{M}$ при оценка v.

3.7. Остойностяващи морфизми

В почти всички раздели на математиката от неоценимо значение са изображения, наречени *морфизми*, при които определени, фундаментални за съответния раздел от математиката свойства, се запазват. Например:

- В линейната алгебра *линейните трансформации* запазват линейните операции: $h(\vec{x}+\vec{y}) = h(\vec{x}) + h(\vec{y})$, $h(a\vec{x}) = ah(\vec{x})$. Когато едно такова изображение е биективно, то се нарича *изоморфизъм*.
- В абстрактната алгебра *хомоморфизмите* запазват операциите в алгебричната структура. Например за хомоморфизмите между полета е вярно, че $h(0) = 0$, $h(1) = 1$, $h(x+y) = h(x) + h(y)$ и $h(xy) = h(x)h(y)$. Когато едно такова изображение е биективно, то се нарича *изоморфизъм*.
- В топологията *непрекъснатите изображения* запазват свойството „допиране“ — ако две множества A и B се допират, то $h(A)$ и $h(B)$ също ще се допират. Когато едно такова изображение е биективно, то се нарича *хомеоморфизъм*.
- В геометрията *гладките функции* запазват гладкостта — ако множеството A е гладко многообразие, то $h(A)$ също ще гладко многообразие (локално). Когато едно такова изображение е биективно, то се нарича *дифеоморфизъм*.

В логиката фундаменталните свойства, които ще искаме да бъдат запазвани от подобни изображения, са свързани със символите за константи, функционалните символи и предикатните символи. В зависимост от това дали тези символи са оценени, или не, различаваме следните видове морфизми $h\colon X \to Y$:

- Когато символите са оценени и в X, и в Y, тогава h се нарича *хомоморфизъм*.
- Когато символите в Y са оценени, а в X не са, тогава h представлява пресмятане стойността на терм или формула.
- Когато символите не са оценени нито в X, нито в Y, тогава h представлява прилагане на субституция.

За хомоморфизми и прилагане на субституции ще говорим по-нататък, а в този раздел ще разгледаме свойствата на остойностяващите морфизми.

3.21. Нека фиксираме една сигнатура. За да не усложняваме формулировките, по-нататък няма да споменаваме тази сигнатура изрично.

3.22. Дефиниция. Казваме, че изображението h е остойностяващ морфизъм в структурата $\mathbf{M}$, ако

- за всеки терм τ, $h(\tau)$ е елемент на универсума на $\mathbf{M}$ и
- за всяка атомарна формула φ, $h(\varphi)$ е твърдение

и освен това са изпълнени следните свойства:

а) $h(\mathtt{c}) = \mathtt{c}^{\mathbf{M}}$ за всеки символ за константа $\mathtt{c}$.

б) $h(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)) = \mathtt{f}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$ за всеки n-местен функционален символ $\mathtt{f}$ и термове $\tau_1, \tau_2, \ldots, \tau_n$.

в) $h(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)) \longleftrightarrow \mathtt{p}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$ за всеки n-местен предикатен символ $\mathtt{p}$ и термове $\tau_1, \tau_2, \ldots, \tau_n$.

Два остойностяващи морфизма h и h' се считат за равни, ако $h(\tau) = h'(\tau)$ за произволен терм τ и $h(\varphi) \longleftrightarrow h'(\varphi)$ за произволна атомарна формула φ.

3.23. Твърдение. *За произволна структура* $\mathbf{M}$ *и оценка* v *в* $\mathbf{M}$ *изображението, което на всеки терм и атомарна формула съпоставя техните стойности в структурата* $\mathbf{M}$ *при оценка* v*, е остойностяващ морфизъм.*

Доказателство. Твърдението става очевидно след като сравним дефиниция 3.22 с дефиниции 3.15 и 3.16. ∎

Едно важно свойство на стойността на терм и формула е това, че стойността не зависи от променливите, които не се срещат в терма или формулата. Например за да пресметнем стойността на терма $\mathtt{x}+(\mathtt{y}+\mathtt{y})$, не е нужно да знаем каква стойност има променливата $\mathtt{z}$. Да докажем този факт.

3.24. Твърдение. *Нека h и h' са остойностяващи морфизми в структурата* $\mathbf{M}$.

а) Ако $h(\mathtt{x}) = h'(\mathtt{x})$ за всяка променлива $\mathtt{x}$, която се среща в терма τ, то $h(\tau) = h'(\tau)$.

б) Ако $h(\mathtt{x}) = h'(\mathtt{x})$ за всяка променлива $\mathtt{x}$, която се среща в атомарната формула φ, то $h(\varphi) \longleftrightarrow h'(\varphi)$.

Доказателство. **(а)** С индукция по терма τ.

Ако $\tau = \mathtt{x}$ е променлива, то $h(\tau) = h'(\tau)$, защото по условие h и h съвпадат за променливи, които се срещат в τ (в случая за $\mathtt{x} = \tau$).

Ако $\tau = \mathtt{c}$ е символ за константа, то както $h(\tau) = h(\mathtt{c})$, така и $h'(\tau) = h'(\mathtt{c})$ е равно на $\mathtt{c}^{\mathbf{M}}$, защото h и h' са остойностяващи морфизми.

Нека $\tau = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$. Понеже h и h' са остойностяващи морфизми, то

$$h(\tau) = \mathtt{f}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$
$$h'(\tau) = \mathtt{f}^{\mathbf{M}}(h'(\tau_1), h'(\tau_2), \ldots, h'(\tau_n))$$

Изразите отдясно на равенствата обаче са равни, защото съгласно индукционното предположение $h(\tau_i) = h'(\tau_i)$.

(б) Нека $\varphi = \mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$. Понеже h и h' са остойностяващи морфизми, то

$$h(\varphi) \longleftrightarrow \mathtt{p}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$
$$h'(\varphi) \longleftrightarrow \mathtt{p}^{\mathbf{M}}(h'(\tau_1), h'(\tau_2), \ldots, h'(\tau_n))$$

Изразите отдясно на еквивалентностите обаче съвпадат, защото в а) доказахме, че $h(\tau_i) = h'(\tau_i)$. ∎

Доказаното твърдение ни показва, че ако $h(\mathtt{x}) = h'(\mathtt{x})$ за всяка променлива $\mathtt{x}$, то h и h' съвпадат. В линейната алгебра се доказва подобно свойство: ако $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$ е базис и линейните трансформации h и h' са такива, че $h(\vec{x}_i) = h'(\vec{x}_i)$, то h съвпада с h'. Това показва, че в нашата теория променливите играят роля подобна на базисът в линейната алгебра. Тази аналогия може да се формализира — както базисът, така

и множеството от променливите са примери за *минимални пораждащи множества*.

От същото твърдение следва още нещо интересно. Ако термът τ не съдържа променливи, то каквито и да са морфизмите h и h', е ясно, че $h(\mathtt{x}) = h(\mathtt{x}')$ за всяка срещаща се в τ променлива. Следователно ако τ не съдържа променливи, то $h(\mathtt{x}) = h(\mathtt{x}')$. Аналогично, ако атомарната формула φ не съдържа променливи, то $h(\varphi) \longleftrightarrow h'(\varphi)$.

***Задача* 21:** 1. Нека термът τ не съдържа нито една променлива. Да се докаже, че за произволна структура $\mathbf{M}$ и оценки v' и v'' в $\mathbf{M}$, стойността на τ в структурата $\mathbf{M}$ при оценка v' е равна на стойността на τ в $\mathbf{M}$ при оценка v''.

2. Нека атомарната формула φ не съдържа нито една променлива. Да се докаже, че за произволна структура $\mathbf{M}$ и оценки v' и v'' в $\mathbf{M}$, стойността на φ в структурата $\mathbf{M}$ при оценка v' е равна на стойността на φ в $\mathbf{M}$ при оценка v''.

3.25. Следствие. *Нека h е остойностяващ морфизъм в структурата $\mathbf{M}$ и оценката v е такава, че $v(\mathtt{x}) = h(\mathtt{x})$ за произволна променлива $\mathtt{x}$. Тогава*

а) за произволен терм τ, $h(\tau)$ е равно на стойността на τ в структурата $\mathbf{M}$ при оценка v;

б) за произволна атомарна формула φ, $h(\varphi)$ е еквивалентно на стойността на φ в структурата $\mathbf{M}$ при оценка v;

в) $h = \bar{v}$.

Доказателство. Да припомним, че $\bar{v}$ е изображението, което на произволен терм или атомарна формула съпоставя техните стойности в структурата $\mathbf{M}$ при оценка v. Съгласно 3.23, $\bar{v}$ е остойностяващ морфизъм. Тъй като $\bar{v}(\mathtt{x}) = v(\mathtt{x}) = h(\mathtt{x})$, то исканото следва от 3.24. ∎

Предното следствие показва, че всеки остойностяващ морфизъм представлява пресмятане на стойността при подходяща оценка.

3.8. Тъждествена вярност и изпълнимост

Обикновено когато пишем някоя клауза ние не си мислим, че променливите в нея имат някакви конкретни стойности. Например клаузата

$$\vdash \mathtt{x} + \mathtt{y} = \mathtt{y} + \mathtt{x}$$

ни казва, че двуместната функция, означена с „+“, е комутативна, т.е. $\mathtt{x}+\mathtt{y} = \mathtt{y}+\mathtt{x}$ при произволни стойности на $\mathtt{x}$ и $\mathtt{y}$. Това показва, че понятието „вярност в структура при оценка“, което вече дефинирахме, не ни дава точно това, което искаме. Например горната клауза ще бъде вярна в някоя структура при определена оценка тогава и само тогава, когато $\mathtt{x}+\mathtt{y} = \mathtt{y}+\mathtt{x}$ е вярно при някакви конкретни стойности на променливите $\mathtt{x}$ и $\mathtt{y}$, определени от оценката, а не при произволни стойности.*

Когато ни е дадена само структура, но не и оценка, можем да различим различни „степени на вярност“ на една формула. Например:

- формулата може да бъде вярна при всяка оценка;
- формулата може да бъде вярна при поне една оценка;
- формулата може да бъде невярна при произволна оценка;

3.26. Дефиниция. а) Атомарната формула φ е *тъждествено вярна* в структурата $\mathbf{M}$, ако φ е вярна в $\mathbf{M}$ при всяка оценка. Използваме следното означение:

$$\mathbf{M} \vDash \varphi$$

б) Атомарната формула φ е *изпълнима* в структурата $\mathbf{M}$, ако съществува оценка, при която φ е вярна в $\mathbf{M}$.

в) Атомарната формула φ е *неизпълнима* в структурата $\mathbf{M}$, ако не съществува оценка, при която φ е вярна в $\mathbf{M}$.

г) Атомарната формула φ е *тъждествено невярна* в структурата $\mathbf{M}$, ако при всяка оценка v в $\mathbf{M}$, формулата φ е невярна в $\mathbf{M}$ при оценка v.

***Задача* 22:** Нека φ е атомарна формула и $\mathbf{M}$ е структура. Докажете, че следните три неща са еквивалентни:

- φ не е изпълнима в $\mathbf{M}$;
- φ е неизпълнима в $\mathbf{M}$;
- φ е тъждествено невярна в $\mathbf{M}$.

***Задача* 23:** Възможно ли е една атомарна формула да не е тъждествено вярна в някоя структура, а пък да бъде изпълнима в същата структура?

*Нека структурата е с универсум положителните реални числа и функционалният символ „+“ е интерпретиран като операцията деление. Тогава горната клауза не е вярна, защото например $2/3 \neq 3/2$. Ако обаче оценката е такава, че $v(\mathtt{x}) = v(\mathtt{y}) = 42$, тогава горната клауза ще бъде вярна при тази оценка, защото $42/42 = 42/42$.

***Задача* 24:** Възможно ли е една атомарна формула да бъде едновременно неизпълнима и тъждествено вярна в някоя структура? Защо при отговора на този въпрос се налага да използваме това, че универсумът на структурите е непразно множество?

Забележка: а) Можем да си позволяваме да изпускаме думата „тъждествено“ от словосъчетанието „тъждествено вярна“ без опасност от недоразумения. Ако кажем, че някоя атомарна формула е „вярна в еди-коя си структура“ без да сме уточнили коя е оценката, е естествено да считаме, че формулата е вярна при произволна оценка.

б) Други термини, които означават същото като „тъждествено вярна“ са *общовалидна* и *валидна*.*

Естествено е когато една атомарна формула не съдържа променливи, нейната вярност да не зависи от оценката по никакъв начин. Това наистина е така и затова за такива формули ще предпочитаме да казваме, че са верни някоя структура, а не че са тъждествено верни..

3.27. Твърдение. *Нека v е оценка в структурата* $\mathbf{M}$*, а атомарната формула φ не съдържа променливи. Тогава следните три са еквивалентни:*

а) φ е (тъждествено) вярна в $\mathbf{M}$*;*

б) φ е изпълнима в $\mathbf{M}$*;*

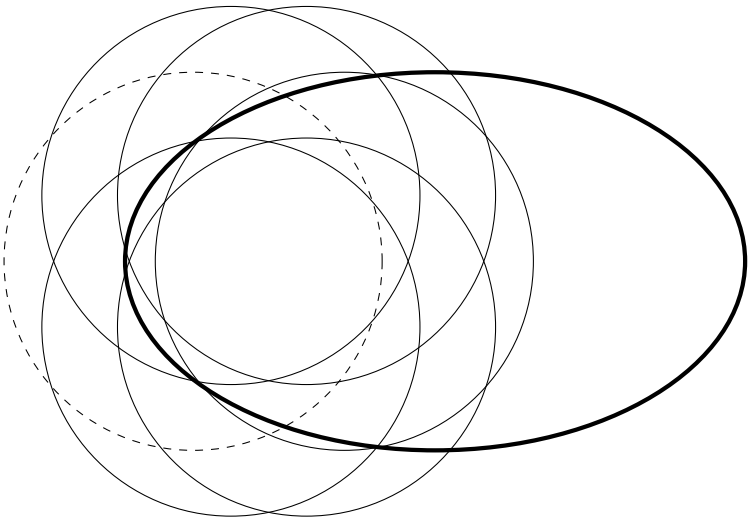
в) φ е вярна в $\mathbf{M}$ *при оценка v.*

Доказателство. Щом φ не съдържа променливи, то каквито и да се оценките v' и v'' в $\mathbf{M}$, те ще съвпадат за променливите, които се съдържат в φ и значи съгласно твърдение 3.24, стойността на φ в $\mathbf{M}$ при оценка v' ще бъде същата, каквато и при оценка v''. С други думи стойността на φ в $\mathbf{M}$ е една и съща при всички оценки. От тук очевидно следва, че да бъде φ тъждествено вярна (т.е. вярна при всички оценки) е същото, каквото да бъде тя вярна при някоя оценка (т.е. изпълнима или вярна при оценка v). ■

3.28. Дефиниция. Клаузата $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ е (тъждествено) вярна в структурата $\mathbf{M}$, ако при всяка оценка, при която предпоставките $\varphi_1, \varphi_2, \ldots, \varphi_n$ са верни, заключението ψ също е вярно. Използваме следното означение:

$$\mathbf{M} \vDash \varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

*На английски universally valid и valid.



Фиг. 8. Диаграма на Вен за вярна клауза

Обърнете внимание, че в общия случай за всяка предпоставка може да има оценки, при които тя е вярна, и оценки, при които тя не е вярна. Също и за заключението може да има оценки, при които то е вярно, и оценки, при които то не е вярно. А за да бъде една клауза тъждествено вярна в структура, ние искаме следното — при онези оценки, при които се окаже, че всички предпоставки са верни, заключението също да бъде вярно. Ако за някоя оценка дори една от предпоставките се окаже невярна, за такава оценка клаузата не казва нищо. Това е илюстрирано във фиг. 8. Всяка от окръжностите изобразява множеството от оценки, при които някоя от предпоставките е вярна. Елипсата с дебела линия пък е множеството от оценки, при които заключението е вярно. Вътрешната бяла област са оценките, при които всяка от предпоставките се оказва вярна и тази бяла област се включва изцяло в удебелената елипса. Забележете, че ако обръщаме внимание не на всички предпоставки, а само на една (напр. на щрихованата окръжност), тогава в общия случай няма да открием никаква зависимост между нея и заключението. Зависимост съществува само когато вземем предвид всички предпоставки едновременно.

3.29. ***Забележка*****:** Когато досега казвахме за някоя клауза, че е вярна в някоя структура, всъщност имахме предвид, че клаузата е тъждествено вярна в структурата. Също както при атомарните формули, така и при клаузите, когато кажем, че някоя клауза е вярна в структура без да уточним коя е оценката, е разумно да считаме, че клаузата е вярна при произволна оценка, т.е. че е тъждествено вярна.

3.30. Дефиниция. Ако формула или клауза φ е тъждествено вярна в структура $\mathbf{M}$ (т.е. $\mathbf{M} \vDash \varphi$), казваме, че $\mathbf{M}$ е модел за φ. Ако всеки елемент на множество Γ от формули или клаузи е тъждествено верен в структура $\mathbf{M}$, казваме, че $\mathbf{M}$ е модел за Γ и пишем $\mathbf{M} \vDash \Gamma$.

3.31. *Забележка*: Понякога в текстове, които не са писани от логици, думата „модел“ се приема за синоним на „структура“. Това е неправилно. Когато кажем, че $\mathbf{M}$ е модел, това означава не само, че $\mathbf{M}$ е структура, но също и че определени (известни може би от контекста) неща са верни в $\mathbf{M}$.

***Задача* 25:** Докажете, че ако някое множество от атомарни формули или клаузи има поне един модел, то то има безброй много модели.

Много полезно се оказва още едно понятие за вярност, при което не само оценката, но и структурата не е фиксирана. За да го дефинираме правилно, нека се запитаме какво всъщност имаме предвид, когато кажем за някоя клауза, че е вярна например във всички полупръстени. Тъй като всеки полупръстен е алгебрична структура от определен вид, то очевидно да бъде една клауза вярна във всички полупръстени трябва да означава, че тя е вярна във всички структури, които са полупръстени.

Вече видяхме какво означава една структура да бъде полупръстен — това означава в нея да са верни аксиомите на равенството (1)–(5), както и аксиомите за полупръстен, дадени на стр. 65. Затова да бъде една клауза вярна във всеки полупръстен означава тя да бъде вярна във всяка структура, в която са верни изброените аксиоми. Това мотивира следната дефиниция:

3.32. Дефиниция. Нека Γ е множество от клаузи. Казваме, че атомарната формула или клаузата φ *следва* от клаузите в Γ, ако φ е тъждествено вярна във всяка структура, в която са тъждествено верни клаузите в Γ. В този случай казваме още и накратко „от Γ следва φ“ и използваме следното означение:

$$\Gamma \vDash \varphi$$

3.33. *Забележка*: Нека не се объркваме и обърнем внимание, че ще използваме знака $\vDash$ за няколко различни цели. Когато Γ е множество от клаузи, тогава $\Gamma \vDash \varphi$ означава „от клаузите в Γ следва φ“. Когато $\mathbf{M}$ е структура, $\mathbf{M} \vDash \varphi$ означава „φ е (тъждествено) вярно в $\mathbf{M}$“. А когато $\mathbf{M}$ е структура и v е оценка, $\mathbf{M} \vDash \varphi[v]$ означава „φ е вярна в $\mathbf{M}$ при оценка v“.

***Задача* 26:** 1. Нека Γ е множество от клаузи и $\varphi \in \Gamma$. Докажете, че от Γ следва φ.

2. Нека Γ и Δ са множества от клаузи и $\Delta \subseteq \Gamma$. Докажете, че ако φ следва от Δ, то φ следва от Γ.
3. Нека Γ и Δ са множества от клаузи и всеки елемент на Δ следва от Γ. Докажете, че ако φ следва от Δ, то φ следва от Γ.

***Забележка*:** Условия а) и в) от задача 26 аксиоматизират това, което в абстрактната алгебрична логика се нарича *релация на следване*.*

В контекста на логическото програмиране е полезно и друго понятие за следване, при което за формулата φ не искаме да бъде тъждествено вярна, а само изпълнима. Нека например Γ съдържа клаузите от някоя програма на пролог. Тогава целта `?-p(X,X)` ще се удовлетвори не когато от програмата следва, че атомарната формула `p(X,X)` е вярна при произволна стойност на променливата `X`, а когато тя е вярна при поне една стойност на `X`.

3.34. Дефиниция. Нека Γ е множество от клаузи. Казваме, че атомарната формула φ се *удовлетворява* от клаузите в Γ, ако φ е изпълнима във всяка структура, в която са тъждествено верни клаузите в Γ.

*На английски consequence relation.

Глава 4

Изводимост

4.1. Увод в права и обратна изводимост

Има два основни начина за компютърно използване на правила. При първия начин, т.н. *права изводимост*, ако в базата знания вече се съдържат предпоставките на дадено правило като установени знания, към нея ще бъде добавено и заключението на правилото. Например, ако имаме правило $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ и предпоставките $\varphi_1, \varphi_2, \ldots, \varphi_n$ са вече известни знания, към базата знания ще бъде добавено и заключението ψ. Когато се използва права изводимост системата се интересува какви знания има до момента и въз основа на тях извежда нови знания. При правата изводимост обаче системата не се интересува каква е крайната цел и затова често генерира излишно много ненужни знания. Това е и основният недостатък на правата изводимост.

При *обратната изводимост* правилата се използват „наопаки“. Тук системата обръща основно внимание на това каква е целта, която искаме да постигнем, и чрез кои правила тя може да се получи. След като открие тези правила, с тяхна помощ се пораждат нови подцели. Тези подцели пораждат нови подцели и т.н. дотогава, докато открием начин да се постигне първоначалната цел. Недостатък на обратната изводимост е това, че системата не се интересува какво всъщност е известно, така че ще пробва много невъзможни начини, за постигане на желаната цел. Обратната изводимост не е много подходяща и за управляващи системи, реагиращи на нововъзникнали събития (например спиране на подаването на гориво към двигателя при поява съответни сигнали от

датчиците).

Има и хибридни системи, при които се използва отчасти права и отчасти обратна изводимост.

Ще илюстрираме правата и обратната изводимост с пример за химичен синтез.* Да допуснем, че разполагаме с неограничено количество въглерод (C), кислород (O_2), водород (H_2) и магнезиев оксид (MgO) и искаме да получим въглеродна киселина (H_2CO_3). За целта можем да използваме следните химични реакции:

$$\mathrm{MgO} + \mathrm{H_2} \longrightarrow \mathrm{Mg} + \mathrm{H_2O}$$
$$\mathrm{C} + \mathrm{O_2} \longrightarrow \mathrm{CO_2}$$
$$\mathrm{CO_2} + \mathrm{H_2O} \longrightarrow \mathrm{H_2CO_3}$$

Наличието на въглерод, кислород, водород и магнезиев оксид можем да представим посредством следните аксиоми/факти:

$$\vdash \mathrm{C} \tag{11}$$
$$\vdash \mathrm{O_2} \tag{12}$$
$$\vdash \mathrm{H_2} \tag{13}$$
$$\vdash \mathrm{MgO} \tag{14}$$

Трите позволени химични реакции пък се свеждат до следните четири правила:

$$\mathrm{MgO}, \mathrm{H_2} \vdash \mathrm{Mg} \tag{15}$$
$$\mathrm{MgO}, \mathrm{H_2} \vdash \mathrm{H_2O} \tag{16}$$
$$\mathrm{C}, \mathrm{O_2} \vdash \mathrm{CO_2} \tag{17}$$
$$\mathrm{CO_2}, \mathrm{H_2O} \vdash \mathrm{H_2CO_3} \tag{18}$$

Целта (която в случая е да получим въглеродна киселина) понякога се представя посредством правило без заключение:

$$\mathrm{H_2CO_3} \vdash$$

Да видим как всичко това работи при използване на права изводимост. Да означим с Δ_i множеството на нещата, които са известни на стъпка i. В началото все още нищо не е известно (т.е. базата знания е празна) и затова

$$\Delta_0 = \varnothing$$

*Примерът е от [5, раздел 2.6].

Единствените правила, които могат да се задействат* при положение, че все още нищо не е известно, са правилата без предпоставки, т.е. (11)–(14). След като тези правила се задействат

$$\Delta_1 = \{\mathrm{C}, \mathrm{O_2}, \mathrm{H_2}, \mathrm{MgO}\}$$

При новото състояние на базата знания правилата без предпоставки отново могат да се задействат, но няма да ни дадат нищо ново. Освен тях обаче вече може да се задействат и правила (15)–(17). С тяхна помощ базата знания става

$$\Delta_2 = \{\mathrm{C}, \mathrm{O_2}, \mathrm{H_2}, \mathrm{MgO}, \mathrm{Mg}, \mathrm{H_2O}, \mathrm{CO_2}\}$$

При новото съдържание на базата знания правила (11)–(17) отново могат да се задействат, но няма да ни дадат нищо ново. Освен тях обаче може да се задейства и правило (18), посредством което и получаваме желаната цел $\mathrm{H_2CO_3}$.

Да видим сега как можем да получим $\mathrm{H_2CO_3}$, използвайки обратна изводимост.

Единственото правило, от което можем да получим $\mathrm{H_2CO_3}$ е (18). Правило (18) ни дава две подцели — $\mathrm{CO_2}$ и $\mathrm{H_2O}$.

Единственото правило, от което можем да получим първата подцел, т.е. $\mathrm{CO_2}$ е (17). От правило (17) получаваме две подподцели — C и $\mathrm{O_2}$, които получаваме веднага от правилата без предпоставки.

Единственото правило, от което можем да получим втората подцел, т.е. $\mathrm{H_2O}$ е (16). От правило (16) получаваме две подподцели — MgO и $\mathrm{H_2}$, които получаваме веднага от правилата без предпоставки.

Да забележим, че обратната изводимост се оказа по-ефективна, защото изобщо не ни се наложи да прилагаме правило (15).

Да забележим също, че и при правия, и при обратната изводимост открихме по същество един и същ начин за получаване на въглеродна киселина от въглерод, кислород, водород и магнезиев оксид. Логиците изобразяват така намерения извод, използвайки дървоподобна структура от следния вид:

$$\dfrac{\dfrac{\mathrm{C} \qquad \mathrm{O_2}}{\mathrm{CO_2}} \qquad \dfrac{\mathrm{MgO} \qquad \mathrm{H_2}}{\mathrm{H_2O}}}{\mathrm{H_2CO_3}}$$

Всяка от хоризонталните черти в този запис отговаря на прилагането на някое от правилата.

*На английски технически жаргон се използва „fire“ вместо „задействат“.

4.1. Тъй като в езика за програмиране пролог се използва обратна изводимост, правилата също се записват наобратно — правилото

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

на пролог изглежда така:

$$\psi :- \varphi_1, \varphi_2, \ldots, \varphi_n.$$

Аксиомата/фактът

$$\vdash \psi$$

на пролог се записва така:

$$\psi.$$

***Задача* 27:** Дадени са правилата

$$\frac{}{\mathtt{p}} \qquad \frac{\mathtt{p} \quad \mathtt{p}}{\mathtt{q}} \qquad \frac{\mathtt{r}}{\mathtt{q}} \qquad \frac{\mathtt{s} \quad \mathtt{q}}{\mathtt{r}} \qquad \frac{\mathtt{q} \quad \mathtt{p}}{\mathtt{s}}$$

Използвайки обратна изводимост, изведете `r`.

4.2. Формална изводимост

Предпоставките и заключенията на правилата от примера за химичен синтез, който в раздел 4.1 използвахме, за да се запознаем с правата и обратната изводимост, не съдържаха променливи, а това — оказва се — спестява невероятно много усложнения. Да разгледаме един нормален пример с променливи.

Нека са дадени клаузите:

$$\vdash \mathtt{x} = \mathtt{f}(\mathtt{x}) \tag{19}$$

$$\mathtt{x} = \mathtt{y},\ \mathtt{y} = \mathtt{z} \vdash \mathtt{x} = \mathtt{z} \tag{20}$$

Тъй като в тези клаузи навсякъде се подразбира, че те са верни при произволни стойности на променливите, то в (19) можем положим $\mathtt{f}(\mathtt{x})$ на мястото на `x`, при което ще получим като частен случай

$$\vdash \mathtt{f}(\mathtt{x}) = \mathtt{f}(\mathtt{f}(\mathtt{x})) \tag{21}$$

Ако в (20) положим $\mathtt{f}(\mathtt{x})$ вместо `y` и $\mathtt{f}(\mathtt{f}(\mathtt{x}))$ вместо `z`, ще получим като частен случай

$$\mathtt{x} = \mathtt{f}(\mathtt{x}),\ \mathtt{f}(\mathtt{x}) = \mathtt{f}(\mathtt{f}(\mathtt{x})) \vdash \mathtt{x} = \mathtt{f}(\mathtt{f}(\mathtt{x}))) \tag{22}$$

Очевидно от (19), (21) и (22) можем да получим като следствие

$$\vdash \mathtt{x} = \mathtt{f}(\mathtt{f}(\mathtt{x})) \tag{23}$$

И така, за да изведем (23), ни се наложи да заместваме променливите с различни изрази, а гледайки единствено изходните клаузи (19) и (20), изобщо не се вижда какви полагания трябва да правим, за да получим нужната цел. Ако целта ни е да разполагаме с някакъв алгоритмичен метод, това е проблем.

Най-простият начин за преодоляване на това затруднение е следният — всяка променлива ще заместваме с всички възможни термове. Така получения метод ще наречем *либерална права изводимост*. Думата „либерална“ в това название показва, че в тази изводимост частните случаи се генерират по всички възможни начини (т.е. давайки на променливите всевъзможни стойности). По-нататък ще дефинираме и други „либерални“ изводимости — *либерална обратна изводимост* и *либерална резолютивна изводимост*. При всяка една от тези изводимости се използват частни случаи като при това тези частни случаи се генерират по всички възможни начини.

Няма защо обясняваме, че либералните изводимости са извънредно неефективни методи — та нали термовете обикновено са безброй много и значи и частните случаи са безброй много! Заради тази неефективност е напълно безсмислено да реализираме тези методи на компютър. И все пак, има смисъл да разработим математическата теория на тези изводимости поради следните причини:

– Математическите свойства на либералните изводимости се доказват сравнително лесно.
– Ще видим, че за всяка либерална изводимост ще можем да получим съответна „нелиберална“ изводимост, която е ефективна.
– Математическите свойства на „нелибералните“ изводимости се получават сравнително лесно като следствия от съответните свойства на либералните изводимости.

Методът, с помощта на който от либералните изводимости получаваме съответни „нелиберални“, се състои в използването на т.н. най-общ унификатор. Засега няма да уточняваме какво точно значи „най-общ унификатор“, а само ще отбележим, че може да си мислим, че благодарение на използването на най-общ унификатор при правене на доказателства компютрите могат да „познават“ кой частен случай ще им бъде полезен и ще им свърши работа. Благодарение на това много често вместо да се генерират безброй много частни случаи е достатъчно да се получи един единствен частен случай.

Между другото, като казваме, че нелибералните изводимости са ефективни, имаме предвид само това, че те са най-ефективният метод, който е измислен. Нелибералната права изводимост се използва, за да тестваме с компютър дали някоя атомарна формула φ се удовлетворява от Γ, където Γ е крайно множество от клаузи. Нека функцията $f\colon \mathbb{N} \to \mathbb{N}$ е такава, че ако в клаузите в $\Gamma \cup \{\varphi\}$ се използват общо n символа, то със сигурност компютърът ще може да установи, че φ се удовлетворява от Γ за не повече от $f(n)$ стъпки. Функцията f е мярка за това колко ефективна е дадена изводимост — колкото по-ефективна е изводимостта, толкова по-малки стойности може да има функцията f. Питаме се колко бързо расте функцията f при „нелибералните“ изводимости (права, обратна, резолютивна). Е, оказва се, че функцията f расте по-бързо от всички функции, които могат да се смятат с алгоритъм. С други думи тя расте толкова бързо, че е трудно да се каже колко точно бързо расте. Да отбележим, че алгоритмите с експоненциална сложност (при които $f(n) = a^n$) се считат за твърде бавни, а е много лесно да смятаме с алгоритъм a^n. Също така е много лесно да смятаме с алгоритъм и така наречената суперекспонента, при която

$$f(n) = \underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{n \text{ броя } a}$$

Можем ли да наречем ефективна една компютърна програма, при която времето на работа се измерва със суперекспонента? Суперекспонентата обаче е толкова мъничка функция в сравнение с функцията $f(n)$, която измерва сложността на „нелибералните“ изводимости, че просто не сме в състояние да кажем колко по-мъничка е тя. И значи „нелибералните“ извосимости, за които казахме, че са „ефективни“, всъщност са толкова неефективни, че просто е невъзможно да кажем колко точно неефективни са.

Има ли тогава изобщо някакъв смисъл да се занимаваме с такива алгоритми и да ги реализираме на компютър? Да, по две причини. Първо, защото тази обезсърчаващо лоша оценка за ефективността на метода, се отнася за възможно най-лошия случай. Това, че в най-лошия случай един метод работи неизползваемо бавно, не значи, че няма ситуации, при които той работи бързо. Второ, защото е доказано, че не съществува метод, който решава задачите, решавани от „нелибералните“ изводимости и чиято теоретична сложност е по-добра. Т.е. няма надежди да измислим метод, чиято теоретична ефективност не е толкова обезсърчаващо лоша.

Тема за размисъл: Езикът пролог се базира на нелибералната обратна изводимост, която току-що „заклеймихме“ като безумно неефективна. Въпреки това алгоритмите, реализирани на пролог, работят общо взето по-бързо, отколкото когато са реализирани например на някой от т.н. „скриптови“ езици като пърл, пайтън и баш. Как е възможно това?

4.2. Дефиниция. а) *Субституцията* е функция, която на всяка променлива съпоставя терм.

б) Ако s е субституция, а τ — терм, атомарна формула или клауза, с $\hat{s}(\tau)$ ще означим израза, който се получава като заместим в τ всяка променлива $\mathtt{x}$ с $s(\mathtt{x})$. Изразът $\hat{s}(\tau)$ се нарича *резултат от прилагането на субституцията s към τ.*

Забележка: Тъй като субституциите се използват в различни алгоритми, дефиницията им обикновено включва допълнителното изискване, че само за краен брой променливи $s(\mathtt{x}) \neq \mathtt{x}$, а за всички останали $s(\mathtt{x}) = \mathtt{x}$. По този начин, за да представим в компютъра една субституция, е достатъчно да помним само двойките $(\mathtt{x}, s(\mathtt{x}))$, при които $\mathtt{x} \neq s(\mathtt{x})$, а те са само краен брой. С цел да не си усложняваме ненужно математическите разсъждения, в нашата дефиниция сме изпуснали това допълнително изискване.

4.3. Пример. Ако субституцията s е такава, че $s(\mathtt{x}) = \mathtt{f(y)}$ и $s(\mathtt{y}) = \mathtt{x}$ и $\varphi = \mathtt{p(x, y, f(x))}$, то $\hat{s}(\varphi) = \mathtt{p(f(y), x, f(f(y)))}$.

4.4. Твърдение. *Нека s е субституция.*

а) Ако τ е терм, то $\hat{s}(\tau)$ е терм.

б) Ако φ е атомарна формула, то $\hat{s}(\varphi)$ е атомарна формула.

в) Ако δ е клауза, то $\hat{s}(\delta)$ е клауза.

Задача 28: Докажете твърдение 4.4.

4.5. Дефиниция. За всяка атомарна формула φ атомарните формули от вида $\hat{s}(\varphi)$ се наричат *частни случаи* на φ. За всяка клауза δ клаузите от вида $\hat{s}(\delta)$ се наричат *частни случаи* на δ.

4.6. Пример. Клауза (22) е частен случай на клауза (20). За да видим това, можем да използваме коя да е субституция, имаща следните свойства: $s(\mathtt{x}) = \mathtt{x}$, $s(\mathtt{y}) = \mathtt{f(x)}$, $s(\mathtt{z}) = \mathtt{f(f(z))}$.

4.7. Твърдение. *Всяка атомарна формула е частен случай на себе си и всяка клауза е частен случай на себе си.*

Доказателство. Използваме субституцията s, която замества всяка променлива $\mathtt{x}$ с $\mathtt{x}$. Прилагането на s към коя да е атомарна формула я остава непроменена. ∎

***Задача* 29:** Вярно ли е, че φ' е частен случай на φ (т.е. съществува ли субституция s, такава че $\hat{s}(\varphi) = \varphi'$), ако

1. $\varphi = \mathtt{p}(\mathtt{x}, \mathtt{c})$ и $\varphi' = \mathtt{p}(\mathtt{c}, \mathtt{c})$;
2. $\varphi = \mathtt{p}(\mathtt{c}, \mathtt{c})$ и $\varphi' = \mathtt{p}(\mathtt{x}, \mathtt{c})$;
3. $\varphi = \mathtt{p}(\mathtt{x}, \mathtt{x})$ и $\varphi' = \mathtt{p}(\mathtt{c}, \mathtt{x})$;
4. $\varphi = \mathtt{p}(\mathtt{x}, \mathtt{y})$ и $\varphi' = \mathtt{p}(\mathtt{f}(\mathtt{y}), \mathtt{f}(\mathtt{x}))$.

В следващата дефиниция може да си мислим, че Γ съдържа някакви аксиоми, които приемаме за верни, а Δ съдържа неща, които вече сме доказали. Грубо казано, казваме, че χ се извежда либерално от Δ посредством Γ, ако е възможно да докажем χ, използвайки едностъпково разсъждение, в което приемаме елементите на Γ и Δ за верни. Получаването на частен случай на нещо, което вече сме доказали, се счита за тривиална операция и не се брои за отделна стъпка.

4.8. Дефиниция. Нека Γ е множество от клаузи, Δ е множество от атомарни формули. Ако някой елемент на Γ има частен случай

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

и φ_1, φ_2,..., φ_n са частни случаи на елементи на Δ, то казваме, че ψ се *извежда либерално* от Δ посредством Γ.

4.9. Пример.
– Ако Γ съдържа клауза (19), то посредством Γ от ∅ се извеждат либерално атомарните формули $\mathtt{x} = \mathtt{f}(\mathtt{x})$ и $\mathtt{f}(\mathtt{x}) = \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
– Ако Γ съдържа клауза (20), то атомарната формула $\mathtt{x} = \mathtt{f}(\mathtt{f}(\mathtt{x}))$ се извежда либерално от $\{\mathtt{x} = \mathtt{f}(\mathtt{x}), \mathtt{f}(\mathtt{x}) = \mathtt{f}(\mathtt{f}(\mathtt{x}))\}$ посредством Γ.

4.10. Дефиниция. Нека Γ е множество от клаузи.

а) За всяко естествено число n ще дефинираме множество ЛПИ$_n$(Γ) от атомарни формули. Нека ЛПИ$_0$(Γ) = ∅. И нека за всяко i множеството ЛПИ$_{i+1}$(Γ) съдържа елементите на ЛПИ$_i$(Γ), както и всички атомарни формули, които се извеждат либерално от ЛПИ$_i$(Γ) посредством Γ.

б) Нека ЛПИ(Γ) е обединението на всички множества ЛПИ$_i$(Γ).

Интуитивно, може да считаме, че ЛПИ$_i$(Γ) съдържа онези атомарни формули, които могат да се докажат с i-стъпково разсъждение, използвайки като аксиоми правилата в Γ. Множеството ЛПИ(Γ) съдържа всички атомарни формули, които могат да се докажат от Γ.

***Задача* 30:** Нека сигнатурата съдържа единствен символ за константа `c`, единствен функционален символ `f` (едноместен) и единствен предикатен символ `p` (също едноместен). Нека множеството Γ има за елементи клаузите $\vdash \mathtt{p(c)}$ и $\mathtt{p(x)} \vdash \mathtt{p(f(x))}$. За всяко естествено число i намерете ЛПИ$_i$(Γ).

***Задача* 31:** Какво ще се промени в предната задача, ако сигнатурата съдържа и други символи освен `c`, `f` и `p`?

***Задача* 32:** Намерете множествата ЛПИ$_i$(Γ) и ЛПИ(Γ), ако Γ съдържа клаузите $\vdash \mathtt{p(c,c)}$ и $\mathtt{p(x,y)} \vdash \mathtt{p(f(x),z)}$.

4.11. Дефиниция. Казваме, че атомарната формула φ се *удовлетворява с либерална права изводимост* от Γ, ако ЛПИ(Γ) съдържа поне един частен случай на φ.

4.12. На пръв поглед не е очевидно, че можем да генерираме елементите на ЛПИ(Γ) алгоритмично, но това наистина е така. Следователно ако φ се удовлетворява с либерална права изводимост от Γ, то има начин това да се установи алгоритмично — просто трябва да си генерираме търпеливо елементите на ЛПИ(Γ), чакайки да се появи някой частен случай на φ. Ако φ се удовлетворява с либерална права изводимост от Γ, то рано или късно ще попаднем на някой частен случай на φ. Ако обаче φ не се удовлетворява с либерална права изводимост от Γ, тогава този алгоритъм ще генерира до безкрайност нови и нови елементи на ЛПИ(Γ) с надеждата да намери някой частен случай на φ, т.е. ще се зацикли. За алгоритъм, имащ тези свойства, казваме, че *полурешава* задачата дали някоя атомарна формула се удовлетворява с либерална права изводимост от Γ. Този дефект за съжаление е неотстраним — в теория на изчислимостта се доказва, че не съществува алгоритъм който *решава* тази задача и никога не се зацикля.

****Задача* 33:** Намерете алгоритъм, който генерира елементите на ЛПИ(Γ) когато Γ е крайно множество от клаузи и сигнатурата съдържа краен брой символи.

В дефиниция 3.34 дефинирахме понятието „φ се удовлетворява от Γ“, което казва, че φ е изпълнима във всяка структура, в която са верни

клаузите от Γ. Това понятие е важно, но очевидно то не е директно проверимо от компютър. Структурите, при които правилата в Γ се оказват верни, могат да бъдат най-разнообразни — толкова разнообразни, че не само компютър, ами и човек не може да ги опише всичките. Как тогава компютър ще може да провери дали във всяка такава структура е изпълнима формулата φ? А дори и да се окаже възможно да опишем всички структури, в които клаузите от Γ са верни, те със сигурност ще бъдат безброй много[*] и няма как с компютър да проверим дали φ е изпълнима във всяка една от тези безброй много структури. Но да допуснем, че по някакви причини се окаже възможно да се ограничим само с една структура. Ако универсумът ѝ е безкраен, задачата пак става нерешима![**]

От друга страна, току що дефинирахме понятието „φ се удовлетворява с либерална права изводимост от Γ“, което може да се полурешава алгоритмично. Изобщо от никъде не личи каква е връзката между двете понятия „φ се удовлетворява от Γ“ и „φ се удовлетворява с либерална права изводимост от Γ“. Дефинициите на тези две понятия са коренно различни! И въпреки това, оказва се, че те са еквивалентни.

4.13. Когато някоя изводимост е такава, че всяко нещо, което може да се докаже е вярно, тогава казваме, че тази изводимост е **коректна**. А когато всяко вярно нещо може да се докаже, тогава казваме, че изводимостта е **пълна**.

Например когато кажем, че либералната права изводимост е коректна, това означава, че ако φ се удовлетворява с либерална права изводимост от Γ, то φ се удовлетворява от Γ. А когато кажем, че тя е пълна, това означава, че ако φ се удовлетворява от Γ, то φ се удовлетворява с либерална права изводимост от Γ.

Теоремата за коректност и пълнота на дадена изводимост е дълбок резултат, от изключителна важност в математическата логика, който се доказва в почти всеки учебник по логика. Ако погледнем по-философски на тази теорема, тя казва, че едно нещо е доказуемо тогава и само тогава, когато то е вярно. Доказуемост и вярност са две напълно различни понятия и тяхната еквивалентност е нещо удивително!

[*] Вж. задача 25.

[**] Например ако универсумът е $\mathbb{N}\setminus\{0\}$, функционалният символ „+“ се интерпретира като събиране, ^ като степенуване и 2 като числото две, то атомарната формула $\mathtt{x}\hat{}(\mathtt{x}'+2)+\mathtt{y}\hat{}(\mathtt{y}'+2) = \mathtt{z}\hat{}(\mathtt{z}'+2)$ е изпълнима тогава и само тогава, когато великата теорема на Ферма е невярна. Няма как да искаме от компютрите да решат задача, която на хората им е отнела векове!

Да разберем, че нещо е доказуемо означава да разполагаме с доказателство, което сме в състояние да проверим стъпка по стъпка и което е достатъчно подробно, така че да можем да осъществим проверката изцяло механично, без да се замисляме за каквото и да е. Но и след най-акуратната проверка на едно такова доказателство ние така и няма да разберем защо всъщност доказаното нещо е вярно. От друга страна, да разберем, че нещо е вярно, означава да се сдобием с интуиция за това как всъщност стоят нещата, която да ни убеди, че нещото наистина е вярно. Но колкото и добра да е интуицията на един математик, винаги има случаи, когато тя го заблуждава, така че единственият сигурен начин да се убедим във верността на нещо е да имаме негово доказателство. По този начин виждаме, че всяко едно от тези две неща — използването на доказателствата като окончателно и безапелативно свидетелство за вярност и придобиването на интуиция, която „от пръв поглед“ да ни казва как стоят нещата, кое е вярно и кое не — е важно и необходимо в математиката.

Интересно е това, че за тези две неща се грижат напълно различни дялове от главния мозък. Светът на лявото полукълбо е подреден свят, където за всичко си има точни правила. Всяко нещо в този свят кристално ясно и съществува необвързано и само по себе си. За речта — говорима или писмена — отговаря лявото полукълбо. От друга страна, светът на дясното полукълбо е объркан. Всичко в този свят е свързано с всичко и се влияе от всичко. В този свят няма „да“ и „не“, няма „вярно“ и „невярно“, а едно цялостно усещане за съотношението на всяко нещо с всички останали неща. Дясното полукълбо е нямо и вместо реч прави музика.*

Математиката е може би най-точната наука и затова изглежда естествено, ако за нея отговаря изцяло лявото полукълбо на мозъка. Оказва се обаче, че това съвсем не е така. За формулирането на логически правилни разсъждения наистина отговаря лявата половина на мозъка, но математическата интуиция и математическите идеи са отдясно. Новата математика се твори от дясната половина на мозъка, а се контролира и проверява от лявата.

Ако четем и заучаваме някое точно математическо доказателство, но интуицията и идеите, скрити в него ни убягват, това ще бъде безполезно. И също така да четем или слушаме нечии математически идеи, които не са съпроводени с точни доказателства, е все едно да слушаме нечии празни фантазии — на такива фантазии никой математик няма

*Повече подробности за функциите на двете полукълба на мозъка може да се научат напр. от [12].

да обърне сериозно внимание (и с право!).

Когато студентите учат някой математически предмет (напр. логическо програмиране), те винаги трябва да търсят скритите идеи и интуицията, обясняващи смисъла дефинициите и изясняващи доказателствата. Също така те трябва да изучават как могат да изказват тези идеи на точен математически език. И когато това стане, те ще могат да се движат свободно в абстрактния свят на математиката и да творят, знаейки, че винаги ще бъдат в състояние да облекат измислените неща в точен математически формализъм, който да им гарантира, че не са измислили някоя глупост и с който ще могат да споделят идеите си. Ако студентите учат нещата наизуст, без всъщност да ги разбират, те най-вероятно ще си вземат изпита, но неприятните усещания, които са изпитвали докато учат, могат да предизвикат у един съпричастен преподавател единствено съчувствие и съжаление. Студенти пък, които са се опитвали да учат идеите без да проследяват доказателствата, са направили нещо безполезно, което е може би по-подходящо за философски факултет, отколкото за ФМИ. Обаче студенти, които са успели да схванат истински нещата, са радост за преподавателите, защото такива студенти са успели поне донякъде да видят красотите в чудния, хем измислен, хем истински свят, в който преподавателите са се опитали да ги заведат.

Забележка: Можем накратко да резюмираме казаното до тук само с едно изречение: „Използвайте целия си мозък, а не само едната му половина!“

4.3. Заместващи морфизми

4.14. Дефиниция. Казваме, че изображението h е заместващ морфизъм, ако h изобразява термове в термове и атомарни формули в атомарни формули по такъв начин, че:

а) $h(\mathtt{c}) = \mathtt{c}$ за всеки символ за константа $\mathtt{c}$.

б) $h(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)) = \mathtt{f}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$ за всеки n-местен функционален символ $\mathtt{f}$.

в) $h(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)) = \mathtt{p}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$ за всеки n-местен предикатен символ $\mathtt{p}$.

Сравнете тази дефиниция с дефиниция 4.14.

4.15. Твърдение. *Нека s е субституция. Изображението, което съпоставя на всеки терм τ терма $\hat{s}(\tau)$ и на всяка атомарна формула φ атомарната формула $\hat{s}(\varphi)$ е заместващ морфизъм.*

Доказателство. Следва непосредствено от дефиниции 4.2 б) и 4.14. ■

Сравнете предното твърдение с твърдение 3.23.

4.16. Твърдение. *Нека h и h' са заместващи морфизми.*

а) Ако $h(\mathtt{x}) = h'(\mathtt{x})$ за всяка променлива $\mathtt{x}$, която се среща в терма τ, то $h(\tau) = h'(\tau)$.

б) Ако $h(\mathtt{x}) = h'(\mathtt{x})$ за всяка променлива $\mathtt{x}$, която се среща в атомарната формула φ, то $h(\varphi) = h'(\varphi)$.

Доказателство. **(а)** С индукция по терма τ.

Ако $\tau = \mathtt{x}$ е променлива, то $h(\tau) = h'(\tau)$, защото по условие h и h съвпадат за променливи, които се срещат в τ (в случая за $\mathtt{x} = \tau$).

Ако $\tau = \mathtt{c}$ е символ за константа, то както $h(\tau) = h(\mathtt{c})$, така и $h'(\tau) = h'(\mathtt{c})$ е равно на $\mathtt{c}$, защото h и h' са заместващи морфизми.

Нека $\tau = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$. Понеже h и h' са заместващи морфизми, то

$$h(\tau) = \mathtt{f}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$
$$h'(\tau) = \mathtt{f}(h'(\tau_1), h'(\tau_2), \ldots, h'(\tau_n))$$

Изразите отдясно на равенствата обаче са равни, защото съгласно индукционното предположение $h(\tau_i) = h'(\tau_i)$.

(б) Нека $\varphi = \mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$. Понеже h и h' са заместващи морфизми, то

$$h(\varphi) = \mathtt{p}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$
$$h'(\varphi) = \mathtt{p}(h'(\tau_1), h'(\tau_2), \ldots, h'(\tau_n))$$

Изразите отдясно на еквивалентностите обаче съвпадат, защото в а) доказахме, че $h(\tau_i) = h'(\tau_i)$. ■

Сравнете предното твърдение с твърдение 3.24.

4.17. Следствие. *Нека h е заместващ морфизъм и субституцията s е такава, че $s(\mathtt{x}) = h(\mathtt{x})$ за произволна променлива $\mathtt{x}$. Тогава*

а) за всеки терм τ, $h(\tau) = \hat{s}(\tau)$;

б) за всяка атомарна формула φ, $h(\varphi) = \hat{s}(\varphi)$;

в) $h = \hat{s}$.

Доказателство. Нека h' е изображението, което на произволен терм τ или атомарна формула φ съпоставя терма $\hat{s}(\tau)$ или съотв. атомарната формула $\hat{s}(\varphi)$. Съгласно 4.15, h' е заместващ морфизъм. Тъй като за произволна променлива $\mathtt{x}$ имаме $h(\mathtt{x}) = s(\mathtt{x}) = h'(\mathtt{x})$, то искането следва от 3.24. ∎

Горното следствие показва, че всеки заместващ морфизъм всъщност представлява прилагане на подходяща субституция. Сравнете това следствие със следствие 3.25.

4.18. Твърдение. *Композиция на заместващи морфизми е заместващ морфизъм.*

Доказателство. Нека h_1 и h_2 са заместващи морфизми и $h(\tau) = h_1(h_2(\tau))$ за всеки терм или атомарна формула τ. Очевидно h изобразява термове в термове и атомарни формули в атомарни формули. Освен това:

– За всеки символ за константа $h(\mathtt{c}) = h_1(h_2(\mathtt{c})) = h_1(\mathtt{c}) = \mathtt{c}$.

– За всеки n-местен функционален символ $\mathtt{f}$

$$h(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)) = h_1(h_2(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)))$$

Тъй като h_2 е заместващ морфизъм, то последното е равно на

$$h_1(\mathtt{f}(h_2(\tau_1), h_2(\tau_2), \ldots, h_2(\tau_n)))$$

Тъй като h_1 също е заместващ морфизъм, това е равно на

$$\mathtt{f}(h_1(h_2(\tau_1)), h_1(h_2(\tau_2)), \ldots, h_1(h_2(\tau_n))) = \mathtt{f}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

– За всеки n-местен предикатен символ $\mathtt{p}$

$$h(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)) = h_1(h_2(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)))$$

Тъй като h_2 е заместващ морфизъм, то последното е равно на

$$h_1(\mathtt{p}(h_2(\tau_1), h_2(\tau_2), \ldots, h_2(\tau_n)))$$

Тъй като h_1 също е заместващ морфизъм, това е равно на

$$\mathtt{p}(h_1(h_2(\tau_1)), h_1(h_2(\tau_2)), \ldots, h_1(h_2(\tau_n))) = \mathtt{f}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

■

4.19. Следствие. *За всеки две субституции s_1 и s_2 можем да намерим такава субституция s, че $\hat{s}(\tau) = \hat{s_2}(\hat{s_1}(\tau))$ за всеки терм τ, $\hat{s}(\varphi) = \hat{s_2}(\hat{s_1}(\varphi))$ за всяка атомарна формула φ и $\hat{s}(\delta) = \hat{s_2}(\hat{s_1}(\delta))$ за всяка клауза δ.*

Доказателство. Най-напред да докажем това следствие в случая на терм и атомарна формула. Съгласно 4.15 прилаганията на s_1 и s_2 са заместващи морфизми. Съгласно 4.18 последователното прилагане на s_1 и s_2 също е заместващ морфизъм. От 4.17 намираме такава субституция s, че този заместващ морфизъм представлява прилагане на субституцията s.

За случая на клауза се налага отделно доказателство, защото сме дефинирали аргументът на заместващите морфизми да бъде терм или атомарна формула, но не и клауза. При клаузи обаче исканото следва лесно от вече доказаното:

$$\begin{aligned}\hat{s}(\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi) &= \hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n) \vdash \hat{s}(\psi) \\ &= \hat{s_2}(\hat{s_1}(\varphi_1)), \hat{s_2}(\hat{s_1}(\varphi_2)), \ldots, \hat{s_2}(\hat{s_1}(\varphi_n)) \vdash \hat{s_2}(\hat{s_1}(\psi)) \\ &= \hat{s_2}(\hat{s_1}(\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi))\end{aligned}$$

■

4.20. Следствие. *Частен случай на частен случай на τ е частен случай на τ*

Доказателство. Частните случаи на τ имат вида $\hat{s_1}(\tau)$ за някоя субституция s_1 и значи частните случаи на частните случаи на τ имат вида $\hat{s_2}(\hat{s_1}(\tau))$ за някои субституции s_1 и s_2. Съгласно следствие 4.19 те имат вида $\hat{s}(\tau)$ за някоя субституция s. ■

***Задача* 34:** Дадени са субституции s_1 и s_2 и нека s бъде субституцията, дефинирана с равенството

$$s(\mathtt{x}) = \hat{s_2}(s_1(\mathtt{x}))$$

Докажете, че последователното прилагане на s_1 и s_2 е еквивалентно на прилагането на субституцията s.

4.21. Твърдение. *Композиция на заместващ морфизъм с остойностяващ морфизъм е остойностяващ морфизъм.*

Доказателство. Нека h_1 е остойностяващ морфизъм в структурата $\mathbf{M}$, h_2 е заместващ морфизъм и $h(\tau) = h_1(h_2(\tau))$ за всеки терм или атомарна формула τ. Изображението h изобразява термовете в елементи на универсума на $\mathbf{M}$ и атомарните формули — в твърдения.

$$\text{терм} \xrightarrow{h_2} \text{терм} \xrightarrow{h_1} \text{елем. на } |\mathbf{M}|$$

$$\text{ат. формула} \xrightarrow{h_2} \text{ат. формула} \xrightarrow{h_1} \text{твърдение}$$

Освен това:

– За всеки символ за константа $h(\mathtt{c}) = h_1(h_2(\mathtt{c})) = h_1(\mathtt{c}) = \mathtt{c}^{\mathbf{M}}$.

– За всеки n-местен функционален символ $\mathtt{f}$

$$h(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)) = h_1(h_2(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)))$$

Тъй като h_2 е заместващ морфизъм, то последното е равно на

$$h_1(\mathtt{f}(h_2(\tau_1), h_2(\tau_2), \ldots, h_2(\tau_n)))$$

Тъй като h_1 остойностяващ морфизъм, това е равно на

$$\mathtt{f}^{\mathbf{M}}(h_1(h_2(\tau_1)), h_1(h_2(\tau_2)), \ldots, h_1(h_2(\tau_n))) = \mathtt{f}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

– За всеки n-местен предикатен символ $\mathtt{p}$

$$h(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)) = h_1(h_2(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)))$$

Тъй като h_2 е заместващ морфизъм, то последното е равно на

$$h_1(\mathtt{p}(h_2(\tau_1), h_2(\tau_2), \ldots, h_2(\tau_n)))$$

Тъй като h_1 е остойностяващ морфизъм, това е еквивалентно на

$$\mathtt{p}^{\mathbf{M}}(h_1(h_2(\tau_1)), h_1(h_2(\tau_2)), \ldots, h_1(h_2(\tau_n))) \longleftrightarrow \mathtt{p}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

■

4.22. Следствие. *За всяка субституция s и оценка v в структурата $\mathbf{M}$ можем да намерим такава оценка w в $\mathbf{M}$, че:*

– *стойността на кой да е терм τ в $\mathbf{M}$ при оценка w е равна на стойността в $\mathbf{M}$ при оценка v на $\hat{s}(\tau)$;*

– *стойността на коя да е атомарна формула φ в $\mathbf{M}$ при оценка w е еквивалентна на стойността в $\mathbf{M}$ при оценка v на $\hat{s}(\varphi)$;*

– $\bar{w}(\tau) = \bar{v}(\hat{s}(\tau))$, $\bar{w}(\varphi) = \bar{v}(\hat{s}(\varphi))$, $\bar{w} = \bar{v} \circ \hat{s}$.

Доказателство. Съгласно 4.15 прилагането на s е заместващ морфизъм, а намирането на стойността при оценка v — остойностяващ морфизъм. Съгласно 4.21 последователното прилагане на тези морфизми е остойностяващ морфизъм. Съгласно 3.25 можем да намерим такава оценка w, че този остойностяващ морфизъм е равносилен на намиране на стойността в $\mathbf{M}$ при оценката w. ∎

Горното следствие показва, че за произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$ съществува такава оценка w, че да приложим най-напред субституцията s и след това да намерим стойността при оценка v на така получения израз е същото каквото директно да намерим стойността на първоначалния терм или атомарна формула при оценката w.

***Задача* 35:** Дадена е субституция s и оценка v в структура $\mathbf{M}$. Нека w бъде оценката в $\mathbf{M}$, която на всяка променлива $\mathtt{x}$ съпоставя стойността на терма $s(\mathtt{x})$ в $\mathbf{M}$ при оценка v. Докажете, че $\bar{v} \circ \hat{s} = \bar{w}$. С други думи, да приложим субституцията s и след това да намерим стойността на получения израз в $\mathbf{M}$ при оценка v е същото каквото директно да намерим стойността на първоначалния израз в $\mathbf{M}$ при оценка w.

4.23. Твърдение. *Всяка атомарна формула, която има изпълним в дадена структура частен случай, е изпълнима в тази структура.*

Доказателство. Да допуснем, че атомарната формула φ има изпълним в структурата $\mathbf{M}$ частен случай $\hat{s}(\varphi)$. Тогава този частен случай е верен в $\mathbf{M}$ при някоя оценка v, т.е. $\bar{v}(\hat{s}(\varphi))$ е истина. Съгласно 4.22 съществува такава оценка w в $\mathbf{M}$, че $\bar{w}(\varphi) \longleftrightarrow \bar{v}(\hat{s}(\varphi))$. Щом φ е вярна при оценка w, значи е изпълнима в $\mathbf{M}$. ∎

4.24. Твърдение. *Частните случаи на тъждествено вярна атомарна формула или клауза са тъждествено верни.*

Доказателство. Ще докажем твърдението поотделно за атомарна формула и клауза. Да припомним, че с $\bar{v}(\varphi)$ означаваме стойността на φ при оценка v.

(формула) Да допуснем, че атомарната формула φ е тъждествено вярна в структурата $\mathbf{M}$ и да разгледаме нейния частен случай $\hat{s}(\varphi)$. За да установим, че този частен случай е тъждествено вярна в $\mathbf{M}$ атомарна формула, да изберем произволна оценка v в $\mathbf{M}$. Съгласно 4.22

съществува такава оценка w в $\mathbf{M}$, че $\bar{v}(\hat{s}(\varphi)) \longleftrightarrow \bar{w}(\varphi)$. Но φ е тъждествено вярна, следователно $\bar{w}(\varphi)$ е истина и значи $\bar{v}(\hat{s}(\varphi))$ също е истина, което означава, че $\hat{s}(\varphi)$ е вярна при оценка v.

(клауза) Да допуснем, че клаузата $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ е тъждествено вярна в структурата $\mathbf{M}$ и да разгледаме нейния частен случай $\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n) \vdash \hat{s}(\psi)$. За да установим, че този частен случай е тъждествено вярна в $\mathbf{M}$ клауза, да изберем такава оценка v в $\mathbf{M}$, че атомарните формули $\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n)$ да бъдат верни в $\mathbf{M}$ при оценка v. Трябва да установим, че $\hat{s}(\psi)$ е вярна в $\mathbf{M}$ при оценка v.

Съгласно 4.22 съществува такава оценка w в $\mathbf{M}$, че $\bar{v} \circ \hat{s} = \bar{w}$. В частност $\bar{v}(\hat{s}(\varphi_i)) \longleftrightarrow \bar{w}(\varphi_i)$ и $\bar{v}(\hat{s}(\psi)) \longleftrightarrow \bar{w}(\psi)$. Понеже формулите $\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n)$ са верни в $\mathbf{M}$ при оценка v, то $\varphi_1, \varphi_2, \ldots, \varphi_n$ са верни в $\mathbf{M}$ при оценка w. Но клаузата $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ е тъждествено вярна в $\mathbf{M}$, значи атомарната формула ψ е вярна в $\mathbf{M}$ при оценка w, откъдето получаваме, че $\hat{s}(\psi)$ е вярна в $\mathbf{M}$ при оценка v. ∎

4.4. Коректност

Типично за формалните изводимости е те да задават набор от аксиоми, които приемаме за доказуеми без доказателство, и набор от правила, с чиято помощ от вече доказаните неща можем да получаваме нови доказани неща. Доказателствата за коректност пък обикновено протичат по следния начин. Първо доказваме, че аксиомите са верни. След това доказваме, че ако прилагаме правилата към верни неща, ще получаваме пак верни неща. От тук следва, че всичко доказуемо е вярно, т.е. получаваме теоремата за коректност.

Конкретно за либералната права изводимост планът ще бъде следният:

- Вече доказахме, че частните случаи на тъждествено верни атомарни формули и клаузи са тъждествено верни (твърдение 4.24).
- От това ще следва, че ако Γ съдържа тъждествено верни клаузи, а Δ — тъждествено верни атомарни формули, то всяко нещо, което се извежда либерално от Δ посредством Γ е тъждествено вярно (вж. дефиниция 4.8).
- От тук ще получим, че ако Γ съдържа тъждествено верни клаузи, то ЛПИ(Γ) ще съдържа тъждествено верни атомарни формули (вж. дефиниция 4.10).
- От това ще следва, че ако φ се удовлетворява с либерална права изводимост от Γ, то φ се удовлетворява от Γ. Това ще бъде така,

защото ако една атомарна формула има тъждествено верен частен случай, то тя е изпълнима.

4.25. Твърдение. *Ако* Γ *съдържа тъждествено верни в структурата* $\mathbf{M}$ *клаузи, а* Δ *— тъждествено верни в* $\mathbf{M}$ *атомарни формули, то всяка атомарна формула, която се извежда либерално от* Δ *посредством* Γ *е тъждествено вярна в* $\mathbf{M}$*.*

Доказателство. Да допуснем, че ψ се извежда либерално от Δ посредством Γ. По дефиниция това означава, че някоя клауза от Γ има такъв частен случай

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi \tag{24}$$

че предпоставките $\varphi_1, \varphi_2, \ldots, \varphi_n$ са частни случаи на елементи на Δ. Съгласно твърдение 4.24 този частен случай е тъждествено верен в $\mathbf{M}$ и предпоставките $\varphi_1, \varphi_2, \ldots, \varphi_n$ също са тъждествено верни в $\mathbf{M}$.

За да докажем, че ψ е тъждествено вярна в $\mathbf{M}$ атомарна формула, да изберем произволна оценка v. Тъй като $\varphi_1, \varphi_2, \ldots, \varphi_n$ са тъждествено верни, то те са верни в $\mathbf{M}$ при оценка v. Но клауза (24) е тъждествено вярна в $\mathbf{M}$, значи заключението ѝ ψ е вярно в $\mathbf{M}$ при оценка v. ∎

4.26. Твърдение. *Ако* Γ *съдържа тъждествено верни в структурата* $\mathbf{M}$ *клаузи, то* $\text{ЛПИ}_0(\Gamma)$*,* $\text{ЛПИ}_1(\Gamma)$*,* $\text{ЛПИ}_2(\Gamma)$*,* $\text{ЛПИ}_3(\Gamma)$*,. . . и* $\text{ЛПИ}(\Gamma)$ *съдържат тъждествено верни в* $\mathbf{M}$ *атомарни формули.*

Доказателство. С индукция по i ще докажем, че всички елементи на $\text{ЛПИ}_i(\Gamma)$ са тъждествено верни в $\mathbf{M}$ атомарни формули, откъдето ще следва, че и $\text{ЛПИ}(\Gamma)$ съдържа такива формули.

При $i = 0$ няма какво да доказваме, защото $\text{ЛПИ}_0(\Gamma) = \varnothing$.

Да допуснем, че $\text{ЛПИ}_i(\Gamma)$ съдържа тъждествено верни формули. Тогава от твърдение 4.26 следва, че всички формули, които се извеждат либерално от $\text{ЛПИ}_i(\Gamma)$ посредством Γ, са тъждествено верни в $\mathbf{M}$. Следователно $\text{ЛПИ}_{i+1}(\Gamma)$ съдържа тъждествено верни в $\mathbf{M}$ атомарни формули. ∎

4.27. Теорема (коректност на либералната права изводимост). *Ако* φ *се удовлетворява с либерална права изводимост от* Γ*, то* φ *се удовлетворява от* Γ*.*

Доказателство. Нека φ се удовлетворява с либерална права изводимост от Γ и да допуснем, че клаузите в Γ са тъждествено верни в структурата $\mathbf{M}$. Трябва да докажем, че φ е изпълнима в $\mathbf{M}$.

Съгласно дефиниция 4.11 атомарната формула φ има частен случай φ', който е елемент на $\text{ЛПИ}(\Gamma)$. Щом φ' е частен случай на φ, то

съществува такава субституцията s, че $\varphi' = \hat{s}(\varphi)$. От твърдение 4.26 получаваме, че $\hat{s}(\varphi)$ е тъждествено вярна в $\mathbf{M}$.

Нека v е произволна оценка в $\mathbf{M}$. Щом формулата $\hat{s}(\varphi)$ е тъждествено вярна в $\mathbf{M}$, значи в частност тя е вярна при оценка v, откъдето заключаваме, че $\hat{s}(\varphi)$ е изпълнима в $\mathbf{M}$. Всяка формула, която има изпълним частен случай, е изпълнима (вж. твърдение 4.23), следователно φ е изпълнима в $\mathbf{M}$. ∎

Да резюмираме стандартния вид на доказателството за коректност. Най-напред доказваме, че всичко, което можем да докажем на една стъпка, използвайки верни предпоставки, е вярно — това беше твърдение 4.25. От тук получаваме като следствие, че всичко, което можем да докажем, изхождайки от верни неща, е вярно — това беше 4.26. И тъй като при либералната права изводимост доказаното нещо искаме да бъде само изпълнимо, а не тъждествено вярно, то се налага и още една стъпка, която е теорема 4.27. При повечето доказателства за коректност тази последна стъпка отсъства и аналогът на твърдение 4.26 е всичко, което трябва да направим.

4.5. Най-малък ербранов модел и пълнота на правата изводимост

След като доказахме коректността на либералната права изводимост идва ред да докажем и нейната пълнота.

Най-общо казано, пълнотата на една изводимост означава, че всичко вярно може да се докаже. Доказателствата за пълнота в математическата логика най-често се правят по един от следните два начина:

– Дефинираме структура, която има следното специално свойство: нещата, които са верни в тази структура са точно нещата, които са доказуеми. Ако едно нещо е вярно по принцип, то в частност то трябва да бъде вярно и в тази специална структура, откъдето ще получим, че то е доказуемо.

– Доказваме контрапозицията на теоремата на пълнота, а именно: ако нещо не е доказуемо, то съществува структура контрапример, която показва, че то може да не е вярно. За да намерим структурата контрапример, постъпваме по следния начин. Въпреки, че нещото, което доказваме, не е доказуемо, нека се опитаме да го докажем по всички възможни начини. Докато правим това ще установим верността на много твърдения. Ако сме достатъчно настойчиви в усилията си, тези твърдения ще станат толкова много,

че еднозначно ще определят каква трябва да бъде структурата контрапример.

Доказателството за пълнота на либералната права изводимост тук ще направим, използвайки първия от тези два начина. Най-напред трябва да видим каква ще бъде специалната структура, в която са верни точно онези неща, които са доказуеми. Трябва да определим следните неща: *първо*, какъв е универсумът на структурата, *второ*, как се интерпретират символите за константи и функционалните символи и *трето*, как се интерпретират предикатните символи.

Първите две неща можем да ги получим без много да ги мислим благодарение на френския логик Жак Ербран (1908 – 1931). Той установил, че ако едно множество от формули (от определен вид) има модел, то това множество ще има и модел, чийто универсум се състои от всички термове без променливи, а стойността на кой да е терм без променливи в тази структура е самият терм. Такива структури се наричат *ербранови*. На практика какъвто и метод за доказателството на теоремата за пълнота да използваме, е достатъчно да се ограничим с ербранови структури.*

Нека отбележим, че ербрановите структури не винаги съществуват. Съгласно задача 10 ако в сигнатурата няма нито един символ за константи, то няма да съществуват термове без променливи, а универсумите на структурите не е позволено да бъдат празното множество. Ако пък в сигнатурата има поне един символ за константа, то този символ сам си е терм без променливи, така че в този случай ербранови структури ще съществуват.

4.28. Всичко в този раздел ще правим при предположението, че в сигнатурата има поне един символ за константи. Това означава, че тук ще докажем пълнотата само при такава сигнатура. По-нататък ще установим пълнотата на друга изводимост за произволни сигнатури, а от това ще получим като следствие и пълнотата на либералната права изводимост при произволни сигнатури. Всъщност на практика ограничението да имаме поне един символ за константа изобщо не е ограничително. Дори и да нямаме такъв символ, можем да променим сигнатурата като добавим в нея някакъв символ за константа, защото тази промяна няма да доведе до никакви съществени изменения.

*Това е вярно не само за теоремата за пълнота на либералната права изводимост, но също и за произволни теореми за пълнота на логики подобни на предикатната логика от първи ред. А при много от некласическите логики вместо ербранови структури можем да използваме т.н. алгебри на Линденбаум.

4.29. Дефиниция. Структурата $\mathbf{H}$ е *ербранова*, ако:

- Универсумът на $\mathbf{H}$ е множеството от всички термове, които не съдържат променливи.
- За всеки символ за константа $\mathtt{c}$, стойността му е самата константа, т.е. $\mathtt{c}^{\mathbf{H}} = \mathtt{c}$. (Да забележим, че $\mathtt{c}$ е терм без променливи.)
- За всеки n-местен функционален символ $\mathtt{f}$ и елементи $\tau_1, \tau_2, \ldots, \tau_n$ на универсума на $\mathbf{H}$ имаме

$$\mathtt{f}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n) = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_2) \tag{25}$$

  (Да забележим, че щом $\tau_1, \tau_2, \ldots, \tau_n$ са от универсума на $\mathbf{H}$, то те са термове без променливи, а значи и $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_2)$ е терм без променливи, т.е. елемент на универсума.)

4.30. Дефиниция. Множеството от всички термове без променливи ще наричаме *ербранов универсум*.

4.31. Обърнете внимание, че скобите от двете страни на равенство (25) имат напълно различен смисъл. Скобите отляво ги използваме, за да покажем, че даваме $\tau_1, \tau_2, \ldots, \tau_n$ като аргументи на функцията $\mathtt{f}^{\mathbf{H}}$. Скобите отдясно пък са просто символи — вторият и последният символ на терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$. Също така напълно различен е и смисълът на запетаите. Използваме запетаите отляво, за да разделим аргументите на функцията $\mathtt{f}^{\mathbf{H}}$. За разлика от тях, запетаите отдясно са просто символи, които се срещат някъде в терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$.

***Задача* 36:** Нека сигнатурата съдържа поне един символ за константа. Докажете подробно, че съществуват ербранови структури.

Тъй като елементите на универсума на една ербранова структура са термове без променливи, то всяка оценка в тази структура съпоставя на променливите термове без променливи. Това означава, че всяка оценка в ербранова структура представлява субституция и значи можем да я прилагаме по два различни начина:

- можем да пресметнем стойността на един терм при тази оценка;
- можем да си мислим, че оценката е субституция и да я приложим към този терм.

Следващото твърдение показва, че при термовете и в двата случая ще получим един и същ резултат, т.е. $\bar{v}(\tau) = \hat{v}(\tau)$.

4.32. Твърдение. *Нека* $\mathbf{H}$ *е ербранова структура и* v *е оценка в* $\mathbf{H}$. *Тогава стойността на кой да е терм* τ *в* $\mathbf{H}$ *при оценка* v *е равна на резултата от прилагането на субституцията* v *към терма* τ.

Доказателство. Ще докажем твърдението с индукция по терма τ. Да припомним, че с $\bar{v}(\tau)$ означаваме стоността на τ при оценка v в структурата на v, а $\hat{v}(\tau)$ е резултатът от прилагането на субституцията v към τ.

Ако $\tau = \mathtt{x}$ е променлива, то стойността на τ при оценка v е стойността на $\mathtt{x}$ при оценка v, т.е. $v(\mathtt{x})$. Да приложим субституцията v към $\mathtt{x}$ означава да заместим $\mathtt{x}$ с $v(\mathtt{x})$ и значи отново получаваме $v(\mathtt{x})$.

Ако $\tau = \mathtt{c}$ е символ за константа, то по дефиниция 3.15 стойността на τ в $\mathbf{H}$ при коя да е оценка е $\mathtt{c}^{\mathbf{H}} = \mathtt{c}$. Резултатът от прилагането на коя да е субституция към терма $\mathtt{c}$ също е $\mathtt{c}$.

Нека $\tau = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$. Стойността на τ в $\mathbf{H}$ при оценка v е $\bar{v}(\tau) = \mathtt{f}^{\mathbf{H}}(\bar{v}(\tau_1), \bar{v}(\tau_2), \ldots, \bar{v}(\tau_n))$. Понеже структурата е ербранова, последното е равно на $\mathtt{f}(\bar{v}(\tau_1), \bar{v}(\tau_2), \ldots, \bar{v}(\tau_n))$, което съгласно индукционното предположение е равно на $\mathtt{f}(\hat{v}(\tau_1), \hat{v}(\tau_2), \ldots, \hat{v}(\tau_n)) = \hat{v}(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)) = \hat{v}(\tau)$. ∎

4.33. Следствие. *Ако* $\mathbf{H}$ *е ербранова структура и* τ *е терм без променливи, то стойността на* τ *в* $\mathbf{H}$ *при коя да е оценка е* τ*.*

Доказателство. Нека v е произволна оценка в $\mathbf{H}$. Съгласно твърдение 4.32 стойността на τ в $\mathbf{H}$ при оценка v е равна на резултата от прилагането на субституцията v към τ. Но τ не съдържа променливи, които субституциите могат да заместват и значи като приложим v към τ получаваме пак τ. ∎

***Задача* 37:** Нека $\mathbf{H}$ е ербранова структура и оценката v в $\mathbf{H}$ е такава, че $v(\mathtt{x}) = \mathtt{f}(\mathtt{f}(\mathtt{c}))$ и $v(\mathtt{y}) = \mathtt{g}(\mathtt{c}, \mathtt{f}(\mathtt{a}))$. Кои от скобите и кои от запетаите в следния израз са символи и кои не са символи:

$$\mathtt{f}(\mathtt{g}^{\mathbf{H}}(v(\mathtt{x}), \hat{v}(\mathtt{g}(\mathtt{a}, \mathtt{y}))))$$

Колко леви скоби и колко запетаи се съдържат в стойността на този израз?

За атомарните формули е невъзможно да бъде вярно твърдение, аналогично на твърдение 4.32. Това е така, защото ако φ е атомарна формула, а v — оценка в ербранова структура, тогава $\bar{v}(\varphi)$ е стойността на φ при оценка v, а стойността на всяка атомарна формула е твърдение. Същевременно, $\hat{v}(\varphi)$ е атомарна формула, защото когато прилагаме субституция към терм, получаваме терм, когато я прилагаме към атомарна формула получаваме атомарна формула, когато я прилагаме към клауза, получаваме клауза и т.н.

Вместо това можем да докажем следното твърдение:

4.34. Твърдение. *Нека* $\mathbf{H}$ *е ербранова структура,* v *е оценка в* $\mathbf{H}$ *и* φ *е атомарна формула. Тогава формулата* φ *е вярна в* $\mathbf{H}$ *при оценка* v *тогава и само тогава, когато формулата* $\hat{v}(\varphi)$ *е вярна в* $\mathbf{H}$*.*

Доказателство. Нека $\varphi = \mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$. Тъй като структурата е ербранова, съгласно твърдение 4.32 стойността на τ_i в $\mathbf{H}$ при оценка v е $\hat{v}(\tau_i)$. Термовете $\hat{v}(\tau_i)$ не съдържат променливи и значи съгласно твърдение 4.33 тяхната стойност също е $\hat{v}(\tau_i)$. Следователно както атомарната формула $\varphi = \mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$, така и атомарната формула $\hat{v}(\varphi) = \mathtt{p}(\hat{v}(\tau_1), \hat{v}(\tau_2), \ldots, \hat{v}(\tau_n))$ е вярна в $\mathbf{H}$ при оценка v тогава и само тогава, когато е истина $\mathtt{p}^{\mathbf{H}}(\hat{v}(\tau_1), \hat{v}(\tau_2), \ldots, \hat{v}(\tau_n))$. Остава само да отбележим, че съгласно твърдение 3.27, да бъде формулата $\hat{v}(\varphi)$ вярна при оценка v е същото, каквото тя да бъде вярна при коя да е оценка. ■

В уводните бележки към този раздел споменахме, че търсим структура, в която са верни точно доказуемите неща. По-конкретно, имаме нужда от структура със следните свойства:

- Структурата е модел на Γ.
- Една атомарна формула $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е вярна в тази структура тогава и само тогава, когато тази формула е доказуема, т.е. когато тя е елемент на ЛПИ(Γ).

Освен това казахме, че ще търсим ербранова структура от този вид. Това подсказва следната дефиниция (словосъчетанието „най-малък модел“ ще бъде обосновано по-нататък):

4.35. Дефиниция. По дадено множество Γ от клаузи ще дефинираме структура $\mathbf{H}$, наречена *най-малък ербранов модел* на Γ.

Универсум на $\mathbf{H}$ е множеството от всички термове без променливи.

За произволен символ за константа $\mathtt{c}$, нека $\mathtt{c}^{\mathbf{H}} = \mathtt{c}$.

За произволен n-местен функционален символ $\mathtt{f}$, нека

$$\mathtt{f}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n) = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_2)$$

За произволен n-местен предикатен символ $\mathtt{p}$, нека $\mathtt{p}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n)$ е твърдението „$\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е елемент на ЛПИ(Γ)“.

Очевидно така дефинираната структура е ербранова. Следващите твърдения 4.38 и 4.37 показват, че наистина имаме право да я наричаме „най-малък ербранов модел на Γ“. Но тъй като това все още не е установено, засега ще слагаме словосъчетанието „най-малък ербранов модел“ в кавички.

4.36. Лема (характеристична лема за най-малкия ербранов модел). *Нека* $\mathbf{H}$ *е „най-малкият ербранов модел“ на* Γ*,* v *е оценка в* $\mathbf{H}$ *и* φ *е атомарна формула. Тогава*

а) ако φ *не съдържа променливи, то* φ *е вярна в* $\mathbf{H}$ *тогава и само тогава, когато* $\varphi \in$ ЛПИ(Γ)*.*

б) φ *е вярна в структурата* $\mathbf{H}$ *при оценка* v *тогава и само тогава, когато* $\hat{v}(\varphi) \in$ ЛПИ(Γ)*;*

Доказателство. **(а)** Нека $\varphi = \mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$. Съгласно следствие 4.33 стойността на τ_i в $\mathbf{H}$ е τ_i. Следователно φ е вярна в $\mathbf{H}$ (все едно при каква оценка) тогава и само тогава, когато е истина $\mathtt{p}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n)$. Съгласно дефиницията на „най-малкия ербранов модел“ това е така тогава и само тогава, когато $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е елемент на ЛПИ(Γ).

(б) Съгласно твърдение 4.34, формулата φ е вярна в $\mathbf{H}$ при оценка v тогава и само тогава, когато $\hat{v}(\varphi)$ е вярна в $\mathbf{H}$. В (а) доказахме, че това се случва когато $\hat{v}(\varphi)$ е елемент на ЛПИ(Γ). ∎

4.37. Твърдение. *Нека* $\mathbf{H}$ *е „най-малкият ербранов модел“ на* Γ*, а* $\mathbf{M}$ *е някакъв ербранов модел на* Γ*. Тогава всяка атомарна формула без променливи, която е вярна в* $\mathbf{H}$*, е вярна също и в* $\mathbf{M}$*.*

Доказателство. Съгласно лема 4.36 а), ако φ е вярна в $\mathbf{H}$, то $\varphi \in$ ЛПИ(Γ). Тъй като $\mathbf{M}$ е модел на Γ, то съгласно твърдение 4.26 елементите на ЛПИ(Γ) са тъждествено верни в $\mathbf{M}$ и значи в частност и φ е вярна в $\mathbf{M}$. ∎

4.38. Твърдение. *Нека* $\mathbf{H}$ *е „най-малкият ербранов модел“ на* Γ*. Тогава* $\mathbf{H}$ *е ербранов модел на* Γ*.*

Доказателство. Трябва да докажем, че всички елементи на Γ са тъждествено верни в $\mathbf{H}$. За целта да изберем произволна клауза $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ от Γ и да допуснем, че v е оценка в $\mathbf{H}$, при която $\varphi_1, \varphi_2, \ldots, \varphi_n$ са верни. Трябва да покажем, че и ψ е вярна в $\mathbf{H}$ при оценка v.

Съгласно лема 4.36 б), $\hat{v}(\varphi_1), \hat{v}(\varphi_2), \ldots, \hat{v}(\varphi_n)$ са елементи на ЛПИ(Γ). Нека $\hat{v}(\varphi_i)$ е елемент на ЛПИ$_{k_i}(\Gamma)$ и $m = \max\{k_1, k_2, \ldots, k_n\}$. Тогава $\hat{v}(\varphi_1), \hat{v}(\varphi_2), \ldots, \hat{v}(\varphi_n)$ са елементи на ЛПИ$_m(\Gamma)$.

Правилото

$$\hat{v}(\varphi_1), \hat{v}(\varphi_2), \ldots, \hat{v}(\varphi_n) \vdash \hat{v}(\psi)$$

е частен случай на правило от Г, следователно $\hat{v}(\psi) \in \text{ЛПИ}_{m+1}(\Gamma)$ и значи $\hat{v}(\psi) \in \text{ЛПИ}(\Gamma)$. От лема 4.36 б) получаваме, че ψ е вярна в $\mathbf{H}$ при оценка v. ∎

4.39. Теорема (пълнота на либералната права изводимост). *Ако φ се удовлетворява от* Г*, то φ се удовлетворява с либерална права изводимост от* Г*.*

Доказателство. Нека $\mathbf{H}$ е най-малкият ербранов модел на Г. Съгласно твърдение 4.38 структурата $\mathbf{H}$ е модел на Г.

Да допуснем, че φ се удовлетворява от Г. По дефиниция от това следва, че φ е изпълнима в $\mathbf{H}$. Нека v е оценка в $\mathbf{H}$, при която φ е вярна. Съгласно лема 4.36 б) формулата $\hat{v}(\varphi)$ е елемент на ЛПИ(Г) и значи ЛПИ(Г) съдържа частен случай на φ. ∎

4.40. Теорема (коректност и пълнота на либералната права изводимост). *Една атомарна формула се удовлетворява от* Г *тогава и само тогава, когато тя се удовлетворява с либерална права изводимост от* Г*.*

Доказателство. Вж. 4.27 и 4.39. ∎

4.6. Обратна изводимост и пролог

При правата изводимост използваме клаузите от Г, които „знаем“, за да доказваме все повече и повече атомарни формули (елементите на ЛПИ(Г)), докато успеем да получим желаната цел — частен случай на φ. При обратната изводимост започваме не с нещата, които знаем, а с целта, която искаме да докажем, и постепенно я свеждаме до полезни за доказване цели. Целите, които се опитваме да докажем“ не са атомарни формули, а конюнкции от атомарни формули.

В този раздел ще се съобразяваме с част от синтактичните правила на пролог. Клаузата

$$\psi_1, \psi_2, \ldots \psi_n \vdash \varphi$$

ще записваме

$$\varphi :- \psi_1, \psi_2, \ldots \psi_n.$$

Ще използваме два възможни начина за запис за клаузи без предпоставки:

$$\varphi :- .$$

или

$$\varphi.$$

За променливите ще използваме главни букви. Понякога обаче ще изпускаме точката в края на клаузите и запитванията.

4.41. Дефиниция. Израз от вида

$$?\text{-}\varphi_1, \varphi_2, \ldots, \varphi_n.$$

където $\varphi_1, \varphi_2, \ldots, \varphi_n$ е крайна редица от атомарни формули се нарича *запитване*. Когато $n = 0$, наричаме това *празното запитване*.

4.42. Дефиниция. Казваме, че едно запитване е изпълнимо в структура, ако съществува оценка, при която всичките му атомарни формули са верни в тази структура.

4.43. Следствие. *Очевидно празното запитване е изпълнимо във всяка структура.*

***Задача* 38:** Защо е възможно да има запитване, което не е изпълнимо в някоя структура $\mathbf{M}$, но всичките му атомарни формули са изпълними в $\mathbf{M}$?

4.44. Дефиниция. Нека s е субституция. Запитванията от вида

$$?\text{-}\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n).$$

се наричат *частни случаи* на запитването

$$?\text{-}\varphi_1, \varphi_2, \ldots, \varphi_n.$$

4.45. Твърдение. *Ако частен случай на едно запитване е изпълним в структура, то това запитване е изпълнимо в тази структура.*

Доказателство. Да разгледаме запитването

$$?\text{-}\varphi_1, \varphi_2, \ldots, \varphi_n.$$

и да допуснем, че неговият частен случай

$$?\text{-}\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n).$$

е изпълним в структурата $\mathbf{M}$. Тогава съществува такава оценка v в $\mathbf{M}$, че атомарните формули $\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n)$ са верни в $\mathbf{M}$ при оценка v. Съгласно твърдение 4.22 съществува такава оценка w в $\mathbf{M}$, че стойността на формулите $\varphi_1, \varphi_2, \ldots, \varphi_n$ при оценка w е същата, каквато на формулите $\hat{s}(\varphi_1), \hat{s}(\varphi_2), \ldots, \hat{s}(\varphi_n)$ при оценка v, т.е. истина. ∎

4.46. Дефиниция. Нека Γ е множество от клаузи. Ако запитването

$$\texttt{?-}\varphi, \varphi_1, \varphi_2, \ldots, \varphi_n.$$

има частен случай от вида

$$\texttt{?-}\varphi', \varphi'_1, \varphi'_2, \ldots, \varphi'_n.$$

и Γ съдържа частен случай от вида

$$\varphi' :\!- \psi_1, \psi_2, \ldots \psi_m.$$

то казваме, че това запитване се *свежда либерално* посредством Γ към запитването

$$\texttt{?-}\psi_1, \psi_2, \ldots \psi_m, \varphi'_1, \varphi'_2, \ldots, \varphi'_n.$$

Забележка: В тази дефиниция позволяваме $m = 0$. В този случай запитването $\texttt{?-}\varphi, \varphi_1, \varphi_2, \ldots, \varphi_n$ се свежда либерално до $\texttt{?-}\varphi_1, \varphi_2, \ldots, \varphi_n$.

4.47. Пример. Нека Γ се състои от следните клаузи:

$$\texttt{p}(\texttt{c}, X).$$
$$\texttt{p}(X, \texttt{f}(Y)) :\!- \texttt{p}(Y, X).$$

В такъв случай запитването $\texttt{?-p}(X, Y), \texttt{q}(Y, X)$ може да се сведе либерално посредством Γ до всяко едно от следните запитвания:

$$\texttt{?-q}(X, \texttt{c}).$$
$$\texttt{?-q}(Y, \texttt{c}).$$
$$\texttt{?-q}(\texttt{f}(\texttt{f}(Z)), \texttt{c}).$$
$$\texttt{?-p}(Y, X), \texttt{q}(\texttt{f}(Y), X).$$

***Задача* 39:** С какви частни случаи са получени запитванията от пример 4.47?

4.48. Твърдение. *Нека* Γ *е множество от клаузи и структурата* $\mathbf{M}$ *е модел на* Γ *(т.е. клаузите от* Γ *са тъждествено верни в* $\mathbf{M}$*). Ако запитването* ε *се свежда либерално посредством* Γ *до запитването* ε' *и* ε' *е изпълнимо в* $\mathbf{M}$*, то запитването* ε *също е изпълнимо в* $\mathbf{M}$*.*

Доказателство. Нека $\mathbf{M}$ модел на Γ и запитването ε има частен случай от вида

$$\texttt{?-}\varphi', \varphi'_1, \varphi'_2, \ldots, \varphi'_n. \tag{26}$$

Нека още Γ съдържа частен случай от вида

$$\varphi' :- \psi_1, \psi_2, \ldots \psi_m. \tag{27}$$

и запитването ε' има вида

$$\texttt{?-}\psi_1, \psi_2, \ldots \psi_m, \varphi'_1, \varphi'_2, \ldots, \varphi'_n.$$

Щом ε' е изпълнимо в $\mathbf{M}$, то значи при някоя оценка v в $\mathbf{M}$ всичките му атомарни формули са верни. От верността на $\psi_1, \psi_2, \ldots \psi_m$ при оценка v и тъждествената вярност на клауза (27) следва, че φ' е вярна при оценка v. Така получихме, че всички атомарни формули от запитването (26) са верни при оценка v и значи ε има изпълним в $\mathbf{M}$ частен случай. От твърдение 4.45 следва, че запитването ε също е изпълнимо в $\mathbf{M}$. ∎

4.49. Дефиниция. Нека Γ е множество от клаузи.

а) Казваме, че редицата от запитвания $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ е *либерално обратно свеждане* спрямо Γ, ако всеки член на редицата (освен последния) се свежда либерално посредством Γ към следващия член.

б) Казваме, че запитването ε се *удовлетворява с либерална обратна изводимост* от Γ, ако съществува либерално обратно свеждане спрямо Γ, чийто пръв член е ε, а последен член е празното запитване.

в) Казваме, че атомарната формула φ се *удовлетворява с либерална обратна изводимост* от Γ, ако от Γ се удовлетворява запитването $\texttt{?-}\varphi$.

4.50. Теорема за коректност на либералната обратна изводимост. *Ако атомарната формула φ се удовлетворява с либерална обратна изводимост от множество* Γ *от клаузи, то φ се удовлетворява от* Γ.

Доказателство. Щом φ се удовлетворява с либерална обратна изводимост от Γ, то значи имаме такова либерално обратно свеждане $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ спрямо Γ, че ε_1 е запитването $\texttt{?-}\varphi$ и ε_n е празното запитване.

Да допуснем, че структурата $\mathbf{M}$ е модел на Γ. Щом ε_n е празното запитване, то значи запитването ε_n е изпълнимо в $\mathbf{M}$. От твърдение 4.48 следва, че ако някой член на редицата $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ е изпълним в $\mathbf{M}$, то и предходния член е изпълним. Следователно и първият член на тази редица е изпълним в $\mathbf{M}$, т.е. запитването $\texttt{?-}\varphi$ е изпълнимо в $\mathbf{M}$. ∎

Следващата дефиниция ще използваме единствено докато доказваме теоремата за пълнота на либералната обратна изводимост. Идеята на доказателството принадлежи на проф. Димитър Скордев.

4.51. Дефиниция. Нека Γ е множество от клаузи. Казваме, че атомарната формула φ е *отстранима* посредством Γ, ако за всяко запитване от вида

$$\texttt{?-}\varphi, \psi_1, \psi_2, \ldots, \psi_n.$$

(където позволяваме $n = 0$) съществува либерално обратно свеждане от Γ, чийто пръв член е горното запитване, а последен — запитването

$$\texttt{?-}\psi_1, \psi_2, \ldots, \psi_n.$$

4.52. Лема. *Нека* Γ *е множество от клаузи и* φ *е атомарна формула без променливи, която е частен случай на някой елемент на* ЛПИ(Γ). *Тогава* φ *е отстранима посредством* Γ.

Доказателство. С индукция по n ще докажем, че лемата е вярна когато φ е частен случай на елемент на ЛПИ$_n(\Gamma)$.

Когато $n = 0$ няма какво да доказваме, защото ЛПИ$_0(\Gamma) = \varnothing$.

Да допуснем, че лемата е вярна за n и φ е атомарна формула без променливи, която е частен случай на елемент на ЛПИ$_{n+1}(\Gamma)$. Ако φ е частен случай на елемент на ЛПИ$_n(\Gamma)$, то получаваме исканото от индукционното предположение.

В противен случай φ е частен случай на φ', където $\varphi' \in$ ЛПИ$_{n+1}(\Gamma)$ и Γ съдържа такъв частен случай

$$\varphi' \coloneq \chi_1, \chi_2, \ldots, \chi_n.$$

че атомарните формули $\chi_1, \chi_2, \ldots, \chi_n$ са частни случаи на елементи на ЛПИ$_n(\Gamma)$.

Нека s е такава субституция, че $\varphi = \hat{s}(\varphi')$. Без ограничение на общността може да считаме, че s замества всички променливи с термове без променливи.*

Съгласно следствие 4.20 частен случай на частен случай е частен случай, следователно клаузата

$$\hat{s}(\varphi') \coloneq \hat{s}(\chi_1), \hat{s}(\chi_2), \ldots, \hat{s}(\chi_n). \tag{28}$$

*Ако променливата $\texttt{x}$ е такава, че $s(\texttt{x})$ съдържа променлива, то няма как $\texttt{x}$ да се съдържа в φ' (защото $\hat{s}(\varphi')$ не съдържа променливи). Нека $s'(\texttt{x}) = s(\texttt{x})$ когато $s(\texttt{x})$ е терм без променливи, а в противен случай нека $s'(\texttt{x})$ е произволен терм без променливи. Тъй като s и s' съвпадат за променливите, срещащи се в φ', то $\varphi = \hat{s}(\varphi') = \hat{s'}(\varphi')$. Следователно може да използваме s' вместо s.

също е частен случай на елемент на Г. Пак от това следствие получаваме, че атомарните формули $\hat{s}(\chi_1), \hat{s}(\chi_2), \ldots, \hat{s}(\chi_n)$ са частни случаи на елементи на ЛПИ$_n$(Г). Поради вида на s те не съдържат променливи и значи от индукционното предположение получаваме, че са отстраними.

След като установихме тези факти, да изберем произволно запитване от вида

$$\texttt{?-}\varphi, \psi_1, \psi_2, \ldots, \psi_m.$$

Използвайки клауза (28) и вземайки предвид, че $\varphi = \hat{s}(\varphi')$, виждаме, че това запитване се свежда либерално до запитването

$$\texttt{?-}\hat{s}(\chi_1), \hat{s}(\chi_2), \ldots, \hat{s}(\chi_n), \psi_1, \psi_2, \ldots, \psi_m.$$

Вече видяхме, че атомарната формула $\hat{s}(\chi_1)$ е отстранима, затова след няколко последователни либерални свеждания горното запитване може да се сведе до

$$\texttt{?-}\hat{s}(\chi_2), \hat{s}(\chi_3), \ldots, \hat{s}(\chi_n), \psi_1, \psi_2, \ldots, \psi_m.$$

Атомарната формула $\hat{s}(\chi_2)$ също е отстранима и значи това запитване пък можем да сведем до

$$\texttt{?-}\hat{s}(\chi_3), \hat{s}(\chi_4), \ldots, \hat{s}(\chi_n), \psi_1, \psi_2, \ldots, \psi_m.$$

По този начин, една по една можем да отстраним всички формули $\hat{s}(\chi_i)$ и да получим

$$\texttt{?-}\psi_1, \psi_2, \ldots, \psi_m.$$

■

4.53. Теорема за пълнота на либералната обратна изводимост. *Ако атомарната формула φ се удовлетворява от множество* Г *от клаузи, то φ се удовлетворява с либерална обратна изводимост от* Г.

Доказателство. Нека **H** е най-малкият ербранов модел на Г. Тъй като **H** е модел на Г и φ се удовлетворява от Г, то атомарната формула φ е изпълнима в **H**. Нека v е оценка в **H**, при която φ е вярна.

От лема 4.36 б) получаваме, че $\hat{v}(\varphi) \in$ ЛПИ(Г). Атомарната формула $\hat{v}(\varphi)$ не съдържа променливи и значи съгласно лема 4.52 тя е отстранима. В частност, ако започнем със запитването $\texttt{?-}\hat{v}(\varphi)$ и приложим тази лема, за да го отстраним, ще получим либерално свеждане, чийто пръв елемент е запитването $\texttt{?-}\hat{v}(\varphi)$, а последен елемент е празното запитване. Ако в това либерално свеждане заменим първия член с $\texttt{?-}\varphi$ отново ще получим либерално свеждане. Това показва, че φ се удовлетворява с либерална обратна изводимост от Г. ■

4.54. Тъй като вместо либерална, езикът пролог използва нелиберална обратна изводимост, при него има значително по-малко (обикновено краен брой) начина едно запитване да бъде сведено до друго. Но въпреки това, в общия случай едно запитване може, и се свежда до повече от едно запитване. Да разгледаме следния пример. Нека програмата Г се състои от следните клаузи:

$$\texttt{p(a)} :\!- \texttt{p(b)}. \tag{29}$$

$$\texttt{p}(X) :\!- \texttt{q}(X). \tag{30}$$

$$\texttt{q(a)}. \tag{31}$$

Клауза (29) свежда запитването

$$\texttt{?-p(a)} \tag{32}$$

към запитването

$$\texttt{?-p(b)}$$

Единствената клауза, която може да използваме за това запитване, е клауза (30). Посредством нея то се свежда до

$$\texttt{?-q(b)}$$

Никоя клауза не може да се използва за това запитване, така че то остава неудовлетворено.

За запитването (32) имаме още една възможност — да приложим клауза (30). Посредством нея то се свежда до

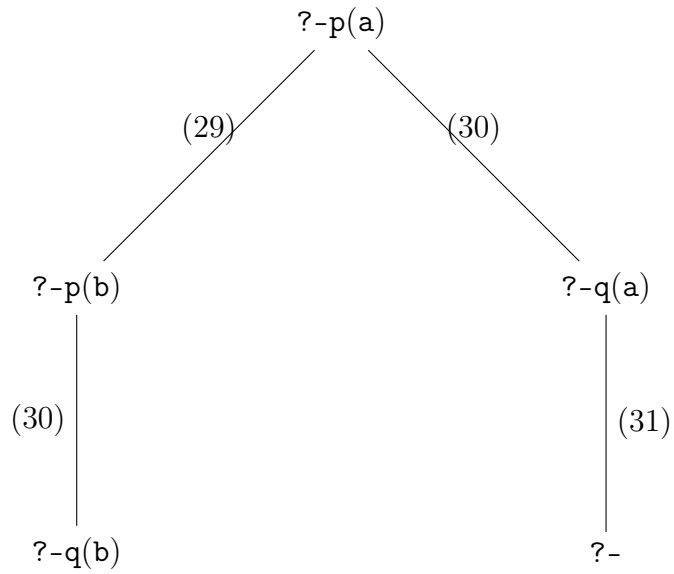
$$\texttt{?-q(a)}$$

Това запитване се удовлетворява от клауза (31).

Всички тези свеждания могат да бъдат изобразени с дърво, както това е направено във фигура 9. Пълнотата на обратната изводимост означава, че ако първоначалната заявка се удовлетворява, то това дърво със сигурност ще съдържа успешен клон (т.е. клон с последен член празното запитване).

Интерпретаторът на пролог обхожда това дърво в дълбочина, започвайки от най-левите клонове. В случая пролог ще обходи най-напред левия клон на дървото, където няма да намери решение на задачата, и след това ще се прехвърли на десния клон.

Когато в дървото има безкраен ляв клон, пролог ще се зацикли без да открие успешен клон. Да разгледаме още един пример. Нека



Фиг. 9

програмата този път се състои от следните клаузи:

$$\mathtt{p}(X) \text{ :- } \mathtt{p}(\mathtt{b}). \tag{33}$$

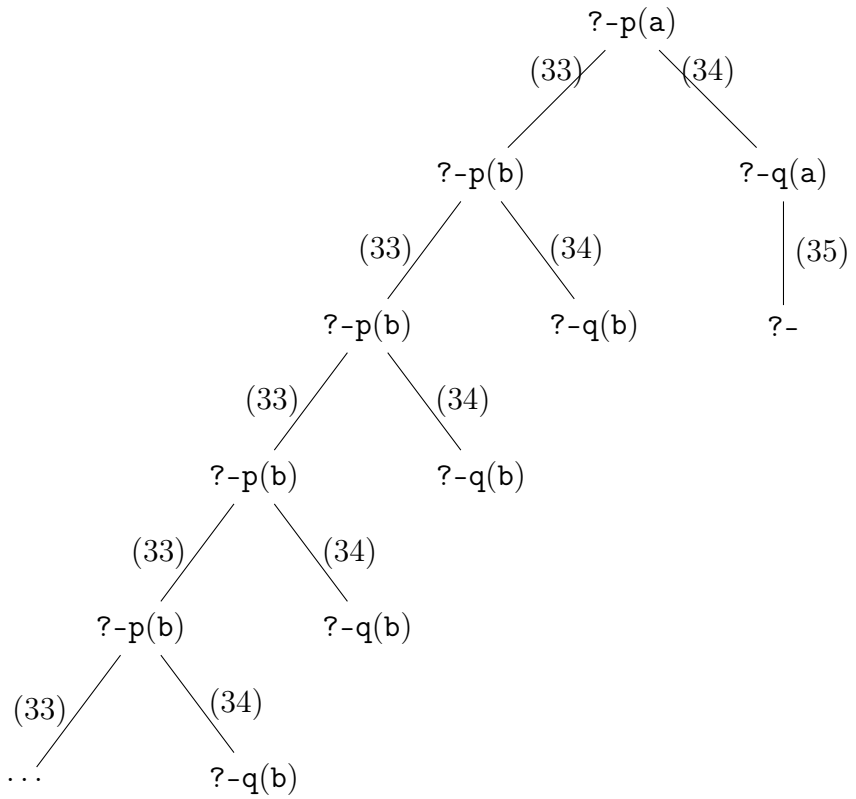
$$\mathtt{p}(X) \text{ :- } \mathtt{q}(X). \tag{34}$$

$$\mathtt{q}(\mathtt{a}). \tag{35}$$

Дървото на извод, което се получава от тази програма при запитване `?-p(a)`, е изобразено на фигура 10. В това дърво има един безкраен ляв клон. Пролог ще тръгне по този клон и никога няма да открие десния и успешен клон.

Всичко това показва, че при пролог е от значение редът, в който са записани клаузите. Когато дървото на извод е крайно, тогава при лошо подреждане на клаузите пролог може да работи значително по-бавно. Например при дървото на фигура 9 пролог ненужно хаби време, за да обхожда левия и неуспешен клон. Много по-лошо е положението, когато дървото е безкрайно. В този случай пролог може до безкрайност да се движи по някой безкраен клон, както се случи при дървото от фигура 10.

В първата от тези програми проблемът се оправя, ако се разменят клаузи (29) и (30), а във втората — като се разменят клаузи (33) и (34). За съжаление обаче, в някои случаи пролог се зацикля без значение в какъв ред подреждаме клаузите.



Фиг. 10

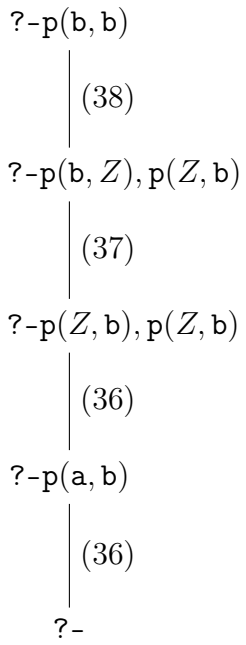
Да разгледаме още един пример (този пример е измислен от проф. Димитър Скордев). Нека програмата се състои от следните клаузи:

$$\mathtt{p}(\mathtt{a},\mathtt{b}). \tag{36}$$

$$\mathtt{p}(X,Y) :\!-\ \mathtt{p}(Y,X). \tag{37}$$

$$\mathtt{p}(X,Y) :\!-\ \mathtt{p}(X,Z),\ \mathtt{p}(Z,Y). \tag{38}$$

По принцип запитването `?-p(b,b)` може да се удовлетвори с обратен извод. Едно успешно свеждане на това запитване до празното запитване е дадено на фигура 11. Въпреки това, както и да подреждаме клаузите в тази програма, винаги ще се оказва, че отляво на този успешен клон, а и изобщо отляво на кой да е успешен клон, в дървото има безкраен клон. Затова при тази програма пролог ще се зацикля без значение как подреждаме клаузите в програмата.



Фиг. 11

4.7. Теорема за компактност и теорема на Ербран за клаузи

Еквивалентността на „вярно“ и „доказуемо“ води до някои непредвидени следствия. Доказателствата са крайни неща и следователно дори и да знаем безброй много твърдения във всяко конкретно доказателство може да използваме само краен брой от тях. Следователно ако Γ съдържа безброй много твърдения и от Γ следва че е вярно (или се удовлетворява) φ то φ ще следва и само от краен брой елементи на Γ. Този интересен резултат се нарича „теорема за компактност“.

4.55. Теорема за компактност за клаузи. *Атомарната формула φ се удовлетворява от множество Γ от клаузи тогава и само тогава, когато съществува такова крайно подмножество Γ' на Γ, че φ се удовлетворява от Γ'.*

Доказателство. Очевидно, че ако φ се удовлетворява от крайно подмножество Γ' на Γ, то φ ще се удовлетворява и от Γ.

Да допуснем, че φ се удовлетворява от Γ. Съгласно теоремата за пълнота на либералната права изводимост (4.39) φ се удовлетворява от Γ с либерална права изводимост. Следователно някой частен случай на φ е елемент на ЛПИ(Γ). Съгласно теоремата за коректност на

либералната права изводимост (4.27) е достатъчно да докажем, че съществува крайно подмножество Γ' на Γ, такъв че този частен случай на φ принадлежи на ЛПИ(Γ').

С индукция по k ще докажем следното твърдение: ако $\psi \in$ ЛПИ$_k(\Gamma)$, то съществува такова крайно подмножество Γ' на Γ, че $\psi \in$ ЛПИ(Γ'). При $k = 0$ няма какво да доказваме, защото ЛПИ$_0(\Gamma) = \varnothing$. Нека твърдението е вярно за k и да допуснем, че $\psi \in$ ЛПИ$_{k+1}(\Gamma)$. Ако $\psi \in$ ЛПИ$_k(\Gamma)$, то получаваме исканото от индукционното предположение. В противен случай ψ следва либерално от ЛПИ$_k(\Gamma)$ посредством Γ. Съгласно дефиниция 4.8 някой елемент на Γ има такъв частен случай

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi \tag{39}$$

че $\varphi_1, \varphi_2, \ldots, \varphi_n$ са частни случаи на елементи на ЛПИ$_k(\Gamma)$. Нека Δ е множеството, което съдържа само оня елемент на Γ, чийто частен случай е клауза (39).

Съгласно индукционното предположение Γ има такива крайни подмножества $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$, че $\varphi_i \in$ ЛПИ(Γ_i). Нека

$$\Gamma' = \Delta \cup \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_n$$

Тогава Γ' е крайно подмножество на Γ и $\varphi_1, \varphi_2, \ldots, \varphi_n \in$ ЛПИ(Γ'). Щом $\varphi_i \in$ ЛПИ(Γ'), то съществува такова число k_i, че $\varphi_i \in$ ЛПИ$_{k_i}(\Gamma')$. Нека $m = \max\{k_1, k_2, \ldots, k_n\}$. Тогава $\varphi_i \in$ ЛПИ$_m(\Gamma')$ и значи $\psi \in$ ЛПИ$_{m+1}(\Gamma')$. ■

***Задача* 40:** Докажете това, за което в първото изречение на горното доказателство се твърди, че е „очевидно“.

4.56. Лема. *Нека* Γ *е множество от клаузи и* Δ *е множеството от всички частни случаи без променливи на клаузи от* Γ. *Ако* $\varphi \in$ ЛПИ(Γ), *то всички частни случаи без променливи на* φ *са елементи на* ЛПИ(Δ).

Доказателство. Ако в сигнатурата няма нито един символ за константа, то φ няма да има нито един частен случай без променливи. Затова, нека считаме, че в сигнатурата има поне един символ за константа.

Щом $\varphi \in$ ЛПИ(Γ), то $\varphi \in$ ЛПИ$_n(\Gamma)$, за някое n. Ще докажем лемата с индукция по n.

При $n = 0$ няма какво да доказваме, защото ЛПИ$_0(\Gamma) = \varnothing$.

Да допуснем, че всички частни случаи без променливи на елементи на ЛПИ$_n(\Gamma)$ са елементи на ЛПИ(Δ). Нека $\varphi \in$ ЛПИ$_{n+1}(\Gamma)$ и $s(\varphi)$ е

произволен частен случай без променливи на φ. Без ограничение на общността може да считаме, че s замества всички променливи с термове без променливи.*

Ако $\varphi \in \text{ЛПИ}_n(\Gamma)$, то получаваме $s(\varphi) \in \text{ЛПИ}(\Delta)$ директно от индукционното предположение.

В противен случай Γ има такъв частен случай

$$\psi_1, \psi_2, \ldots, \psi_n \vdash \varphi \tag{40}$$

че $\psi_1, \psi_2, \ldots, \psi_n$ са частни случаи на елементи на $\text{ЛПИ}_n(\Gamma)$. Клаузата

$$s(\psi_1), s(\psi_2), \ldots, s(\psi_n) \vdash s(\varphi)$$

е частен случай на клауза (40), която пък е частен случай на клауза от Γ. Същевременно тази клауза не съдържа променливи, защото s замества всички променливи с термове без променливи; следователно тя е елемент на Δ. Тъй като формулите ψ_i са частни случаи на елементи на $\text{ЛПИ}_n(\Gamma)$, то предпоставките на тази клауза $s(\psi_i)$ са частни случаи без променливи на елементи на $\text{ЛПИ}_n(\Gamma)$. Съгласно индукционното предположение тези предпоставки са елементи на $\text{ЛПИ}(\Delta)$, а значи и заключението $s(\varphi)$ е елемент на $\text{ЛПИ}(\Delta)$. ∎

4.57. Теорема на Ербран за клаузи. *Нека сигнатурата съдържа поне един символ за константа. Атомарната формула φ се удовлетворява от множество Γ от клаузи тогава и само тогава, когато съществува такова крайно множество Δ от частни случаи без променливи на клаузи от Γ и такъв частен случай без променливи ψ на φ, че ψ се удовлетворява от Δ.*

Доказателство. Обратната посока на твърдението е очевидна, защото ако някой частен случай на φ се удовлетворява от множество от частни случаи на елементи на Γ, то φ ще се удовлетворява от Γ.

За да докажем правата посока, да допуснем, че φ се удовлетворява от Γ. Тогава φ се удовлетворява от Γ с либерална права изводимост. По дефиниция това означава, че съществува частен случай φ' на φ, който принадлежи на $\text{ЛПИ}(\Gamma)$.

Нека Γ' е множеството от всички частни случаи без променливи на клаузи от Γ. От предната лема получаваме, че φ' има частен случай

*Ако променливата $\mathtt{x}$ е такава, че $s(\mathtt{x})$ съдържа променлива, то няма как $\mathtt{x}$ да се съдържа в φ (защото $\hat{s}(\varphi)$ не съдържа променливи). Нека $s'(\mathtt{x}) = s(\mathtt{x})$ когато $s(\mathtt{x})$ е терм без променливи, а в противен случай нека $s'(\mathtt{x})$ е произволен терм без променливи. Тъй като s и s' съвпадат за променливите, срещащи се в φ, то $\varphi = \hat{s}(\varphi) = \hat{s'}(\varphi)$. Следователно може да използваме s' вместо s.

без променливи ψ, който е елемент на ЛПИ(Γ'). Последното означава, че ψ се удовлетворява от Γ' с либерална права изводимост. От теоремата за компактност за клаузи заключаваме, че ψ се удовлетворява от някое крайно подмножество Δ на Γ'. Остава само да отбележим, че частен случай на частен случай е частен случай и значи ψ е частен случай на φ. ■

***Задача* 41:** Докажете обратната посока на теоремата на Ербран, за която в горното доказателство се твърди, че е „очевидна“.

Тук няма да обосноваваме това, но да проверим дали някоя атомарна формула без променливи φ се удовлетворява от краен брой клаузи без променливи Γ е задача, която може да се реши напълно алгоритмично. Един примерен метод е следният: за всяка атомарна формула, която се среща в клауза от Γ разглеждаме две възможности — да бъде вярна или да бъде невярна. Ако открием комбинация, при която клаузите от Γ стават верни, но φ не е сред верните атомарни формули, значи φ не се удовлетворява от Γ. Ако пък при всички комбинации, при които клаузите от Γ са верни, формулата φ също е вярна, тогава φ се удовлетворява от Γ.

Това подсказва следния метод за полупроверка дали φ се удовлетворява от множество Γ от клаузи: Генерираме всички крайни множества от частни случаи без променливи на елементи на Γ и генерираме всички частни случаи без променливи на φ. За всяко генерирано крайно множество Δ от клаузи без променливи и частен случай ψ без променливи проверяваме дали ψ се удовлетворява от Δ. Ако да — чудесно, защото установихме, че φ се удовлетворява от Γ. Ако не — продължаваме да генерираме до безкрайност множества Δ и формули ψ.

Също както либералната права изводимост, и този метод е много неефективен поради огромния брой частни случаи, които се получават. Тези частни случаи не са толкова много, колкото при либералната изводимост, защото се правят само за изходните клаузи и са винаги без променливи, но въпреки това методът в този си вид на практика е неизползваем. Струваше си обаче да го спомена, защото на такъв принцип е работила първата компютърна програма за автоматично доказателство на теореми за предикатната логика от Пол Гилмор [7] (1960 г.).

Има обаче един специален случай, когато този метод е сравнително ефективен и при това никога не се зацикля. Ако сигнатурата е такава, че в нея няма функционални символи, а само краен брой символи за константи, тогава единствените термове без променливи са символите за константи, които са краен брой. Това означава, че всяка клауза

от Γ има само краен брой частни случаи без променливи и ако Γ е крайно, то и множеството от всички частни случаи без променливи на елементи на Γ също е крайно. Нека това множество е Δ. Тогава φ се удовлетворява от Γ тогава и само тогава, когато някой частен случай без променливи на φ (такива също има краен брой) се удовлетворява от Δ. Именно такава ситуация възниква, когато използваме логическо програмиране за заявки към база данни — тогава няма функционални символи и това гарантира, че заявката ще се изпълни за крайно време.

Глава 5

Предикатна логика от първи ред

5.1. Свободни и свързани променливи

Променливите в математиката и програмирането са два вида — свободни и свързани. Двата вида променливи са много различни един от друг. Да разгледаме израза

$$\sum_{i=1}^{n} i^2$$

За да пресметнем този израз е нужно да знаем каква е стойността на променливата n, но не и стойността на променливата i. Въпреки, че в този израз се срещат две променливи — n и i – очевидно тези променливи са много различни. Ако искаме с горния израз да дефинираме функция, тази функция ще зависи само от n, но не и от i:

$$f(n) = \sum_{i=1}^{n} i^2$$

Променливите, подобни на n, се наричат *свободни променливи*, а подобните на i се наричат *свързани променливи*.

Ето още няколко примера. В следващия израз променливата x е свързана, а y е свободна:

$$\int_0^1 f(x, y)dx$$

В следния програмен блок променливата `i` е свързана, а променливите `a` и `x` са свободни:

```
{
   int i;
   for(i=1;i<1000;i++){
      x[i] = (x[i-1]*x[i-1]) % a;
   }
}
```

И така, да обобщим — стойността (смисъла) на един израз зависи от стойността на свободните променливи, срещащи се в него, но не зависи от стойността на свързаните променливи в израза.

Едно друго различие между свободните и свързаните променливи е следното. Ако в един израз заменим всяко срещане на свободната променлива x с някакъв израз, ще се получи пак смислен израз. Да заменяме свързана променлива с израз не е позволено.

Например ако в израза

$$\sum_{i=1}^{n} i^2$$

заменим n с k^2+5 получаваме израза

$$\sum_{i=1}^{k^2+5} i^2$$

Ако в израза

$$\int_0^1 f(x,y)dx$$

заменим y с z^2+5 получаваме израза

$$\int_0^1 f(x,z^2+5)dx$$

Ако в горния програмен блок заменим променливата `a` с израза `a*a+5` получаваме новия програмен блок:*

*При подобни замени трябва да се спазват типовете. Например ако променливата `a` се срещаше отляво на оператор за призвояване, тогава бихме могли да заменяме тази променлива само с израз, притежаващ определен адрес в паметта. В този случай замяната на `a` с `a*a+5` не би била коректна, но замяната с `x[4]` ще бъде коректна.

```
{
   int i;
   for(i=1;i<1000;i++){
      x[i] = (x[i-1]*x[i-1]) % (a*a+5);
   }
}
```

Ако по подобен начин заменяме свързаните променливи с изрази, обикновено се получават безсмислици. Например ако в израза

$$\sum_{i=1}^{n} i^2$$

заменим i с k^2+5 получаваме

$$\sum_{(k^2+5)=1}^{n} (k^2+5)^2$$

Ако в израза

$$\int_0^1 f(x,y)dx$$

заменим x с z^2+5 получаваме

$$\int_0^1 f(z^2+5,y)d(z^2+5)$$

Ако в горния програмен блок заменим `i` с `a*a+5` получаваме

```
{
   int a*a+5;
   for(a*a+5=1;a*a+5<1000;(a*a+5)++){
      x[i] = (x[i-1]*x[i-1]) % (a*a+5);
   }
}
```

Да забележим, че във всеки от горните случаи свързаната променлива има нещо като „свързващ оператор“, от който съвсем ясно личи, че се получава безсмислица. Това са $\sum_{(k^2+5)=1}^{n}$, $\int_0^1 \ldots d(z^2+5)$ и декларацията `int a*a+5;`.

При преименуване на свързана променлива смисълът на израза се запазва. Например

$$\sum_{i=1}^{n} i^2 = \sum_{j=1}^{n} j^2 \neq \sum_{i=1}^{m} i^2$$

$$\int_0^1 f(x,y)dx = \int_0^1 f(z,y)dz \neq \int_0^1 f(x,z)dx$$

Също и в горния програмен блок смисълът се запазва, ако преименуваме `i` на `j`, но не и ако преименуваме `a` на `b`.

Една друга разлика между свободните и свързаните променливи е това, че свободните променливи имат глобална видимост, а свързаните — локална. Да разгледаме следния програмен фрагмент:

```
{
   int i, j;
   i = a * a;
   {
      int i;
      i = b + 5;
      j = 3;
   }
   {
      int i, a;
      i = b + j;
      a = i * i;
      b = a * i;
   }
}
```

В този фрагмент са декларирани три променливи с име `i`. Въпреки че използват едно и също име, тези три променливи са напълно различни една от друга. Освен това всяка си има своя област на видимост. Променливите `i`, декларирани във вътрешните блокове, са видими само в рамките на тези блокове. Променливата `i`, декларирана във външния блок, също е видима само в рамките на този блок, но не и извън него. Освен това вътрешните променливи `i` скриват външната променлива `i`, в следствие на което тя не се вижда във вътрешните блокове. Променливата `j` обаче не се скрива от вътрешна декларация и затова се вижда и във вътрешните блокове.

В този програмен фрагмент се използват и две свободни променливи — `a` и `b`. Всяко срещане на променлива `b` в този програмен фрагмент се отнася за една и съща променлива. Във втория вътрешен блок обаче е декларирана свързана променлива `a`. Тази свързана променлива `a` скрива свободната променлива `a`.

При свързаните променливи може да има различни свързани променливи с едно и също име, но при свободните това е невъзможно — едноименните свободни променливи винаги са една и съща променлива.

Казаното дотук за свободните и свързаните променливи е обобщено в таблица 2.

свободни променливи	***свързани променливи***
изразът зависи от стойността им	стойността им не е релевантна
може да се заместват с изрази	не може да се заместват
не може да се преименуват	може да се преименуват
глобална видимост	локална видимост

Таблица 2. Свойства на свободните и свързаните променливи

5.2. Формули

Далеч не всяко математическо твърдение може да бъде формулирано, използвайки клаузи. *Предикатната логика от първи ред* е логически език с много по-голяма изразителна сила, отколкото езикът на клаузите. Това е така поради следните две причини:

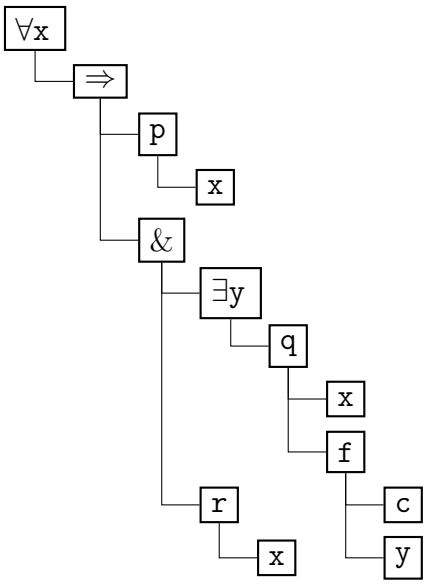
- във формулите може да използваме произволни логически операции — конюнкция ($\&$), дизюнкция ($\vee$), отрицание ($\neg$), импликация ($\Rightarrow$) и еквиваленция ($\Leftrightarrow$);
- във формулите може да се използват така наречените квантори — *кванторът за всеобщност* $\forall$, наречен още *универсален квантор* и *кванторът за съществуване* $\exists$, наречен още *екзистенциален квантор*.

5.1. При записване на формули ще изпускаме някои от скобите. Например може да считаме, че всяка формула от вида $\varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$ е съкратен запис на формулата $(((\varphi_1 \vee \varphi_2) \vee \varphi_3) \vee \varphi_4)$. Приемаме, че импликацията ($\Rightarrow$) и еквиваленцията ($\Leftrightarrow$) са с най-малък приоритет, конюнкцията ($\&$) и дизюнкцията ($\vee$) са със среден (и равен по между си) приоритет и унарните операции ($\neg$, $\forall$ и $\exists$) са с най-голям приоритет. Например всяка формула от вида $\varphi_1 \,\&\, \varphi_2 \Rightarrow \varphi_3 \vee \varphi_4$ е съкратен запис на формулата $((\varphi_1 \,\&\, \varphi_2) \Rightarrow (\varphi_3 \vee \varphi_4))$.

5.2. Също както термовете, атомарните формули и клаузите, така и формулите могат да бъдат считани както за редици от символи, така и за дървета. Например формулата

$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\; \mathtt{q}(\mathtt{x}, \mathtt{f}(\mathtt{c}, \mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x}))$$

може да бъде представена като дърво както това е показано във фигура 12.



Фиг. 12. Синтактично дърво за формулата .
$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x}, \mathtt{f}(\mathtt{c}, \mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x}))$

***Задача* 42:** Кое е дървото на следната формула:

$$\exists \mathtt{x}\,(\neg \forall \mathtt{y}\,\mathtt{q}(\mathtt{x}, \mathtt{f}(\mathtt{c}, \mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x}))$$

5.3. Когато се дава точната дефиниция на понятието „формула“, не е нужно в нея да се споменават изрично всички логически операции. Например в дефиниция 5.5 няма да кажем нищо за еквиваленцията. Това не създава проблеми, защото можем да считаме, че всяка формула от вида $\varphi \Leftrightarrow \psi$ представлява съкратен запис на формулата $((\varphi \Rightarrow \psi) \,\&\, (\psi \Rightarrow \varphi))$.

5.4. Пример. Нека $=$ и $\in$ са двуместни предикатни символи, а $\mathtt{x}$, $\mathtt{y}$ и $\mathtt{z}$ са променливи. Тогава изразът[*]

$$\forall \mathtt{x}\,\forall \mathtt{y}\,(\forall \mathtt{z}\,(\mathtt{z} \in \mathtt{x} \Leftrightarrow \mathtt{z} \in \mathtt{y}) \Rightarrow \mathtt{x} = \mathtt{y})$$

е съкратен запис на формулата

$$\forall \mathtt{x}\,\forall \mathtt{y}\,(\forall \mathtt{z}\,((\in(\mathtt{z}, \mathtt{x}) \Rightarrow \in(\mathtt{z}, \mathtt{y})) \,\&\, (\in(\mathtt{z}, \mathtt{y}) \Rightarrow \in(\mathtt{z}, \mathtt{x}))) \Rightarrow =(\mathtt{x}, \mathtt{y}))$$

Да си припомним, че дефиницията на терм (вж. дефиниция 2.4) бе индуктивна. В нея от по-прости термове конструирame по-сложни. Дефиницията на формула също ще бъде индуктивна. В нея от по-прости формули конструираме по-сложни.

[*]В теория на множествата този израз се нарича аксиома за екстенсионалност или аксиома за обемност.

5.5. Дефиниция. Нека е дадена сигнатура $\mathfrak{sig}$. *Формулите* при сигнатура $\mathfrak{sig}$ се дефинират индуктивно:

а) ако φ е атомарна формула при сигнатура $\mathfrak{sig}$, то φ е формула при сигнатура $\mathfrak{sig}$;

б) ако φ и ψ са формули при сигнатура $\mathfrak{sig}$, то $(\varphi \,\&\, \psi)$, $(\varphi \vee \psi)$ и $(\varphi \Rightarrow \psi)$ са формули при сигнатура $\mathfrak{sig}$;

в) ако φ е формула при сигнатура $\mathfrak{sig}$, то $\neg\varphi$ е формула при сигнатура $\mathfrak{sig}$;

г) ако $\mathtt{x}$ е променлива от сигнатурата $\mathfrak{sig}$ и φ е формула при сигнатура $\mathfrak{sig}$, то $\forall\mathtt{x}\,\varphi$ и $\exists\mathtt{x}\,\varphi$ са формули при сигнатура $\mathfrak{sig}$.

5.6. Дефиниция. Когато формулата ψ е част от формулата φ, казваме, че ψ е *подформула* на φ. Когато формулата φ съдържа подформула от вида $\forall\mathtt{x}\,\psi$ или $\exists\mathtt{x}\,\psi$ (където ψ е формула), казваме, че тази подформула е *областта на действие* на квантора $\forall\mathtt{x}$ или $\exists\mathtt{x}$, с който започва подформулата.

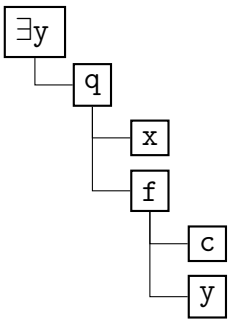
5.7. За да разберем правилно смисълът на току-що дадената дефиниция, е най-добре да си мислим формулите като дървета, а не като редици от символи. Ако си мислим формулата като дърво, някои от поддърветата ще бъдат подформули, а други (по-малките) — термове. Подформулите на една формула са всички поддървета, които са формули, а не термове. Областта на действие на всеки квантор е поддървото, което се намира под съответния квантор. Например областта на действие на квантора $\forall\mathtt{y}$ от формулата на фигура 12 е подформулата, илюстрирана на фигура 13.

Когато си мислим формулите не като дървета, а като редици от символи, тогава дефиниция 5.6 ще бъде вярна само тогава, когато изписваме формулата изцяло, без да пропускаме нито едни скоби. Да разгледаме отново формулата

$$\forall\mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists\mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x}))$$

Според дефиниция 5.6, ако една подформула започва с квантор, тогава тази подформула е областта на действие на този квантор. Една подформула на горната формула е формулата $\exists\mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x})$. Тази формула започва с квантора $\exists\mathtt{y}$ и значи тя трябва да е областта на действие на този квантор. Достатъчно е обаче да погледнем дървото от фигура 12, за да видим, че това е невярно!

Тази грешка се обяснява по следния начин — не е вярно, че формулата $\exists\mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x})$ започва с квантор. Ако я напишем изцяло,



Фиг. 13. Синтактично дърво на областта на действие на квантора ∃y от формулата на фигура 12.

без да пропускаме скоби, тогава ще видим че тази формула започва не с квантор, а със скоба, защото дефиниция 5.5 б) изисква около всяка конюнкция да се слагат скоби:

$$(\exists \mathtt{y}\, \mathtt{q}(\mathtt{x}, \mathtt{f}(\mathtt{c}, \mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x}))$$

Тъй като е много неудобно всеки път да преценяваме къде се слагат всички нужни скоби и дали дадена подформула започва с квантор, или със скоба, която не е написана, най-добре е когато откриваме подформулите на една формула, да си мислим тази формула като дърво, а не като редица от символи.

***Задача* 43:** Намерете всички подформули, на формулата от фигура 12.

5.8. В клаузите и атомарните формули няма свързани променливи. В следствие на кванторите обаче, във формулите на предикатната логика от първи ред може да има както свързани, така и свободни променливи. Една променлива x е свързана, ако попада в областта на действие на някой квантор ∀x или ∃x , а свободни са променливите, които се срещат на места, които не са в областта на действие на такъв квантор.

Много е важно да се разберат следващите примери.

5.9. Пример. Всички променливи, които се срещат в безкванторна формула са нейни свободни променливи. Например променливите x, y и z са свободните променливи на формулата

$$(\mathtt{p}(\mathtt{x}, \mathtt{c}) \vee \neg \mathtt{q}(\mathtt{y})) \Rightarrow \mathtt{p}(\mathtt{z}, \mathtt{x})$$

Атомарните формули са вид безкванторни формули, следователно всички променливи, които се срещат в една атомарна формула са нейни свободни променливи.

5.10. Пример. Променливите $\mathtt{x}$ е единствената свободна променлива на формулата

$$\forall \mathtt{x}\, \forall \mathtt{y}\, (\mathtt{p}(\mathtt{x}, \mathtt{c}) \vee \neg \mathtt{q}(\mathtt{y})) \Rightarrow \forall \mathtt{z}\, \mathtt{p}(\mathtt{z}, \mathtt{x})$$

В тази формула променливата $\mathtt{x}$ от подформулата $\mathtt{p}(\mathtt{x}, \mathtt{c})$ е свързана, защото попада в областта на действие на квантора $\forall \mathtt{x}$. Същевременно променливата $\mathtt{x}$ от подформулата $\mathtt{p}(\mathtt{z}, \mathtt{x})$ е свободна, защото не попада в областта на действие на квантор $\forall \mathtt{x}$ или $\exists \mathtt{x}$.

Също както в езиците за програмиране декларирането на локална променлива в даден програмен блок скрива едноименните променливи, декларирани в по-външен блок, така и кванторите скриват едноименните свободни променливи, както и свързаните променливи с по-външен квантор.

5.11. Пример. Във формулата

$$\forall \mathtt{x}\, (\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}) \vee \mathtt{p}(\mathtt{x})) \vee \mathtt{p}(\mathtt{x})$$

в първата атомарна формула $\mathtt{p}(\mathtt{x})$ променливата $\mathtt{x}$ е свързана с втория квантор, във втората атомарна формула $\mathtt{p}(\mathtt{x})$ променливата $\mathtt{x}$ е свързана с втория квантор, а в третата — променливата $\mathtt{x}$ е свободна.

За понятието „свободна променлива на формула“ ще ни трябва точна математическа дефиниция.

5.12. Дефиниция. а) Променливата $\mathtt{x}$ е *свободна променлива* на формулата φ, ако $\mathtt{x}$ се среща в φ на място, което не попада в областта на действие на никой квантор $\forall \mathtt{x}$ или $\exists \mathtt{x}$.

б) Когато променливата $\mathtt{x}$ се среща във формула φ на място, което не попада в областта на действие на никой квантор $\forall \mathtt{x}$ или $\exists \mathtt{x}$, за това срещане на $\mathtt{x}$ в φ казваме, че е *свободно срещане* на $\mathtt{x}$ в φ.

***Задача* 44:** За всяка променлива, срещаща се във формулата

$$\forall \mathtt{x}\, (\forall \mathtt{x}\, \exists \mathtt{y}\, \mathtt{q}(\mathtt{x}, \mathtt{f}(\mathtt{c}, \mathtt{y})) \,\&\, \mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x}, \mathtt{y}))$$

определете дали е свободна или свързана. Кои са кванторите на свързаните променливи? Коя е областта на действие на всеки един от кванторите? Кое е множеството от свободните променливи на тази формула?

В уводните бележки на тази глава бе споменато, че свързаните променливи в един израз може да се преименуват без това да промени смисъла на израза. Ако формулата φ може да се сведе до формулата ψ посредством преименуване на свързаните променливи, казваме, че тези две формули са *конгруентни* (точната дефиниция на това понятие е дадена малко по-долу).

5.13. Пример. Формулата $\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}, \mathtt{y})$ е конгруентна с формулата $\forall \mathtt{z}\, \mathtt{p}(\mathtt{z}, \mathtt{y})$ и се получава като преименуваме $\mathtt{x}$ на $\mathtt{z}$. Тя обаче не е конгруентна с формулата $\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}, \mathtt{z})$, защото променливата $\mathtt{y}$ е свободна и не може да се преименува.

Ако във формулата $\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}, \mathtt{y})$ преименуваме свързаната променлива $\mathtt{x}$ на $\mathtt{y}$, получаваме формулата $\forall \mathtt{y}\, \mathtt{p}(\mathtt{y}, \mathtt{y})$. Тези две формули обаче не са конгруентни, защото новопоявилият се квантор $\forall \mathtt{y}$ обхваща и свободната променлива $\mathtt{y}$ и я превръща от свободна в свързана.

5.14. Дефиниция. а) Нека формулата φ съдържа подформула от вида $\forall \mathtt{x}\, \psi$ или $\exists \mathtt{x}\, \psi$ и променливата $\mathtt{y}$ не се среща никъде в тази подформула (нито като свързана, нито като свободна). Ако в тази подформула заменим всички променливи $\mathtt{x}$ с $\mathtt{y}$, казваме, че новата формула е получена посредством *еднократно преименуване на свързана променлива* във φ.

б) Ако във формулата φ извършваме последователно няколко пъти (може и нула) еднократни преименувания на свързани променливи, то казваме, че получената накрая формула ψ е получена от φ посредством *преименуване на свързаните променливи*.

в) Формулата φ е *конгруентна* с формулата ψ, ако ψ може да се получи от φ посредством преименуване на свързаните променливи. Когато φ е конгруентна с ψ, записваме това така:

$$\varphi \equiv \psi$$

Нормално е, когато преименуваме променливата на даден квантор, да преименуваме само променливите, които се „управляват“ от този квантор. Например ако във формулата

$$\forall \mathtt{x}\, (\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}) \vee \mathtt{p}(\mathtt{x})) \vee \mathtt{p}(\mathtt{x}) \tag{41}$$

решим да преименуваме първия квантор на $\forall \mathtt{y}$, тогава трябва да получим

$$\forall \mathtt{y}\, (\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}) \vee \mathtt{p}(\mathtt{y})) \vee \mathtt{p}(\mathtt{x}) \tag{42}$$

Въпреки това, ако извършим преименуването така, както е определено в дефиниция 5.14 а), ще получим

$$\forall \mathtt{y}\,(\forall \mathtt{y}\,\mathtt{p}(\mathtt{y}) \vee \mathtt{p}(\mathtt{y})) \vee \mathtt{p}(\mathtt{x}) \tag{43}$$

Дефиницията е дадена по този начин, за да опростим формулировката ѝ и някои от доказателствата, но всъщност това по никакъв начин не е променило кои формули ще се окажат конгруентни и кои не. Наистина, нищо не ни пречи от формула (41) да получим най-напред формула (43), а след това с още едно преименуване от формула (43) да получим формула (42).

5.15. Твърдение. *а)* $\varphi \equiv \varphi$

б) ако $\varphi \equiv \psi$*, то* $\psi \equiv \varphi$

в) ако $\varphi \equiv \psi$ *и* $\psi \equiv \chi$*, то* $\varphi \equiv \chi$

Доказателство. **(а)** φ се получава от φ посредством нула на брой еднократни преименувания на свързани променливи.

(б) Да забележим, че ако можем да извършим еднократно преименуване на променливата $\mathtt{x}$ на $\mathtt{y}$, нищо не ни пречи да извършим преименуването и в обратната посока — от $\mathtt{y}$ на $\mathtt{x}$. Следователно ако ψ е получена от φ с помощта на няколко еднократни преименувания на свързани променливи, то нищо не ни пречи да получим φ от ψ като прилагаме преименуванията наобратно и в обратен ред.

(в) Ако от φ с няколко преименувания получим ψ и после с още няколко — χ, то значи от φ с няколко преименувания можем да получим χ. ∎

***Задача* 45:** Конгруентни ли са формулите:

1. $\forall \mathtt{x}\,\forall \mathtt{y}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$ и $\forall \mathtt{x}\,\forall \mathtt{z}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$
2. $\forall \mathtt{x}\,\forall \mathtt{y}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$ и $\forall \mathtt{y}\,\forall \mathtt{x}\,\forall \mathtt{x}\,\mathtt{p}(\mathtt{y},\mathtt{x})$
3. $\forall \mathtt{x}\,\forall \mathtt{y}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$ и $\forall \mathtt{y}\,\forall \mathtt{y}\,\forall \mathtt{x}\,\mathtt{p}(\mathtt{y},\mathtt{x})$

В теорията на предикатната логика от първи ред ще искаме да отъждествяваме конгруентните формули, т.е. да работим „с точност до конгруентност“. Това означава, че трябва да бъдат верни твърдения, подобни на следващото.

5.16. Твърдение. *Ако формулите* φ *и* φ' *са конгруентни, то те имат едни и същи свободни променливи.*

Доказателство. Достатъчно е да докажем, че ако една формула е получена посредством еднократно преименуване на свързана променлива, то тя има същите свободни променливи, както и първоначалната формула. Нека формулата φ има подформула от вида $\forall \mathtt{x}\, \psi$, променливата $\mathtt{y}$ не се среща никъде в тази подформула и ψ' е получена като заменим в ψ всички срещания на $\mathtt{x}$ с $\mathtt{y}$. Нека формулата φ' се получава от φ като заменим подформулата $\forall \mathtt{x}\, \psi$ с $\forall \mathtt{y}\, \psi'$. В такъв случай формулата φ изглежда по следния начин (областта на действие на квантора $\forall \mathtt{x}$ е подчертана):

$$\varphi = \ldots\ldots\ldots \underline{\forall \mathtt{x}\, \psi} \ldots\ldots\ldots$$

а формулата φ' изглежда така (областта на действие на квантора $\forall \mathtt{y}$ е подчертана):

$$\varphi' = \ldots\ldots\ldots \underline{\forall \mathtt{y}\, \psi'} \ldots\ldots\ldots$$

Ако променливата $\mathtt{z}$ е различна както от $\mathtt{x}$, така и от $\mathtt{y}$, и има свободно срещане в φ, то $\mathtt{z}$ ще се среща свободно и в φ'. И обратно, ако $\mathtt{z}$ се среща свободно в φ', то $\mathtt{z}$ ще се среща свободно и в φ'. Това е така, защото разликите между φ и φ' са свързани единствено с променливите $\mathtt{x}$ и $\mathtt{y}$ и по никакъв начин не влияят на променливата $\mathtt{z}$ и кванторите $\forall \mathtt{z}$ и $\exists \mathtt{z}$.

Всички променливи $\mathtt{x}$ в подформулата $\forall \mathtt{x}\, \psi$ са свързани, а променливата $\mathtt{y}$ не се среща в тази подформула. Освен това променливата $\mathtt{x}$ не се среща в подформулата $\forall \mathtt{y}\, \psi'$, а всички променливи $\mathtt{y}$ в тази подформула са свързани. Това означава, че всички свободни срещания на $\mathtt{x}$ или $\mathtt{y}$ в φ или φ' са извън тези две подформули, на мястото, означено по-горе с точки. Тъй като при преименуването не правим промени на местата, означени по-горе с точки, то $\mathtt{x}$ или $\mathtt{y}$ се среща свободно в φ, тя ще се среща свободно и в φ' и ако се среща свободно в φ', то тя ще се среща свободно и в φ.

Разгледахме случая на преименуване в подформула от вида $\forall \mathtt{x}\, \psi$. Случаят на подформула $\exists \mathtt{x}\, \psi$ е аналогичен. ∎

5.17. Лема за преименуване на свързаните променливи. *За всяка формула φ и крайно множество Θ от променливи съществува формула, конгруентна на φ, в която не се срещат променливи от Θ, всички квантори са с различни променливи и никоя от кванторните променливи не е свободна.*

Доказателство. Нека Θ' е обединението на Θ с множеството от всички променливи (свързани и несвързани), които се срещат в φ. Множеството Θ' е крайно.

Ще построим редица от конгруентни формули $\varphi = \varphi_0, \varphi_1, \varphi_2, \ldots, \varphi_n$, в която последната формула ще отговаря на изискванията на лемата — всички кванторни променливи ще са различни и никоя кванторна променлива няма да е елемент на Θ'.

Ако разполагаме с формулата φ_i, можем да получим формулата φ_{i+1} по следния начин. Нека $\forall \mathtt{z}\, \psi$ или $\exists \mathtt{z}\, \psi$ е произволна подформула на φ_i, чиято променлива $\mathtt{z}$ е елемент на Θ' и в ψ не се срещат квантори с променливи от Θ'. Нека $\mathtt{z}'$ е произволна променлива, която не се среща нито в φ_i, нито е елемент на Θ'. Да преименуваме навсякъде в подформулата $\forall \mathtt{z}\, \psi$ или $\exists \mathtt{z}\, \psi$ променливата $\mathtt{z}$ на $\mathtt{z}'$. Нека така получената формула бъде φ_{i+1}.

Процесът на строене на тази редица от конгруентни формули спира тогава, когато стигнем до формула φ_n, в която няма квантори с променливи от Θ'. Тъй като при всяко преименуване избираме напълно нова променлива, която не се среща нито в φ_i, нито е елемент на Θ, то φ_n ще отговаря на условията на лемата. ∎

5.3. Субституции

> Повечето формулировки на правилото за субституция, които са били публикувани — дори и от най-способните логици — преди 1940, са явно сбъркани.
>
> Хаскел Къри и Робер Фей [6]

За да завършим синтактичните разглеждания за предикатната логика от първи ред, остава да видим как можем да прилагаме субституция към предикатна формула. При термовете, атомарните формули и клаузите беше лесно. Например за произволна атомарна формула φ ако s е субституция, то формулата $\hat{s}(\varphi)$ се получава като заменим в φ всяка променлива $\mathtt{x}$ с $s(\mathtt{x})$. Можехме да използваме такава проста дефиниция, защото в термовете, атомарните формули и клаузите няма свързани променливи.

Да видим защо при наличието на свързани променливи прилагането на субституции трябва да се прави по-внимателно. Това е така, защото при прилагането на субституция към израз със свързани променливи може да се допуснат два вида грешки.

5.18. Първия вид грешка вече я споменахме — при прилагане на субституция не трябва да заменяме свързаните променливи, защото се

получават безсмислици. Например ако в израза

$$\sum_{i=1}^{100} i^2$$

заменим i с x^2, ще получим безсмислицата

$$\sum_{x^2=1}^{100} (x^2)^2$$

и ако в предикатната формула

$$\forall \mathtt{x}\, \mathtt{p}(\mathtt{x})$$

заменим $\mathtt{x}$ с терма $\mathtt{f}(\mathtt{y})$, ще получим безсмислицата

$$\forall(\mathtt{f}(\mathtt{y}))\, \mathtt{p}(\mathtt{f}(\mathtt{y}))$$

Извод: Когато прилагаме субституция, трябва да игнорираме променливите, които са свързани от някой квантор. Например ако към формулата

$$\mathtt{p}(\mathtt{x}) \vee \forall \mathtt{x}\, \mathtt{q}(\mathtt{x})$$

искаме да приложим субституция, заменяща променливата $\mathtt{x}$ с терма $\mathtt{f}(\mathtt{y})$, ще получим формулата

$$\mathtt{p}(\mathtt{f}(\mathtt{y})) \vee \forall \mathtt{x}\, \mathtt{q}(\mathtt{x})$$

5.19. Вторият вид грешка е по-труден за откриване. Да предположим, че сме дефинирали числовата функция f така:

$$f(y) = \int_0^1 (x^2 + y^3) dx$$

Това, че в дефиницията на тази функция сме използвали променливата x не означава, че от тук нататък до края на света нямаме право да използваме променливи x за други цели. Да допуснем, че в даден момент сме дефинирали някаква променлива x. На колко ще бъде равно $f(2x)$? Ако просто навсякъде заменим y с $2x$ и кажем, че

$$f(2x) = \int_0^1 (x^2 + (2x)^3) dx$$

това ще бъде грешка! Да забележим, че изразът от дясната страна на равенството по никакъв начин не зависи от стойността на x, защото променливата x е свързана. Верният отговор е следният:

$$f(2x) = \int_0^1 (z^2 + (2x)^3)dz$$

Разбира се, вместо z можем да използваме коя да е друга променлива, която не се използва за други цели.

Ето друг пример. Нека

$$a_n = \sum_{i=1}^{n} i^3$$

Ако след време дефинираме променлива i, на колко ще бъде равно a_{2i}? Верният отговор е

$$a_{2i} = \sum_{j=1}^{2i} j^3$$

Тук също, вместо j можем да използваме коя да е друга променлива, която не се използва за друго цели.

Обмислете добре тези примери, преди да продължите четенето на този раздел!

***Задача* 46:** Нека субституцията s заменя променливата $\mathtt{x}$ с терма $\mathtt{f(y)}$ и не променя променливата $\mathtt{y}$, т.е. $s(\mathtt{y}) = \mathtt{y}$. Още е рано да дадем точната дефиниция на $\hat{s}(\varphi)$ при произволна субституция s и формула φ. Въпреки това, имайки предвид съображенията изказани дотук, преценете какви трябва да бъдат

1. $\hat{s}(\forall \mathtt{x}\, \mathtt{p(x, y)} \vee \mathtt{q(x)})$
2. $\hat{s}(\forall \mathtt{y}\, \mathtt{p(x, y)} \vee \mathtt{q(y)})$

Анализът на грешките от първи и втори тип (вж. 5.19 и 5.19) показва, че когато субституцията замества едни променливи с термове, съдържащи други променливи, тези променливи — както заменените, така и новопоявилите се — не трябва да имат нищо общо с кванторните променливи. Ако една заменена променлива е била в областта на действие на квантор, получаваме грешка от първи тип, а когато новопоявила се променлива попадне в областта на действие на квантор, получаваме грешка от втори тип. Тези, потенциално опасни, променливи ще наричаме активни променливи.

5.20. Дефиниция. Следните променливи ще наричаме *активни променливи* във формулата φ при субституция s:

- свободните променливи на φ;
- променливите, които се срещат в термовете $s(\mathtt{x})$, където $\mathtt{x}$ е свободна променлива на φ.

Активни са променливите, които имат някакво отношение към промените, които прави субституцията. Първото от условията в горната дефиниция включва променливите, които се заменят от субституцията, а второто — променливите, които се появяват след прилагане на субституцията.

5.21. Дефиниция. Нека s и φ са произволни субституция и формула и Θ е крайното множество от всички активни във φ променливи при субституция s. Ако φ не съдържа квантори с променливи от Θ, нека $\varphi' = \varphi$, а в противен случай нека φ' е конгруентна с φ формула, която не съдържа квантори с променливи от Θ (такава съществува благодарение на лема 5.17).

В такъв случай, с $\hat{s}(\varphi)$ ще означим израза, който се получава като заместим във φ' всяка свободна променлива $\mathtt{x}$ с $s(\mathtt{x})$. Изразът $\hat{s}(\varphi)$ се нарича *резултат от прилагането на субституцията s към φ*.

Да забележим, че според тази дефиниция заменяме само свободните променливи. Ако променливата $\mathtt{z}$ не е свободна, тогава $\mathtt{z}$ не се заменя с терма $s(\mathtt{z})$.

Всичко, което трябва да знаем за прилагането на субституция към формула, е включено в следващото следствие. Точният вид на дефиниция 5.21 няма да ни бъде нужен за нищо друго, освен за да гарантираме, че това следствие е вярно.

5.22. Следствие. *а) Ако формулата φ не съдържа квантори с променливи, които са активни във φ при субституция s, тогава $\hat{s}(\varphi)$ се получава като заменим във φ всяка свободна променлива $\mathtt{x}$ с $s(\mathtt{x})$.*

б) За всяка субституция s и формула φ съществува такава конгруентна с φ формула φ', че φ' не съдържа квантори с активни при субституция s променливи и $\hat{s}(\varphi) = \hat{s}(\varphi')$.

в) Ако формулата φ не съдържа квантори с променливи, които са активни във φ при субституция s и субституцията s не променя кванторните променливи във φ (т.е. за всяка такава променлива $s(\mathtt{z}) = \mathtt{z}$), тогава $\hat{s}(\varphi)$ се получава като заменим във φ всяка променлива $\mathtt{x}$ с $s(\mathtt{x})$.

Доказателство. **(а)** следва непосредствено от дефиницията, **(б)** също става очевидно, ако забележим, че φ и φ' имат едни и същи свободни променливи и значи при субституция s едни и същи променливи са активни във φ и във φ' и **(в)** следва от (а). ∎

5.23. Атомарните формули не съдържат квантори и в частност не съдържат квантори с променливи, които са активни. Това означава, че според току-що дадената дефиниция, ако φ е атомарна, то $\hat{s}(\varphi)$ се получава като заменим във φ всяка свободна променлива $\mathtt{x}$ с $s(\mathtt{x})$. От друга страна, според дефиниция 4.2 б), за да получим $\hat{s}(\varphi)$, трябва да заменим във φ всяка променлива $\mathtt{x}$ с $s(\mathtt{x})$. Тъй като в една атомарна формула всички променливи са свободни, то тези две дефиниции не си противоречат.

В дефиниция 5.21 нарекохме $\hat{s}(\varphi)$ просто „израз“, а не формула. Всъщност $\hat{s}(\varphi)$ е формула, но това се нуждае от доказателство. Най-напред ще докажем следната проста лема:

5.24. Лема. *Нека s е субституция, която не променя кванторните променливи във формулата φ (т.е. за всяка такава променлива $s(\mathtt{z}) = \mathtt{z}$). Тогава изразът, който ще се получи от φ като заменим във φ всяка променлива $\mathtt{x}$ с $s(\mathtt{x})$, е формула.*

Доказателство. Лемата може да се докаже с проста индукция по построението на формулата φ (т.е. индукция по дефиниция 5.5).

Ако φ е атомарна, тогава съответният израз е просто $\hat{s}(\varphi)$ (вж. дефиниция 4.2 б)) и значи също е атомарна формула.

Ако $\varphi = (\psi_1 \,\&\, \psi_2)$, то изразът, който се получава от φ като заменим всяка променлива $\mathtt{z}$ с $s(\mathtt{z})$, има вида $(\psi_1' \,\&\, \psi_2')$, където ψ_1' и ψ_2' се получават по същия начин съответно от ψ_1 и ψ_2. Съгласно индукционното предположение, ψ_1' и ψ_2' са формули и значи $(\psi_1' \,\&\, \psi_2')$ също е формула.

Аналогично се разглеждат случаите когато $\varphi = (\psi_1 \vee \psi_2)$ и $\varphi = (\psi_1 \Rightarrow \psi_2)$.

Ако $\varphi = \neg\psi$, то изразът, който се получава от φ като заменим всяка променлива $\mathtt{z}$ с $s(\mathtt{z})$, има вида $\neg\psi'$, където ψ' се получава по същия начин от ψ. Съгласно индукционното предположение, ψ' е формула и значи $\neg\psi'$ също е формула.

Ако $\varphi = \forall\mathtt{x}\,\psi$, тъй като s не променя кванторните променливи, то $s(\mathtt{x}) = \mathtt{x}$ и значи изразът, който се получава от φ като заменим всяка променлива $\mathtt{z}$ с $s(\mathtt{z})$, има вида $\forall\mathtt{x}\,\psi'$, където ψ' се получава по същия начин от ψ. Съгласно индукционното предположение, ψ' е формула и значи $\forall\mathtt{x}\,\psi'$ също е формула.

Аналогично се разглежда случаят когато $\varphi = \exists \mathtt{x}\, \psi$. ∎

5.25. Твърдение. *За всяка субституция s и формула φ изразът $\hat{s}(\varphi)$ е формула.*

Доказателство. Нека φ' е такава конгруентна с φ формула, че $\hat{s}(\varphi) = \hat{s}(\varphi')$ и φ' не съдържа квантори с активни при субституция s променливи. Нека s' е следната субституция:

$$s'(\mathtt{z}) = \begin{cases} s(\mathtt{z}), & \text{ако } \mathtt{z} \text{ е свободна във } \varphi \text{ (и значи и във } \varphi') \\ \mathtt{z}, & \text{иначе} \end{cases}$$

Изразът $\hat{s}(\varphi')$ се получава от φ' като заменим всички свободни променливи $\mathtt{x}$ в φ' с $s(\mathtt{x})$. Субституцията s' съвпада с s за всички такива променливи и значи $\hat{s}(\varphi')$ може да се получи от φ' като заместим всяка свободна променлива $\mathtt{x}$ в φ' с $s'(\mathtt{x})$.

Тъй като φ' не съдържа квантори с променливи, които са активни при субституция s, то в частност φ' не съдържа квантори с променливи, които са свободни във φ'. Следователно за всяка кванторна променлива $s'(\mathtt{z}) = \mathtt{z}$ и значи $\hat{s}(\varphi')$ може да се получи от φ като заменим в φ' всяка променлива $\mathtt{z}$ с $s'(\mathtt{z})$. От лема 5.24 следва, че $\hat{s}(\varphi')$ е формула. ∎

Вече споменахме, че ще искаме да отъждествяваме конгруентните формули. Това означава, че ще трябва да докажем, че ако две формули φ_1 и φ_2 са конгруентни, то за произволна субституция s, формулите $\hat{s}(\varphi_1)$ и $\hat{s}(\varphi_2)$ също са конгруентни. Доказателството на това твърдение е сложно и ще изисква отначало да докажем две помощни леми.

5.26. Лема. *Нека Θ е крайно множество от променливи и φ_1 и φ_2 са конгруентни формули, които не съдържат квантори с променливи от Θ. Тогава съществува такава редица от формули*

$$\varphi_1 = \psi_1, \psi_2, \psi_3, \ldots, \psi_n = \varphi_2$$

че членовете на тази редица не съдържат квантори с променливи от Θ и всеки член се получава от предходния посредством еднократно преименуване на свързана променлива.

Доказателство. Съгласно дефиницията на конгруентни формули, съществува такава редица от формули $\varphi_1 = \psi'_1, \psi'_2, \ldots, \psi'_n = \varphi_2$, че всеки член се получава от предходния посредством еднократно преименуване на свързана променлива. Нека $\mathtt{x}$ е някоя променлива от Θ, която се среща в квантор на някоя формула от тази редица и нека $\mathtt{y}$ е променлива, която не е елемент на Θ и не се среща никъде в тази редица. Може

да забележим, че ако навсякъде в тази редица заменим променливата $\mathtt{x}$ с $\mathtt{y}$, отново ще получим редица от формули $\varphi_1 = \psi''_1, \psi''_2, \ldots, \psi''_n = \varphi_2$, в която всеки член се получава от предходния посредством еднократно преименуване на свързана променлива. В новата редица обаче ще се срещат по-малко квантори с променливи от Θ. Ако повторим това преобразование нужния брой пъти, ще получим редица със свойствата, изисквани от условието на лемата. ∎

5.27. Лема. *Ако формулите φ_1 и φ_2 са конгруентни и не съдържат променливи, които са активни при субституция s, то $\hat{s}(\varphi_1)$ и $\hat{s}(\varphi_2)$ са конгруентни.*

Доказателство. Съгласно твърдение 5.16, конгруентните формули имат едни и същи свободни променливи, от където следва, че активните в φ_1 и φ_2 променливи при субституция s са едни и същи. Да означим с Θ множеството на активните променливи в тези формули. Съгласно лема 5.26 съществува редица $\varphi_1 = \psi_1, \psi_2, \ldots, \psi_n = \varphi_2$, в която всеки член се получава от предходния с еднократно преименуване на свързана променлива и никой член не съдържа квантори с променливи от Θ. Тъй като членовете на тази редица са конгруентни, те имат едни и същи свободни променливи и значи Θ е множеството от активните променливи при субституция s на всяка една от тези формули. Следователно формулите $\hat{s}(\psi_i)$ могат да се получат от ψ_i като заменим всяка свободна променлива $\mathtt{x}$ в ψ_i с $\hat{s}(\mathtt{x})$.

Формулата ψ_{i+1} е получена от ψ_i посредством еднократно преименуване на свързана променлива. Това че ψ_i и ψ_{i+1} не съдържат квантори с променливи от Θ ни гарантира, че можем да направим аналогично преименуване, за да получим $\hat{s}(\psi_{i+1})$ от $\hat{s}(\psi_i)$. Следователно $\hat{s}(\varphi_1) = \hat{s}(\psi_1) \equiv \hat{s}(\psi_2) \equiv \hat{s}(\psi_3) \equiv \ldots \equiv \hat{s}(\psi_n) = \hat{s}(\varphi_2)$. ∎

5.28. Твърдение. *Ако $\varphi_1 \equiv \varphi_2$, то $\hat{s}(\varphi_1) \equiv \hat{s}(\varphi_2)$. С други думи, ако формулите φ_1 и φ_2 са конгруентни, то за произволна субституция s, формулите $\hat{s}(\varphi_1)$ и $\hat{s}(\varphi_2)$ са конгруентни.*

Доказателство. Нека φ'_1 и φ'_2 са такива формули, че $\varphi_1 \equiv \varphi'_1$, $\varphi_2 \equiv \varphi'_2$, $\hat{s}(\varphi_1) = \hat{s}(\varphi'_1)$ и $\hat{s}(\varphi_2) = \hat{s}(\varphi'_2)$ и φ'_1 и φ'_2 не съдържат квантори с променливи, които са активни при субституцията s. Тъй като φ_1 и φ_2, са конгруентни, то φ'_1 и φ'_2 също са конгруентни и затова получаваме искането от лема 5.27. ∎

Сравнете следващото твърдение с твърдение 4.16.

5.29. Твърдение. *Нека субституциите s_1 и s_2 съвпадат за всички свободни променливи на формулата φ. Тогава $\hat{s_1}(\varphi) \equiv \hat{s_2}(\varphi)$.*

Доказателство. Нека φ' е конгруентна с φ формула, която не съдържа квантори с променливи, които са активни при субституция s_1. Формулата φ' има същите свободни променливи като φ, а за всяка такава свободна променлива субституциите s_1 и s_2 съвпадат. Това означава, че формулата φ' не съдържа квантори с активни променливи също и при субституция s_2. От дефиницията за прилагане на субституция (дефиниция 5.21) следва, че $\hat{s_1}(\varphi') = \hat{s_2}(\varphi')$.

От тук получаваме исканото, защото от една страна $\hat{s_1}(\varphi) \equiv \hat{s_1}(\varphi')$, а от друга $\hat{s_2}(\varphi') \equiv \hat{s_2}(\varphi)$. ∎

Сравнете следващото твърдение със следствие 4.19.

5.30. Твърдение. *За всеки две субституции s_1 и s_2, съществува такава субституция s, че за всяка формула φ*

$$\hat{s}(\varphi) \equiv \hat{s_2}(\hat{s_1}(\varphi))$$

Доказателство. Нека s е субституцията, за която $s(\mathtt{z}) = \hat{s_2}(s_1(\mathtt{z}))$ за всяка променлива $\mathtt{z}$. Ще докажем, че за произволна формула $\hat{s}(\varphi) \equiv \hat{s_2}(\hat{s_1}(\varphi))$.

Нека Θ е множеството от свободните променливи на φ. Нека Θ_1 е обединението на Θ с множеството от всички променливи, които се срещат в термовете $s_1(\mathtt{z})$, където $\mathtt{z} \in \Theta$. С други думи, Θ_1 е множеството от активните променливи в φ при субституция s_1. Нека Θ_2 е обединението на Θ_1 с множеството от всички променливи, които се срещат в термовете $s_2(\mathtt{z})$, където $\mathtt{z} \in \Theta_1$. Множеството Θ_2 е крайно, значи от лемата за преименуване на свързаните променливи 5.17 можем да намерим формула ψ, която е конгруентна с φ, и която не съдържа квантори с променливи от Θ_2.

Тъй като Θ_1 е множеството от активните променливи в φ при субституция s_1 и $\Theta_1 \subseteq \Theta_2$, то ψ не съдържа квантори с променливи, които са активни при субституция s_1. Следователно $\hat{s_1}(\psi)$ се получава като заменим в ψ всяка свободна променлива $\mathtt{x}$ с терма $s_1(\mathtt{x})$. Това означава, че свободните променливи на $\hat{s_1}(\psi)$ са елементи на Θ_1, а формулата $\hat{s_1}(\psi)$ не съдържа квантори с променливи, които са активни при субституция s_2, следователно $\hat{s_2}(\hat{s_1}(\psi))$ се получава от $\hat{s_1}(\psi)$ като заменим всяка свободна променлива $\mathtt{y}$ с терма $s_2(\mathtt{y})$. Тъй като всички свободни променливи в $\hat{s_1}(\psi)$ се съдържат в термовете $s_1(\mathtt{x})$, които са заменили свободните променливи $\mathtt{x}$ в ψ, то това означава, че $\hat{s_2}(\hat{s_1}(\psi))$ може да се получи от ψ като заменим всяка свободна променлива $\mathtt{x}$ в ψ с терма $\hat{s_1}(s_1(\mathtt{x}))$.

Тъй като ψ не съдържа квантори с променливи от Θ_2, то ψ не съдържа квантори с променливи, които са активни при субституция s.

Следователно формулата $\hat{s}(\psi)$ може да се получи от ψ като заменим всяка свободна променлива $\mathtt{x}$ в ψ с терма $s(\mathtt{x})$, т.е. с терма $\hat{s_1}(s_1(\mathtt{x}))$.

Доказахме, че $\hat{s}(\psi) = \hat{s_2}(\hat{s_1}(\psi))$, откъдето следва $\hat{s}(\varphi) \equiv \hat{s_2}(\hat{s_1}(\varphi))$. ■

5.4. Стойност на формула в структура

Да си припомним, че стойността на една атомарна формула в структура зависи от оценката, при която оценяваме формулата. Това е така, защото структурата не дава стойност на променливите, които се срещат в атомарната формула. Също така да си припомним, че за да оценим една атомарна формула е достатъчно да знаем каква е стойността на променливите, които се срещат в нея (вж. твърдение 3.24).

Подобно се оказва и положението при формулите с тази разлика, че тук съществена е само стойността на свободните променливи. Да разгледаме например формулата

$$\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{t})$$

За да пресметнем стойността на тази формула е необходимо:

- структурата да каже кое е множеството, в което „работи“ кванторът $\forall \mathtt{x}$, т.е. универсумът;
- структурата да каже как се интерпретира предикатният символ $\mathtt{p}$;
- оценката да каже каква е стойността на променливите $\mathtt{y}$, $\mathtt{z}$ и $\mathtt{t}$. Стойността на останалите променливи, включително на $\mathtt{x}$, е без значение.

Например ако универсумът на структурата $\mathbf{M}$ е множеството на естествените числа и $\mathtt{p}^{\mathbf{M}}$ е предикатът

$$\mathtt{p}^{\mathbf{M}}(n, m, k, l) \longleftrightarrow m^{n+3} + m^{n+3} \neq l^{n+3}$$

тогава стойността на горната формула при оценка v трябва да бъде твърдението:

„За всяко естествено число n, $(v(\mathtt{y}))^{n+3} + (v(\mathtt{z}))^{n+3} \neq (v(\mathtt{t}))^{n+3}$.“

В общия случай, ако универсумът на структурата $\mathbf{M}$ е $|\mathbf{M}|$, тогава стойността в $\mathbf{M}$ на горната формула при оценка v е твърдението:

„За всеки елемент μ на $|\mathbf{M}|$ е вярно $\mathtt{p}^{\mathbf{M}}(\mu, v(\mathtt{y}), v(\mathtt{z}), v(\mathtt{t}))$.“

Да разгледаме формула, в която след квантора стои не атомарна формула, а произволна формула

$$\forall \mathtt{x}\, \varphi$$

Кога трябва да считаме, че тази формула е вярна в структурата $\mathbf{M}$ при оценка v? Като пръв и най-естествен опит може да пробваме със следното твърдение:

> „При всяка възможна стойност на $\mathtt{x}$ от универсума на $\mathbf{M}$ формулата φ е вярна.“

Един проблем в тази формулировка е това, че в нея нищо не се казва за останлите променливи във φ и за оценката, която дава стойност на тези променливи. На втори опит получаваме следното:

> „При всяка възможна стойност на $\mathtt{x}$ от универсума на $\mathbf{M}$ формулата φ е вярна при оценка v.“

В тази формулировка обаче възниква друг проблем. Какво означава „вярна при оценка v при еди-каква си стойност на променливата $\mathtt{x}$“? Та нали оценката v дава стойност на всички променливи, включително и на $\mathtt{x}$? На трети опит получаваме следното твърдение:

> „За всеки елемент μ на универсума на $\mathbf{M}$ формулата φ е вярна при модифицираната оценка v', която се дефинира по следния начин:
>
> $$v'(\xi) = \begin{cases} \mu, & \text{ако } \xi = \mathtt{x} \\ v(\xi), & \text{иначе“} \end{cases}$$

Именно на този вариант ще се спрем, когато дефинираме стойността на формула в структура при оценка. Преди това обаче нека дефинираме по-формално понятието „модифицирана оценка“.

5.31. Дефиниция. Нека v е оценка в структурата $\mathbf{M}$, $\mathtt{x}$ е променлива и μ е елемент на универсума на $\mathbf{M}$. Тогава оценката

$$v'(\xi) = \begin{cases} \mu, & \text{ако } \xi = \mathtt{x} \\ v(\xi), & \text{иначе“} \end{cases}$$

се нарича *модифицирана оценка* и ще бъде означавана $v^{\mu}_{\mathtt{x}}$.

Модифицираната оценка също може да бъде модифицирана като вместо $(((v^{\mu_1}_{\mathtt{x}_1})^{\mu_2}_{\mathtt{x}_2})\dots)^{\mu_n}_{\mathtt{x}_n}$ ще пишем $v^{\mu_1\mu_2\dots\mu_n}_{\mathtt{x}_1\mathtt{x}_2\dots\mathtt{x}_n}$. С други думи,

$$v^{\mu_1\mu_2\dots\mu_n}_{\mathtt{x}_1\mathtt{x}_2\dots\mathtt{x}_n}(\xi) = \begin{cases} \mu_n, & \text{ако } \xi = \mathtt{x}_n \\ \mu_{n-1}, & \text{ако } \xi = \mathtt{x}_{n-1} \text{ и } \xi \neq \mathtt{x}_n \\ \dots & \\ \mu_1, & \text{ако } \xi = \mathtt{x}_1 \text{ и } \xi \neq \mathtt{x}_1,\, \xi \neq \mathtt{x}_2, \dots,\, \xi \neq \mathtt{x}_{n-1} \end{cases}$$

***Задача* 47:** Възможно ли е $v^{\mu\nu}_{\mathtt{xy}}(\mathtt{x}) \neq \mu$?

Използвайки модифицирани оценки, можем да дадем дефиницията за стойност на формула по следния начин:

5.32. Дефиниция. Нека v е оценка в структурата $\mathbf{M}$. Стойността на формула φ в структурата $\mathbf{M}$ при оценка v е твърдение, което ще означаваме с

$$\mathbf{M} \vDash \varphi[v]$$

и се дефинира индуктивно посредством следните правила:

а) ако φ е атомарна, тогава стойността ѝ се определя съгласно дефиниция 3.16;

б) ако $\varphi = \psi_1 \,\&\, \psi_2$, тогава $\mathbf{M} \vDash \varphi[v]$ е твърдението
„$\mathbf{M} \vDash \psi_1[v]$ и $\mathbf{M} \vDash \psi_2[v]$“;

в) ако $\varphi = \psi_1 \vee \psi_2$, тогава $\mathbf{M} \vDash \varphi[v]$ е твърдението
„$\mathbf{M} \vDash \psi_1[v]$ или $\mathbf{M} \vDash \psi_2[v]$“;

г) ако $\varphi = \psi_1 \Rightarrow \psi_2$, тогава $\mathbf{M} \vDash \varphi[v]$ е твърдението
„ако $\mathbf{M} \vDash \psi_1[v]$, то $\mathbf{M} \vDash \psi_2[v]$“;

д) ако $\varphi = \neg\psi$, тогава $\mathbf{M} \vDash \varphi[v]$ е твърдението
„не е вярно, че $\mathbf{M} \vDash \psi[v]$“;

е) ако $\varphi = \forall \mathtt{x}\, \psi$, тогава $\mathbf{M} \vDash \varphi[v]$ е твърдението
„за всеки елемент μ на универсума на $\mathbf{M}$ е вярно $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$“;

ж) ако $\varphi = \exists \mathtt{x}\, \psi$, тогава $\mathbf{M} \vDash \varphi[v]$ е твърдението
„за някой елемент μ на универсума на $\mathbf{M}$ е вярно $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$“.

5.33. Дефиниция. а) Една формула е *вярна* в структура $\mathbf{M}$ при оценка v, ако стойността ѝ в структурата $\mathbf{M}$ при оценка v е истина. Следователно

$$\mathbf{M} \vDash \varphi[v]$$

е истина тогава и само тогава, когато формулата φ е вярна в структурата $\mathbf{M}$ при оценка v.

б) Формула φ е *тъждествено вярна* в структура $\mathbf{M}$, ако е вярна в структурата $\mathbf{M}$ при произволна оценка v. Записваме това така:

$$\mathbf{M} \vDash \psi$$

в) Формула φ е *тъждествено вярна* или *предикатна тавтология*, ако φ е тъждествено вярна във всяка структура. Записваме това така:

$$\vDash \psi$$

г) Две формули φ и ψ са *еквивалентни*, ако формулата $\varphi \Leftrightarrow \psi$ е предикатна тавтология. Записваме това така:

$$\vDash \varphi \Leftrightarrow \psi$$

5.34. Твърдение. *Две формули φ и ψ са еквивалентни тогава и само тогава, когато при произволна структура $\mathbf{M}$ и оценка v в $\mathbf{M}$, $\mathbf{M} \vDash \varphi[v]$ е еквивалентно на $\mathbf{M} \vDash \psi[v]$.*

Доказателство. Да си припомним, че формулата $\varphi \Leftrightarrow \psi$ е съкращение на формулата $(\varphi \Rightarrow \psi)\ \&\ (\psi \Rightarrow \varphi)$. Затова от дефиниции 5.33 г) и 5.33 б) следва, че φ и ψ са еквивалентни тогава и само тогава, когато при произволна структура $\mathbf{M}$ и оценка v в $\mathbf{M}$, от $\mathbf{M} \vDash \varphi[v]$ следва $\mathbf{M} \vDash \psi[v]$ и от $\mathbf{M} \vDash \psi[v]$ следва $\mathbf{M} \vDash \varphi[v]$. Това означава, че $\mathbf{M} \vDash \varphi[v]$ и $\mathbf{M} \vDash \psi[v]$ са еквивалентни. ■

5.35. Следствие. *а)* $\vDash \varphi \Leftrightarrow \varphi$

б) ако $\vDash \varphi \Leftrightarrow \psi$, *то* $\vDash \psi \Leftrightarrow \varphi$

в) ако $\vDash \varphi \Leftrightarrow \psi$ *и* $\vDash \psi \Leftrightarrow \chi$, *то* $\vDash \varphi \Leftrightarrow \chi$

Доказателство. Следва непосредствено от твърдение 5.34. ■

5.36. Твърдение. *Нека формулата φ има подформула ψ и формулата ψ е еквивалентна на ψ'. Ако формулата φ' се получава от φ като заменим подформулата ψ с ψ', то формулите φ и φ' са еквивалентни.*

Доказателство. С индукция по построението на формулата φ.

Ако $\psi = \varphi$, то като заменим ψ с еквивалентна на нея формула ψ', ще получим $\varphi' = \psi'$ и значи φ и φ' ще са еквивалентни. Остава да разгледаме случая когато $\psi \neq \varphi$.

Ако φ е атомарна, то единствената ѝ подформула е $\psi = \varphi$, а вече разгледахме този случай.

Ако $\varphi = \chi_1\ \&\ \chi_2$ и $\psi \neq \varphi$, то ψ ще е подформула на χ_1 или χ_2. Нека за определеност ψ е подформула на χ_1 и като заменим ψ на ψ' от χ_1 получаваме χ_1'. Съгласно индукционното предположение χ_1 и χ_1' са еквивалентни. Тъй като $\varphi' = \chi_1'\ \&\ \chi_2$, от тук получаваме, че φ и φ' също са еквивалентни.*

Случаите когато $\varphi = \chi_1 \vee \chi_2$ и $\varphi = \chi_1 \Rightarrow \chi_2$ се разглеждат аналогично.

Ако $\varphi = \neg\chi$ и $\psi \neq \varphi$, то ψ ще е подформула на χ. Като заменим ψ на ψ' от χ получаваме χ'. Съгласно индукционното предположение χ и

*По-подробно това се обосновава така. Щом при произволна структура $\mathbf{M}$ и

χ' са еквивалентни. Тъй като $\varphi' = \neg\chi'$, от тук получаваме, че φ и φ' също са еквивалентни.

Ако $\varphi = \forall\mathtt{x}\,\chi$ и $\psi \neq \varphi$, то ψ е подформула на χ. Като заменим ψ с ψ' от χ получаваме χ'. Съгласно индукционното предположение χ и χ' са еквивалентни. Тъй като $\varphi' = \forall\mathtt{x}\,\chi'$, то от тук получаваме, че φ и φ' също са еквивалентни.*

Случаят когато $\varphi = \exists\mathtt{x}\,\chi$ се разглежда аналогично. ∎

Следващата лема ще използваме, за да докажем няколко изключително важни свойства. След като докажем тези свойства, тя повече няма да ни бъде нужна.

В тази лема с $\bar{v}(\tau)$ е означена стойността на терма τ при оценка v.

5.37. Лема. *За всяка формула φ, която не съдържа квантори с една и съща променлива, е вярно следното твърдение:*

> *Ако субституцията s е такава, че φ не съдържа квантори с активни при субституция s променливи, а оценките v и w в структура $\mathbf{M}$ са такива, че за всяка свободна променлива $\mathtt{x}$ във формулата φ е изпълнено $w(\mathtt{x}) = \bar{v}(s(\mathtt{x}))$, то $\mathbf{M} \vDash \varphi[w]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\varphi))[v]$.*

Доказателство. Ще докажем това твърдение с индукция по формулата φ (т.е. индукция по дефиниция 5.5).

Когато φ е атомарна, нека w' е субституцията, за която $w'(\mathtt{z}) = \bar{v}(s(\mathtt{z}))$ за всяка променлива $\mathtt{z}$. От твърдение 3.24 следва, че $\mathbf{M} \vDash \varphi[w]$

оценка v, $\mathbf{M} \vDash \chi_1[v]$ е еквивалентно на $\mathbf{M} \vDash \chi_1'[v]$, значи

$$\begin{aligned}\mathbf{M} \vDash \varphi[v] &\longleftrightarrow \mathbf{M} \vDash (\chi_1 \,\&\, \chi_2)[v]\\ &\longleftrightarrow \mathbf{M} \vDash \chi_1[v] \text{ и } \mathbf{M} \vDash \chi_2[v]\\ &\longleftrightarrow \mathbf{M} \vDash \chi_1'[v] \text{ и } \mathbf{M} \vDash \chi_2[v]\\ &\longleftrightarrow \mathbf{M} \vDash (\chi_1' \,\&\, \chi_2)[v]\\ &\longleftrightarrow \mathbf{M} \vDash \varphi'[v]\end{aligned}$$

*По-подробно това се обосновава така. Щом при произволна структура $\mathbf{M}$ и оценка v, $\mathbf{M} \vDash \chi[v]$ е еквивалентно на $\mathbf{M} \vDash \chi'[v]$, значи

$$\begin{aligned}\mathbf{M} \vDash \varphi[v] &\longleftrightarrow \mathbf{M} \vDash \forall\mathtt{x}\,\chi[v]\\ &\longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \chi[v^{\mu}_{\mathtt{x}}]\\ &\longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \chi'[v^{\mu}_{\mathtt{x}}]\\ &\longleftrightarrow \mathbf{M} \vDash \forall\mathtt{x}\,\chi'[v]\\ &\longleftrightarrow \mathbf{M} \vDash \varphi'[v]\end{aligned}$$

е еквивалентно на $\mathbf{M} \vDash \varphi[w']$, а от следствие 4.22 получаваме, че последното е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\varphi))[v]$.

Когато $\varphi = \psi_1 \,\&\, \psi_2$, то съгласно индукционното предположение* $\mathbf{M} \vDash \psi_1[w]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\psi_1))[v]$ и $\mathbf{M} \vDash \psi_2[w]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\psi_2))[v]$. Следователно

$$\begin{aligned}
\mathbf{M} \vDash \varphi[w] &\longleftrightarrow \mathbf{M} \vDash (\psi_1 \,\&\, \psi_2)[w] \\
&\longleftrightarrow \mathbf{M} \vDash \psi_1[w] \text{ и } \mathbf{M} \vDash \psi_2[w] \\
&\longleftrightarrow \mathbf{M} \vDash (\hat{s}(\psi_1))[v] \text{ и } \mathbf{M} \vDash (\hat{s}(\psi_2))[v] \\
&\longleftrightarrow \mathbf{M} \vDash (\hat{s}(\psi_1) \,\&\, \hat{s}(\psi_2))[v] \\
&\longleftrightarrow \mathbf{M} \vDash (\hat{s}(\psi_1 \,\&\, \psi_2))[v] \\
&\longleftrightarrow \mathbf{M} \vDash (\hat{s}(\varphi))[v]
\end{aligned}$$

Когато $\varphi = \psi_1 \vee \psi_2$ или $\varphi = \psi_1 \Rightarrow \psi_2$, се разсъждава аналогично.

Когато $\varphi = \neg\psi$, то съгласно индукционното предположение $\mathbf{M} \vDash \psi[w]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\psi))[v]$. Следователно

$$\begin{aligned}
\mathbf{M} \vDash \varphi[w] &\longleftrightarrow \mathbf{M} \vDash (\neg\psi)[w] \\
&\longleftrightarrow \text{не е вярно, че } \mathbf{M} \vDash \psi[w] \\
&\longleftrightarrow \text{не е вярно, че } \mathbf{M} \vDash (\hat{s}(\psi))[v] \\
&\longleftrightarrow \mathbf{M} \vDash (\neg\hat{s}(\psi))[v] \\
&\longleftrightarrow \mathbf{M} \vDash (\hat{s}(\neg\psi))[v] \\
&\longleftrightarrow \mathbf{M} \vDash (\hat{s}(\varphi))[v]
\end{aligned}$$

Когато $\varphi = \forall \mathtt{x}\, \psi$, нека s' е субституцията

$$s'(\mathtt{z}) = \begin{cases} s(\mathtt{z}), & \text{ако } \mathtt{z} \text{ е свободна променлива на } \varphi \\ \mathtt{z}, & \text{иначе} \end{cases}$$

Понеже трябва да докажем, че $\mathbf{M} \vDash \forall \mathtt{x}\, \psi[w]$ е еквивалентно на $\mathbf{M} \vDash \hat{s}(\forall \mathtt{x}\, \psi)[v]$, да разпишем тези два израза, за да видим какво ще се получи. За първия израз имаме:

$$\mathbf{M} \vDash \forall \mathtt{x}\, \psi[w] \longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \psi[w^{\mu}_{\mathtt{x}}] \qquad (44)$$

Преди да разпишем втория израз, да съобразим, че тъй като φ и ψ не съдържат квантори с променливи, които са активни при субституция s, то същото важи и за субституцията s' и значи от следствие 5.22 а)

*Защо можем да прилагаме индукционното предположение? Защо ψ_1 и ψ_2 не съдържат квантори с променливи, които са активни при субституция s?

получаваме, че $\hat{s}(\varphi) = \hat{s'}(\varphi)$, защото s и s' съвпадат за свободните променливи в φ. Тъй като единствено $\mathtt{x}$ може да бъде свободна променлива на ψ без да е свободна променлива на $\forall \mathtt{x}\, \psi$, а $s'(\mathtt{x}) = \mathtt{x}$, то единствено $\mathtt{x}$ може да бъде активна променлива в ψ при субституция s' без да е активна в $\forall \mathtt{x}\, \psi$ при субституция s'. Тъй като φ не съдържа различни квантори с една и съща променлива, то в ψ не може да има квантор с променливата $\mathtt{x}$ и значи не само в $\forall \mathtt{x}\, \psi$, но и в ψ няма квантори с променливи, които са активни при субституция s'. Следователно $\hat{s'}(\forall \mathtt{x}\, \psi)$ се получава от $\forall \mathtt{x}\, \psi$ като заменим в $\forall \mathtt{x}\, \psi$ всяка свободна променлива $\mathtt{z}$ с $s'(\mathtt{z})$. Но s' не променя несвободните променливи и значи може да си мислим, че $\hat{s'}(\forall \mathtt{x}\, \psi)$ се получава от $\forall \mathtt{x}\, \psi$ като заменим в $\forall \mathtt{x}\, \psi$ всички променливи $\mathtt{z}$ (а не само свободните) с $s'(\mathtt{z})$. Също и $\hat{s'}(\psi)$ се получава от ψ като заменим в ψ всички променливи $\mathtt{z}$ с $s'(\mathtt{z})$. Следователно

$$\hat{s'}(\forall \mathtt{x}\, \psi) = \forall \mathtt{x}\, \hat{s'}(\psi)$$

Вече сме готови да разпишем $\mathbf{M} \vDash \hat{s}(\forall \mathtt{x}\, \psi)[v]$:

$$\begin{aligned} \mathbf{M} \vDash (\hat{s}(\forall \mathtt{x}\, \psi))[v] &\longleftrightarrow \mathbf{M} \vDash (\hat{s'}(\forall \mathtt{x}\, \psi))[v] \\ &\longleftrightarrow \mathbf{M} \vDash \forall \mathtt{x}\, (\hat{s'}(\psi))[v] \\ &\longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash (\hat{s'}(\psi))[v^{\mu}_{\mathtt{x}}] \end{aligned} \tag{45}$$

Като сравним (44) с (45), виждаме, че остава да докажем единствено това, че $\mathbf{M} \vDash \psi[w^{\mu}_{\mathtt{x}}]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s'}(\psi))[v^{\mu}_{\mathtt{x}}]$. Това следва непосредствено от индукционното предположение за ψ. Да проверим обаче, че имаме право да приложим индукционното предположение за s', $w^{\mu}_{\mathtt{x}}$ и $v^{\mu}_{\mathtt{x}}$. Вече видяхме, че ψ не съдържа квантори с активни променливи при субституция s'. Освен това за всяка свободна променлива $\mathtt{y}$ на ψ трябва да докажем, че е вярно $w^{\mu}_{\mathtt{x}}(\mathtt{y}) = \bar{v^{\mu}_{\mathtt{x}}}((s'(\mathtt{y})))$. Наистина, ако $\mathtt{y} = \mathtt{x}$, то

$$w^{\mu}_{\mathtt{x}}(\mathtt{y}) = w^{\mu}_{\mathtt{x}}(\mathtt{x}) = \mu = \bar{v^{\mu}_{\mathtt{x}}}(\mathtt{x}) = \bar{v^{\mu}_{\mathtt{x}}}(s'(\mathtt{x})) = \bar{v^{\mu}_{\mathtt{x}}}(s'(\mathtt{y}))$$

Ако пък $\mathtt{y}$ е различна от $\mathtt{x}$, то $\mathtt{y}$ ще трябва да бъде свободна променлива и на φ и затова

$$w^{\mu}_{\mathtt{x}}(\mathtt{y}) = w(\mathtt{y}) = \bar{v}(s(\mathtt{y}))$$

Тъй като φ не съдържа квантори с променливи, които са активни при субституция s, то термът $s(\mathtt{y})$ не съдържа променливата $\mathtt{x}$ и затова можем да продължим горната редица от равенства по следния начин:

$$\bar{v}(s(\mathtt{y})) = \bar{v^{\mu}_{\mathtt{x}}}(s(\mathtt{y})) = \bar{v^{\mu}_{\mathtt{x}}}(s'(\mathtt{y}))$$

Когато $\varphi = \exists \mathtt{x}\, \psi$, се разсъждава аналогично. ∎

5.38. Лема. *Нека формулата φ не съдържа променливата* y *(нито като свободна, нито като свързана) и φ' се получава от φ като заместим всички променливи* x *във φ с* y*. Нека s е субституцията*

$$s(\xi) = \begin{cases} \mathtt{y}, & \text{ако } \xi = \mathtt{x} \\ \xi, & \text{иначе} \end{cases}$$

Тогава формулите $\hat{s}(\varphi)$ и φ' са конгруентни.

Доказателство. Нека Θ е множеството, което съдържа x, y и всички променливи, които се срещат във φ и φ'. Да изберем произволен квантор от φ, чиято променлива е елемент на Θ и да извършим еднократно преименуване на тази свързана променлива с променлива, която не е елемент на Θ. Да означим така получената формула с φ_1. Да направим аналогично преименуване и в φ' и да означим получената формула с φ'_1. По същия начин от φ_1 и φ'_1 можем да получим φ_2 и φ'_2, после φ_2 и φ'_2 и т.н. Това можем да продължим докато получим формули φ_n и φ'_n, които не съдържат квантори с променливи от Θ. На всяка стъпка формулите φ_i не съдържат променливата y и ако заменим в φ_i всички променливи x с y, ще получим формулата φ'_i. В частност φ'_n може да се получи от φ_n като заменим всички променливи x с y. Формулата φ_n не съдържа активни променливи при субституция s и затова $\hat{s}(\varphi_n) = \varphi'_n$. Но формулата φ_n е конгруентна с φ, а φ'_n — с φ', така че получаваме исканото от твърдение 5.28. ∎

Тъй като искаме да отъждествяваме конгруентните формули, трябва да докажем, че стойностите на кои да е две конгруентни формули са еквивалентни.

5.39. Твърдение. *Ако две формули са конгруентни, то те са еквивалентни.*

Доказателство. С пълна математическа индукция по броя на символите във формулата φ ще докажем, че за всяка формула ψ, ако $\varphi \equiv \psi$, то $\vDash \varphi \Leftrightarrow \psi$. Индукционното предположение ще бъде, че когато някоя формула съдържа по-малко символи от φ, за нея това твърдение ще бъде вярно.

Тъй като конгруентните формули се получават посредством последователност от еднократни преименувания на свързани променливи, всяко от които не променя броя на символите във формулата, достатъчно ще бъде да докажем, че ако извършим еднократно преименуване на свързана променлива във φ, ще получим еквивалентна формула.

Нека формулата φ' се получава от φ като преименуваме всички променливи $\mathtt{x}$ в някоя нейна подформула на $\mathtt{y}$. Ако тази подформула не съвпада с φ, то тя ще съдържа по-малко символи от φ. Съгласно индукционното предположение тази подформула ще бъде еквивалентна на подформулата, която ще се получи като заменим $\mathtt{x}$ на $\mathtt{y}$, така че еквивалентността на φ и φ' ще следва от твърдение 5.36.

Остава да разгледаме случая когато $\varphi = \forall \mathtt{x}\, \psi$ и $\varphi' = \forall \mathtt{y}\, \psi'$ (случаят $\varphi = \exists \mathtt{x}\, \psi$ е аналогичен). За произволна оценка v в структура $\mathbf{M}$

$$\mathbf{M} \vDash \varphi[v] \longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно } \mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$$

и

$$\mathbf{M} \vDash \varphi'[v] \longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно } \mathbf{M} \vDash \psi'[v^{\mu}_{\mathtt{y}}]$$

Следователно трябва да докажем, че $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$ е еквивалентно на $\mathbf{M} \vDash \psi'[v^{\mu}_{\mathtt{y}}]$. Нека s е субституцията

$$s(\xi) = \begin{cases} \mathtt{y}, & \text{ако } \xi = \mathtt{x} \\ \xi, & \text{иначе} \end{cases}$$

Тъй като $\mathtt{y}$ не се среща в ψ, то за всяка свободна променлива на ψ е вярно, че $v^{\mu}_{\mathtt{x}}(\mathtt{z}) = v^{\mu}_{\mathtt{y}}(s(\mathtt{z}))$ и значи от лема 5.37 получаваме, че $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\psi))[v^{\mu}_{\mathtt{y}}]$. От тук посредством индукционното предположение следва искането, защото съгласно лема 5.38, формулата $\hat{s}(\psi)$ е конгруентна с ψ'. ∎

Сравнете следващото твърдение с твърдение 3.24.

5.40. Твърдение. *Нека v и v' са оценки в структура $\mathbf{M}$. Ако v и v' съвпадат за всички свободни променливи на формулата φ, то $\mathbf{M} \vDash \varphi[v]$ е еквивалентно на $\mathbf{M} \vDash \varphi[v']$.*

Доказателство. Нека s е субституцията идентитет, която не променя нито една променлива. Нека ψ е формула, конгруентна с φ, в която нито една кванторна променлива не е свободна променлива и в която всички квантори са с различни променливи. В такъв случай ψ няма да съдържа квантори с променливи, активни при субституция s. Съгласно дефиницията за прилагане на субституция към формула (вж. дефиниция 5.21) следва, че $\hat{s}(\psi) = \psi$ и затова от лема 5.37 получаваме, че $\mathbf{M} \vDash \psi[v]$ е еквивалентно на $\mathbf{M} \vDash \psi[v']$. От тук следва искането, защото съгласно твърдение 5.39 формулите φ и ψ са еквивалентни. ∎

5.41. Когато за една формула използваме израза $\varphi[\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n]$, където $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ са различни помежду си променливи, с това ще имаме

предвид, че всички свободни променливи на φ са измежду $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$. В този случай, ако $\mu_1, \mu_2, \ldots, \mu_n$ са елементи на универсума на структурата $\mathbf{M}$, с

$$\mathbf{M} \vDash \varphi[\mu_1, \mu_2, \ldots, \mu_n]$$

ще означаваме твърдението, че формулата φ е вярна в структурата $\mathbf{M}$ при оценка v, за която $v(\mathtt{x}_1) = \mu_1$, $v(\mathtt{x}_2) = \mu_2$, $\ldots$, $v(\mathtt{x}_n) = \mu_n$. Съгласно твърдение 5.40, точният избор на оценката v не е от значение.

5.42. Дефиниция. *Затворена формула* означава формула, която не съдържа свободни променливи.

5.43. Съгласно твърдение 5.40 верността на една затворена формула в структура $\mathbf{M}$ не зависи по никакъв начин от оценката, защото всеки две оценки съвпадат за свободните променливи на една затворена формула. Следователно има смисъл когато говорим за затворена формула да четем „$\mathbf{M} \vDash \varphi$“ като „$\varphi$ е вярна в $\mathbf{M}$“, изпускайки наречието „тъждествено“.

5.44. Твърдение. *Ако φ е затворена формула, а $\mathbf{M}$ — произволна структура, то за всяка оценка v в $\mathbf{M}$*

$$\mathbf{M} \vDash \varphi[v] \longleftrightarrow \mathbf{M} \vDash \varphi$$

Доказателство. Да припомним, че $\mathbf{M} \vDash \varphi$ означава, че формулата φ е тъждествено вярна в $\mathbf{M}$, т.е. φ е вярна при всяка оценка.

Съгласно твърдение 5.40 за кои да е две оценки v_1 и v_2 в $\mathbf{M}$ твърдението $\mathbf{M} \vDash \varphi[v_1]$ е еквивалентно на $\mathbf{M} \vDash \varphi[v_2]$. Следователно ако $\mathbf{M} \vDash \varphi[v]$ е вярно за някоя оценка v, то $\mathbf{M} \vDash \varphi[v]$ ще е вярно за всяка оценка v. ∎

Следващото твърдение обобщава лема 5.37 и допълва следствие 4.22.

5.45. Твърдение. *За всяка субституция s и оценка v в структура $\mathbf{M}$ съществува такава оценка w в $\mathbf{M}$, че за всяка формула φ, $\mathbf{M} \vDash \varphi[w]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\varphi))[v]$.*

Доказателство. Нека w е оценката $w(\mathtt{z}) = \bar{v}(s(\mathtt{z}))$, където с $\bar{v}(s(\mathtt{z}))$ е означена стойността на терма $s(\mathtt{z})$ при оценка v.

Нека φ е произволна формула. Нека ψ е конгруентна с φ формула, която не съдържа квантори с активни при субституция s променливи и в която всички квантори имат различни променливи. Съгласно лема 5.37, $\mathbf{M} \vDash \psi[w]$ е еквивалентно на $\mathbf{M} \vDash (\hat{s}(\psi))[v]$. От тук получаваме исканото, защото формулите φ и $\hat{s}(\varphi)$ са конгруентни, а значи и еквивалентни с формулите ψ и $\hat{s}(\psi)$. ∎

5.5. Интуиционистка логика

Класическа и конструктивна математика

При класическия начин за правене на математика твърденията, които изказваме, имат дескриптивен, т.е. описателен характер. Във всяко математическо твърдение се говори за свойствата, притежавани от математическите обекти. Самите математически обекти са статични и неизменни — по времето на древните гърци те са притежавали същите свойства, които притежават и днес. Математическите обекти сякаш живеят в свой идеален и неизменен свят. Математикът не се меси в този свят, не нарушава спокойствието му, а само като страничен наблюдател описва свойствата на математическите обекти. Ако изкажем твърдението, че всяко реално число, повдигнато на квадрат, е неотрицателно, верността на това твърдение няма нищо общо с това какво можем или не можем, какво искаме или не искаме, какво знаем или не знаем.

Конструктивният начин за правене на математика е алтернатива на класическия. При него също можем да изказваме дескриптивни твърдения, но освен това обръщаме внимание и на това какво ние самите можем да правим с математическите обекти. Две числа не просто имат сбор, но могат да бъдат събирани, едно уравнение не просто има решение, но може да бъде решавано, производната на една функция не просто съществува, но може да бъде пресмятана и т.н. Например когато разработваме алгебрата конструктивно, ние не се задоволяваме с това да изкажем твърдение според което полиномите с комплексни коефициенти имат комплексен корен, а започваме да изследваме въпроса дали има начин, с който да можем да намерим корена. С други думи, когато правим математиката конструктивно, в математическите твърдения се говори не само за идеални математически обекти, но и за нас самите — какво можем или не можем да правим с математическите обекти.

Забележка: Има различни философски теории за математиката. Въз основа на тези философии много от конструктивистите не биха се съгласили с обясненията, които ще дадем за това какво представлява и как се прави конструктивната математика. Тук обаче няма да обръщаме никакво внимание на тези философии, нито на това кое според тях е „правилно“ и кое „неправилно“. В частност няма да се интересуваме от това дали някоя математическа теория — конструктивна или не — противоречи на едни или други философски принципи. Математическа значимост имат единствено следните три неща:

– Кои твърдения могат да бъдат формулирани, използвайки езика на дадена математическа теория?

– Каква част от твърденията теорията може да докаже?

– Какви допускания теорията приема без доказателство?

Ще отбележим все пак едно по-съществено несъответствие между тукашните обяснения и едно от по-съществените коструктивни направления в математиката — *интуиционизмът*. Според обясненията които даваме тук, също както и в класическата математика, ако допуснем, че даден обект не съществува и стигнем до противоречие, това означава, че този обект все пак съществува, макар и не винаги да сме състояние да го намерим. Това означава, че в нашите разсъждения предполагаме съществуването на някакъв „свят“, в който съществуват математическите обекти. Ако допуснем, че в този свят даден обект не съществува и стигнем до противоречие, значи въпросният обект съществува. От друга страна, според математическия интуиционизъм не е правомерно да преполагаме съществуването на математически свят, в който има математически обекти, за които не можем дори въобразаемо да допуснем, че могат да бъдат конструирани. Можем да считаме, че съществува само това, което наистина можем конструираме и можем да считаме за вярно само това, което можем да докажем, че е вярно. Ако допуснем, че даден обект не съществува, ние всъщност допускаме, че не сме в състояние да конструираме въпросния обект. Ако това ни доведе до противоречие, това означава, че никога няма да можем да стигнем до извода, че не можем да конструираме дадения обект. С други думи, ако допуснем, че даден обект не съществува и стигнем до противоречие, това означава само, че винаги ще съществува надеждата, че този обект съществува, но само това — няма никакви гаранции, че обектът наистина съществува.

Интуиционистка логика

Това, че в конструктивната математика можем да изказваме и доказваме твърдения, които не могат дори да се формулират на езика на класическата математика, може да ни наведе на мисълта, че конструктивната математика се нуждае от логика с нови логически операции, при която твърденията трябва да се доказват по принципно нов начин. За щастие, оказва се, че това съвсем не е така. Например всички неща, които са доказани в този курс до момента, са доказани конструктивно и за да направим това дори не ни се наложи да предупредим предварително. Използваните тук доказателства на различните твърдения са конструктивни, но в същото време те могат да се четат и от хора, които никога не са чували за конструктивната математика, нито пък знаят какво означава едно доказателство да бъде конструктивно.

Как става възможно това? Начинът е следният — ще използваме обичайните логически операции, но ще ги тълкуваме по нов начин. Това между другото означава, че конструктивната математика се прави донякъде на принципа „говорим едно, ама разбираме друго“.

Когато един конструктивист изкаже твърдение от вида

„съществува такова x, че ...“,

той всъщност има предвид

„може да се намери такова x, че ...“.

Когато пък конструктивистът изкаже твърдение от вида

„A или B“,

той всъщност има предвид

„може да се установи дали A, или B“.

За останалите логически операции може да считаме, че те в конструктивната математика се тълкуват по същия начин, както и в класическата.

Това, че тълкуваме някои от логическите операции по друг начин, означава ли, че в конструктивната математика трябва да се използват по-различни доказателства, в сравнение с класическата? Удивително е, че отговорът на този въпрос е отрицателен — логиката, която се използва в конструктивната математика, наречена *интуиционистка логика*, не се нуждае от по-различен вид доказателства в сравнение с класическата логика.* Достатъчно ще бъде да спазваме две прости правила, и това ще ни гарантира, че доказателството ще бъде конструктивно.

Първото правило е следното: не трябва да използаме метода „допускане на противното“. Ако допуснем A и стигнем до противоречие, това ще ни гарантира, че A не е вярно, но ако допуснем, че A не е вярно и стигнем до противоречие, ние ще докажем не твърдението A, а неговото двойно отрицание. Ще видим, че когато разсъждаваме конструктивно, двойното отрицание на едно твърдение не винаги може да се счита еквивалентно на самото твърдение.

Второто правило за правене на конструктивни доказателства е следното: имаме право да разглеждаме случаи за това дали едно твърдение е вярно, или не, само тогава, когато разполагаме с метод, посредством който можем да проверим дали твърдението е вярно, или не.

*Всъщност ако интуиционистката логика се нуждаеше от по-различен вид доказателства, то не би имало особен смисъл да използваме обичайните класически логически операции, пък да ги тълкуваме по друг начин. Вместо „съществува x“ можеше да казваме „може да се намери x“ и вместо „A или B“ можеше да казваме „може да установим дали A, или B“.

По-нататък в този раздел ще видим с конкретни примери как спазването на тези две прости правила наистина ще ни позволи да доказваме твърденията по такъв начин, че доказателствата да са верни без значение дали тълкуваме твърденията както в конструктивната математика, или както в класическата.

Двойно отрицание

Нека видим защо в интуиционистката логика двойното отрицание не се унищожава. Нека A е следното твърдение:

> „Съществува цифра, която се среща безброй много пъти в десетичното представяне на π“.

Тъй като има само десет различни цифри — 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9 — то няма как всяка от тях да се среща само краен брой пъти в десетичното представяне на числото π. Следователно твърдението A е вярно.

Ако обаче изтълкуваме това твърдение конструктивно, смисълът му ще стане следният:

> „Може да се намери цифра, която се среща безброй много пъти в десетичното представяне на π“.

Засега за нито една от десетте цифри не знаем със сигурност дали се среща безброй много пъти в десетичното представане на π и затова ако тълкуваме твърдението A конструктивно, то това твърдение не може да се счита за доказано. Най-вероятно всяка цифра среща безброй много пъти, но засега това не е нищо повече от предположение.

Да допуснем, че твърдението A е невярно, означава да допуснем, че не съществува цифра, която се среща безброй много пъти в десетичното представяне на π. От тук не е трудно да се стигне до противоречие, следователно това допускане не е вярно. Щом като допускането, че твърдението A не е вярно, води до противоречие, значи е вярно двойното отрицание на A.

И така, доказахме двойното отрицание на A, макар че засега самото твърдение A остава недоказано.

При интуиционистката логика всяко математическо твърдение носи определена информация. Когато информацията представлява някой конкретен математически обект или метод за преобразуване на математически обекти, казваме, че тази информация е конструктивна. Например информацията за конкретна цифра, която се среща безброй пъти в π е конструктивна. Когато пък информацията не ни дава конкретни математически обекти или методи, а само сведения за свойствата на математическите обекти, казваме, че тази информация е дескриптивна.

Например информацията, която казва, че има цифра, която се среща безброй пъти в π, но не ни казва коя точно е тази цифра, е дескриптивна.

Какъв е интуитивният смисъл на двойното отрицание? Оказва се, че можем да използваме двойно отрицание, за да премахнем конструктивната информация от едно твърдение. Например твърдението A ни дава конкретна цифра, която се среща безброй пъти в десетичното представяне на π. Тъй като засега не знаем коя е тази цифра, то твърдението A все още няма конструктивно доказателство. Отрицанието на A казва, че няма цифра, която се среща безброй пъти, и значи двойното отрицание на A казва, че не е възможно да няма цифра, която се среща безброй пъти. Е, щом не е възможно да няма цифра, която се среща безброй пъти в десетичното представяне на π, значи такава цифра съществува. Твърдението „не не A“ обаче не ни дава информация коя е тази цифра — видяхме, че ние можем да докажем това твърдение дори и без да знаем коя точно цифра се среща безброй пъти.

5.46. Факт. *Твърдението A е дескриптивно, т.е. не носи конструктивна информация, тогава и само тогава, когато A е еквивалентно на „не не A“.*

Да отбележим, че няма смисъл да правим тройно отрицание на едно твърдение, защото тройното отрицание е еквивалентно на единичното. Това е така, защото единичното отрицание на едно твърдение просто казва, че нещо е невъзможно или невярно, и не ни носи никаква конструктивна информация. Единственото нещо, което прави двойното отрицание, пък е това да премахне конструктивната информация от едно твърдение. Затова ако приложим двойно отрицание към твърдение, което не съдържа конструктивна информация, например към единично отрицание, резултатът ще бъде еквивалентно на него твърдение.

Твърдение. *За произволно твърдение A*

а) от A следва „не не A“;

б) от „не A“ следва „не не не A“;

в) от „не не не A“ следва „не A“;

г) „не не не A“ е еквивалентно на „не A“.

Доказателство. **(а)** Да допуснем, че A е вярно. Трябва да докажем „не не A“. За да докажем това, да допуснем, че „не A“. От това веднага получаваме противоречие (имаме едновременно A и не A) и значи допускането „не A“ е невъзможно. Значи е вярно „не не A“.

(б) се получава от (а) като сложим „не A“ на мястото на A.

(в) Да допуснем, че „не не не A“. Трябва да докажем, че „не A“. За да докажем това, да допуснем, че A е вярно. От тук и от (а) получаваме „не не A“, а това ни дава противоречие (имаме „не не A“ и отрицанието на „не не A“). Противоречието се дължи на допускането, че A е вярно и значи A не е вярно.

(г) следва от (б) и (в). ∎

Забележка: От тук нататък, когато искаме да изкажем твърдение с двойно отрицание, ще използваме изрази от вида „неможе да не е вярно, че ...“. Например следното твърдение е вярно конструктивно: „не може да не съществува цифра, която се среща безброй много пъти в десетичното представяне на числото π“.

Превод на класическата логика в интуиционистката

Благодарение на двойното отрицание, всичко, което можем да правим в класическата математика, може да бъде направено и в конструктивната. Вече споменахме, че има две логически операции, които в интуиционистката логика се тълкуват по-различно, отколкото в класическата — кванторът за съществуване и дизюнкцията. Конструктивното съществуване ни казва кой точно е съществуващият обект, докато класическото съществуване ни казва, че обектът съществува по принцип, без да ни дава метод как да намерим този обект. Конструктивната дизюнкция „A или B“ ни казва дали A е вярно, или B, докато при класическата дизюнкция може и да не знаем кое точно от тези две твърдения е вярно. Също така видяхме, че можем да използваме двойно отрицание, за да премахваме конструктивната информация от едно твърдение. Това означава, че можем да използваме двойно отрицание, за да преведем всяко твърдение, използващо класическата логика, на езика на интуиционистката логика.

5.47. Факт. *Ако вземем твърдение, изказано използвайки класическата логика, и пред всеки квантор за съществуване, както и пред всяка дизюнкция, сложим двойно отрицание, ще получим равносилно на него твърдение, изказано използвайки интуиционистката логика.*

Този превод на класическата логика в интуиционистката се нарича *отрицателен превод*.

Конструктивната математика включва в себе си класическата математика, но я разширява по съществен и нетривиален начин. Така например Аритметиката на Пеано представлява формална теория, в която могат да се изказват твърдения за естествени числа. Конструктивен аналог на Аритметиката на Пеано е Аритметиката на Хейтинг. Всяко твърдение, което може да се формулира на езика на Аритметиката на Пеано, може да се преведе и на езика на Аритметиката на Хейтинг и всяко твърдение, което може да се докаже от първата теория, може да се докаже и от втората. На езика на Аритметиката на Хейтинг обаче могат да се формулират и много твърдения, които не могат да се изкажат на езика на Аритметиката на Пеано.*

Аналогично е положението и с Теорията на Цермело – Френкел (ZF), която представлява формална теория, в която могат да се изказват твърдения за множества и обикновено се използва като основа на съвременната класическа математика. Конструктивен аналог на ZF е Интуиционистката теория на Цермело – Френкел (IZF). Всяко твърдение, което може да се формулира на езика на ZF, може да се преведе и на езика на IZF и всяко твърдение, което може да се докаже от първата теория, може да се докаже и от втората. На езика на IZF обаче могат да се формулират и много твърдения, които не могат да се изкажат на езика на ZF.

Забележка: Исторически конструктивната математика възникнала във връзка със желанието да се даде по-обоснован фундамент на математиката. Считало се, че класическата математика е необоснована и заплашена от вътрешни противоречия, докато твърдения в конструктивната математика имат ясен смисъл и затова е много невероятно в нея да се открие противоречие. Реалното сравнение на теоретикодоказателствената сила на различни класически и конструктивни теории показва, че това мнение е неправилно. Например ако някога се открие противоречие в теорията ZF, то и теорията IZF ще се окаже противоречива. Въпросът, който има реално значение за основите на математиката, е не това дали да правим математиката конструктивно, или не, а това дали да позволяваме т.н. *непредикативни дефиниции*. Мнението, че конструктивната математика е „по-обоснованият“ начин за правене на математика, се дължи на това, че теорията ZF е непредика-

*Едно такова твърдение е *тезисът на Чърч*. На езика на Аритметиката на Хейтинг тезисът на Чърч може да се формулира така:

$$\forall \mathtt{x}\, \exists \mathtt{y}\, \varphi[\mathtt{x}, \mathtt{y}] \Rightarrow \exists \mathtt{e}\, \forall \mathtt{x}\, (!\{\mathtt{e}\}(\mathtt{x}) \,\&\, \varphi[\mathtt{x}, \{\mathtt{e}\}(\mathtt{x})])$$

тивна, докато почти всички математици-конструктивисти предпочитат да правят математиката предикативно. Вместо IZF те използват теорията CZF (която е предикативен аналог на IZF), теория на типовете или дори език без множества.

Изоморфизъм на Къри – Хауърд

Приложенията на интуиционистката логика далеч не се изчерпват само с това, че тя ни позволява да разработваме математиката конструктивно. Оказва се, че всяко твърдение, изказано посредством интуиционистката логика, може да се интерпретира като тип данни, а всяко интуиционистко доказателство ни дава обект, притежаващ съответния тип данни. Вярно е и обратното — типовете данни в езиците за програмиране може да се интерпретират като интуиционистки твърдения, а обектите, дефинирани в компютърните програми — като интуиционистки доказателства. Тази връзка между логика и програмиране се нарича *изоморфизъм на Къри – Хауърд*.

За да си обясним как е възможно твърденията да бъдат типове данни, а програмните обекти — доказателства, да разгледаме един пример. Нека A е твърдението

„Можем да намерим мегапираня“,

а B е твърдението

„Можем да намерим хипопозавър“.

Конструктивната информация, която се съдържа в твърдението A, е метод за намиране на мегапираня, а конструктивната информация, която се съдържа в твърдението B, е метод за намиране на хипопозавър. Да докажем твърдението A означава да посочим метод за намиране на мегапираня, а да докажем B означава да посочим метод за намиране на хипопозавър. На твърдението A съответства типът данни „метод за откриване на мегапираня“, а на твърдението B — „метод за откриване на хипопозавър“. Всеки обект, чийто тип е „метод за откриване на мегапираня“, може да се счита за доказателство на твърдението A и всеки обект, чийто тип е „метод за откриване на хипопозавър“, може да се счита за доказателство на твърдението B.

Нека видим как изглежда това съответствие между твърдения и типове и между доказателства и програмни обекти при различните логически операции.

Конюнкция

Твърдението „A и B“ казва, че A и B са верни, т.е. разполагаме с метод за откриване на мегапираня и метод за откриване на хипопозавър. Това означава, че информацията, която се съдържа в твърдението „A и B“ е комбинация от информацията, която се съдържа в A, и информацията, която се съдържа в B.

Следователно на твърдението „A и B“ съответства следният тип данни: „наредена двойка от метод за откриване на мегапираня и метод за откриване на хипопозавър“. На хаскел този тип се обозначава така:

`(`A`,`B`)`

В математиката доказваме конюнкцията по следния начин:

– Доказваме A.

– Доказваме B.

– От тук заключаваме, че е вярно „A и B“.

Ако x е обект от тип A, а y е обект от тип B, тогава наредената двойка от x и y на хаскел се обозначава така:

`(`x`,`y`)`

Следователно, ако си мислим x като доказателство на A и y — като доказателство на B, то `(`x`,`y`)` ще бъде доказателство на „A и B“

Обратно, ако вече сме доказали „A и B“, от тук директно можем да получим като следствия A и B. Също и на хаскел ако вече разполагаме с обект z от тип `(`A`,`B`)`, тогава

`fst` z

ще ни даде първия елемент на z, а

`snd` z

ще ни даде втория. Следователно ако z е доказателство на „A и B“, то „`fst` z“ ще бъде доказателство на A, а „`snd` z“ ще бъде доказателство на B.

Дизюнкция

Когато тълкуваме дизюнкцията конструктивно, твърдението „A или B“ има следния смисъл:

> „Може да се установи дали можем да намерим мегапираня,
> или можем да намерим хипопозавър.“

Следователно твърдението „A или B“ ни дава метод за откриване на мегапираня или метод за откриване на хипопозавър.

На твърдението „A и B“ съответства тип данни, който в известен смисъл представлява обединение на типовете A и B. Всеки обект от тип „A или B“ ни дава обект от тип A или обект от тип B. На хаскел този тип се обозначава така:

`Either` A B

В математиката доказваме дизюнкцията по следните два начина:

– Доказваме A.

– От тук заключаваме, че е вярно „A или B“.

Втори начин:

– Доказваме B.

– От тук заключаваме, че е вярно „A или B“.

Също и на хаскел ако x е обект от тип A, тогава

`Left(`x`)`

е обект от тип „`Either` A W“, където на мястото на W може да стои произволен тип. Значи ако си мислим x като доказателство на A, тогава „`Left(`x`)`“ представлява доказателство на „A или B“.

Аналогично, ако y е обект от тип B, тогава

`Right(`y`)`

е обект от тип „`Either` W B“, където на мястото на W може да стои произволен тип. Ако си мислим y като доказателство на B, тогава „`Right(`y`)`“ представлява доказателство на „A или B“.

Обратно, ако вече сме доказали „A или B“, начинът да използваме това твърдение е да разглеждаме случаи:

– Първи случай — вярно е A. Използвайки A, доказваме някакво твърдение C.

– Втори случай — вярно е B. Използвайки B, доказваме някакво твърдение C.

– От тук заключаваме, че C е вярно.

Аналогично на това, и на хаскел ако вече разполагаме с обект z от тип `Either` A B, един начин да го използваме е следната конструкция:

```
u = case z of
   Left(x) →
      използвайких намирамеобектоттипC
   Right(y) →
      използвайкиу намирамеобектоттипC
```

Импликация

Твърдението „ако A, то B“ има следния смисъл:

> „Ако можем да намерим мегапираня, то ще можем да намерим хипопозавър.“

Следователно твърдението „ако A, то B“ ни дава метод, посредством който ако ни бъде даден метод за откриване на мегапираня, ще можем да получим метод за откриване на хипопозавър.

На твърдението „ако A, то B“ съответства типът данни „функция, чийто аргумент е от тип A (т.е. метод за откриване на мегапираня), а стойността — от тип B (т.е. метод за откриване на хипопозавър)“. На хаскел този тип се обозначава така:

```
A -> B
```

В математиката доказваме импликацията по следния начин:

– Допускаме, че A е вярно.

– Започваме да правим разсъждения, използващи A.

– В края на тези разсъждения доказваме B

– От тук заключаваме, че от A следва B.

На доказателство от този вид съответства приблизително следният код на хаскел:

```
\x →
   let ...
       ...използвайки x дефинирамеразниобекти
       ...
   in
      изразоттипB
```

Този код представлява анонимна функция с аргумент `x` от тип A, която връща стойност от тип B.

За да можем да дадем на хаскел по-конкретен пример, да разгледаме твърдението „ако A и B, то B и A“. Това твърдение е очевидно вярно. На хаскел на него съответства функция, която взема като аргумент наредена двойка `(a,b)` и връща като стойност наредената двойка `(b,a)`:

```
\x → -- x еаргументътнафункцията
   let a = fst(x) -- приемаме, чех е
       b = snd(x) -- нареденатадвойка(a,b)
   in
      (b,a) -- връщамекатостойност(b,a)
```

Ако вече сме доказали твърденията A и „ако A, то B“, то от тях ще получим като следствие и твърдението B.

Аналогично на това, и на хаскел ако x е обект от тип A и f е функция от тип A->B, то $f(x)$ ще бъде обект от тип B.

Отрицание

Отрицанието в интуиционистката логика има две особености, които ще приемем без да ги обосноваваме.

Първата особеност на отрицанието е следната: приемаме, че ако дадено твърдение не е вярно, то от него следва всяко твърдение. Тази особеност на интуиционистката логика е присъща и на класическата логика, а значи и на цялата математика.

Втората особеност на отрицанието е следната. Да си представим на ум, че по някакъв начин сме се сдобили с мегапираня, и да допуснем, че от тук можем да стигнем до противоречие. С други думи, допускането, че по някакъв начин сме намерили мегапираня, води до противоречие. Ще можем ли да стигнем до противоречие ако допуснем, че съществува мегапираня без да допускаме, че сме намерили тази мегапираня? От правилата на интуиционистката логика можем да заключим, че отговорът на този въпрос трябва да е положителен.

Смисълът на твърдението „не A“ е следният:

> „Не съществува мегапираня.“

Начинът, по който доказваме такова твърдение е следният:

– Допускаме, че A е вярно.

– Започваме да правим разсъждения, използващи A.

– В края на тези разсъждения стигаме до противоречие

– От тук заключаваме, че A не е вярно.

На хаскел няма непосредствен начин за представяне на отрицанието. Но тъй като твърденията „A не е вярно“ и „от A следва противоречие“ са еквивалентни, то може да считаме, че на твърдението „не A“ съответства типът A `-> Impossible`, където `Impossible` е празният тип, т.е. тип който няма нито една стойност. Празният тип не е вграден, но в хаскел 2010 той може да се дефинира така:

```
data Impossible
```

Квантор за всеобщност

Да разгледаме твърдението

> „За всеки зъб на мегапираня може да се намери мегапираня, която го е притежавала.“

Това твърдение ни дава метод, посредством който ако ни бъде даден зъб от мегапираня, ще можем да получим метод за намиране на мегапиранята, която е притежавала този зъб. На това съответства типът данни „функция, чийто аргумент е обект x от тип „мегапирански зъб“, а стойността от тип „мегапираня, притежател на x“.

На пръв поглед, разгледан като тип данни, кванторът за всеобщност се интерпретира по същия начин, както и импликацията — като функция. Има обаче и една съществена разлика. При импликацията типовете на аргумента и на стойността са фиксирани и отнапред известни. При квантора за всеобщност аргументът също е има фиксиран тип, но типът на върнатата стойност зависи от аргумента, защото в израза „мегапираня, притежател на x“ се споменава x. Такива типове се наричат *зависими* (dependent types).

Един друг пример за зависим тип ни дава функцията, която получава като аргумент някакво естествено число n и връща като стойност n-мерен нулев вектор. Аргументът на тази функция е от тип „естествено число“, а връщаната стойност — „n-мерен вектор“ е зависим тип, защото зависи от n.

Хаскел няма вградена поддръжка за зависими типове,[*] но често няма проблем да пишем програми и да си мислим, че те използват зависими типове — просто компилаторът ще пресметне някакви по-общи типове. Например разгледаната по-горе функция, която връща n-мерен нулев вектор, ще връща стойност от тип „вектор“ без да се уточнява, че векторът е n-мерен.

В математиката доказваме твърдение от вида „за всяко x е вярно $C(x)$“ по следния начин:

- Избираме произволен обект x.
- Започваме да правим разсъждения за x.
- В края на тези разсъждения доказваме $C(x)$.
- От тук заключаваме, че за всяко x е вярно $C(x)$.

На доказателство от този вид съответства приблизително следният код на хаскел:

[*] Има начин поведението им донякъде да се имитира, вж. [11].

```
\x →
   let ...
       ...използвайки x дефинирамеразниобекти
       ...
   in
      изразоттип C(x)
```

Този код представлява анонимна функция с аргумент `x`, която връща стойност от тип $C(x)$.

Ако вече сме доказали твърдението „за всяко x е вярно $C(x)$“, то за всеки конкретен обект x ще можем да получим като следствие $C(x)$.

Аналогично на това, и на хаскел ако f е функция която по аргумент x ни връща обект от тип $C(x)$, то $f(x)$ ще бъде обект от тип $C(x)$.

Квантор за съществуване

Да разгледаме твърдението

> „Съществува мегапираня, която е загубила зъб в река Ориноко.“

От това твърдение можем да получим следната информация:

– някоя мегапираня;

– зъб, загубен от тази мегапираня в река Ориноко.

На това съответства типът данни наредена двойка, чийто пръв елемент x е от тип „мегапираня“, а вторият — от тип „зъб, загубен от x в река Ориноко“.

На пръв поглед, разгледан като тип данни, кванторът за съществуване се интерпретира по същия начин както конюнкцията — като наредена двойка. Има обаче и една съществена разлика — типът на втория елемент на тази наредена двойка зависи от x и значи е зависим тип.

Няколко примера на хаскел

В следващите няколко примера най-напред ще дадем програмен код на хаскел, а след това ще коментираме как той може да се интерпретира като математическо доказателство.

Когато на хаскел дефинираме някакъв обект A, с инструкцията `:t` A можем да получим неговия тип. Навсякъде в дадените примери `Prelude` е командният показалец на интерпретатора — текстът след

него се въвежда от нас. Текстът пък, който не е предшестван от команден показалец, се отпечатва от интерпретатора. Например в следния пример

```
Prelude> let x = 'q'
Prelude> :t x
x :: Char
```

на първия ред дефинираме променлива `x` със стойност `'q'`. На втория ред питаме интерпретатора какъв е типът на `x`. На третия ред интерпретаторът отговаря, че типът на `x` е `Char`.

Първият пример, с който ще илюстрираме съответствието между програми и доказателства, е следният:

```
Prelude> let composition(f,g) = h
Prelude|         where h(x) = g(f(x))
Prelude|
Prelude> :t composition
composition :: (t2 -> t1, t1 -> t) -> t2 -> t
```

От програмна гледна точка тук дефинираме функцията `composition`, която получава като аргумент наредена двойка от две функции `f` и `g` и връща като стойност тяхната композиция `h`. Ако превърнем типа на функцията `composition` в логическа формула ще получим формулата

$$(\varphi \Rightarrow \psi)\ \&\ (\psi \Rightarrow \chi) \Rightarrow (\varphi \Rightarrow \chi)$$

в която φ, ψ и χ са формулите, отговарящи съответно на автоматичните типове `t2`, `t1` и `t`. Да забележим, че тази формула е тавтология. Функцията `f`, която `composition` получава като аргумент, е от тип `t2->t1`, т.е. $\varphi \Rightarrow \psi$, а функцията `g` — от тип `t1->t`, т.е. $\psi \Rightarrow \chi$. Стойността `h`, която `composition` трябва да върне е от тип $\varphi \Rightarrow \chi$. За да получим `h`, да допуснем, че притежаваме обект `x` от тип φ. Ако към `x` приложим `f`, ще получим обект от тип ψ. Ако към този обект приложим `g`, ще получим обект от тип χ. По този начин получаваме функция, която получава аргумент от тип φ и връща стойност от тип χ и значи типът на тази функция е $\varphi \Rightarrow \chi$.

Това е „програмистката“ интерпретация на случващото се във функцията `composition`. Да видим сега какво доказателство съответства на тази функция. Тъй като аргументът ѝ е от тип $(\varphi \Rightarrow \psi)\ \&\ (\psi \Rightarrow \chi)$, то доказателството започва с изречението „Да допуснем, че е вярно $(\varphi \Rightarrow \psi)\,\&\,(\psi \Rightarrow \chi)$“. В дефиницията на функцията `composition`, аргументът ѝ е наредената двойка `(f,g)`. Това е все едно в доказателството за удобство да означим $(\varphi \Rightarrow \psi)$ с `f` и $(\psi \Rightarrow \chi)$ с `g`. Тъй като `composition` връща като стойност функцията `h`, то доказателството продължава според

дефиницията на функцията `h`. Аргументът `x` на `h` е от тип φ, и значи доказателството продължава с изречението „Да допуснем, че е вярно φ“. След това h връща `g(f(x))`. На това отговаря следното изречение „От φ и `f` получаваме ψ, а от тук и `g` получаваме χ“.

И така, на дадената по-горе програма отговаря следното доказателство:

Да допуснем, че $(\varphi \Rightarrow \psi)$ & $(\psi \Rightarrow \chi)$. Тогава

$$\varphi \Rightarrow \psi \qquad (\texttt{f})$$

и

$$\psi \Rightarrow \chi \qquad (\texttt{g})$$

Да допуснем, че е вярно φ. От φ и `f` получаваме ψ, а от ψ и `g` получаваме χ.

Допуснахме φ и доказахме χ. Значи е вярно

$$\varphi \Rightarrow \chi \qquad (\texttt{h})$$

Допуснахме $(\varphi \Rightarrow \psi)$ & $(\psi \Rightarrow \chi)$ и доказахме $\varphi \Rightarrow \chi$, значи е вярно

$$(\varphi \Rightarrow \psi) \,\&\, (\psi \Rightarrow \chi) \Rightarrow \varphi \Rightarrow \chi \qquad (\texttt{composition})$$

Да разгледаме друг пример.

```
Prelude> let dis(x) = Left x
Prelude> :t dis
dis :: a -> Either a b
```

От програмна гледна точка тук дефинираме функцията `dis`, която получава обект от произволен тип `a` и връща като стойност обекта `Left(x)` от тип `Either a b`. Да си припомним, че на типа `Either a b` съответства формулата $\varphi \vee \psi$, където φ е формулата, съответстваща на типа `a`, а ψ е формулата, съответстваща на типа `b`. Следователно формулата, съответстваща на типа на функцията `dis` е

$$\varphi \Rightarrow \varphi \vee \psi$$

Да забележим, че тази формула е тавтология.

Извикването на функцията `dis` се свежда до извикването на конструктора `Left`. Типът `Either a b` има още един конструктор — `Right`, с чиято помощ можем да докажем и формулата $\psi \Rightarrow \varphi \vee \psi$.

Разгледаният пример с дизюнкция дава тривиално от математическа гледна точка доказателство. Ето един по-интересен пример, също използващ дизюнкция:

```
Prelude> let cases(f,g) = h
Prelude|         where
Prelude|             h(x) = case x of
Prelude|                   Left(y) -> f(y)
Prelude|                   Right(z) -> g(z)
Prelude|
Prelude> :t cases
cases :: (t1 -> t, t2 -> t) -> Either t1 t2 -> t
```

Преведен като формула, типът на функцията `cases` е

$$(\varphi \Rightarrow \chi)\ \&\ (\psi \Rightarrow \chi) \Rightarrow (\varphi \vee \psi \Rightarrow \chi)$$

където φ, ψ и χ са формулите, съответстващи на автоматичните типове `t1`, `t2` и `t`. Функцията `h`, която функцията `cases` връща като стойност, има за аргумент обект `x` от тип `Either t1 t2`, т.е. $\varphi \vee \psi$. Тази функция разглежда два случая в зависимост от това дали `x` има вида `Left(y)`, където `y` е от тип φ, или `Right(z)`, където `y` е от тип ψ. В първия случай `h` връща като стойност `f(y)`, а във втория — `g(z)`.

Да видим как да интерпретираме това като математическо доказателство на формулата $(\varphi \Rightarrow \chi)\ \&\ (\psi \Rightarrow \chi) \Rightarrow (\varphi \vee \psi \Rightarrow \chi)$. Щом като аргументът на `cases` е от тип $(\varphi \Rightarrow \chi)\ \&\ (\psi \Rightarrow \chi)$, то значи доказателството започва с думите „Да допуснем, че $(\varphi \Rightarrow \chi)\ \&\ (\psi \Rightarrow \chi)$“. Аргументът на `h` има вида `(f,g)`, което е все едно в математическото доказателство за краткост да означим $\varphi \Rightarrow \chi$ с `f` и $\psi \Rightarrow \chi$ с `g`. Функцията `cases` връща като стойност функцията h, така че доказателството продължава според дефиницията на `h`. Функцията `h` има аргумент от тип $\varphi \vee \psi$, което значи, че доказателството продължава с изречението „Да допуснем, че е вярно $\varphi \vee \psi$“. След това в `h` се разглеждат случаи според вида на аргумента на `h`. В математическото доказателство това отговаря на това да разгледаме случаи в зависимост от това дали е вярно φ или ψ. При първия случай пресмятаме `f(y)`, където `y` е от тип φ, а `f` е от тип $\varphi \Rightarrow \chi$. В математическото доказателство това отговаря на изречението: „Щом като φ е вярно, а според `f` от φ следва χ, то значи χ е вярно.“ Във втория случай пресмятаме `g(z)`, където `z` е от тип ψ, а `g` е от тип $\psi \Rightarrow \chi$. В математическото доказателство това отговаря на изречението: „Щом като ψ е вярно, а според `g` от ψ следва χ, то значи и χ е вярно.“

И така, на дадената по-горе програма отговаря следното доказателство:

Да допуснем, че $(\varphi \Rightarrow \chi)\ \&\ (\psi \Rightarrow \chi)$ Тогава

$$\varphi \Rightarrow \chi \qquad (\texttt{f})$$

и

$$\psi \Rightarrow \chi \qquad (\texttt{g})$$

Да допуснем, че

$$\varphi \vee \psi \qquad (\texttt{x})$$

и да разгледаме случаи в зависимост от това дали е вярно φ или ψ.

В първия случай, когато е вярно φ, от `f` получаваме χ.

Във втория случай, когато е вярно ψ, от `g` отново получаваме χ.

И в двата случая получихме χ.

Допуснахме $\varphi \vee \psi$ и доказахме χ, значи е вярно

$$\varphi \vee \psi \Rightarrow \chi \qquad (\texttt{h})$$

Допуснахме $(\varphi \Rightarrow \chi)\,\&\,(\psi \Rightarrow \chi)$ и доказахме $\varphi \vee \psi \Rightarrow \chi$, значи е вярно

$$(\varphi \Rightarrow \chi)\,\&\,(\psi \Rightarrow \chi) \Rightarrow (\varphi \vee \psi \Rightarrow \chi) \qquad (\texttt{cases})$$

5.6. Нормални форми

В много дялове на математиката се използват т.н. нормални форми. Когато всеки обект от някой тип може да бъде представен посредством израз, имащ определен вид, казваме, че този израз е нормалната форма на обекта. Когато искаме да дефинираме в компютърна програма тип, чиито обекти притежават нормална форма, не е нужно да се грижим за компютърно представяне на изрази, които не са в нормална форма.

Ето няколко примера.

– От дискретната математика знаем, че булевите функции могат да се представят в конюнктивна нормална форма, в дизюнктивна нормална форма и като полиноми.

– Всеки алгоритъм, обработващ естествени числа, може да се реализира посредством компютърна програма, имаща единствено оператори за присвояване, оператори за четене и запис на входни и изходни данни, само един оператор за цикъл и никакви други оператори. Това е известно като нормална форма на Клѝни.

– В алгебрата всяка матрица може да се представи в жорданова нормална форма.

– Представянето на всяко естествено число като произведение на прости числа също може да се разглежда като нормална форма.

– В ламбда-смятането се дефинира понятието редукция на ламбда-термове. Когато един ламбда-терм бъде доведен посредством редукция до ламбда-терм, който не може повече да се редуцира, казваме, че сме получили бета нормална форма. Бета нормалната форма е резултатът от изчислението (следователно зациклящите се изчислителни процеси не притежават бета нормална форма).

В този раздел ще дефинираме няколко нормални форми за формули класическата предикатна логика от първи ред.* Нещата, които са верни само при класическата логика, но не и при интуиционистката логика, ще бъдат отбелязвани с **(клас)**.

5.48. Да изберем някоя затворена формула, която е невярна във всяка структура, и да я означим с $\bot$. (Такава формула винаги съществува. Ако φ е произволна формула, нека $\varphi' = \forall \mathtt{x}_1 \forall \mathtt{x}_2 \ldots \forall \mathtt{x}_n(\varphi)$, където всички свободни променливи на φ са измежду променливите $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$. Формулата φ' е затворена. Тогава нека $\bot$ бъде формулата $\varphi' \,\&\, \neg\varphi'$.)

5.49. Твърдение. *За произволна формула φ:*

а) $\vDash \neg\varphi \Leftrightarrow (\varphi \Rightarrow \bot)$

б) $\vDash \neg\neg\varphi \Leftrightarrow \varphi$ **(клас)**

Доказателство. **(а)** Да си припомним, че $\neg\varphi \Leftrightarrow (\varphi \Rightarrow \bot)$ е съкращение на формулата $(\neg\varphi \Rightarrow (\varphi \Rightarrow \bot)) \,\&\, ((\varphi \Rightarrow \bot) \Rightarrow \neg\varphi)$. Следователно трябва да докажем, че $\vDash \neg\varphi \Rightarrow (\varphi \Rightarrow \bot)$ и $\vDash (\varphi \Rightarrow \bot) \Rightarrow \neg\varphi$.

Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$. Формулата $\neg\varphi \Rightarrow (\varphi \Rightarrow \bot)$ е вярна в структурата $\mathbf{M}$ при оценка v тогава и само тогава, когато е вярно твърдението

ако A е лъжа, то от A следва лъжа

Това твърдение е очевидно вярно.

Формулата $(\varphi \Rightarrow \bot) \Rightarrow \neg\varphi$ е вярна в структурата $\mathbf{M}$ при оценка v тогава и само тогава, когато е вярно твърдението

ако от A следва лъжа, то A е лъжа

Това твърдение също е очевидно вярно.

(б) Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$. Трябва да докажем, че твърдението „не не A“ еквивалентно на A. При използване на класическата логика това е очевидно. ∎

*Няма хубави нормални форми за интуиционистката предикатна логика.

5.50. Твърдение. *За произволни формули φ и ψ:*

а) $\models \neg(\varphi \vee \psi) \Leftrightarrow \neg\varphi \,\&\, \neg\psi$

б) $\models \neg(\varphi \,\&\, \psi) \Leftrightarrow \neg\varphi \vee \neg\psi^{*}$ **(клас)**

в) $\models \neg(\varphi \Rightarrow \psi) \Leftrightarrow \varphi \,\&\, \neg\psi^{**}$ **(клас)**

Доказателство. **(а)** Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \models \varphi[v]$, а B е твърдението $\mathbf{M} \models \psi[v]$.

Най-напред да докажем, че $\mathbf{M} \models \neg(\varphi \vee \psi) \Rightarrow \neg\varphi \,\&\, \neg\psi[v]$. За целта да допуснем, че не е вярно твърдението „A или B“. Трябва да докажем, че не е вярно A и не е вярно B. За да докажем, че не е вярно A, да допуснем, че A е вярно. Щом A е вярно, значи е вярно и твърдението „A или B“. Но това е противоречие, значи A не е вярно. Аналогично се вижда, че и B не е вярно.

За да докажем, че $\mathbf{M} \models \neg\varphi \,\&\, \neg\psi \Rightarrow \neg(\varphi \vee \psi)[v]$, да допуснем, че не е вярно A и не е вярно B. Трябва да докажем, че не е вярна дизюнкцията „A или B“. За да докажем, че тази дизюнкция не е вярна, да допуснем, че тя е вярна и да разгледаме случаи в зависимост от това дали е вярно A или B. Когато е вярно A получаваме противоречие, защото знаем, че A не е вярно. Но когато е вярно B също получаваме противоречие, защото знаем, че B не е вярно. И в двата случая стигаме до противоречие. То се дължи на допускането, че дизюнкцията „A или B“ е вярна.

(б) Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \models \varphi[v]$, а B е твърдението $\mathbf{M} \models \psi[v]$.

Най-напред да докажем, че $\mathbf{M} \models \neg(\varphi \,\&\, \psi) \Rightarrow \neg\varphi \vee \neg\psi[v]$. За целта да допуснем, че не е вярно твърдението „A и B“. Трябва да докажем, че A е невярно или B е невярно. От конструктивна гледна точка това означава, че ни е дадена функция, която получава аргументи от типове A и B и връща стойност от тип „противоречие“. Имайки само такава функция няма начин да направим функция, която получава само един аргумент от тип A и връща противоречие, нито пък функция, която получава само B и връща противоречие. Следователно няма как да конструираме обект от тип $\neg\varphi \vee \neg\psi$. Това ни подсказва, че ще трябва да допуснем противното.

И така, да допуснем, че не е вярно твърдението „A не е вярно или B не е вярно“. Ако допуснем, че A е невярно, ще получим че е вярно твърдението, за което току-що допуснахме, че е невярно. Значи твърдението A не може да не е вярно. Аналогично се вижда, че и B не може

[*] Интуиционистки е вярно $\models \neg(\varphi \,\&\, \psi) \Leftrightarrow \neg\neg(\neg\varphi \vee \neg\psi)$.

[**] Интуиционистки е вярно $\models \neg(\varphi \Rightarrow \psi) \Leftrightarrow \neg\neg\varphi \,\&\, \neg\psi$.

да не е вярно. Това обаче противоречи на дизюнкцията, казваща, че A не е вярно или B не е вярно.

За да докажем обратната посока, да допуснем, че не е вярно A или не е вярно B. Трябва да докажем, че не е вярно твърдението „A и B“. За целта да допуснем, че твърдението „A и B“ е вярно. Значи твърденията A и B са верни, но това веднага ни дава противоречие с дизюнкцията, според която не е вярно A или не е вярно B. ∎

5.51. Твърдение. *За произволна формула* φ

а) $\models \neg\exists \mathtt{x}\, \varphi \Leftrightarrow \forall \mathtt{x}\, \neg\varphi$

б) $\models \neg\forall \mathtt{x}\, \varphi \Leftrightarrow \exists \mathtt{x}\, \neg\varphi$ [*] **(клас)**

Доказателство. **(а)** Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. За произволен елемент μ на универсума на A, нека $A(\mu)$ е твърдението $\mathbf{M} \models \varphi[v^{\mu}_{\mathtt{x}}]$.

Най-напред ще докажем, че $\mathbf{M} \models \neg\exists \mathtt{x}\, \varphi \Rightarrow \forall \mathtt{x}\, \neg\varphi[v]$. За целта да допуснем, че не съществува $\mu \in |\mathbf{M}|$, за което $A(\mu)$ е истина. Трябва да докажем, че за всяко $\mu \in |\mathbf{M}|$ твърдението $A(\mu)$ е лъжа. Да изберем произволно $\mu \in |\mathbf{M}|$ и да допуснем, че $A(\mu)$ е истина. Вижда се, че това противоречи на допускането, че не съществува такова μ.

За да докажем обратната посока, да допуснем, че за всяко $\mu \in |\mathbf{M}|$ твърдението $A(\mu)$ е лъжа. Трябва да докажем, че не съществува $\mu \in |\mathbf{M}|$, за което $A(\mu)$ е истина. Да допуснем, че такова μ съществува. Тогава за това μ твърдението $A(\mu)$ ще бъде истина, което противоречи на допускането, че за всяко μ твърдението $A(\mu)$ е лъжа.

(б) Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. За произволен елемент μ на универсума на A, нека $A(\mu)$ е твърдението $\mathbf{M} \models \varphi[v^{\mu}_{\mathtt{x}}]$.

Най-напред ще докажем, че $\mathbf{M} \models \neg\forall \mathtt{x}\, \varphi \Rightarrow \exists \mathtt{x}\, \neg\varphi[v]$. Да допуснем, че не е вярно, че за всеки елемент μ на универсума на $\mathbf{M}$ е вярно $A(\mu)$. От конструктивна гледна точка това означава, че ни е дадена функция, която ще ни върне обект от тип противоречие, ако ѝ дадем като аргумент функция, която за всяко μ връща обект от тип $A(\mu)$. Не се вижда как, разполагайки с такава функция, ще можем да намерим обект μ, за който не е вярно $A(\mu)$. Следователно твърдението не може да се докаже конструктивно и значи трябва да допуснем противното.

[*] Интуиционистки е вярно $\models \neg\forall \mathtt{x}\, \varphi \Leftrightarrow \neg\neg\exists \mathtt{x}\, \neg\varphi$.

И така, да допуснем, че не съществува $\mu \in |\mathbf{M}|$, за което не е вярно $A(\mu)$. Използвайки доказаното в а), можем да заключим, че за всяко $\mu \in |\mathbf{M}|$ твърдението $A(\mu)$ е вярно.* Това е противоречие.

За да докажем обратната посока, да допуснем, че не съществува елемент μ на универсума на $\mathbf{M}$, за който е вярно твърдението $A(\mu)$. Трябва да докажем, че за всеки елемент μ на универсума на $\mathbf{M}$, твърдението $A(\mu)$ е невярно. Това е очевидно. ∎

5.52. Твърдение. *За произволни формули φ и ψ:*

$$\vDash (\varphi \Rightarrow \psi) \Leftrightarrow (\neg\varphi \vee \psi)^{**} \qquad \textbf{(клас)}$$

Доказателство. Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$, а B е твърдението $\mathbf{M} \vDash \psi[v]$.

Най напред ще докажем, че $\mathbf{M} \vDash (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)[v]$. Да допуснем, че от A следва B. Трябва да докажем, че A не е вярно или B е вярно. От конструктивна гледна точка това означава, че имаме функция, на която ако ѝ дадем аргумент от тип A, ще ни върне стойност от тип B. Не се вижда как имайки такава функция това ще ни позволи да установим кой от членовете на исканата дизюнкция е верен. Това ни подсказва, че желаното свойство не може да бъде доказано конструктивно и значи трябва да допуснем противното.

И така, да допуснем, че не е вярна дизюнкцията, казваща, че A е невярно или B е вярно. Това означава, че A е истина и B е лъжа (вж. доказателството на твърдение 5.50 а)). Но щом A е истина и от A следва B, значи B е истина. Стигнахме до противоречие.

За да докажем обратната посока, да допуснем, че A е невярно или B е вярно. Трябва да докажем, че от A следва B. За целта да допуснем, че A е вярно. Щом A е вярно, то значи дизюнкцията „A е невярно или B е вярно“ ни казва, че B е вярно. И така, допуснахме, че A е вярно и доказахме, че B е вярно. Следователно от A следва B. ∎

5.53. Преди много хилядолетия в град Лагаш, намиращ се в днешен Ирак, живял бог Нингирсу, който бил юначен ловец и воин. Веднъж, по време на един от ловните си походи, Нингирсу влязал в бой с многоглавия дракон Мушмаху. Всеки път когато Нингирсу отрязвал една от

*Всъщност от доказаното в а) можем да заключим, че за всяко $\mu \in |\mathbf{M}|$ твърдението $A(\mu)$ не може да не е вярно. Използвайки класическата логика, от тук, разбира се, получаваме, че за всяко $\mu \in |\mathbf{M}|$ твърдението $A(\mu)$ е вярно.

**Интуиционистки не е възможно да изразим импликацията посредством останалите съждителни операции. Вярно е обаче следното: $\vDash \neg\neg(\varphi \Rightarrow \psi) \Leftrightarrow \neg\neg(\neg\varphi \vee \psi)$.

главите на Мушмаху, на него му пораствали много нови глави. Отначало Нингирсу решил, че работата е безнадеждна, но после забелязал, че всеки път когато отрязвал някоя глава, шиите на новопоникналите глави били най-малко с един сантиметър по-къси. Понеже разбирал от математика, Нингирсу съобразил, че ако не се отказва, със сигурност ще се стигне до момент, когато драконът ще остане без глави. След като победил, Нингирсу се отправил към град Нипур, където се намирал зигуратът Екур — жилището на боговете. Нингирсу бил изпълнен с боен плам, надавал викове, пеел песни, с които прославял геройството си, и извършвал разни вандалства с цел да му обръщат внимание. Боговете в Нипур се уплашили и му казали, че ако се поуспокои малко, ще го възнаградят. Като стигнал Екур, той показал на удивените богове бойните си трофеи.

Логиците използват т.н. *ординални числа* с цел да „измерят“ колко е сложно дадено разсъждение, използващо индукция. Ако използваме обикновената индукция за естествени числа по възможно най-естествения начин, казваме, че сме използвали индукция до ординала ω. Ако обаче докато правим индукционната стъпка, вътре в нея за втори път започваме да правим индукция, казваме че сме използвали индукция до ординала ω^2. Ако пък и в индукционната стъпка на втората индукция пак правим индукция, тогава използваме индукция до ω^3. Според „великото предположение“ на Харви Фридман, всяка теорема, публикувана в математическо списание като „Annals of Mathematics“, чиято формулировка използва само крайни обекти, може да се докаже с индукция до ω^3. Въпреки това, оказва се, че само с индукция до ω^3 не можем да докажем, че драконът Мушмаху ще бъде победен. За целта ни е нужна индукция до $\omega^\omega = \sup\{\omega, \omega^2, \omega^3, \omega^4, \dots\}$.

Въпреки, че драконът Мушмаху бил убит, той си оставил достоен потомък — Лернейската хидра. Също както и при Мушмаху, и на Лернейската хидра за всяка отсечена глава ѝ пораства нови глави. Новите глави на Хидрата обаче растяли по малко по-различни правила, така че понякога се случвало шиите на новите глави да не са по-къси от шията на току-що отсечената глава. Хидрата била убита от Херкулес, защото отново се оказало, че без значение как ѝ режем главите, след краен брой стъпки тя ще остане без глави. Това обаче е изключително трудно да се докаже, защото изисква индукция до ординала $\varepsilon_0 = \sup\{\omega, \omega^\omega, \omega^{\omega^\omega}, \omega^{\omega^{\omega^\omega}}, \dots\}$.* Оказва се, че в стандартната аксиома-

*Всеки може сам да се постави на мястото на Херкулес и да се опита да победи Лернейската хидра, използвайки джава аплет от адрес `http://math.andrej.com/2008/02/02/the-hydra-game/`.

тична теория на аритметиката — Аритметиката на Пеано — е невъзможно да правим толкова сложни индукции и затова в тази теория не може да се докаже, че Хидрата ще бъде убита от Херкулес.

В следващата дефиниция да си мислим, че редицата от числа $(a_n)_{n=1}^{\infty}$ измерва броя на главите на дракона Мушмаху. Числото a_1 е равно на броя на главите, чиято шия е дълга 1 сантиметър, a_2 е равно на броя на главите, чиято шия е равна на 2 сантиметра и т.н.

5.54. Дефиниция. Нека $(a_n)_{n=1}^{\infty}$ е редица от естествени числа и редицата $(b_n)_{n=1}^{\infty}$ се получава като извършим следните промени в редицата $(a_n)_{n=1}^{\infty}$:

- избираме такова естествено число i, че $a_i \neq 0$;
- намаляваме a_i;
- заменяме $a_1, a_2, \ldots, a_{i-1}$ с произволни естествени числа.

В такъв случай казваме, че редицата $(a_n)_{n=1}^{\infty}$ се *конвертира* в редицата $(b_n)_{n=1}^{\infty}$. Също ще казваме, че редицата $(b_n)_{n=1}^{\infty}$ се получава от редицата $(a_n)_{n=1}^{\infty}$ посредством *конверсия*. Числото i ще наричаме *ранг на конверсията*.

5.55. Лема за фундираност на конверсията. *Нека $(a_n)_{n=1}^{\infty}$ е редица от естествени числа, в която има само краен брой ненулеви елементи. Да разгледаме следния процес: прилагаме конверсия към редицата $(a_n)_{n=1}^{\infty}$, после прилагаме конверсия към новополучената редица, след това отново прилагаме конверсия и т.н. Без значение какви точно конверсии извършваме, със сигурност след краен брой стъпки ще стигнем до редица, съдържаща само нули.*

Доказателство. Да забележим, че не е възможно да прилагаме безброй много конверсии, чийто ранг е 1. Наистина, след всяка такава конверсия първият член на редицата намалява, а това не може да се случва до безкрайност. Това означава, че както и да прилагаме конверсиите, след краен брой стъпки или ще стигнем до редица, съдържаща само нули, или ще стигнем до конверсия от ранг поне 2.

Да забележим също, че не е възможно да прилагаме безброй много конверсии, чийто ранг е 1 или 2. Наистина, вече видяхме, че не можем да прилагаме безброй конверсии само от ранг 1, следователно ако прилагаме само конверсии от ранг 1 или 2, ще трябва да прилагаме безброй пъти конверсии от ранг 2. Това обаче е невъзможно, защото след всяка конверсия от ранг 2 вторият член на редицата намалява, а това не може да се случва до безкрайност (да забележим, че конверсиите от

ранг 1 не променят втория член на редицата). Това означава, че както и да прилагаме конверсиите, след краен брой стъпки или ще стигнем до редица, съдържаща само нули, или ще стигнем до конверсия от ранг поне 3.

Обаче не е възможно също да прилагаме безброй много конверсии, чийто ранг е 1, 2 или 3. Наистина, вече видяхме, че не можем да прилагаме безброй конверсии само от рангове 1 или 2, следователно ако прилагаме само конверсии от ранг 1, 2 или 3, ще трябва да прилагаме безброй пъти конверсии от ранг 3. Това обаче е невъзможно, защото след всяка конверсия от ранг 3 третият член на редицата намалява, а това не може да се случва до безкрайност (да забележим, че конверсиите от рангове 1 или 2 не променят третия член на редицата). Това означава, че както и да прилагаме конверсиите, след краен брой стъпки или ще стигнем до редица, съдържаща само нули, или ще стигнем до конверсия от ранг поне 4.

Разсъждавайки по подобен начин, можем да стигнем до извода, че както и да прилагаме конверсиите, след краен брой стъпки или ще стигнем до редица, съдържаща само нули, или ще стигнем до конверсия от ранг поне k, където k е произволно предварително избрано естествено число. Нека числото k е такова, че всички ненулеви членове в $(a_n)_{n=1}^{\infty}$ са измежду $a_1, a_2, \ldots, a_{k-1}$. В такъв случай няма да бъде възможна конверсия от ранг k или по-голям, следователно както и да прилагаме конверсиите, със сигурност ще стигнем до редица, съдържаща само нули. ∎

5.56. Дефиниция. Казваме, че дадена формула е в *отрицателна нормална форма*, ако:

- във формулата няма други логически операции, освен конюнкция ($\&$), дизюнкция ($\vee$), отрицание ($\neg$) и кванторите за всеобщност ($\forall$) и съществуване ($\exists$);
- всички отрицания във формулата се намират пред атомарни подформули.

5.57. Твърдение. *За всяка формула може да се намери еквивалентна на нея (според класическата логика) формула, която е в отрицателна нормална форма.*

Доказателство. Нека φ е произволна формула. Най-напред да елиминираме импликациите във φ по следния начин: да заменяме докато може всяка подформула във φ, имаща вида $\psi' \Rightarrow \psi''$, с формулата $\neg\psi' \vee \psi''$. Съгласно твърдение 5.52 след всяка такава замяна получаваме еквивалентна формула. Тъй като след всяка замяна броят на

импликациите намалява, след краен брой стъпки ще получим формула без импликации, което значи, че във формулата няма други логически операции, освен конюнкция (&), дизюнкция ($\vee$), отрицание ($\neg$) и кванторите за всеобщност ($\forall$) и съществуване ($\exists$).

Остава да „преместим“ отрицанията пред атомарни формули. За целта да прилагаме докато може замени от следния вид:

$$\neg\neg\psi \longmapsto \psi \tag{46}$$
$$\neg(\psi' \vee \psi'') \longmapsto \neg\psi' \,\&\, \neg\psi'' \tag{47}$$
$$\neg(\psi' \,\&\, \psi'') \longmapsto \neg\psi' \vee \neg\psi'' \tag{48}$$
$$\neg\exists \mathtt{x}\, \psi \longmapsto \forall \mathtt{x}\, \neg\psi \tag{49}$$
$$\neg\forall \mathtt{x}\, \psi \longmapsto \exists \mathtt{x}\, \neg\psi \tag{50}$$

Съгласно твърдения 5.49 б), 5.50 а), 5.50 б), 5.51 а) и 5.51 б), след всяка такава замяна получаваме еквивалентна формула. Ако след краен брой стъпки получим формула, към която не можем да приложим никоя от тези замени, то това означава, че сме стигнали до формула в отрицателна нормална форма. Това ни дава т.н. частична коректност на алгоритъма за привеждане в отрицателна нормална форма. Остава да видим защо този алгоритъм никога не се зацикля, т.е. трябва да докажем, че можем да прилагаме замени от горния вид само краен брой пъти. Това ще направим по няколко начина.

(I начин) Ще използваме фундираността на конверсията (лема 5.55).

Нека χ е произволна формула. *Височина* на χ ще наричаме дължината на най-дългата редица от вида

$$\chi_1, \chi_2, \chi_3, \ldots, \chi_k$$

където $\chi_1 = \chi$, $\chi_i \neq \chi_{i+1}$ и χ_{i+1} е подформула на χ_i. Ако си мислим формулата като дърво, тогава височината ѝ е равна на дължината на най-дългия клон в това дърво.

За всяка формула φ ще дефинираме редица $(a_n)_{n=1}^{\infty}$, която ще наречем *редица на* φ, по следния начин: нека a_i е равно на броя на подформулите на φ, които започват с отрицание и са с височина точно i.

Може да се забележи, че ако формулата φ' се получава от φ посредством някоя от замените (46)–(50), тогава редицата на φ' може да се получи от редицата на φ посредством конверсия. Съгласно лемата за фундираност на конверсията, след краен брой стъпки или ще стигнем формула, към която не можем да приложим никоя от замените (46)–(50), или ще стигнем формула, чиято редица се състои само от

нули. Последното би означавало, че във формулата няма нито едно отрицание, но в този случай също е невъзможно да приложим замените (46)–(50).

(II начин) Да разгледаме следния начин, по който от формула получаваме аритметичен израз. Да заменим всяка атомарна формула с числото две. Да заменим всяка подформула от вида $\psi' \,\&\, \psi''$, $\psi' \vee \psi''$ и $\psi' \Rightarrow \psi''$ с $x + y$, където x и y са изразите, получени съответно от ψ' и ψ''. Да заменим всяка подформула от вида $\neg\psi$ с x^2, където x е изразът, получен от ψ. Да заменим всяка подформула от вида $\forall \mathtt{x}\, \psi$ и $\exists \mathtt{x}\, \psi$ с $1 + x$, където x е изразът, получен от ψ. Може да се забележи, че така полученият израз е естествено число, не по-малко от 2. Тъй като за всеки естествени числа по-големи или равни на 2 е изпълнено $(x^2)^2 > x$, $(x + y)^2 > x^2 + y^2$ и $(1 + x)^2 > 1 + x^2$, то стойността на получения израз ще намалява след всяка замяна от горния вид, а това не може да продължи до безкрайност. ∎

5.58. ***Забележка*:** Когато използваме първия начин за да докажем, че алгоритъмът за привеждане в отрицателна нормална форма спира, не е нужно да се използва толкова сложно твърдение като фундираността на конверсията. Нека редицата, съответстваща на някоя формула е $(a_n)_{n=1}^{\infty}$ и да разгледаме числото

$$\sum_{i=1}^{\infty} a_i 3^i$$

(сумата е коректна, защото само краен брой от числата a_i са ненулеви). Докато при дракона Мушмаху когато някоя негова глава бъде отсечена, може да му пораснат произволен брой нови глави, то тук, когато вкараме някое отрицание „навътре“ във формулата, то се заменя най-много с две „по-малки“ отрицания. Това означава, че след всяка замяна от вида (46)–(50), така дефинираното число ще намалява, а това не може да продължи до безкрайност.

5.59. Твърдение. *За произволни формули φ и ψ, ако* x *не е свободна променлива на φ, то:*

а) $\models \varphi \,\&\, \exists \mathtt{x}\, \psi \Leftrightarrow \exists \mathtt{x}\, (\varphi \,\&\, \psi)$ $u \models \exists \mathtt{x}\, \psi \,\&\, \varphi \Leftrightarrow \exists \mathtt{x}\, (\psi \,\&\, \varphi)$;

б) $\models \varphi \,\&\, \forall \mathtt{x}\, \psi \Leftrightarrow \forall \mathtt{x}\, (\varphi \,\&\, \psi)$ $u \models \forall \mathtt{x}\, \psi \,\&\, \varphi \Leftrightarrow \forall \mathtt{x}\, (\psi \,\&\, \varphi)$;

в) $\models \varphi \vee \exists \mathtt{x}\, \psi \Leftrightarrow \exists \mathtt{x}\, (\varphi \vee \psi)$ $u \models \exists \mathtt{x}\, \psi \vee \varphi \Leftrightarrow \exists \mathtt{x}\, (\psi \vee \varphi)$;

г) $\models \varphi \vee \forall \mathtt{x}\, \psi \Leftrightarrow \forall \mathtt{x}\, (\varphi \vee \psi)$ $u \models \forall \mathtt{x}\, \psi \vee \varphi \Leftrightarrow \forall \mathtt{x}\, (\psi \vee \varphi)$. **(клас)**

Доказателство. **(а)** Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$ и за произволен елемент μ на универсума на $\mathbf{M}$, нека $B(\mu)$ е твърдението $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$. Тъй като $\mathtt{x}$ не е свободна променлива на φ, то за всеки елемнт μ на универсума на $\mathbf{M}$, твърдението $\mathbf{M} \vDash \varphi[v^{\mu}_{\mathtt{x}}]$ е вярно тогава и само тогава, когато е вярно твърдението A (което очевидно не зависи от μ).

Най-напред ще докажем, че $\mathbf{M} \vDash \varphi \,\&\, \exists \mathtt{x}\, \psi \Rightarrow \exists \mathtt{x}\, (\varphi \,\&\, \psi)[v]$. Да допуснем, че твърдението A е вярно и съществува такова $\mu \in |\mathbf{M}|$, че твърдението $B(\mu)$ е вярно. В такъв случай очевидно съществува такова $\mu \in |\mathbf{M}|$, че твърдението „A и $B(\mu)$“ е вярно.

За да докажем обратната посока, да допуснем, че съществува такова $\mu \in |\mathbf{M}|$, че твърдението „A и $B(\mu)$“ е вярно. В такъв случай очевидно твърдението A ще бъде вярно и освен това ще съществува $\mu \in |\mathbf{M}|$, за което твърдението $B(\mu)$ е вярно.

(б) Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$ и за произволен елемент μ на универсума на $\mathbf{M}$, нека $B(\mu)$ е твърдението $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$. Тъй като $\mathtt{x}$ не е свободна променлива на φ, то за всеки елемнт μ на универсума на $\mathbf{M}$, твърдението $\mathbf{M} \vDash \varphi[v^{\mu}_{\mathtt{x}}]$ е вярно тогава и само тогава, когато е вярно твърдението A (което очевидно не зависи от μ).

Най-напред ще докажем, че $\mathbf{M} \vDash \varphi \,\&\, \forall \mathtt{x}\, \psi \Rightarrow \forall \mathtt{x}\, (\varphi \,\&\, \psi)[v]$. Да допуснем, че е вярно A и че за всяко $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$. В такъв случай очевидно за всяко $\mu \in |\mathbf{M}|$ ще бъде вярно твърдението „A и $B(\mu)$“.

За да докажем обратната посока, да допуснем, че за всяко $\mu \in |\mathbf{M}|$ е вярно твърдението „A и $B(\mu)$“. Тъй като универсумът на $\mathbf{M}$ е непразен, като приложим току-що допуснатото за някой елемент на универсума, ще получим, че твърдението A е вярно. Освен това очевидно за всяко $\mu \in |\mathbf{M}|$ ще бъде вярно твърдението $B(\mu)$.

(в) Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$ и за произволен елемент μ на универсума на $\mathbf{M}$, нека $B(\mu)$ е твърдението $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$. Тъй като $\mathtt{x}$ не е свободна променлива на φ, то за всеки елемнт μ на универсума на $\mathbf{M}$, твърдението $\mathbf{M} \vDash \varphi[v^{\mu}_{\mathtt{x}}]$ е вярно тогава и само тогава, когато е вярно твърдението A (което очевидно не зависи от μ).

Най-напред ще докажем, че $\mathbf{M} \vDash \varphi \vee \exists \mathtt{x}\, \psi \Rightarrow \exists \mathtt{x}\, (\varphi \vee \psi)[v]$. Да допуснем, че е вярно A или за някое $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$. В първия случай, когато е вярно A, ще съществува $\mu \in |\mathbf{M}|$, за което е вярно твърдението „A или $B(\mu)$“, защото кой да е елемент μ на универсума ще ни свърши

работа (а такъв има, защото универсъмът е непразно множество). Във втория случай, когато съществува $\mu \in |\mathbf{M}|$, за което е вярно $B(\mu)$, очевидно също ще съществува $\mu \in |\mathbf{M}|$, за което е вярно твърдението „A или $B(\mu)$“.

За да докажем обратната посока, да допуснем, че съществува $\mu \in |\mathbf{M}|$, за което е вярно твърдението „A или $B(\mu)$“. Ако за това μ, е вярно A, то A е вярно (това твърдение не зависи от μ). Ако пък за съществуващото μ е вярно $B(\mu)$, то значи съществува $\mu \in |\mathbf{M}|$, за което е вярно $B(\mu)$. Следователно A е вярно или съществува $\mu \in |\mathbf{M}|$, за което е вярно $B(\mu)$.

(г) Да изберем произволни структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$ и за произволен елемент μ на универсума на $\mathbf{M}$, нека $B(\mu)$ е твърдението $\mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]$. Тъй като $\mathtt{x}$ не е свободна променлива на φ, то за всеки елемнт μ на универсума на $\mathbf{M}$, твърдението $\mathbf{M} \vDash \varphi[v^{\mu}_{\mathtt{x}}]$ е вярно тогава и само тогава, когато е вярно твърдението A (което очевидно не зависи от μ).

Най-напред ще докажем, че $\mathbf{M} \vDash \varphi \vee \forall \mathtt{x}\, \psi \Rightarrow \forall \mathtt{x}\, (\varphi \vee \psi)[v]$. За целта да допуснем, че е вярно A или за всяко $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$. В първия случай очевидно за всяко $\mu \in |\mathbf{M}|$ ще бъде вярно твърдението „A или $B(\mu)$“. Във втория случай — също.

За да докажем обратната посока, да допуснем, че за всяко $\mu \in |\mathbf{M}|$ е вярно твърдението „A или $B(\mu)$“. От конструктивна гледна точка това означава, че ни е дадена функция, която по дадено μ ни казва дали е вярно A или $B(\mu)$. Ако решим „да пробваме“ тази функция за различни елементи на универсума и всеки път получаваме, че е вярно $B(\mu)$, това няма да означава, че $B(\mu)$ е вярно винаги — възможно е просто да не сме имали късмет и ако бяхме улучили „правилното“ μ, щяхме да установим, че е вярно A. Следователно няма как имайки такава функция да разберем дали е вярно A, или за всяко $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$. Това ни подсказва, че твърдението не е вярно конструктивно и значи трябва да допуснем противното.

И така, да допуснем, че не е вярно твърдението „A е вярно или за всяко $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$“. Това означава (вж. доказателството на твърдение 5.50 а)), че твърдението A не е вярно и твърдението „за всяко $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$“ също не е вярно. Но ние знаем, че за всяко $\mu \in |\mathbf{M}|$ е вярно „A или $B(\mu)$“. Тъй като току-що установихме, че A не е вярно, то значи за всяко $\mu \in |\mathbf{M}|$ е вярно $B(\mu)$. Това е противоречие. ∎

5.60. Твърдение. *За произволни формули φ и ψ, ако* $\mathtt{x}$ *не е свободна променлива на φ, то:*

а) $\models (\varphi \Rightarrow \forall \mathtt{x}\, \psi) \Leftrightarrow \forall \mathtt{x}\, (\varphi \Rightarrow \psi)$;

б) $\models (\varphi \Rightarrow \exists \mathtt{x}\, \psi) \Leftrightarrow \exists \mathtt{x}\, (\varphi \Rightarrow \psi)$; **(клас)**

в) $\models (\exists \mathtt{x}\, \psi \Rightarrow \varphi) \Leftrightarrow \forall \mathtt{x}\, (\psi \Rightarrow \varphi)$;

г) $\models (\forall \mathtt{x}\, \psi \Rightarrow \varphi) \Leftrightarrow \exists \mathtt{x}\, (\psi \Rightarrow \varphi)$. **(клас)**

5.61. Дефиниция. Казваме, че една формула е в *пренексна нормална форма*, ако тя има вида

$$\mathfrak{Q}_1 \mathtt{x}_1 \mathfrak{Q}_2 \mathtt{x}_2 \ldots \mathfrak{Q}_n \mathtt{x}_n (\varphi)$$

където $\mathfrak{Q}_1, \mathfrak{Q}_2, \ldots, \mathfrak{Q}_n$ са квантори ($\forall$ или $\exists$), а формулата φ не съдържа квантори.

5.62. Твърдение. *За всяка формула може да се намери еквивалентна на нея (според класическата логика) формула, която е в пренексна нормална форма.*

Доказателство. Нека ни е дадена произволна формула. Съгласно лема 5.17 може да намерим конгруентна, а значи и еквивалентна на нея формула, в която всички квантори имат различни променливи и никоя кванторна променлива не е свободна променлива във формулата. Да прилагаме в така получената формула докато може замени от следния вид:

$$\begin{gathered}
\varphi \,\&\, \exists \mathtt{x}\, \psi \longmapsto \exists \mathtt{x}\, (\varphi \,\&\, \psi) \\
\exists \mathtt{x}\, \psi \,\&\, \varphi \longmapsto \exists \mathtt{x}\, (\psi \,\&\, \varphi) \\
\varphi \,\&\, \forall \mathtt{x}\, \psi \longmapsto \forall \mathtt{x}\, (\varphi \,\&\, \psi) \\
\forall \mathtt{x}\, \psi \,\&\, \varphi \longmapsto \forall \mathtt{x}\, (\psi \,\&\, \varphi) \\
\varphi \vee \exists \mathtt{x}\, \psi \longmapsto \exists \mathtt{x}\, (\varphi \vee \psi) \\
\exists \mathtt{x}\, \psi \vee \varphi \longmapsto \exists \mathtt{x}\, (\psi \vee \varphi) \\
\varphi \vee \forall \mathtt{x}\, \psi \longmapsto \forall \mathtt{x}\, (\varphi \vee \psi) \\
\forall \mathtt{x}\, \psi \vee \varphi \longmapsto \forall \mathtt{x}\, (\psi \vee \varphi) \\
\varphi \Rightarrow \forall \mathtt{x}\, \psi \longmapsto \forall \mathtt{x}\, (\varphi \Rightarrow \psi) \\
\varphi \Rightarrow \exists \mathtt{x}\, \psi \longmapsto \exists \mathtt{x}\, (\varphi \Rightarrow \psi)
\end{gathered}$$

$$\begin{gathered}
\exists \mathtt{x}\, \psi \Rightarrow \varphi \longmapsto \forall \mathtt{x}\, (\psi \Rightarrow \varphi) \\
\forall \mathtt{x}\, \psi \Rightarrow \varphi \longmapsto \exists \mathtt{x}\, (\psi \Rightarrow \varphi) \\
\neg \exists \mathtt{x}\, \varphi \longmapsto \forall \mathtt{x}\, \neg \varphi \\
\neg \forall \mathtt{x}\, \varphi \longmapsto \exists \mathtt{x}\, \neg \varphi
\end{gathered}$$

Ако имаме подформула от вида напр. $\varphi \,\&\, \exists \mathtt{x}\, \psi$, то във φ променливата $\mathtt{x}$ няма да бъде свободна, защото ако $\mathtt{x}$ бе свободна във φ, то тъй като работим с формула, в която няма два квантора с една и съща променлива, то променливата $\mathtt{x}$ би била свободна и в цялата формула, а пък си осигурихме да няма кванторна променлива, която да е свободна в цялата формула. Това означава, че съгласно твърдения 5.59 б), 5.59 а), 5.59 г), 5.59 в), 5.60 а), 5.60 б), 5.60 в), 5.60 г), 5.51 а) и 5.51 б) при всяка една от тези замени ще получаваме еквивалентни формули. Ако след краен брой стъпки стигнем формула, към която не може да прилагаме никоя от по-горните замени, то значи сме получили формула в пренексна нормална форма. Това ни дава частичната коректност на алгоритъма за привеждане в пренексна нормална форма. Остава да докажем, че алгоритъмът винаги спира.

(I начин) За произволна формула χ да наречем *дълбочина* на подформулата χ' на χ дължината на най-дългата редица от вида

$$\chi_1, \chi_2, \chi_3, \ldots, \chi_k$$

където $\chi_1 = \chi$, $\chi_k = \chi'$, $\chi_i \neq \chi_{i+1}$ и χ_{i+1} е подформула на χ_i. Ако си мислим формулата като дърво, тогава дълбочината на една подформула е равна на разтоянието между корена на поддървото и корена на цялото дърво.

Да забележим, че замените от горния вид не променят броя на подформулите, започващи с квантор. Всяка една такава замяна обаче намалява дълбочината на някоя от подформулите, започващи с квантор, а това не може да продължи до безкрайност.

(II начин) Да разгледаме следния начин, по който от формула получаваме аритметичен израз. Да заменим всяка атомарна подформула с числото 2. Да заменим всяка подформула от вида $\chi' \,\&\, \chi'$, $\chi' \vee \chi''$ и $\chi' \Rightarrow \chi''$ с $x + y$, където x и y са изразите, получени съответно от χ' и χ''. Да заменим всяка подформула от вида $\neg\chi$ с $1 + x$, където x е изразът, получен от χ. Да заменим всяка подформула от вида $\forall \mathtt{x}\, \chi$ и $\exists \mathtt{x}\, \chi$ с x^2, където x е изразът, получен от χ. Може да се забележи, че така полученият израз е естествено число, не по-малко от 2. Тъй като за всеки естествени числа, не по-малки от 2, е изпълнено $x + y^2 < (x + y)^2$, $x^2 + y < (x + y)^2$ и $1 + x^2 < (1 + x)^2$, то след всяко преобразование на формулата по описание по-горе начин, стойността на съответния аритметичен израз ще се увеличи. Това обаче не може да продължи до безкрай, защото броят на символите в тези аритметични изрази се

запазва след всяко такова преобразование и значи може да се получат само краен брой аритметични изрази. ■

5.63. ***Забележка:*** Въпреки, че правилата за замяна в алгоритъма за привеждане в пренексна нормална форма са много на брой, те се помнят лесно. Нека да забележим, че когато кванторът излиза пред отрицание или се е намирал отляво на импликация, той се променя, т.е. от $\forall$ става $\exists$ и от $\exists$ става $\forall$. Във всички останали случаи кванторът просто се премества без промяна.

5.64. Твърдение. *За всяка формула може да се намери еквивалентна на нея (според класическата логика) формула, която е едновременно в пренексна нормална форма и в отрицателна нормална форма.*

Доказателство. Непосредствено се проверява, че ако преобразуваме според алгоритъма за привеждане в пренексна нормална форма формула, която се намира в отрицателна нормална форма, то ще получаваме формули в отрицателна нормална форма. Това означава, че можем просто да приведем формулата в отрицателна нормална форма и след това да приложим алгоритъмът за привеждане в пренексна нормална форма.

Също така можем непосредствено да проверим, че ако преобразуваме според алгоритъма за привеждане в отрицателна нормална форма формула, която се намира в пренексна нормална форма, то ще получаваме формули в пренексна нормална форма. Това означава, че можем просто да приведем формулата в пренексна нормална форма и след това да приложим алгоритъмът за привеждане в отрицателна нормална форма.

Също така, напълно допустимо е да прилагаме разбъркано замените от двата алгоритъма. В този случай процесът също със сигурност ще бъде краен и ще даде желания резултат, но тук няма да доказваме това. ■

5.65. ***Забележка:*** Ако формулата вече е приведена в отрицателна нормална форма и приложим към нея алгоритъма за привеждане в пренексна нормална форма, няма да ни се наложи да прилагаме нито веднъж замени, при които кванторът се променя, т.е. от $\forall$ става $\exists$ или от $\exists$ става $\forall$. Това означава, че не е нужно да работим стъпка по стъпка, а можем просто да преместим наведнъж всички квантори отпред на формулата (без да променяме реда им!) и ще получим пренексна нормална форма.

5.66. Твърдение. *Формула от вида* $\forall \mathtt{x}\,(\varphi)$ *е тъждествено вярна в някоя структура* $\mathbf{M}$ *тогава и само тогава, когато в* $\mathbf{M}$ *е тъждествено вярна формулата* φ*.*

Доказателство. Да допуснем, че $\mathbf{M} \vDash \forall \mathtt{x}\,(\varphi)$, т.е. за всяка оценка v в $\mathbf{M}$ е вярно $\mathbf{M} \vDash \forall \mathtt{x}\,(\varphi)[v]$. По дефиниция това означава, че за всяка оценка v в $\mathbf{M}$ и за всеки елемент μ на универсума на $\mathbf{M}$ е вярно $\mathbf{M} \vDash \varphi[v^{\mu}_{\mathtt{x}}]$. В частност, ако изберем $\mu = v(\mathtt{x})$, ще получим $v^{\mu}_{\mathtt{x}} = v$ и значи за всяка оценка v в $\mathbf{M}$ ще бъде вярно $\mathbf{M} \vDash \varphi[v]$. Следователно φ е тъждествено вярна в $\mathbf{M}$.

Обратната посока се вижда още по-лесно. Да допуснем, че φ е тъждествено вярна в $\mathbf{M}$. Това значи, че φ ще бъде вярна при произволна оценка. В частност φ ще бъде вярна и при всяка модифицирана оценка $v^{\mu}_{\mathtt{x}}$, и значи формулата $\forall \mathtt{x}\,\varphi$ е тъждествено вярна в $\mathbf{M}$. ∎

Прилагателното „скулемов“ от следващата дефиниция произлиза от името на откривателя на преобразованието, наречено *скулемизация* — норвежкия логик Тӯралф Скӯлем.

5.67. Дефиниция. а) Нека е дадена формула от вида $\exists \mathtt{x}\,(\varphi)$, в която няма свободни променливи, а $\mathtt{c}$ е символ за константа. Нека s е субституцията, която заменя $\mathtt{x}$ с $\mathtt{c}$ и оставя непроменени всички други променливи. Формулата $\hat{s}(\varphi)$ се нарича *скулемово усилване* (от първи вид) на $\exists \mathtt{x}\,(\varphi)$.

б) Нека е дадена формула от вида $\exists \mathtt{x}\,(\varphi)$, в която всички свободни променливи са измежду $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$, а $\mathtt{f}$ е n-местен функционален символ. Нека s е субституцията, която заменя $\mathtt{x}$ с терма $\mathtt{f}(\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n)$ и оставя непроменени всички други променливи. Формулата $\hat{s}(\varphi)$ се нарича *скулемово усилване* (от втори вид) на $\exists \mathtt{x}\,(\varphi)$.

в) Символът за константа $\mathtt{c}$ от а) и функционалният символ $\mathtt{f}$ от б) се наричат *скулемов символ за константа* и *скулемов функционален символ*.

5.68. Пример. Да разгледаме формулата $\varphi = \exists \mathtt{x}\,\mathtt{p}(\mathtt{x})$. Едно нейно скулемово усилване е формулата $\varphi' = \mathtt{p}(\mathtt{c})$. Във всяка структура $\mathbf{M}$ формулата φ „казва“, че в универсума на $\mathbf{M}$ има елемент, за който предикатът $\mathtt{p}^{\mathbf{M}}$ е истина. Формулата φ' пък казва, че предикатът $\mathtt{p}^{\mathbf{M}}$ е истина не за някой неопределен елемент, а за $\mathtt{c}^{\mathbf{M}}$. Следователно скулемовото усилване φ' казва повече, отколкото φ и ако формулата φ' е вярна в някоя структура, то и φ ще бъде вярна. По-долу твърдение 5.70 ще покаже, че това се случва винаги, а не само при този конкретен пример.

Ако имаме право сами да решим каква да бъде интерпретацията на символа $\mathtt{c}$ и формулата φ е вярна в $\mathbf{M}$, то ще можем да интерпретираме $\mathtt{c}$ така, че $\mathtt{c}^{\mathbf{M}}$ да бъде оня елемент от универсума, чието съществуване се твърди от формулата φ. Ако променим $\mathbf{M}$ по този начин, формулата φ' ще стана вярна. По-долу твърдение 5.71 ще покаже, че това се случва винаги при скулемово усилване от първи вид, а не само при този конкретен пример.

5.69. Пример. Да разгледаме формулата $\varphi = \exists \mathtt{y}\, \mathtt{p}(\mathtt{x}, \mathtt{y})$. Едно нейно скулемово усилване е формулата $\varphi' = \mathtt{p}(\mathtt{x}, \mathtt{f}(\mathtt{x}))$. Във всяка структура $\mathbf{M}$ формулата φ „казва“, че за всеки елемент μ на универсума на $\mathbf{M}$ съществува такъв елемент ν, че $\mathtt{p}^{\mathbf{M}}(\mu, \nu)$ е истина. Формулата φ' пък казва, че $\mathtt{p}^{\mathbf{M}}(\mu, \nu)$ е истина не за някое неопределено ν, а за $\nu = \mathtt{f}^{\mathbf{M}}(\mu)$. Следователно скулемовото усилване φ' казва повече, отколкото φ и ако формулата φ' е вярна в някоя структура, то и φ ще бъде вярна. По-долу твърдение 5.70 ще покаже, че това се случва винаги, а не само при този конкретен пример.

Ако имаме право сами да решим каква да бъде интерпретацията на символа $\mathtt{f}$ и формулата φ е вярна в $\mathbf{M}$, то ще можем да интерпретираме $\mathtt{f}$ така, че $\mathtt{f}^{\mathbf{M}}(\mu)$ да бъде оня елемент ν от универсума, чието съществуване се твърди от формулата φ. Ако променим $\mathbf{M}$ по този начин, формулата φ' ще стана вярна. По-долу твърдение 5.72 ще покаже, че това се случва винаги при скулемово усилване от втори вид, а не само при този конкретен пример.

Следващото твърдение показва, че усилването на една формула казва повече неща, отколкото самата формула.

5.70. Твърдение. *Ако скулемовото усилване на една формула е тъждествено вярно в структура* $\mathbf{M}$*, то и самата формула е тъждествено вярна в* $\mathbf{M}$*.*

Доказателство. Нека $\hat{s}(\varphi)$ е скулемово усилване на $\exists \mathtt{x}\,(\varphi)$ и $\mathbf{M}$ е структура, в която формулата $\hat{s}(\varphi)$ е тъждествено вярна. За да докажем, че $\exists \mathtt{x}\,(\varphi)$ е тъждествено вярна в $\mathbf{M}$, да изберем произволна оценка v в $\mathbf{M}$. Ще докажем, че формулата $\exists \mathtt{x}\,(\varphi)$ е вярна в $\mathbf{M}$ при оценка v.

Съгласно твърдение 5.45 съществува такава оценка w в $\mathbf{M}$, че формулата $\hat{s}(\varphi)$ е вярна при оценка v тогава и само тогава, когато формулата φ е вярна при оценка w. При това оценката w има следния вид: за всяка променлива $\mathtt{z}$ стойността на $w(\mathtt{z})$ е равна на стойността на терма $s(\mathtt{z})$ при оценка v. Съгласно дефиниция 5.67, $s(\mathtt{z}) = \mathtt{z}$ за всички

променливи, освен $\mathtt{x}$. Следователно

$$w(\mathtt{z}) = \begin{cases} \mu, & \text{ако } \mathtt{z} = \mathtt{x}, \text{ където } \mu \text{ е стойността на } s(\mathtt{x}) \text{ при оценка } v \\ v(\mathtt{z}), & \text{иначе} \end{cases}$$

Това равенство означава, че $w = v^{\mu}_{\mathtt{x}}$. Тъй като формулата $\hat{s}(\varphi)$ е вярна при оценка v, то формулата φ е вярна при оценка w, т.е. φ е вярна при оценка $v^{\mu}_{\mathtt{x}}$. Последното означава, че формулата $\exists \mathtt{x}\,(\varphi)$ е вярна при оценка v. ■

5.71. Твърдение. *Нека $\hat{s}(\varphi)$ е скулемово усилване от първи вид на $\exists \mathtt{x}\,(\varphi)$ и скулемовият символ за константа не се среща никъде във формулата φ. Ако формулата $\exists \mathtt{x}\,(\varphi)$ е тъждествено вярна в структура $\mathbf{M}$, то съществува структура $\mathbf{K}$, която съвпада с $\mathbf{M}$ във всичко, освен може би при интерпретацията на скулемовия символ, и в която е тъждествено вярна формулата $\hat{s}(\varphi)$.*

Доказателство. Съгласно дефиниция 5.67 субституцията s има вида

$$s(\mathtt{z}) = \begin{cases} \mathtt{c}, & \text{ако } \mathtt{z} = \mathtt{x} \\ \mathtt{z}, & \text{иначе} \end{cases}$$

Освен това формулата $\exists \mathtt{x}\,(\varphi)$ няма свободни променливи и значи във φ няма други свободни променливи, освен може би $\mathtt{x}$.

Нека u е някоя оценка в $\mathbf{M}$. Щом формулата $\exists \mathtt{x}\,(\varphi)$ е тъждествено вярна в $\mathbf{M}$, то значи тя е вярна при оценка u. По дефиниция това означава, че съществува такъв елемент μ на универсума на $\mathbf{M}$, че φ е вярна при оценка $u^{\mu}_{\mathtt{x}}$. Нека структурата $\mathbf{K}$ е идентична с $\mathbf{M}$ във всичко, освен в интерпретацията на символа $\mathtt{c}$ — нека $\mathtt{c}^{\mathbf{K}} = \mu$.

За да докажем, че формулата $\hat{s}(\varphi)$ е тъждествено вярна в $\mathbf{K}$, да изберем произволна оценка v. Съгласно твърдение 5.45 съществува такава оценка w в $\mathbf{M}$, че стойността на коя да е формула и в частност на φ в $\mathbf{K}$ при оценка w е същата като стойността на $\hat{s}(\varphi)$ в $\mathbf{K}$ при оценка v. При това стойността на коя да е променлива $\mathtt{z}$ при оценка w е равна на стойността на терма $s(\mathtt{z})$ в $\mathbf{K}$ при оценка v. От вида на субституцията s следва, че

$$w(\mathtt{z}) = \begin{cases} \mu, & \text{ако } \mathtt{z} = \mathtt{x} \\ v(\mathtt{z}), & \text{иначе} \end{cases}$$

Това равенство означава, че $w = v^{\mu}_{\mathtt{x}}$.

Тъй като символът $\mathtt{c}$, който е единствената разлика между $\mathbf{M}$ и $\mathbf{K}$, не се среща във формулата φ, а φ е вярна в $\mathbf{M}$ при оценка $u^{\mu}_{\mathtt{x}}$, то φ е

вярна и в $\mathbf{K}$ при оценка $u^{\mu}_{\mathtt{x}}$. Тъй като единствената възможна свободна променлива във φ е $\mathtt{x}$, а за $\mathtt{x}$ оценките $u^{\mu}_{\mathtt{x}}$ и $v^{\mu}_{\mathtt{x}}$ съвпадат, то φ е вярна в $\mathbf{K}$ и при оценка $v^{\mu}_{\mathtt{x}}$. Тъй като $v^{\mu}_{\mathtt{x}} = w$, то φ е вярна при оценка w и значи $\hat{s}(\varphi)$ е вярна при оценка v. ∎

5.72. Твърдение. *Нека $\hat{s}(\varphi)$ е скулемово усилване от втори вид на $\exists\mathtt{x}\,(\varphi)$ и скулемовият функционален символ не се среща никъде във формулата φ. Ако формулата $\exists\mathtt{x}\,(\varphi)$ е тъждествено вярна в структура $\mathbf{M}$, то съществува структура $\mathbf{K}$, която съвпада с $\mathbf{M}$ във всичко, освен може би при интерпретацията на скулемовия символ, и в която е тъждествено вярна формулата $\hat{s}(\varphi)$.*

Доказателство. Съгласно дефиниция 5.67 субституцията s има вида

$$s(\mathtt{z}) = \begin{cases} \mathtt{f}(\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n), & \text{ако } \mathtt{z} = \mathtt{x} \\ \mathtt{z}, & \text{иначе} \end{cases}$$

Освен това всички свободни променливи във формулата $\exists\mathtt{x}\,(\varphi)$ са измежду $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ и значи всички свободни променливи във φ са измежду $\mathtt{x}, \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$.

Нека $\mu_1, \mu_2, \ldots, \mu_n$ са произволни елементи на универсума на $\mathbf{M}$. Нека u е някоя оценка в $\mathbf{M}$, за която $u(\mathtt{x}_i) = \mu_i$. Щом формулата $\exists\mathtt{x}\,(\varphi)$ е тъждествено вярна в $\mathbf{M}$, то значи тя е вярна при оценка u. По дефиниция това означава, че съществува такъв елемент μ на универсума на $\mathbf{M}$, че φ е вярна при оценка $u^{\mu}_{\mathtt{x}}$. Нека $f\colon |\mathbf{M}|^n \to |\mathbf{M}|$ е онази функция, която съпоставя на всеки така избрани $\mu_1, \mu_2 \ldots, \mu_n$ така намереното μ.*

Нека структурата $\mathbf{K}$ е идентична с $\mathbf{M}$ във всичко, освен в интерпретацията на символа $\mathtt{f}$ — нека $\mathtt{f}^{\mathbf{K}} = f$.

За да докажем, че формулата $\hat{s}(\varphi)$ е тъждествено вярна в $\mathbf{K}$, да изберем произволна оценка v. Съгласно твърдение 5.45 съществува такава оценка w в $\mathbf{M}$, че стойността на коя да е формула и в частност на φ в $\mathbf{K}$ при оценка w е същата като стойността на $\hat{s}(\varphi)$ в $\mathbf{K}$ при оценка v. При това стойността на коя да е променлива $\mathtt{z}$ при оценка w е равна на стойността на терма $s(\mathtt{z})$ в $\mathbf{K}$ при оценка v. Тъй като стойността на терма $\mathtt{f}(\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n)$ е $f(v(\mathtt{x}_1), v(\mathtt{x}_2), \ldots, v(\mathtt{x}_n))$, то от вида на субституцията s следва, че

$$w(\mathtt{z}) = \begin{cases} f(v(\mathtt{x}_1), v(\mathtt{x}_2), \ldots, v(\mathtt{x}_n)), & \text{ако } \mathtt{z} = \mathtt{x} \\ v(\mathtt{z}), & \text{иначе} \end{cases}$$

Това равенство означава, че $w = v^{\mu}_{\mathtt{x}}$, където $\mu = f(v(\mathtt{x}_1), v(\mathtt{x}_2), \ldots, v(\mathtt{x}_n))$.

*На това място използваме т.н. *аксиома за избора*.

От дефиницията на f следва, че съществува такава оценка u, че $u(\mathtt{x}_i) = v(\mathtt{x}_i)$, а формулата φ е вярна в $\mathbf{M}$ при оценка $u^\mu_{\mathtt{x}}$. Тъй като символът $\mathtt{f}$, който е единствената разлика между $\mathbf{M}$ и $\mathbf{K}$, не се среща във формулата φ, а φ е вярна в $\mathbf{M}$ при оценка $u^\mu_{\mathtt{x}}$, то φ е вярна и в $\mathbf{K}$ при оценка $u^\mu_{\mathtt{x}}$. Тъй като всички свободни променливи във φ са измежду $\mathtt{x}, \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ и за тях оценките $u^\mu_{\mathtt{x}}$ и $v^\mu_{\mathtt{x}}$ съвпадат, то φ е вярна в $\mathbf{K}$ и при оценка $v^\mu_{\mathtt{x}}$. Тъй като $v^\mu_{\mathtt{x}} = w$, то φ е вярна при оценка w и значи $\hat{s}(\varphi)$ е вярна при оценка v. ∎

Забележка: В доказателството на твърдение 5.72 използвахме т.н. *аксиома за избора*. Един начин да формулираме тази аксиома е следният:

> Ако за всяко $x \in X$ съществува $y \in Y$, за които е верен някакъв предикат $p(x, y)$, то тогава съществува такава функция f, че за всяко $x \in X$ е вярно $p(x, f(x))$.

Ако тук интерпретираме квантора „съществува“ конструктивно, тази аксиома е съвсем естествена, защото в този случай ще разполагаме с конкретен метод, посредством който по дадено x можем да получим нужното y. Когато интерпретираме квантора класически, не разполагаме с никакъв метод, посредством който по x можем да получим y и затова аксиомата за избора дълго време е била една от най-оспорваните аксиоми на теория на множествата. Всъщност методът на скулемизацията, може да се обоснове и без да се използва аксиомата за избора, но доказателството става значително по-усложнено.

5.73. Дефиниция. Множество от формули е *изпълнимо*, ако съществува структура, в която са тъждествено верни всички формули от множеството. Множество от формули е *неизпълнимо*, ако не е изпълнимо.

5.74. (скулемизация) Нека ни е дадено крайно множество от формули Γ в пренексна нормална форма и да разгледаме следния процес, който ще наречем *скулемизация*. Избираме произволна формула от Γ, която съдържа квантор. Ако формулата започва с квантор $\forall$, то отстраняваме този квантор. Съгласно твърдение 5.66 ще получим такова множество от формули Γ', че Γ е изпълнимо тогава и само тогава, когато Γ е изпълнимо. Ако пък избраната формула започва с квантор $\exists$, то да заменим тази формула с такова нейно скулемово усилване, че СКУЛЕМОВИЯТ ФУНКЦИОНАЛЕН СИМВОЛ ДА НЕ СЕ СРЕЩА В НИКОЯ ФОРМУЛА ОТ Γ.* Съгласно твърдение 5.70, ако Γ' е изпълнимо, то и

*Разбира се, може да направим това само ако в сигнатурата има достатъчно символи.

Γ ще е изпълнимо. Ще докажем и обратното — ако Γ е изпълнимо, то и Γ′ е изпълнимо. Да допуснем, че Γ е изпълнимо и нека **M** е някоя структура, в която са тъждествено верни формулите от Γ. Съгласно твърдение 5.71 или 5.72 съществува структура **K**, в която е вярно скулемовото усилване на формулата, започваща с ∃. Останалите формули са идентични в Γ и Γ′. Те обаче не съдържат скулемовия символ, а структурите **M** и **K** съвпадат за всички символи, освен евентуално за скулемовия. Тъй като тези формули са тъждествено верни в **M**, то те са тъждествено верни и в **K**.

Тъй като на всяка стъпка от описания процес изчезва по един квантор, то след краен брой стъпки ще получим множество, в което всички формули са безкванторни. При това, така намереното множество не се различава от първоначалното по отношение на своята изпълнимост — ако първоначалното множество е изпълнимо, то и новополученото ще е неизпълнимо, и ако новополученото е изпълнимо, то значи и първоначалното множество е било изпълнимо.

И така, доказахме следната теорема:

5.75. Теорема. *Нека* Γ *е крайно*[*] *множество от формули. Ако в сигнатурата има достатъчно символи, то ще можем да намерим такова крайно множество* Γ′ *от безкванторни формули, че* Γ *да бъде изпълнимо тогава и само тогава, когато е изпълнимо* Γ′.

Доказателство. Да приведем формулите от Γ в пренексен вид и след това да приложим скулемизация (вж. 5.74). ∎

5.76. Дефиниция. Казваме, че една формула е в *скулемова нормална форма*, ако:

- формулата е в пренексна нормална форма;
- формулата не съдържа нито един квантор ∃;
- формулата не съдържа свободни променливи.

5.77. Теорема за скулемизацията. *Нека* Γ *е крайно*[**] *множество от формули. Ако в сигнатурата има достатъчно символи, то ще можем да намерим такова крайно множество* Γ′ *от формули в скулемова нормална форма, че* Γ *да бъде изпълнимо тогава и само тогава, когато е изпълнимо* Γ′.

[*] Теоремата е вярна и за безкрайно Γ, но тук няма да обосноваваме това.

[**] Теоремата е вярна и за безкрайно Γ, но тук няма да обосноваваме това.

Доказателство. Най-напред, използвайки теорема 5.75 можем да получим крайно множество Δ от безкванторни формули, което е изпълнимо тогава и само тогава, когато е изпълнимо Γ. Ако пред всяка от безкванторните формули в Δ сложим достатъчно квантори за всеобщност, ще получим множество Γ' в слулемова нормална форма. Освен това, съгласно твърдение 5.66, формулите от множеството Γ' са тъждествено верни точно в онези структури, в които са тъждествено верни и формулите от Δ. Следователно Δ е изпълнимо тогава и само тогава, когато е изпълнимо Γ'. ∎

5.78. Твърдение. *За всеки три формули φ, ψ и χ:*

а) $\vDash \varphi \vee (\psi \,\&\, \chi) \Leftrightarrow (\varphi \vee \psi) \,\&\, (\varphi \vee \chi)$

б) $\vDash (\psi \,\&\, \chi) \vee \varphi \Leftrightarrow (\psi \vee \varphi) \,\&\, (\chi \vee \varphi)$

Доказателство. Да изберем произволна структура $\mathbf{M}$ и оценка v в $\mathbf{M}$. Нека A е твърдението $\mathbf{M} \vDash \varphi[v]$, B е твърдението $\mathbf{M} \vDash \psi[v]$ и C е твърдението $\mathbf{M} \vDash \chi[v]$. Трябва да докажем, че са верни твърденията

> „Твърдението (A или (B и C)) е вярно тогава и само тогава, когато е вярно ((A или B) и (A или C))“

и

> „Твърдението ((B и C) или A) е вярно тогава и само тогава, когато е вярно ((B или A) и (C или A))“

И двете твърдения са очевидни. ∎

5.79. Дефиниция. а) *Литерал* означава атомарна формула или отрицание на атомарна формула.

б) *Елементарна дизюнкция* означава безкванторна формула, в която единствените логически операции са дизюнкция и отрицание и всяко отрицание се намира пред атомарна формула.

в) Една безкванторна формула е в *конюнктивна нормална форма*, ако представлява конюнкция от елементарни конюнкции.*

5.80. Твърдение. *За всяка безкванторна формула можем да намерим еквивалентна на нея формула в конюнктивна нормална форма.*

*По точно, можем да дадем следната индуктивна дефиниция на това какво значи *конюнктивна нормална форма*:

а) всяка елементарна дизюнкция е в конюнктивна нормална форма;

б) конюнкция на две формули в конюнктивна нормална форма е формула в конюнктивна нормална форма.

Доказателство. Нека φ е произволна безкванторна формула. Да я приведем в отрицателна нормална форма.* По този начин ще получим безкванторна формула, в която единствените логически операции са конюнкция, дизюнкция и отрицания и всяко отрицание се намира пред атомарна формула. Да прилагаме към така получената формула докато може замени от следния вид:

$$\varphi \vee (\psi \,\&\, \chi) \longmapsto (\varphi \vee \psi) \,\&\, (\varphi \vee \chi) \tag{51}$$
$$(\psi \,\&\, \chi) \vee \varphi \longmapsto (\psi \vee \varphi) \,\&\, (\chi \vee \varphi) \tag{52}$$

Съгласно твърдение 5.78 при замени от този вид ще получаваме еквивалентни формули. Освен това след всяка такава замяна формулата ще остава безкванторна, няма да се появят други операции, освен конюнкция, дизюнкция и отрицание и всяко отрицание ще си остане пред атомарната си формула. Може да забележим обаче, че ако след краен брой стъпки стигнем до формула, към която не можем да приложим замяна от горния вид, това ще означава, че сме получили формула в конюнктивна нормална форма. Това показва частичната коректност на алгоритъма за привеждане в конюнктивна нормална форма. Остава да видим защо алгоритъмът винаги свършва след краен брой стъпки.

(I начин) За произволна формула χ да наречем *конюнктивна височина* на χ дължината на най-дългата редица от вида

$$\chi_1, \chi_2, \chi_3, \ldots, \chi_k$$

където χ_1 е подформула на χ, χ_{i+1} е подформула на χ_i, $\chi_i \neq \chi_{i+1}$ и формулите χ_i са от вида $\varphi' \,\&\, \varphi''$. Ако си мислим формулата като дърво, тогава конюнктивната ѝ височина е равна на броя на конюнкциите в клона, съдържащ най-много конюнкции.

За всяка формула φ ще дефинираме редица $(a_n)_{n=1}^{\infty}$, която ще наречем *редица на* φ, по следния начин: нека a_i е равно на броя на под-

*Да припомним, че за целта е достатъчно да прилагаме докато може замени от следния вид:

$$\varphi \Leftrightarrow \psi \longmapsto (\varphi \Rightarrow \psi) \,\&\, (\psi \Rightarrow \varphi)$$
$$\varphi \Rightarrow \psi \longmapsto \neg\varphi \vee \psi$$
$$\neg\neg\varphi \longmapsto \varphi$$
$$\neg(\varphi \vee \psi) \longmapsto \neg\varphi \,\&\, \neg\psi$$
$$\neg(\psi \,\&\, \varphi) \longmapsto \neg\psi \vee \neg\varphi$$

формулите на φ, които имат вида $\varphi' \vee \varphi''$ и са с конюнктивна височина точно i.

Може да се забележи, че ако формулата ψ се получава от φ посредством някоя от замените (51) и (52), тогава редицата на ψ може да се получи от редицата на φ посредством конверсия. Съгласно лемата за фундираност на конверсията, след краен брой стъпки или ще стигнем формула, към която не можем да приложим никоя от тези замени.

(II начин) Да разгледаме следния начин, по който от безкванторна формула в отрицателна нормална форма получаваме аритметичен израз. Да заменим всеки литерал с числото 2. Да заменим всяка подформула от вида $\chi' \,\&\, \chi'$ с $x + y$, където x и y са изразите, получени съответно от χ' и χ''. Да заменим всяка подформула от вида $\chi' \vee \chi'$ с $x.y$, където x и y са изразите, получени съответно от χ' и χ''. След всяко преобразование на формулата от горния вид, стойността на съответния аритметичен израз няма да се промени, защото $x.(y + z) = x.y + x.z$ и $(y + z).x = y.x + z.x$. Същевременно след всяко такова преобразование се увеличава броят на числата 2 в така получения аритметичен израз. Това не може да продължи до безкрайност, защото за числа не по-малки от 2 е вярно $x + y \leqq x.y$ и значи стойността на всеки такъв аритметичен израз със сигурност е не по-малка от удвоения брой на числата 2 в него. ∎

5.7. Влагания

В раздел 3.7 споменахме, че в почти всеки математически дял важна роля играят различни видове морфизми. В същия раздел дефинирахме понятието остойностяващ морфизъм. Остойностяващите морфизми изобразяват синтактични обекти (термове и атомарни формули) в семантични обекти (съотв. елементи на универсума на структура и твърдения). В раздел 4.3 пък дефинирахме заместстващите морфизми. Те са изображения, които преобразуват синтактични обекти в синтактични обекти (термове в термове и атомарни формули в атомарни формули). В този раздел ще дефинираме понятието „влагане“. Влаганията представляват морфизми, които изобразяват елементите на универсума на една структура в елементи на друга структура.

5.81. Дефиниция. Изображението h е *влагане*[*] на структурата $\mathbf{M}$ в структурата $\mathbf{K}$, ако h изобразява всеки елемент на универсума на $\mathbf{M}$ в

[*] На английски: embedding.

елемент на универсума на $\mathbf{K}$ и освен това:

а) за всеки символ за константа $\mathtt{c}$, $h(\mathtt{c}^{\mathbf{M}}) = \mathtt{c}^{\mathbf{K}}$;

б) за всеки n-местен функционален символ $\mathtt{f}$ и елементи $\mu_1, \mu_2, \ldots, \mu_n$ на универсума на $\mathbf{M}$

$$h(\mathtt{f}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)) = \mathtt{f}^{\mathbf{K}}(h(\mu_1), h(\mu_2), \ldots, h(\mu_n))$$

в) за всеки n-местен предикатен символ $\mathtt{p}$ и елементи $\mu_1, \mu_2, \ldots, \mu_n$ на универсума на $\mathbf{M}$

$$\mathtt{p}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n) \longleftrightarrow \mathtt{p}^{\mathbf{K}}(h(\mu_1), h(\mu_2), \ldots, h(\mu_n))$$

Когато h е влагане от $\mathbf{M}$ в $\mathbf{K}$, записваме това така:

$$h\colon \mathbf{M} \to \mathbf{K}$$

Забележка: Влаганията не са единственият вид морфизми, които могат да се дефинират между структури. *Хомоморфизмите* се дефинират по същия начин, както влаганията, но пункт 5.81 в) е отслабен по следния начин: искаме единствено от верността на $\mathtt{p}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)$ да следва верността на $\mathtt{p}^{\mathbf{K}}(h(\mu_1), h(\mu_2), \ldots, h(\mu_n))$. Хомоморфизмите, които се дефинират в алгебрата, са частен случай на това понятие за хомоморфизъм. Понятието хомоморфизъм между графи също е негов частен случай.*

Друг важен вид морфизми са т.н. *елементарни влагания*. Пункт 5.81 в) от дефиницията на влагане изисква влаганията да запазват верността на атомарните формули. При елементарните влагания обаче се иска те да запазват верността не само на атомарните формули, но и на произволни предикатни формули.

Освен хомоморфизми, влагания и елементарни влагания, друг вид морфизми между структури са т.н. *силни хомоморфизми*.

От всички тези различни видове морфизми между структури, в нашия курс ще използваме само влаганията.

5.82. Дефиниция. а) Ако $h\colon \mathbf{M} \to \mathbf{K}$ е влагане, което е биекция между универсумите на $\mathbf{M}$ и $\mathbf{K}$, казваме, че h е *изоморфизъм*.

б) Когато $h\colon \mathbf{M} \to \mathbf{M}$ е влагане на една структура в същата структура, което е изоморфизъм, казваме, че h е *автоморфизъм*.

*Да си припомним примери 3.11 и 3.12, в които се вижда, че както структурите, използвани в алгебрата, така и графите са примери за структури по смисъла на този курс.

в) Когато съществува изоморфизъм между структурите $\mathbf{M}$ и $\mathbf{K}$, казваме, че $\mathbf{M}$ е *изоморфна* на $\mathbf{K}$.

Забележка: Въпреки, че има различни понятия за морфизъм между структури, оказва се, че понятието „изоморфизъм“ е абсолютно. Едно изображение е биективно влагане тогава и само тогава, когато е биективен хомоморфизъм, тогава и само тогава, когато е биективен силен хомоморфизъм и тогава и само тогава, когато е биективно елементарно влагане. В този курс няма да доказваме това, тъй като влаганията ще бъдат единствените морфизми между структури, които ще използваме.

5.83. Твърдение. *а) Всяка структура е изоморфна сама себе си.*

б) Ако структурата $\mathbf{M}$ е изоморфна с $\mathbf{K}$, то $\mathbf{K}$ е изоморфна с $\mathbf{M}$.

в) Ако структурата $\mathbf{M}$ е изоморфна с $\mathbf{N}$ и $\mathbf{N}$ е изоморфна с $\mathbf{K}$, то $\mathbf{M}$ е изоморфна с $\mathbf{K}$.

Доказателство. Вж. твърдения 5.84, 5.85 и 5.86. ∎

5.84. Твърдение. *Нека $\mathbf{M}$ е структура, а h е изображението, което изобразява всеки елемент на универсума на $\mathbf{M}$ в същия елемент. Тогава h е изоморфизъм.*

Доказателство. Очевидно h е биекция. Освен това непосредствено може да се провери, че h удовлетворява изискванията на дефиницията за влагане. ∎

5.85. Твърдение. *Ако $h\colon \mathbf{M} \to \mathbf{K}$ е изоморфизъм, то обратното изображение $h^{-1}\colon \mathbf{K} \to \mathbf{M}$ също е изоморфизъм.*

Доказателство. Да бъде h е биекция означава за произволни μ от универсума на $\mathbf{M}$ и κ от универсума на $\mathbf{K}$ да е вярно $h^{-1}(h(\mu)) = \mu$ и $h(h^{-1}(\kappa)) = \kappa$. Тъй като тези условия са симетрични спрямо μ и κ, то значи функцията h е обратна на h^{-1}, откъдето следва, че h^{-1} също е биекция.

Остава да докажем, че h^{-1} е влагане. Това е така, защото

– за всеки символ за константа $\mathtt{c}$, $h^{-1}(\mathtt{c}^{\mathbf{K}}) = h^{-1}(h(\mathtt{c}^{\mathbf{M}})) = \mathtt{c}^{\mathbf{M}}$.

– за всеки n-местен функционален символ $\mathtt{f}$

$$\begin{aligned} h^{-1}(\mathtt{f}^{\mathbf{K}}(\kappa_1, \kappa_2, \ldots, \kappa_n)) = \\ &= h^{-1}(\mathtt{f}^{\mathbf{K}}(h(h^{-1}(\kappa_1)), h(h^{-1}(\kappa_2)), \ldots, h(h^{-1}(\kappa_n)))) \\ &= h^{-1}(h(\mathtt{f}^{\mathbf{K}}(h^{-1}(\kappa_1), h^{-1}(\kappa_2), \ldots, h^{-1}(\kappa_n)))) \\ &= \mathtt{f}^{\mathbf{K}}(h^{-1}(\kappa_1), h^{-1}(\kappa_2), \ldots, h^{-1}(\kappa_n)) \end{aligned}$$

– за всеки n-местен предикатен символ $\mathtt{p}$

$$\begin{aligned}\mathtt{p}^{\mathbf{K}}(\kappa_1, \kappa_2, \ldots, \kappa_n) &\longleftrightarrow \\ &\longleftrightarrow \mathtt{p}^{\mathbf{K}}(h(h^{-1}(\kappa_1)), h(h^{-1}(\kappa_2)), \ldots, h(h^{-1}(\kappa_n))) \\ &\longleftrightarrow \mathtt{p}^{\mathbf{K}}(h^{-1}(\kappa_1), h^{-1}(\kappa_2), \ldots, h^{-1}(\kappa_n))\end{aligned}$$

■

В раздел 4.3 показахме, че композиия на два заместващи морфизма е заместващ морфизъм и композиция на заместващ морфизъм с остойностяващ морфизъм е остойностяващ морфизъм. Тук ще покажем, че композиция на влагане с влагане е влагане и композиция на остойностяващ морфизъм с влагане е остойностяващ морфизъм.

5.86. Твърдение. *Нека $h_2\colon \mathbf{M} \to \mathbf{N}$ и $h_1\colon \mathbf{N} \to \mathbf{K}$ са влагания и h е композицията на h_1 и h_2, т.е. $h(\mu) = h_1(h_2(\mu))$ за всеки елемент μ на универсума на $\mathbf{M}$. Тогава $h\colon \mathbf{M} \to \mathbf{K}$ е влагане.*

Доказателство. Очевидно h изобразява елементите на $|\mathbf{M}|$ в елементи на $|\mathbf{K}|$. Освен това:

– За всеки символ за константа $h(\mathtt{c}^{\mathbf{M}}) = h_1(h_2(\mathtt{c}^{\mathbf{M}})) = h_1(\mathtt{c}^{\mathbf{N}}) = \mathtt{c}^{\mathbf{K}}$.

– За всеки n-местен функционален символ и елементи $\mu_1, \mu_2, \ldots, \mu_n$ на универсума на $\mathbf{M}$

$$h(\mathtt{f}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)) = h_1(h_2(\mathtt{f}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)))$$

Тъй като h_2 е влагане, то последното е равно на

$$h_1(\mathtt{f}^{\mathbf{N}}(h_2(\mu_1), h_2(\mu_2), \ldots, h_2(\mu_n)))$$

Тъй като h_1 също е влагане, това е равно на

$$\mathtt{f}^{\mathbf{K}}(h_1(h_2(\mu_1)), h_1(h_2(\mu_2)), \ldots, h_1(h_2(\mu_n))) = \mathtt{f}(h(\mu_1), h(\mu_2), \ldots, h(\mu_n))$$

– Нека $\mathtt{p}$ е n-местен предикатен символ и $\mu_1, \mu_2, \ldots, \mu_n$ са елементи на универсума на $\mathbf{M}$. Тъй като h_2 е влагане, то

$$\mathtt{p}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n) \longleftrightarrow \mathtt{p}^{\mathbf{N}}(h_2(\mu_1), h_2(\mu_2), \ldots, h_2(\mu_n))$$

Тъй като h_1 също е влагане, това е еквивалентно на

$$\begin{aligned}\mathtt{p}^{\mathbf{K}}(h_1(h_2(\mu_1)), &h_1(h_2(\mu_2)), \ldots, h_1(h_2(\mu_n))) \longleftrightarrow \\ &\longleftrightarrow \mathtt{f}^{\mathbf{K}}(h(\mu_1), h(\mu_2), \ldots, h(\mu_n))\end{aligned}$$

■

5.87. Твърдение. *Нека h_2 е остойностяващ морфизъм в структурата* $\mathbf{M}$*, $h_1 \colon \mathbf{M} \to \mathbf{K}$ е влагане и h е изображението, за което $h(\tau) = h_2(h_1(\tau))$ за всеки терм τ и $h(\varphi) = h_2(\varphi)$ за всяка атомарна формула φ. Тогава h е остойностяващ морфизъм в* $\mathbf{K}$*. Изображението h се нарича композиция на h_1 и h_2.*

Доказателство. Очевидно h изобразява термовете в елементи на универсума на $\mathbf{K}$ и атомарните формули в твърдения. Освен това

– За всеки символ за константа $h(\mathtt{c}^{\mathbf{M}}) = h_1(h_2(\mathtt{c})) = h_1(\mathtt{c}^{\mathbf{M}}) = \mathtt{c}^{\mathbf{K}}$.

– За всеки n-местен функционален символ $\mathtt{f}$

$$h(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)) = h_1(h_2(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)))$$

Тъй като h_2 е остойностяващ морфизъм, то последното е равно на

$$h_1(\mathtt{f}^{\mathbf{M}}(h_2(\tau_1), h_2(\tau_2), \ldots, h_2(\tau_n)))$$

Тъй като h_1 влагане, това е равно на

$$\mathtt{f}^{\mathbf{K}}(h_1(h_2(\tau_1)), h_1(h_2(\tau_2)), \ldots, h_1(h_2(\tau_n))) = \mathtt{f}^{\mathbf{K}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

– За всеки n-местен функционален символ $\mathtt{p}$

$$h(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)) \longleftrightarrow h_2(\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n))$$

Тъй като h_2 е остойностяващ морфизъм, то последното е еквивалентно на

$$\mathtt{p}^{\mathbf{M}}(h_2(\tau_1), h_2(\tau_2), \ldots, h_2(\tau_n))$$

Тъй като h_1 е влагане, то това е еквивалентно на

$$\mathtt{p}^{\mathbf{K}}(h_1(h_2(\tau_1)), h_1(h_2(\tau_2)), \ldots, h_1(h_2(\tau_n))) \longleftrightarrow \\ \longleftrightarrow \mathtt{p}^{\mathbf{K}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

■

5.88. Дефиниция. Ако $h \colon \mathbf{M} \to \mathbf{K}$ е влагане, то за всяка оценка v в $\mathbf{M}$ с $h \circ v$ ще означаваме оценката в $\mathbf{K}$, за която за всяка променлива $\mathtt{z}$

$$(h \circ v)(\mathtt{z}) = h(v(\mathtt{z}))$$

5.89. Следствие. *Нека v е оценка в структурата* $\mathbf{M}$ *и $h \colon \mathbf{M} \to \mathbf{K}$ е влагане. Тогава*

а) h изобразява стойността на кой да е терм в $\mathbf{M}$ при оценка v в стойността на този терм в $\mathbf{K}$ при оценка $h \circ v$;

б) една атомарна формула е вярна в $\mathbf{M}$ при оценка v тогава и само тогава, когато тя е вярна в $\mathbf{K}$ при оценка $h \circ v$.

Доказателство. Нека $\bar{v}$ е изображението, което на всеки терм или атомарна формула τ съпоставя неговата или нейната стойност при оценка v. Съгласно твърдение 3.23 изображението $\bar{v}$ е остойностяващ морфизъм. Аналогично нека $w = h \circ v$ и $\bar{w}$ е изображението, което на всеки терм или атомарна формула τ съпоставя неговата или нейната стойност при оценка w. То също е остойностяващ морфизъм. Нека h' е композицията на $\bar{v}$ и h по смисъла на твърдение 5.87. И то е остойностяващ морфизъм.

Съгласно дефиницията на h', за произволна променлива $\mathtt{x}$ е вярно, че $h'(\mathtt{x}) = h(\bar{v}(\mathtt{x})) = h(v(\mathtt{x})) = w(\mathtt{x}) = \bar{w}(\mathtt{x})$. Щом морфизмите h' и $\bar{w}$ дават една и съща стойност за всички променливи, съгласно твърдение 3.24 те съвпадат. ∎

5.90. Твърдение. *Нека $h\colon \mathbf{M} \to \mathbf{K}$ е влагане. Тогава всяка клауза, която е тъждествено вярна в $\mathbf{K}$, е тъждествено вярна и в $\mathbf{M}$.*

Доказателство. Нека

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi \tag{53}$$

е произволна тъждествено вярна в $\mathbf{K}$ клауза. Нека v е такава оценка, че атомарните формули $\varphi_1, \varphi_2, \ldots, \varphi_n$ са верни в $\mathbf{M}$ при оценка v. Трябва да докажем, че ψ е вярна в $\mathbf{M}$ при оценка v.

Нека w е оценка в $\mathbf{K}$, дефинирана както в следствие 5.89. Щом $\varphi_1, \varphi_2, \ldots, \varphi_n$ са верни в $\mathbf{M}$ при оценка v, то те са верни и в $\mathbf{K}$ при оценка w. Щом клауза (53) тъждествено вярна, то значи атомарната формула ψ е вярна в $\mathbf{K}$ при оценка w. Следователно ψ е вярна в $\mathbf{M}$ при оценка v. ∎

5.91. Твърдение. *Нека $\mathbf{M}$ е произволна структура. Тогава съществуват ербранова структура $\mathbf{H}$ и влагане $h\colon \mathbf{H} \to \mathbf{M}$.*

Доказателство. Нека h е изображението, което на всеки терм без променливи съпоставя неговата стойност в $\mathbf{M}$ (тъй като термът е без променливи, съгласно твърдение 3.24 стойността му в $\mathbf{M}$ е една и съща при коя да е оценка). Нека $\mathbf{H}$ е ербрановата структура, за която за всеки n-местен предикатен символ $\mathtt{p}$ и термове без променливи $\tau_1, \tau_2, \ldots, \tau_n$

$$\mathtt{p}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n) \longleftrightarrow \mathtt{p}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

Тогава h е влагане от $\mathbf{H}$ в $\mathbf{M}$, защото

- за всеки символ за константа $\mathtt{c}$, $h(\mathtt{c}^{\mathbf{H}}) = h(\mathtt{c}) = \mathtt{c}^{\mathbf{M}}$;
- за всеки n-местен функционален символ f и термове $\tau_1, \tau_2, \ldots, \tau_n$ без променливи

  $$h(\mathtt{f}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n)) = h(\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n))$$

  Последното е равно на стойността на терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ в $\mathbf{M}$, т.е. на

  $$\mathtt{f}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

  където $h(\tau_i)$ са стойностите на τ_i в $\mathbf{M}$.
- за всеки n-местен предикатен символ f и термове $\tau_1, \tau_2, \ldots, \tau_n$ без променливи

  $$\mathtt{p}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n)) \longleftrightarrow \mathtt{p}^{\mathbf{M}}(h(\tau_1), h(\tau_2), \ldots, h(\tau_n))$$

  съгласно дефиницията на $\mathbf{H}$.

■

5.92. Твърдение. *Структурата* $\mathbf{H}$ *от предното твърдение е единствена.*

Доказателство. Нека $\mathbf{M}$ е структура, $\mathbf{H}'$ и $\mathbf{H}''$ са ербранови структури и $h' \colon \mathbf{H}' \to \mathbf{M}$ и $h'' \colon \mathbf{H}'' \to \mathbf{M}$ са влагания. Трябва да докажем, че структурите $\mathbf{H}'$ и $\mathbf{H}''$ са равни.

Тъй като те са ербранови, то универсумите им съвпадат, а символите за константи и функционалните символи се интерпретират идентично.

Освен това, за всеки n-местен предикатен символ $\mathtt{p}$, $\mathtt{p}^{\mathbf{H}'}(\tau_1, \tau_2, \ldots, \tau_n)$ е истина тогава и само тогава, когато атомарната формула $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е вярна в $\mathbf{H}'$ (все едно при каква оценка, защото в $\tau_1, \tau_2, \ldots, \tau_n$ няма променливи). Съгласно следствие 5.89 това е така тогава и само тогава, когато формулата $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е вярна в $\mathbf{M}$ (при коя да е оценка). Напълно аналогично можем да докажем, че и $\mathtt{p}^{\mathbf{H}''}(\tau_1, \tau_2, \ldots, \tau_n)$ е истина тогава и само тогава, когато формулата $\mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$ е вярна в $\mathbf{M}$. ■

5.93. Едно от нещата, които се изследват в теория на моделите, е това какви класове от структури може да се аксиоматизират, използвайки само формули от определен вид. Вече доказахме две теоретикомоделни свойства на клаузите: всяко множество от клаузи има най-малък ербранов модел и твърдение 5.90. Ще използваме тези свойства, за да

докажем, че полетата не могат да се аксиоматизират, използвайки клаузи.

Нека сигнатурата съдържа три символа за константи: „0“, „1“ и „c“, два двуместни функционални символа „$+$“ и „$.$“ и два едноместен функционални символа „$-$“ и „$^{-1}$“. Да допуснем, че има множество Γ от клаузи, което аксиоматизира алгебричното понятие за поле.

Нека $\mathbf{R}_0$ е полето, модел на Γ, с универсум — множеството на реалните числа, в което символите „0“, „1“, „$+$“, „$.$“, „$-$“ и „$^{-1}$“ се интерпретират по обичайния начин и $\mathsf{c}^{\mathbf{R}_0} = 0$. Нека $\mathbf{R}_1$ се дефинира аналогично, но $\mathsf{c}^{\mathbf{R}_1} = 1$. Нека $\mathbf{H}_0$ и $\mathbf{H}_1$ са ербранови структури, за които съществуват влагания $h_0 \colon \mathbf{H}_0 \to \mathbf{R}_0$ и $h_1 \colon \mathbf{H}_1 \to \mathbf{R}_1$ (такива ербранови структури съществуват съгласно твърдение 5.91). Тъй като клаузата

$$\vdash \mathsf{c}.\mathsf{c}^{-1} = 1$$

не е вярна в $\mathbf{R}_0$, то съгласно твърдение 5.90 тя не е вярна и в $\mathbf{H}_0$. Тъй като клаузата

$$\vdash \mathsf{c} = 0$$

не е вярна в $\mathbf{R}_1$, то съгласно твърдение 5.90 тя не е вярна и в $\mathbf{H}_1$. От това следва, че двете атомарни формули $\mathsf{c} = 0$ и $\mathsf{c}.\mathsf{c}^{-1} = 1$ не са верни в най-малкия ербранов модел на Γ. В този най-малък ербранов модел са верни аксиомите за поле, стойността на c е различна от нулевия елемент на това поле, но въпреки това тя няма обратен елемент. Това е противоречие.

Да отбележим, че понятието поле се аксиоматизира много лесно, използвайки произволни формули. Например следният списък от без-

кванторни формули ще ни свърши работа:

$$\mathtt{x} + (\mathtt{y} + \mathtt{z}) = (\mathtt{x} + \mathtt{y}) + \mathtt{z}$$
$$\mathtt{x}.(\mathtt{y}.\mathtt{z}) = (\mathtt{x}.\mathtt{y}).\mathtt{z}$$
$$\mathtt{x} + \mathtt{y} = \mathtt{y} + \mathtt{x}$$
$$\mathtt{x}.\mathtt{y} = \mathtt{y}.\mathtt{x}$$
$$\mathtt{x}.(\mathtt{y} + \mathtt{z}) = \mathtt{x}.\mathtt{y} + \mathtt{x}.\mathtt{z}$$
$$\mathtt{x} + 0 = \mathtt{x}$$
$$\mathtt{x}.1 = \mathtt{x}$$
$$\neg(0 = 1)$$
$$\mathtt{x} + (-\mathtt{x}) = 0$$
$$\neg(\mathtt{x} = 0) \Rightarrow \mathtt{x}.\mathtt{x}^{-1} = 1$$
$$\mathtt{x} = \mathtt{x}$$
$$\mathtt{x} = \mathtt{y} \Rightarrow \mathtt{y} = \mathtt{x}$$
$$\mathtt{x} = \mathtt{y} \,\&\, \mathtt{y} = \mathtt{z} \Rightarrow \mathtt{x} = \mathtt{z}$$
$$\mathtt{x}_1 = \mathtt{x}_2 \,\&\, \mathtt{y}_1 = \mathtt{y}_2 \Rightarrow \mathtt{x}_1 + \mathtt{y}_1 = \mathtt{x}_2 + \mathtt{y}_2$$
$$\mathtt{x}_1 = \mathtt{x}_2 \,\&\, \mathtt{y}_1 = \mathtt{y}_2 \Rightarrow \mathtt{x}_1.\mathtt{y}_1 = \mathtt{x}_2.\mathtt{y}_2$$
$$\mathtt{x}_1 = \mathtt{x}_2 \Rightarrow -\mathtt{x}_1 = -\mathtt{x}_2$$
$$\mathtt{x}_1 = \mathtt{x}_2 \Rightarrow \mathtt{x}_1^{-1} = \mathtt{x}_2^{-1}$$

5.94. Твърдение. *Нека $h\colon \mathbf{M} \to \mathbf{K}$ е сюрективно влагане. Формулата φ е вярна в $\mathbf{M}$ при някоя оценка v в $\mathbf{M}$ тогава и само тогава, когато тя е вярна в $\mathbf{K}$ при оценка $h \circ v$.*

Доказателство. С индукция по построението на формулата φ.

Когато φ е атомарна, твърдението вече е доказано — вж. следствие 5.89.

Ако $\varphi = \psi_1 \,\&\, \psi_2$, то съгласно индукционното предположение $\mathbf{M} \vDash \psi_1[v]$ е еквивалентно на $\mathbf{K} \vDash \psi_1[h \circ v]$ и $\mathbf{M} \vDash \psi_2[v]$ е еквивалентно на $\mathbf{K} \vDash \psi_2[h \circ v]$. Следователно

$$\begin{aligned}
\mathbf{M} \vDash \varphi[v] &\longleftrightarrow \mathbf{M} \vDash (\psi_1 \,\&\, \psi_2)[v] \\
&\longleftrightarrow \mathbf{M} \vDash \psi_1[v] \text{ и } \mathbf{M} \vDash \psi_2[v] \\
&\longleftrightarrow \mathbf{K} \vDash \psi_1[h \circ v] \text{ и } \mathbf{K} \vDash \psi_2[h \circ v] \\
&\longleftrightarrow \mathbf{K} \vDash (\psi_1 \,\&\, \psi_2)[h \circ v] \\
&\longleftrightarrow \mathbf{K} \vDash \varphi[h \circ v]
\end{aligned}$$

Когато $\varphi = (\psi_1 \vee \psi_2)$ и $\varphi = (\psi_1 \Rightarrow \psi_2)$, се разсъждава аналогично.

Ако $\varphi = \neg\psi$, то съгласно индукционното предположение $\mathbf{M} \vDash \psi[v]$ е еквивалентно на $\mathbf{K} \vDash \psi[h \circ v]$. Следователно

$$\begin{aligned}\mathbf{M} \vDash \varphi[v] &\longleftrightarrow \mathbf{M} \vDash (\neg\psi)[v] \\ &\longleftrightarrow \text{не е вярно, че } \mathbf{M} \vDash \psi[v] \\ &\longleftrightarrow \text{не е вярно, че } \mathbf{K} \vDash \psi[h \circ v] \\ &\longleftrightarrow \mathbf{K} \vDash (\neg\psi)[h \circ v] \\ &\longleftrightarrow \mathbf{K} \vDash \varphi[h \circ v]\end{aligned}$$

Когато $\varphi = \forall \mathtt{x}\, \psi$

$$\begin{aligned}\mathbf{M} \vDash \varphi[v] &\longleftrightarrow \mathbf{M} \vDash (\forall \mathtt{x}\, \psi)[v] \\ &\longleftrightarrow \text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]\end{aligned}$$

Съгласно индукционното предположение това е еквивалентно на

$$\text{за всяко } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \psi[h \circ v^{\mu}_{\mathtt{x}}] \tag{54}$$

От друга страна

$$\begin{aligned}\mathbf{K} \vDash \varphi[h \circ v] &\longleftrightarrow \mathbf{K} \vDash (\forall \mathtt{x}\, \psi)[h \circ v] \\ &\longleftrightarrow \text{за всяко } \kappa \in |\mathbf{K}| \text{ е вярно, че } \mathbf{K} \vDash \psi[(h \circ v)^{\kappa}_{\mathtt{x}}]\end{aligned} \tag{55}$$

За всяко $\mu \in |\mathbf{M}|$ ако $\kappa = h(\mu)$, то оценките $h \circ v^{\mu}_{\mathtt{x}}$ и $(h \circ v)^{\kappa}_{\mathtt{x}}$ ще съвпадат. Следователно от (55) следва (54). Освен това за всяко $\kappa \in |\mathbf{K}|$ поради сюрективността на h съществува такова $\mu \in |\mathbf{M}|$, че $\kappa = h(\mu)$ и значи оценките $h \circ v^{\mu}_{\mathtt{x}}$ и $(h \circ v)^{\kappa}_{\mathtt{x}}$ ще съвпадат. Следователно от (54) следва (55).

Когато $\varphi = \exists \mathtt{x}\, \psi$

$$\begin{aligned}\mathbf{M} \vDash \varphi[v] &\longleftrightarrow \mathbf{M} \vDash (\exists \mathtt{x}\, \psi)[v] \\ &\longleftrightarrow \text{за някое } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \psi[v^{\mu}_{\mathtt{x}}]\end{aligned}$$

Съгласно индукционното предположение това е еквивалентно на

$$\text{за някое } \mu \in |\mathbf{M}| \text{ е вярно, че } \mathbf{M} \vDash \psi[h \circ v^{\mu}_{\mathtt{x}}] \tag{56}$$

От друга страна

$$\begin{aligned}\mathbf{K} \vDash \varphi[h \circ v] &\longleftrightarrow \mathbf{K} \vDash (\exists \mathtt{x}\, \psi)[h \circ v] \\ &\longleftrightarrow \text{за някое } \kappa \in |\mathbf{K}| \text{ е вярно, че } \mathbf{K} \vDash \psi[(h \circ v)^{\kappa}_{\mathtt{x}}]\end{aligned} \tag{57}$$

За всяко $\mu \in |\mathbf{M}|$ ако $\kappa = h(\mu)$, то оценките $h \circ v^{\mu}_{\mathtt{x}}$ и $(h \circ v)^{\kappa}_{\mathtt{x}}$ ще съвпадат. Следователно от (56) следва (57). Освен това за всяко $\kappa \in |\mathbf{K}|$ поради сюрективността на h съществува такова $\mu \in |\mathbf{M}|$, че $\kappa = h(\mu)$ и значи оценките $h \circ v^{\mu}_{\mathtt{x}}$ и $(h \circ v)^{\kappa}_{\mathtt{x}}$ ще съвпадат. Следователно от (57) следва (54). ∎

Забележка: В доказателството на твърдение 5.94 разгледайте по-внимателно случаите $\varphi = \forall \mathtt{x}\, \psi$ и $\varphi = \exists \mathtt{x}\, \psi$. Открийте разликата в доказателството на тези два случая.

В следващото следствие използваме означенията, въведени в 5.41.

5.95. Следствие. *Нека $\varphi[\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n]$ е формула и $h\colon \mathbf{M} \to \mathbf{K}$ е сюрективно влагане. Тогава за произволни елементи $\mu_1, \mu_2, \ldots, \mu_n$ на универсума на* $\mathbf{M}$

$$\mathbf{M} \vDash \varphi[\mu_1, \mu_2, \ldots, \mu_n] \longleftrightarrow \mathbf{K} \vDash \varphi[h(\mu_1), h(\mu_2), \ldots, h(\mu_n)]$$

Доказателство. Нека v е произволна оценка в $\mathbf{M}$, за която $v(\mathtt{x}_1) = \mu_1$, $v(\mathtt{x}_2) = \mu_2$, $\ldots$, $v(\mathtt{x}_n) = \mu_n$. Съгласно 5.41 това, което всъщност твърдим в това следствие, е

$$\mathbf{M} \vDash \varphi[v] \longleftrightarrow \mathbf{K} \vDash \varphi[h \circ v]$$

Твърдение 5.94 казва точно това. ∎

Да напомним, че съгласно 5.41 когато пишем $\varphi[\mathtt{x}]$ имаме предвид, че единствено $\mathtt{x}$ може да бъде свободна променлива на φ. С други думи $\mathtt{x}$ е единствената свободна променлива на φ или φ няма свободни променливи. Когато пишем $\varphi[\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n]$, това ще означава, че всички свободни променливи на φ са измежду променливите $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$.

5.96. Дефиниция. Нека $\mathbf{M}$ е структура.

а) Подмножество X на универсума на $\mathbf{M}$ е *определимо* посредством формулата $\varphi[\mathtt{x}]$, ако за всяко $\mu \in |\mathbf{M}|$

$$\mu \in X \longleftrightarrow \mathbf{M} \vDash \varphi[\mu]$$

б) Подмножество $X \subseteq |\mathbf{M}|^n$ е *определимо* посредством формулата $\varphi[\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n]$, ако за произволни елементи $\mu_1, \mu_2, \ldots, \mu_n$ на универсума на $\mathbf{M}$

$$(\mu_1, \mu_2, \ldots, \mu_n) \in X \longleftrightarrow \mathbf{M} \vDash \varphi[\mu_1, \mu_2, \ldots, \mu_n]$$

5.97. Теорема за определимите множества. *Нека $\mathbf{M}$ е структура и $h\colon \mathbf{M} \to \mathbf{M}$ е произволен автоморфизъм.*

а) Ако подмножество X на универсума на $\mathbf{M}$ е определимо, то за всеки елемент μ на универсума на $\mathbf{M}$

$$\mu \in X \longleftrightarrow h(\mu) \in X$$

б) Ако подмножество $X \subseteq |\mathbf{M}|^n$ *е определимо, то за произволни елементи* $\mu_1, \mu_2, \ldots, \mu_n$ *на универсума на* $\mathbf{M}$

$$(\mu_1, \mu_2, \ldots, \mu_n) \in X \longleftrightarrow (h(\mu_1), h(\mu_2), \ldots, h(\mu_n)) \in X$$

Доказателство. Нека множеството X е определимо посредством формулата φ. Тъй като всеки автоморфизъм е сюрективно влагане, съгласно следствие 5.95

$$\begin{aligned}(\mu_1, \mu_2, \ldots, \mu_n) \in X &\longleftrightarrow \mathbf{M} \vDash \varphi[\mu_1, \mu_2, \ldots, \mu_n] \\ &\longleftrightarrow \mathbf{M} \vDash \varphi[h(\mu_1), h(\mu_2), \ldots, h(\mu_n)] \\ &\longleftrightarrow (h(\mu_1), h(\mu_2), \ldots, h(\mu_n)) \in X\end{aligned}$$

■

Глава 6

Метод на резолюциите

6.1. Дефиниция и коректност на метода

Да припомним следната дефиниция:

6.1. Дефиниция. а) *Литерал* означава атомарна формула или отрицание на атомарна формула.

б) *Елементарна дизюнкция* означава безкванторна формула, в която единствените логически операции са дизюнкция и отрицание и всяко отрицание се намира пред атомарна формула.

в) Една безкванторна формула е в *конюнктивна нормална форма*, ако представлява конюнкция от елементарни конюнкции.[*]

6.2. Твърдение. *Ако в сигнатурата разполагаме с достатъчно неизползвани символи за константи и функционални символи,*[**] *то за всяко крайно множество*[***] *от формули можем да намерим такова*

[*] По точно, можем да дадем следната индуктивна дефиниция на това какво значи *конюнктивна нормална форма*:

а) всяка елементарна дизюнкция е в конюнктивна нормална форма;

б) конюнкция на две формули в конюнктивна нормална форма е формула в конюнктивна нормална форма.

[**] На практика това не е сериозно ограничение, защото при нужда винаги можем да заменим сигнатурата с нова сигнатура, в която има много нови символи.

[***] Това твърдение е вярно и за безкрайни множества, но тук няма да доказваме това.

крайно множество от елементарни дизюнкции, че (от гледна точка на класическата логика) първоначалното множество е изпълнимо тогава и само тогава, когато е изпълнимо множеството от елементарни дизюнкции.

Доказателство. Съгласно теорема 5.75 за всяко крайно множество от формули можем да намерим крайно множество от безкванторни формули, което е изпълнимо тогава и само тогава, когато е изпълнимо първоначалнорагато множество. Също така видяхме, че за всяка безкванторна формула можем да намерим еквивалентна на нея безкванторна формула в конюнктивна нормална форма.

Всяка формула от вида $\varphi \,\&\, \psi$ е тъждествено вярна в някоя структура $\mathbf{M}$ тогава и само тогава, когато формулата $\varphi \,\&\, \psi$ е вярна при всяка оценка в $\mathbf{M}$, а това е така тогава и само тогава, когато при всяка оценка в $\mathbf{M}$ са верни двете формули φ и ψ, и значи тогава и само тогава, когато тези две формули са тъждествено верни в $\mathbf{M}$. Това означава, че ако в така полученото множество заменяме докато може всяка формула от вида $\varphi \,\&\, \psi$ с двете формули φ и ψ, ще получаваме формули, които са изпълними тогава и само тогава, когато е изпълнимо първоначалното множеството от формули. Тъй като след всяка такава замяна броят на конюнкциите намалява, то след краен брой стъпки ще стигнем до множество от елементарни конюнкции. ∎

Твърдение 6.2 показва, че когато се интересуваме от изпълнимостта на множество от формули, може да считаме, че елементите на това множество имат сравнително прост вид — елементарни дизюнкции. Методът на резолюциите е метод, посредством който можем да полурешаваме въпроса дали множество от елементарни дизюнкции е неизпълнимо. За да ни бъде по-удобно да разработим теорията на резолюциите, вместо с елементарни дизюнкции, ще работим с по-просто устроени обекти, наречени дизюнкти.

6.3. Дефиниция. а) *Дизюнкт* е крайно (може и празно) множество от литерали.

б) Един дизюнкт е верен в структура $\mathbf{M}$ при оценка v в $\mathbf{M}$, ако не може да няма литерал от дизюнкта, който е верен в $\mathbf{M}$ при оценка v.

в) Един дизюнкт е (тъждествено) верен в структура $\mathbf{M}$, ако е верен при всяка оценка в $\mathbf{M}$.

г) Множество от дизюнкти е *изпълнимо*, ако съществува структура, в която са тъждествено верни всички дизюнкти от множеството. Множество от дизюнкти е *неизпълнимо*, ако не е изпълнимо.

6.4. ***Забележка*:** а) Разбира се, когато разсъждаваме класически, изразът „не може да няма литерал от дизюнкта, който е верен в $\mathbf{M}$ при оценка v“ означава просто „има литерал от дизюнкта, който е верен в $\mathbf{M}$ при оценка v. Тъй като методът на резолюциите (във вида, който е описан тук) се използва за класическата, а не за интуиционистката логика, то спокойно можем да използваме и по-кратката формулировка „има литерал“. Все пак тук ще използваме по-дългата формулировка „не може да няма“, за да се подсещаме на кои места в доказателствата ще ни се наложи да използваме метода „допускане на противното“.

б) В литературата на английски език обектите, които тук наричаме „дизюнкти“, най-често се наричат клаузи (clauses). Терминът „клауза“ обаче се използва и за други видове обекти, докато „дизюнкт“ означава винаги това, което е казано в дефиниция 6.3 а).

6.5. Следствие. *Ако в сигнатурата разполагаме с достатъчно неизползвани символи за константи и функционални символи, то за всяко крайно множество*[*] *от формули можем да намерим такова крайно множество от дизюнкти, че (от гледна точка на класическата логика) първоначалното множество е изпълнимо тогава и само тогава, когато е изпълнимо множеството от дизюнкти.*

Доказателство. Съгласно твърдение 6.2 можем да намерим крайно множество от елементарни дизюнкции което е изпълнимо тогава и само тогава, когато е изпълнимо първоначалното множество. Да заменим всяка елементарна дизюнкция от така полученото множество с дизюнкта, съдържащ литералите в тази елементарна дизюнкция. Очевидно е, (при използване на класическата логика) че всяка така заменена елементарна дизюнкция е вярна в някоя структура $\mathbf{M}$ при оценка v тогава и само тогава, когато в $\mathbf{M}$ при оценка v е верен дизюнкта, с който заменихме елементарната дизюнкция. ■

6.6. Дефиниция. Празното множество от литерали се нарича *празен дизюнкт* и най-често се означава с $\square$, ■ или $\{\}$.

Следващото твърдение показва, че можем да си мислим, че празният дизюнкт е твърдение, което е винаги невярно.

6.7. Твърдение. *Празният дизюнкт не е тъждествено верен в никоя структура.*

[*]Твърдението е вярно и за безкрайни множества, но тук няма да доказваме това.

Доказателство. Празният дизюнкт не съдържа литерали и значи не съдържа литерали, които да са верни в структура все едно при каква оценка. ■

6.8. Дефиниция. а) Нека s е субституция, а δ — дизюнкт. Дизюнктът, който се получава като заменим всеки литерал φ от δ с $\hat{s}(\varphi)$, се нарича *резултат от прилагането* на субституцията s към δ и ще означаваме с $\hat{s}(\delta)$.

б) Всеки дизюнкт от вида $\hat{s}(\delta)$, където s е произволна субституция, а δ е дизюнкт, се нарича *частен случай* на дизюнкта δ.

Дефиницията на частен случай на дизюнкт и следващото твърдение са аналогични на дефиниция 4.5 и твърдение 4.24, които казват същото нещо, но за клаузи.

6.9. Твърдение. *Частните случаи на дизюнкт, който е тъждествено верен в структура* $\mathbf{M}$*, също са тъждествено верни в* $\mathbf{M}$*.*

Доказателство. Нека дизюнктът δ е тъждествено верен в структура $\mathbf{M}$, а $\hat{s}(\delta)$ е някой негов частен случай. Нека v е произволна оценка в $\mathbf{M}$. Трябва да докажем, че частният случай $\hat{s}(\delta)$ е верен в $\mathbf{M}$ при оценка v. Съгласно твърдение 5.45 съществува такава оценка w, че всеки литерал φ от δ е верен в $\mathbf{M}$ при оценка w тогава и само тогава, когато литералът $\hat{s}(\varphi)$ е верен в $\mathbf{M}$ при оценка v. Тъй като дизюнктът δ е тъждествено верен в $\mathbf{M}$, то не може да няма литерал φ от δ, който е верен в $\mathbf{M}$ при оценка w, а значи не може да няма литерал φ от δ, такъв че $\hat{s}(\varphi)$ е верен в $\mathbf{M}$ при оценка v. Остава да забалежим, че съгласно дефиниция 6.8 а), дизюнктът $\hat{s}(\delta)$ съдържа точно литералите $\hat{s}(\varphi)$, където φ е литерал от δ. ■

6.10. Дефиниция. а) За всяка атомарна формула φ нека $\overline{\varphi} = \neg\varphi$ и $\overline{\neg\varphi} = \varphi$. Очевидно $\overline{\overline{\psi}} = \psi$ за всеки литерал ψ.

б) Ако два дизюнкта имат частни случаи от вида $\{\varphi\} \cup \delta'$ и $\{\overline{\varphi}\} \cup \delta''$, то дизюнктът $\delta' \cup \delta''$ се нарича тяхна *либерална резолвента*.

Следващото твърдение показва, че ако два дизюнкта са „верни“, то и тяхната резолвента е „вярна“. Същността на метода на резолюциите се състои в това, че ако целта ни е да стигнем до противоречие, то може да ограничим разсъжденията си единствено до правене на резолвенти. Съгласно твърдение 6.7 да стигнем до противоречие посредством резолюции означава да получим като следствие празният дизюнкт.

6.11. Твърдение. *Либералната резолвента на дизюнкти, които са тъждествено верни в структура* **M**, *също е тъждествено вярна в* **M**.

Доказателство. Нека двата дизюнкта имат частни случаи $\{\varphi\}\cup\delta'$ и $\{\overline{\varphi}\}\cup\delta''$, а а резолвентата е $\delta'\cup\delta''$. Съгласно твърдение 6.9, тези частни случаи са тъждествено верни в **M**. Нека v е произволна оценка в **M**. Трябва да докажем, че резолвентата $\delta'\cup\delta''$ е вярна в **M** при оценка v, т.е. че тя не може да не съдържа литерал, който е верен в **M** при оценка v. За да докажем това, да допуснем, че резолвентата не съдържа литерал, който е верен при оценка v. Това означава, че литералите от δ', както и литералите от δ'' са неверни в **M** при оценка v. Тъй като частният случай $\{\varphi\}\cup\delta'$ не може да не съдържа литерал, който е верен в **M** при оценка v, то литералът φ не може да не е верен в **M** при оценка v и значи литералът $\overline{\varphi}$ не е верен в **M** при оценка v. Така доказахме, че всички литерали на частният случай $\{\overline{\varphi}\}\cup\delta''$ са неверни в **M** при оценка v, а това е противоречие. ∎

6.12. Дефиниция. Нека Γ е множество от дизюнкти.

а) За всяко естествено число n ще дефинираме множество ЛРИ$_n$(Γ) от дизюнкти. Нека ЛРИ$_0$(Γ) = Γ. И нека за всяко i множеството ЛРИ$_{i+1}$(Γ) съдържа елементите на ЛРИ$_i$(Γ), както и всички либерални резолвенти на елементи на ЛРИ$_i$(Γ).

б) Нека ЛРИ(Γ) е обединението на всички множества ЛРИ$_i$(Γ).

Интуитивно ЛРИ$_i$(Γ) съдържа всички дизюнкти, които могат да се докажат посредством i-стъпково резолютивно разсъждение, използващо като аксиоми дизюнктите от Γ. Множеството ЛРИ(Γ) съдържа всички дизюнкти, които могат да се докажат резолютивно от Γ.

6.13. Твърдение. *Ако* Γ *е множество от тъждествено верни в структура* **M** *дизюнкти, то елементите на* ЛРИ$_0$(Γ), ЛРИ$_1$(Γ), ЛРИ$_2$(Γ), ЛРИ$_3$(Γ),... *и* ЛРИ$_0$(Γ) *съдържат тъждествено верни в* **M** *дизюнкти.*

Доказателство. С индукция по i ще докажем, че всички елементи на ЛРИ$_i$(Γ) са тъждествено верни в **M**, от където ще следва, че и елементите на ЛРИ(Γ) са такива дизюнкти.

При $i = 0$ няма какво да доказваме, защото ЛРИ$_0$(Γ) = Γ.

Да допуснем, че ЛРИ$_i$(Γ) съдържа тъждествено верни в **M** дизюнкти. От твърдение 6.11 следва, че всички либерални резолвенти на елементи на ЛРИ$_i$(Γ) са тъждествено верни в **M** и значи всички елементи на ЛРИ$_{i+1}$(Γ) са тъждествено верни в **M**. ∎

6.14. Теорема за коректност на либералната резолютивна изводимост. *Ако множеството от дизюнкти* Г *е изпълнимо, то празният дизюнкт не е елемент на* ЛРИ(Г).

Доказателство. Да допуснем, че дизюнктите от Г са тъждествено верни в структура **M**. Съгласно твърдение 6.13 елементите на ЛРИ(Г) също ще бъдат тъждествено верни в **M**. Съгласно твърдение 6.7 пък, празният дизюнкт не е тъждествено верен в **M**. ■

Приложение А

Реализация на език за програмиране ИМПЕРАТОР

В това приложение ще видим как можем да напишем на пролог интерпретатор на прост императивен език за програмиране и то така, че този език да стана част от пролог. Понеже всеки език за програмиране трябва да си има име, нека дадем на нашия език името ИМПЕРАТОР.

Най-напред ще определим синтаксиса на ИМПЕРАТОР. Всяка програма на ИМПЕРАТОР ще се състои просто от ключовата дума `император`, следвана от списък от команди, разделени със символа точка и запетая. Всяка команда ще има един от следните пет вида.

Първо, оператор за присвояване. Отляво трябва да стои идентификатор, който си го мислим като име на променлива на ИМПЕРАТОР:

```
променлива := израз
```

Второ, присвояване на стойност на променлива на пролог. Отляво трябва да стои име на променлива на пролог, която все още не е получила стойност:

```
променлива <== израз
```

Трето, изпълнение на предикат на пролог (също както в програма на пролог можем да изпълняваме код, написан на ИМПЕРАТОР, така и в програма на ИМПЕРАТОР можем да извикваме предикати на пролог):

```
пролог p(T1,T2,...,Tn)
```

Четвърто, оператор за цикъл while. Ще считаме, че цикълът се изпълнява докато стойността на контролиращия израз е различна от нула:

```
while израз loop (
   оператор 1;
   оператор 2;
   ...
   оператор n
)
```

Пето, оператор за цикъл for:

```
for i in A..B loop (
   оператор 1;
   оператор 2;
   ...
   оператор n
)
```

Шесто, условен оператор. Приемаме, че изразът е истина, ако стойността му е различна от нула:

```
if израз then (
   оператор 1;
   оператор 2;
   ...
   оператор n
)
```

или

```
if израз then (
   оператор 1;
   оператор 2;
   ...
   оператор n
) else (
   оператор 1;
   оператор 2;
   ...
   оператор m
)
```

Използвайки този език, ето как можем да дефинираме на пролог предикат за пресмятане на факториела на едно число:

```
% по дадено естествено число N записва в F неговия факториел
факториел(N,F) :- императ
   f := 1;
   for i in 2..N loop (
      f := f * i
   );
   F <== f.
```

В последния ред от тази програма присвояваме на променливата с име F на пролог стойността, която в този момент променливата с име f на ИМПЕРАТОР има.

Пълната реализация на ИМПЕРАТОР е дадена в края на това приложение. Преди това обаче, ще коментираме някои по-трудни за разбиране неща. Първо, да видим как е реализиран синтаксисът на ИМПЕРАТОР.

Операторите `император`, `:=`, `<==` и `пролог` са дефинирани така:

```
:- op(1150, fx, [император]).
:- op(990, xfx, [':=', '<==']).
:- op(990, fx, [пролог]).
```

Тук приоритетът 1150 е подбран така, че да бъде по-малко число (т.е. по-голям приоритет) от приоритета на `:-`, но по-голямо число от приоритета на оператора точка и запетая. Първото ни гарантира, че няма нужда да ограждаме програмата на ИМПЕРАТОР в скоби така: `p :- (император ...)`. Второто ни позволява да не слагаме скоби след `император` само заради това, че в програмата се съдържат команди, разделени с точка и запетая ето така: `p :- император (... ; ...)`.*

Приоритетът 990 е подбран така, че бъде по-голямо число (т.е. по-малък приоритет) от приоритетите на всички аритметични операции в пролог, но по-малко число, от приоритета на оператора точка и запетая. Първото ни гарантира, че няма нужда да пишем скоби в израза след `:=` и `<==`, например `i := (1+2)`. Второто пък означава, че няма нужда да пишем скоби около самия оператор за присвояване, например `(i:=1);(f:=1)`.

Синтаксисът на оператора за цикъл while на ИМПЕРАТОР използва две различни ключови думи — `while` и `loop`. Всяка една от тях трябва

*Обаче скоби около програмата на ИМПЕРАТОР все пак трябва да се поставят, ако тя не е единственото нещо, което се прави в предиката на пролог:

```
p(X) :- q(X),
        (император
            тук е програмата на ИМПЕРАТОР
        ),
        r(X).
```

да бъде дефинирана като префиксен, постфиксен или инфиксен оператор на пролог със свой собствен приоритет. Един начин това да бъде направено е да дефинираме `while` като префиксен оператор, а `loop` — като инфиксен с два аргумента. Ще дадем на `while` по-голям приоритет, отколкото на `loop`, така че в израза

`while` израз `loop (` команди `)`

пролог да постави скобите по следния начин:

`(while` израз `) loop (` команди `)`

Записан като терм същият цикъл изглежда така:

`loop(while(`израз`),` команди`)`

Ще изберем така приоритетите на `while` и `loop`, че те да бъдат по-големи от приоритета на оператора точка и запетая, но по-малки от приоритетите на операторите `:=`, `<==` и `пролог`:

```
:- op(994, fx, [while]).
:- op(995, xfx, [loop]).
```

Операторът за цикъл for използва операторите `for`, `in`, `..` и `loop`. Така ще ги дефинираме, че в израза

`for` i `in` a`..`b `loop (` команди `)`

пролог да постави скобите по следния начин:

`(for (`i `in (`a`..`b`))) loop (` команди `)`

Записан като терм същият цикъл изглежда така:

`loop(for(in(i,(a..b))),` команди`)`

Ще изберем така приоритетите на `for`, `in` и `..`, че те да бъдат по-големи от приоритета на `loop` и по-малки от приоритетите на аритметичните оператори:

```
:- op(994, fx, [for]).
:- op(993, xfx, [in]).
:- op(992, xfx, ['..']).
```

Условният оператор на ИМПЕРАТОР използва две или три ключови думи — `if`, `then` и може би също `else`:

```
:- op(995, fx, [if]).
:- op(993, xfx, [then]).
:- op(994, xfx, [else]).
```

Така избраните приоритети означават, че пролог ще интерпретира изразите

```
if израз then ( команди )
if израз then ( команди1 ) else ( команди2 )
```

като термовете

```
if(then(израз,команди))
if(else(then(израз,команди1),команди2)
```

За разлика от пролог, на ИМПЕРАТОР променливите могат да си променят стойността. Това означава, че не можем да помним стойността на променливите на ИМПЕРАТОР в променливи на пролог. Вместо това ще използваме базата знания на пролог.

Да припомним, че вграденият предикат `assert` може да се използва, за да добавим факти към базата знания на пролог. Например след изпълнението на `assert(a(5))` пролог „ще знае“, че `a(5)` е вярно. Предикатът `retract` пък прави обратното, например след изпълнението на `retract(a(5))` пролог „ще забрави“, че `a(5)` е вярно.

За да запомним, че стойността на променливата `x` е 5, ще добавим към базата знания на пролог факт от вида

```
стойност_на_променлива(x, 5)
```

За да дадем на променливата `x` нова стойност, например 7, първо трябва да премахнем от базата знания старата стойност, т.е. факта, имащ вида `стойност_на_променлива(x, _)` и едва след това да добавим новия факт `стойност_на_променлива(x, 7)`.

Следва пълната реализация на ИМПЕРАТОР:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% © 2015 Антон Зиновиев
%
% Тази програма може бъде използвана без ограничения от
% всеки, който е получил нейно копие, в това число да бъде
% разпространявана, изменяна, лицензирана и прелицензирана
% при условие, че тази бележка за авторските права не се
% трие.  ПРОГРАМАТА СЕ ПРЕДОСТАВЯ БЕЗ КАКВИТО И ДА Е
```

```
% ГАРАНЦИИ, ЧЕ СТАВА ЗА НЕЩО СМИСЛЕНО И НЯМА ДА НАВРЕДИ.
% Без изрично разрешение, имената на носителите на
% авторските права не могат да бъдат използвани за
% популяризиране на продукти, базирани на тази програма.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(1150, fx, [император]).
:- op(990, xfx, [':=', '<==']).
:- op(990, fx, [пролог]).

:- op(994, fx, [while]).
:- op(995, xfx, [loop]).

:- op(994, fx, [for]).
:- op(993, xfx, [in]).
:- op(992, xfx, ['..']).

:- op(995, fx, [if]).
:- op(993, xfx, [then]).
:- op(994, xfx, [else]).

%% стойност(Израз, Стойност) -- пресмята в Стойност
%%            стойността на Израз
стойност(Пром, Стойност) :-
        atom(Пром),
        стойност_на_променлива(Пром, Стойност).
стойност(Число, Число) :-
        number(Число).
стойност(A + B, Z) :-
        стойност(A, X), стойност(B, Y),
        Z is X + Y.
стойност(A - B, Z) :-
        стойност(A, X), стойност(B, Y),
        Z is X - Y.
стойност(A * B, Z) :-
        стойност(A, X), стойност(B, Y),
        Z is X * Y.
стойност(A / B, Z) :-
        стойност(A, X), стойност(B, Y),
        Z is X / Y.
стойност(A = B, Z) :-
```

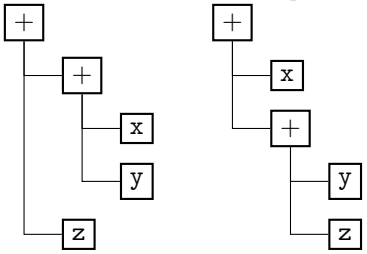
```
        стойност(A, X), стойност(B, Y),
        (X = Y -> Z = 1 ; Z = 0).
стойност(A\= B, Z) :-
        стойност(A, X), стойност(B, Y),
        (X \= Y -> Z = 1 ; Z = 0).
стойност(A < B, Z) :-
        стойност(A, X), стойност(B, Y),
        (X < Y -> Z = 1 ; Z = 0).
стойност(A > B, Z) :-
        стойност(A, X), стойност(B, Y),
        (X > Y -> Z = 1 ; Z = 0).
стойност(A =< B, Z) :-
        стойност(A, X), стойност(B, Y),
        (X =< Y -> Z = 1 ; Z = 0).
стойност(A >= B, Z) :-
        стойност(A, X), стойност(B, Y),
        (X >= Y -> Z = 1 ; Z = 0).

император X ; Y :-
        (император X), (император Y).
император V := E :-
        стойност(E, EVal), !,
        (
         %% премахваме старата стойност на V
         retract(стойност_на_променлива(V, _))
        ;
         true
        ), !,
        %% добавяме новата стойност
        assert(стойност_на_променлива(V, EVal)).
император V <== E :-
        стойност(E, V), !.
император пролог Цел :-
        call(Цел).
император if E then X :-
        стойност(E, EVal), !,
        (EVal \= 0 -> (император X) ; true).
император if E then X else Y :-
        стойност(E, EVal), !,
        (EVal \= 0 -> (император X) ; (император Y)).
император while E loop X :-
```

```
        стойност(E, EVal), !,
        (
         EVal \= 0
        ->
         (император X), !, (император while E loop X)
        ;
         true
        ).
император for I in A..B loop X :- император
        I := A;
        while I =< B loop (
           X;
           I := I + 1
        ).
```

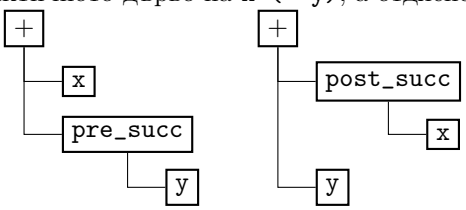
Приложение Б

Решения на задачите

Задача 1. Отляво е синтактичното дърво на `(x+y)+z`, `x+y+z`, `((x+y))+z` и `(x)+y+z`, а отдясно — синтактичното дърво на `x+(y+z)`:



Задача 2. Тъй като на си има префикснен `++` и постфиксен `++`, за префиксния ще използваме етикет `pre_succ`, а за постфиксния — `post_succ`. В такъв случай отляво е синтактичното дърво на `x+(++y)`, а отдясно — на `(x++)+y`:



Компилаторите на си ще интерпретират `x+++y` като `(x++)+y`,* но не знам дали това е произволно решение или е предписание на стандарта на си.

Задача 3. `f` е двуместен функционален символ, `g` и `h` са едноместни функционални символи, `c` и `a` са символи за константи, а `x` и `y` са променливи.

Задача 4. `f(c)` е терм, `c(f)` не е терм, защото константата `c` не може да има аргументи, а функционалният символ `f` трябва да има аргументи, `c` е терм, `f` не е терм,

*И също `x+++++y` като `((x++)++)+y`.

защото функционалният символ $\mathtt{f}$ трябва да има поне един аргумент, $\mathtt{x}(\mathtt{c})$ не е терм, защото променливите не могат да имат аргументи, $\mathtt{f}(\mathtt{f}(\mathtt{x}))$ е терм, $\mathtt{f}(\mathtt{f}(\mathtt{f}(\mathtt{c})))$ е терм, $\mathtt{f}(\mathtt{f}(\mathtt{f}(\mathtt{c},\mathtt{c})))$ не е терм, защото $\mathtt{f}$ не може да бъде едновременно едноместен и двуместен, $\mathtt{g}(\mathtt{g}(\mathtt{x},\mathtt{x}),\mathtt{g}(\mathtt{x},\mathtt{y}))$ е терм.

Задача 5. Термът $\mathtt{f}(\mathtt{c})$ съдържа скоби, но не съдържа запетаи. Няма термове, които съдържат запетаи, но не съдържат скоби. Във всички термове има равен брой леви и десни скоби.

Задача 6. Например $\mathtt{f}(\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c})$ и $\mathtt{g}(\mathtt{g}(\mathtt{h}(\mathtt{c},\mathtt{c},\mathtt{c})))$. Във втория от тези два терма може ли да заменим $\mathtt{h}$ с $\mathtt{g}$?

Задача 7. Термът $+(\mathtt{c},*(\mathtt{x},\mathtt{a}))$ съдържа 4 скоби — две леви и две десни.

Задача 8. Тъй като $\mathtt{c}$ е символ за константа константа, от правило 2.4 б) получаваме, че $\mathtt{c}$ е терм.

Тъй като $\mathtt{x}$ е променлива, от правило 2.4 а) получаваме, че $\mathtt{x}$ е терм.

Тъй като вече сме доказали, че $\mathtt{c}$ и $\mathtt{x}$ са термове, а $\mathtt{f}$ е двуместен функционален символ, от правило 2.4 в) получаваме, че $\mathtt{f}(\mathtt{c},\mathtt{x})$ е терм.

Тъй като вече сме доказали, че $\mathtt{c}$ и $\mathtt{f}(\mathtt{c},\mathtt{x})$ са термове, а $\mathtt{f}$ е двуместен функционален символ, от правило 2.4 в) получаваме, че $\mathtt{f}(\mathtt{c},\mathtt{f}(\mathtt{c},\mathtt{x}))$ е терм.

Задача 9. Ако заменим константата с $\mathtt{c}$, едноместния функционален символ с $\mathtt{f}$ и двуместния с $\mathtt{g}$, то тези изрази ще добият следния вид:

$$\boxed{\mathtt{c}}\ \boxed{\mathtt{f}(\mathtt{c})}\ \boxed{\mathtt{f}(\mathtt{f}(\mathtt{c}))}\ \boxed{\mathtt{g}(\mathtt{c},\mathtt{c})}\ \boxed{\mathtt{g}(\mathtt{f}(\mathtt{c}),\mathtt{g}(\mathtt{c},\mathtt{c}))}$$

Задача 10. С индукция по построението на терма ще докажем, че всеки терм съдържа поне една променлива.

а) Ако $\mathtt{x}$ е променлива, то очевидно $\mathtt{x}$ съдържа променлива.

б) Символи за константи няма.

в) Нека $\mathtt{f}$ е n-местен функционален символ и $\tau_1, \tau_2, \ldots, \tau_n$ са термове, всеки от които съдържа поне една променлива.[*] В такъв случай очевидно и термът $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$ ще съдържа поне една променлива (тук неявно използваме, че $n \geqq 1$).

Задача 11. а) Ако $\mathtt{x}$ е променлива, то $\mathtt{x}$ съдържа равен брой запетаи и десни скоби (нула), защото променливите не са запазени символи, а запетаята и дясната скоба са.

б) Ако $\mathtt{c}$ е символ за константа, то $\mathtt{c}$ съдържа равен брой запетаи и десни скоби (нула), защото символите за константи не са запазени символи, а запетаята и дясната скоба са.

в) Нека $\mathtt{f}$ е функционален символ (по услови той трябва да бъде двуместен). Да допуснем, че τ_1 и τ_2 са термове с равен брой запетаи и десни скоби.[**] Нека k_i е броят на запетаите в τ_i (същото число е равно и на броя на десните скоби). Тъй като $\mathtt{f}$ не е запазен символ, то $\mathtt{f}$ не запетая или скоба. Следователно броят на запетаите в терма $\mathtt{f}(\tau_1, \tau_2)$ е $1 + k_1 + k_2$ — една запетая между τ_1 и τ_2 плюс запетаите в τ_1 и τ_2. Броят на десните скоби е същият — затворената скоба в края на терма плюс скобите в τ_1 и τ_2.

Задача 12. Индуктивният принцип казва следното:

Нека $\mathtt{p}$ е свойство, за което е вярно, че:

[*] Това е индукционното предположение.

[**] Това е индукционното предположение.

1. свойството $\mathfrak{p}$ е вярно за 5;
2. свойството $\mathfrak{p}$ е вярно за 8;
3. ако свойството $\mathfrak{p}$ е вярно за буртантите n и m, то то ще бъде вярно и за nm^2.

Тогава свойството $\mathfrak{p}$ ще бъде вярно за всички буртанти.

Ще използваме този индуктивен принцип в случая, когато $\mathfrak{p}$ е свойството, че числото събрано с 1 се дели на 3 без остатък. За целта трябва да докажем, че са верни трите изисквания в индуктивния принцип за буртанти.

1. Очевидно 5 събрано с 1 се дели на 3 без остатък.
2. Очевидно 8 събрано с 1 се дели на 3 без остатък.
3. Да допуснем, че n и m са такива буртанти, че $n+1$ и $m+1$ се делят на 3 без остатък.* Нека $n+1 = 3k$ и $m+1 = 3l$. Тогава

$$\begin{aligned} nm^2 + 1 &= ((n+1)-1)((m+1)-1)^2 + 1 \\ &= (3k-1)(3l-1)^2 \\ &= 3k(3l-1)^2 - (3l-1)^2 + 1 \\ &= 3k(3l-1)^2 - 9l^2 + 6l - 1 + 1 \\ &= 3(k(3l-1)^2 - 3l^2 + 2l) \end{aligned}$$

Следователно $nm^2 + 1$ се дели на три без остатък.

Доказахме, че трите условия от принципа за индукция за буртанти са изпълнени, от където следва, че всички буртанти събрани с единица се делят на 3 без остатък.

Задача 13. `[[[1],2],[3,4],5,6]`. За синтактичното дърво вж. фиг. 14.

Задача 14. `:- op(800,fy,[not]).`

Задача 15.

```
:- op(800,fy,[не]).
:- op(1000, xfy, [и]).
:- op(1100, xfy, [или]).
не Нещо :- not(call(Нещо)).
Едно и Друго :- call(Едно), call(Друго).
Едно или Друго :- call(Едно); call(Друго).
```

Задача 16. `c` е символ за константа и значи терм, а не атомарна формула, `f(c)` е терм без променливи, а не атомарна формула, `p(c)` и `p(f(f(f(c))))` са атомарни формули без променливи, `p(f(x))` е атомарна формула, която съдържа променливи, `f(p(x))` и `p(p(x))` не са нито термове, нито атомарни формули, `f(f(x))` не е атомарна формула, а терм, който съдържа променливи.

Задача 17. (8) За произволни реални числа x, y и z ако $x < y$ и $y < z$, то $x < z$.

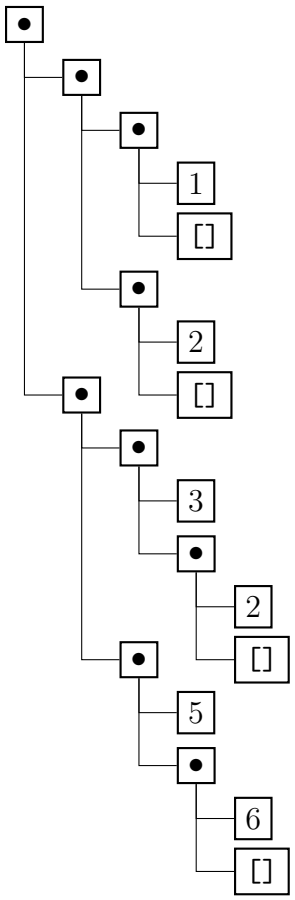
(9) За произволни реални числа x, y и z ако $x < y$, то $x + z < y + z$.

(10) Ако $3 < 3$, то всяко реално число е по-малко от себе си.

Задача 18. 1. $\mathtt{p(x,y)} \vdash \mathtt{p(y,x)}$

2. $\vdash \mathtt{p(x,y)}$

*Това че $n+1$ и $m+1$ се делят на 3 без остатък е индукционното предположение.



Фиг. 14. Синтактично дърво за
`[[[1|[]]|[2|[]]]|[[3|[4|[]]]|[5|[6|[]]]]]` и
`•(•(•(1,[]),•(2,[])),•(•(3,•(4,[])),•(5,•(6,[]))))`

Задача 19. Тъй като универсумът на $\mathbf{M}$ е непразно множество, то той съдържа поне един елемент μ. Да дефинираме оценката v така: $v(\mathtt{x}) = \mu$ за всяка променлива $\mathtt{x}$.

Задача 20. Нека стойността на φ в структурата $\mathbf{M}$ при оценка v е твърдението $\mathfrak{p}$. Тогава φ е невярна тогава и само тогава, когато $\mathfrak{p}$ е лъжа, тогава и само тогава, когато $\mathfrak{p}$ не е истина, тогава и само тогава, когато φ не е вярна.

Задача 21. 1. Тъй като τ не съдържа променливи, то $v'(\mathtt{x}) = v''(\mathtt{x})$ за всяка променлива $\mathtt{x}$, която се среща в τ. Затова искането следва от 3.23 и 3.24.

2. се доказва аналогично на а).

Задача 22. Непосредствено от дефиницията следва, че φ не е изпълнима в $\mathbf{M}$ тогава и само тогава, когато φ е неизпълнима в $\mathbf{M}$.

Ако φ е неизпълнима, то не съществува оценка, при която φ е вярна, следователно каквато и оценка да изберем, φ няма да е вярна при тази оценка и значи φ е невярна при всяка оценка, т.е. φ е тъждествено невярна.

Ако φ е тъждествено невярна в $\mathbf{M}$, то φ е невярна при всяка оценка, значи няма как да намерим оценка, при която φ е вярна, следователно φ е неизпълнима.

Задача 23. Да. Например атомарната формула $\mathtt{x} = \mathtt{c}$ е изпълнима във всяка структура, защото е вярна когато оценката дава на $\mathtt{x}$ същата стойност, каквато структурата дава на $\mathtt{c}$. Но от друга страна тази формула няма да бъде тъждествено вярна, ако в универсума на структурата се съдържат поне два елемента, защото тогава ще съществуват оценки, при които стойността на $\mathtt{x}$ се различава от стойността на $\mathtt{c}$.

Задача 24. Ако φ е тъждествено вярна, то φ е вярна при всяка оценка. Нека v е произволно избрана оценка. Тогава φ е вярна при оценка v, значи φ е изпълнима в структурата и следователно не е неизпълнима в структурата.

В това доказателство използваме, че универсумът на структурата е непразно множество, по следния начин: за да можем да изберем произволна оценка v, е нужно да сме сигурни, че съществува поне една оценка. Ако ни е даден поне един елемент от универсума, тогава със сигурност ще съществува поне една оценка — да вземем например оценката, която на всички променливи дава като стойност дадения елемент от универсума. Ако обаче универсумът бе празното множество, то не би съществувала нито една оценка, защото за произволна променлива $\mathtt{x}$, $v(\mathtt{x})$ трябва да бъде елемент на универсума.

Задача 25. Задачата може да се реши лесно, използвайки свойствата на хомоморфизмите, които ще докажем по-нататък, но всъщност тя има много и най-различни решения. Ето едно (упътване):

Нека $\mathbf{M}$ е модел за Γ. За всяко естествено число m ще дефинираме различни по между си структури $\mathbf{K}_m$, които са модели за Γ. Универсум на $\mathbf{K}_m$ е $|\mathbf{M}| \times \{m\}$. За всеки символ за константа нека $\mathtt{c}^{\mathbf{K}_m} = (\mathtt{c}^{\mathbf{K}}, m)$. За всеки функционален символ нека $\mathtt{f}^{\mathbf{K}_m}((\mu_1, m), (\mu_2, m), \ldots, (\mu_n, m)) = (\mathtt{f}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n), m)$. И за всеки предикатен символ нека $\mathtt{p}^{\mathbf{K}_m}((\mu_1, m), (\mu_2, m), \ldots, (\mu_n, m)) = \mathtt{p}^{\mathbf{M}}(\mu_1, \mu_2, \ldots, \mu_n)$.

За произволна оценка v в $\mathbf{K}_m$ да означим с v' оценката в $\mathbf{M}$, такава че за произволна променлива $\mathtt{x}$, $v'(\mathtt{x})$ е равно на първия елемент на наредената двойка $v(\mathtt{x})$.

С индукция по термовете докажете, че за произволен терм τ и оценка v в $\mathbf{K}_m$, ако μ е стойността на τ в $\mathbf{M}$ при оценка v', то (μ, m) е стойността на τ в $\mathbf{K}_m$ при оценка v.

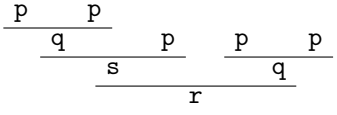
След това докажете още, че за произволна атомарна формула φ и оценка v в $\mathbf{K}_m$, стойността на φ в $\mathbf{M}$ при оценка v' е еквивалентна на стойността на φ в $\mathbf{K}_m$ при оценка v.

От тук заключете, че ако някоя атомарна формула или клауза е тъждествено вярна в $\mathbf{M}$, то тя ще бъде тъждествено вярна и в $\mathbf{K}_m$.*

Задача 26.

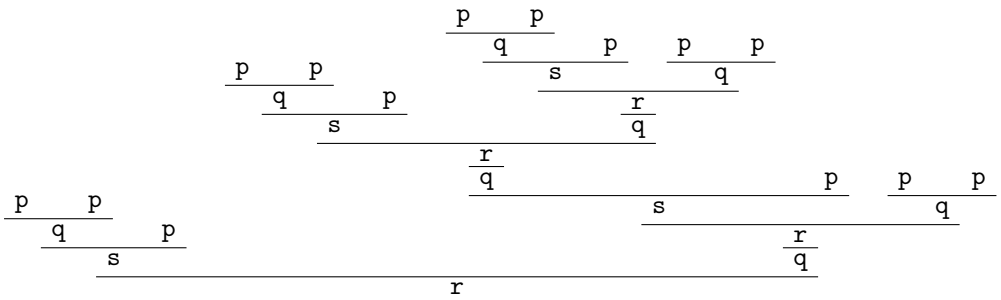
1. Ако структурата $\mathbf{M}$ е такава, че в нея са тъждествено верни всички елементи на Γ, то в частност и φ ще бъде тъждествено вярно в $\mathbf{M}$.
2. следва от в).
3. Да допуснем, че φ следва от Δ. Да изберем произволна структура, в която са тъждествено верни елементите на Γ. Тъй като всеки от елементите на Δ следва от Γ, то в тази структура ще бъдат тъждествено верни и елементите на Δ. Но φ следва от Δ, значи в тази структура и φ е тъждествено вярно.

Задача 27. Един възможен извод е следният:



*Всъщност е вярно и обратното.

Ето друг възможен извод:



Кой е възможно най-големият извод на $\mathtt{r}$, който можете да измислите?

Задача 28. 1. С индукция по терма τ. Ако $\tau = \mathtt{x}$ е променлива, то $\hat{s}(\tau) = s(\mathtt{x})$, следователно $\hat{s}(\tau)$ е терм. Ако $\tau = \mathtt{c}$ е константа, то $\hat{s}(\tau) = \mathtt{c}$ и е терм. Ако $\tau = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$, то съгласно индукционното предположение $\hat{s}(\tau_1)$, $\hat{s}(\tau_2)$,..., $\hat{s}(\tau_n)$ са термове, следователно $\hat{s}(\tau) = \mathtt{f}(\hat{s}(\tau_1), \hat{s}(\tau_2), \ldots, \hat{s}(\tau_n))$ също е терм (вж. дефиниция 2.4 в)).

2. Ако $\varphi = \mathtt{p}(\tau_1, \tau_2, \ldots, \tau_n)$, то от а) следва, че $\hat{s}(\tau_1)$, $\hat{s}(\tau_2)$,..., $\hat{s}(\tau_n)$ са термове, следователно $\hat{s}(\varphi) = \mathtt{p}(\hat{s}(\tau_1), \hat{s}(\tau_2), \ldots, \hat{s}(\tau_n))$ също е атомарна формула (вж. дефиниция 3.2 б)).
3. следва от б).

Задача 29. 1. Да, ако $s(\mathtt{x}) = \mathtt{c}$.

2. Не, защото φ не съдържа променливи и значи $\hat{s}(\varphi) = \varphi$ за всяка субстуция s.
3. Не, защото за да може първият аргумент на $\mathtt{p}$ да бъде $\mathtt{c}$, е нужно $s(\mathtt{x}) = \mathtt{c}$, но тогава и вторият аргумент ще стане $\mathtt{c}$. Изобщо $\hat{s}(\varphi)$ винаги има вида $\mathtt{p}(\sigma, \sigma)$ за някой терм σ.
4. Да, ако $s(\mathtt{x}) = \mathtt{f}(\mathtt{y})$ и $s(\mathtt{y}) = \mathtt{f}(\mathtt{x})$.

Задача 30. По дефиниция $\text{ЛПИ}_0(\Gamma) = \varnothing$.

Клаузата $\vdash \mathtt{p}(\mathtt{c})$ няма променливи и затова неин единствен частен случай е самата тя. С нейна помощ можем да получим $\mathtt{p}(\mathtt{c})$ като елемент на $\text{ЛПИ}_1(\Gamma)$. Всички частни случаи на другата клауза от Γ имат една предпоставка и затова няма как с нея да изведем някоя атомарна формула от $\varnothing$. Следователно $\text{ЛПИ}_1(\Gamma) = \{\mathtt{p}(\mathtt{c})\}$.

Единствената формула, която можем да получим използвайки клаузата $\vdash \mathtt{p}(\mathtt{c})$ е $\mathtt{p}(\mathtt{c})$, а тя вече е елемент на $\text{ЛПИ}_1(\Gamma)$. Следователно, за да получим някакъв нов елемент на $\text{ЛПИ}_2(\Gamma)$ има смисъл да използваме единствено другото правило $\mathtt{p}(\mathtt{x}) \vdash \mathtt{p}(\mathtt{f}(\mathtt{x}))$. Единственият частен случай на единственият елемент на $\text{ЛПИ}_1(\Gamma)$ е $\mathtt{p}(\mathtt{c})$. Единственият частен случай на това правило, който има за предпоставка $\mathtt{p}(\mathtt{c})$ е $\mathtt{p}(\mathtt{c}) \vdash \mathtt{p}(\mathtt{f}(\mathtt{c}))$. С негова помощ получаваме $\mathtt{p}(\mathtt{f}(\mathtt{c}))$ като нов елемент на $\text{ЛПИ}_2(\Gamma)$. Следователно $\text{ЛПИ}_2(\Gamma) = \{\mathtt{p}(\mathtt{c}), \mathtt{p}(\mathtt{f}(\mathtt{c}))\}$.

За улеснение да се уговорим с $\mathtt{p}(\mathtt{f}^n(\mathtt{c}))$ да записваме формулата имаща вида $\mathtt{p}(\mathtt{f}(\mathtt{f}(\mathtt{f}(\ldots\mathtt{f}(\mathtt{c})\ldots))))$, която съдържа точно n символа $\mathtt{f}$.

С индукция по i ще докажем, че $\text{ЛПИ}_i(\Gamma)$ съдържа всички атомарни формули от вида $\mathtt{p}(\mathtt{f}^k(\mathtt{c}))$, където $0 \leqq k \leqq i$. Да допуснем, че вече сме доказали това за i ($i \geqq 1$). Ще го докажем и за $i+1$.

По дефиниция $\text{ЛПИ}_{i+1}(\Gamma)$ съдържа всички елементи на $\text{ЛПИ}_i(\Gamma)$. Да видим какви нови елементи може да съдържа това множество.

Вече отбелязахме, че първата клауза от Γ има за единствен частен случай клаузата $\vdash \mathtt{p(c)}$, а с нейна помощ можем да получим единствено атомарната формула $\mathtt{p(c)}$, за която съгласно индукционното предположение вече знаем, че е елемент на $\text{ЛПИ}_i(\Gamma)$. Да видим какво можем да получим, използвайки второто правило.

Съгласно индукционното предположение елементите на $\text{ЛПИ}_i(\Gamma)$ не съдържат променливи. Следователно единствени техни частни случаи са самите те. Единственият частен случай на второто правило от Γ, който можем комбинираме с атомарна формула от вида $\mathtt{p(f}^k\mathtt{(c))}$, е $\mathtt{p(f}^k\mathtt{(c))} \vdash \mathtt{p(f}^{k+1}\mathtt{(c))}$. Следователно от $\mathtt{p(f}^k\mathtt{(c))}$ получаваме $\mathtt{p(f}^{k+1}\mathtt{(c))}$. Имайки предвид какво съгласно индукционното предположение съдържа $\text{ЛПИ}_i(\Gamma)$, установяваме, че $\text{ЛПИ}_{i+1}(\Gamma)$ съдържа точно това, което трябва.

Задача 31. Клаузите от Γ ще имат повече частни случаи. Тези частни случаи обаче ще бъдат неизползваеми, защото атомарните формуле в $\text{ЛПИ}_i(\Gamma)$ не съдържат променливи и следователно техни единствени частни случаи са само те. Същевременно единствените символи, срещащи се в тези атомарни формули са $\mathtt{c}$, $\mathtt{f}$ и $\mathtt{p}$. Следователно множествата $\text{ЛПИ}_i(\Gamma)$ съдържат същите елементи, както и в предната задача.

Задача 32. Да означим с $\mathtt{p(f}^n\mathtt{(c))}$ формулата, имаща вида $\mathtt{p(f(f(f(\ldots f(c)\ldots))))}$ и съдържаща точно n символа $\mathtt{f}$. Тогава $\text{ЛПИ}_0(\Gamma) = \varnothing$ и $\text{ЛПИ}_1(\Gamma) = \{\mathtt{p(c,c)}\}$, а $\text{ЛПИ}_i(\Gamma)$ съдържа формулата $\mathtt{p(c,c)}$ и всички формули от вида $\mathtt{p(f}^k\mathtt{(c)},\tau)$, където $1 \leqq k \leqq i-1$ и τ е произволен терм от сигнатурата.

Задача 33. (*Упътване.*) Основната трудност е това, че клаузите могат да имат има безброй много частни случаи. Например клаузата $\vdash \mathtt{x} = \mathtt{x}$ може да има частни случаи $\vdash \mathtt{f(x)} = \mathtt{f(x)}$, $\vdash \mathtt{f(f(x))} = \mathtt{f(f(x))}$, $\vdash \mathtt{f(f(f(x)))} = \mathtt{f(f(f(x)))}$ и много други.

Тъй като по условие сигнатурата съдържа краен брой символи, то за всяко естествено число n съществуват краен брой термове, съставени от не повече от n символа. Следователно всяка клауза ще има краен брой частни случаи, при които заменяме променливите с термове, всеки от които съдържа не повече от n символа. Да наречем такива частни случаи *ограничени* до n.

Ще дефинираме множества $\Delta_0, \Delta_1, \Delta_2, \ldots$, които са аналогични на $\text{ЛПИ}_i(\Gamma)$, но са крайни и могат да се генерират алгоритмично.

Нека $\Delta_0 = \varnothing$.

За всяко i нека Δ_{i+1} съдържа елементите на Δ_i, както и всички атомарни формули ψ, такива че някой елемент на Γ има ограничен до i частен случай

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$$

и $\varphi_1, \varphi_2, \ldots, \varphi_n$ са ограничени до i частни случаи на елементи на Δ_i. Тъй като има краен брой такива частни случаи и множеството Δ_i е крайно, не е проблем да намерим всички такива ψ (те също са краен брой).

Нека Δ е обединението на всички Δ_i. Въпреки че Δ е може би безкрайно множество, видяхме, че елементите му могат да се генерират алгоритмично.

Остава да докажем, че $\Delta = \text{ЛПИ}(\Gamma)$. Включването $\Delta \subseteq \text{ЛПИ}(\Gamma)$ следва непосредствено от дефинициите, защото при Δ_i вземахме ограничени частни случаи, а при $\text{ЛПИ}_i(\Gamma)$ произволни, така че всичко каквото можем да правим при Δ_i, можем да го правим и при $\text{ЛПИ}_i(\Gamma)$ и значи $\Delta_i \subseteq \text{ЛПИ}_i(\Gamma)$.

Обратното включване се вижда по-трудно. Нека $\varphi \in \text{ЛПИ}_i(\Gamma)$. В такъв случай някой елемент на Γ е имал частен случай

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \varphi$$

такъв че $\varphi_1, \varphi_2, \ldots, \varphi_n$ са частни случаи на елементи на ЛПИ$_{i-1}(\Gamma)$. Всяко от тези φ_i на свой ред е получено от частен случай на елемент на Γ и краен брой частни случаи на елементи на ЛПИ$_{i-2}(\Gamma)$. Те пък са били получени от някои елементи на ЛПИ$_{i-3}(\Gamma)$ и т.н. По този начин установяваме, че „вкарването“ на φ в ЛПИ$_i(\Gamma)$ зависи в някакъв смисъл от краен брой атомарни формули. Нека n е броят на символите на най-големия терм, съдържащ се в някоя от тези атомарни формули. В такъв случай може да се докаже, че $\varphi \in \Delta_{i+n}$ и значи $\varphi \in \Delta$.

Задача 34. Съгласно 4.15 прилаганията на s_1 и s_2 са заместващи морфизми. Съгласно 4.18 последователното прилагане на s_1 и s_2 също е заместващ морфизъм. Съгласно 4.15 и прилагането на s е заместващ морфизъм. Съгласно 4.16 за да докажем, че последователното прилагане на s_1 и s_2 е еквивалентно на прилагането на s, е достатъчно да видим, че тези две преобразования дават един и същ резултат, ако ги приложим към променлива. Но това наистина е така, защото $\hat{s_2}(\hat{s_1}(\mathtt{x})) = \hat{s_2}(s_1(\mathtt{x})) = s(\mathtt{x})$.

Задача 35. Съгласно 4.15 прилагането на s е заместващ морфизъм. Съгласно 3.23 намирането на стойността в $\mathbf{M}$ при оценка v е остойностяващ морфизъм. Съгласно 4.21 последователното прилагане тези морфизми е остойностяващ морфизъм. Съгласно 3.23 намирането на стойността в $\mathbf{M}$ при оценка w е остойностяващ морфизъм. Съгласно 3.24 за да докажем, че тези два остойностяващи морфизма съвпадат, е достатъчно да видим, че те съвпадат, ако ги приложим към променлива. Но това наистина е така, защото стойността на $\mathtt{x}$ в $\mathbf{M}$ при оценка w е равна на $w(\mathtt{x})$, което по условие е равно на стойността на терма $s(\mathtt{x})$ в $\mathbf{M}$ при оценка v, а $s(\mathtt{x}) = \hat{s}(\mathtt{x})$.

Задача 36. Бърз преглед на дефиницията на структура 3.10 показва, че за да дефинираме коя да е структура (вкл. ербранова структура) трябва да определим следното: кой е универсумът, как се интерпретират символите за константите, как се интерпретират функционалните символи и как се интерпретират предикатните символи.

В случая можем да вземем универсъмът на структурата да бъде множеството от всички термове без променливи. Да забележим, че това множество е непразно, защото сигнатурата съдържа поне един символ за константа и този символ е терм без променливи.

Интерпретацията на символите за константи и функционалните символи на всяка една ербранова структура е еднозначно определена, така че не се налага да я измисляме: $\mathtt{c}^{\mathbf{H}} = \mathtt{c}$ и $\mathtt{f}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_n) = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$. Няма никакви ограничения за интерпретацията на предикатните символи. За определеност може да считаме, че $\mathtt{p}^{\mathbf{H}}(\tau_1, \tau_2, \ldots, \tau_N)$ е истина за всеки предикатен символ $\mathtt{p}$ и елементи $\tau_1, \tau_2, \ldots, \tau_n$ на универсума на $\mathbf{H}$.

Тази дефиниция на ербрановата структура $\mathbf{H}$ можем да изкажем по-формално по следния начин.

Нека $\mathbf{H} = (\Omega, \mathtt{i})$, където Ω е множеството от всички термове без променливи.

За всеки символ за константа $\mathtt{c}$ нека $\mathtt{i}(\mathtt{c}) = \mathtt{c}$.

За всеки n-местен функционален символ $\mathtt{f}$ нека $\mathtt{i}(\mathtt{f}) = f$, където $f\colon \Omega^n \to \Omega$ е функцията

$$f(\tau_1, \tau_2, \ldots, \tau_n) = \mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$$

(Отляво на горното равенство скобите и запетаите разграничават аргументите на функцията f, докато отдясно те са просто символи, срещащи се някъде в терма $\mathtt{f}(\tau_1, \tau_2, \ldots, \tau_n)$.)

За всеки n-местен предикатен символ $\mathtt{p}$ нека $\mathtt{i}(\mathtt{p})$ е n-местният предикат p, за който

$$p(\tau_1, \tau_2, \ldots, \tau_n) \longleftrightarrow 1 = 1$$

Така дефинирана, структурата $\mathbf{H}$ ще бъде ербранова.

Задача 37. Ако означаваме скобите и запетаите, които са символи, с $\mathtt{,}$, $\mathtt{(}$ и $\mathtt{)}$, тогава можем да препишем израза от условието на задачата така:

$$\mathtt{f}\big(\mathtt{g}^{\mathbf{H}}(v(\mathtt{x}), \hat{v}(\mathtt{g}\big(\mathtt{a}\mathtt{,}\mathtt{y}\big)))\big)$$

Стойността на този израз в $\mathbf{H}$ е термът $\mathtt{f}(\mathtt{g}(\mathtt{f}(\mathtt{f}(\mathtt{c})), \mathtt{g}(\mathtt{a}, \mathtt{g}(\mathtt{c}, \mathtt{f}(\mathtt{a})))))$. С просто преброяване установяваме, че той съдържа 7 леви скоби и 3 запетаи.

Задача 38. Да бъде една атомарна формула изпълнима в $\mathbf{M}$ означава тя да е вярна при някоя оценка. Ако всички атомарни формули на едно запитване са изпълними, това означава, че има оценки в $\mathbf{M}$, при които те са верни. Да бъде запитването изпълнимо означава атомарните му формули да са верни при една и съща оценка.

Да разгледаме запитването

$$\mathtt{?\text{-}x} > 0,\ \mathtt{x} < 0.$$

Ако универсумът на структурата се състои от числа и символите $<$, $>$ и 0 са интерпретирани по обичайния начин, то това запитване няма да бъде изпълнимо, но поотделно всяка от атомарните му формули ще бъде изпълнима.

Задача 39.

- $\mathtt{?\text{-}p}(\mathtt{c}, X), \mathtt{q}(X, \mathtt{c})$ и $\mathtt{p}(\mathtt{c}, X)$:–
- $\mathtt{?\text{-}p}(\mathtt{c}, Y), \mathtt{q}(Y, \mathtt{c})$ и $\mathtt{p}(\mathtt{c}, Y)$:–
- $\mathtt{?\text{-}p}(\mathtt{c}, \mathtt{f}(\mathtt{f}(Z))), \mathtt{q}(\mathtt{f}(\mathtt{f}(Z)), \mathtt{c})$ и $\mathtt{p}(\mathtt{c}, \mathtt{f}(\mathtt{f}(Z)))$:–
- $\mathtt{?\text{-}p}(X, \mathtt{f}(Y)), \mathtt{q}(\mathtt{f}(Y), X)$ и $\mathtt{p}(X, \mathtt{f}(Y))$:– $\mathtt{p}(Y, X)$

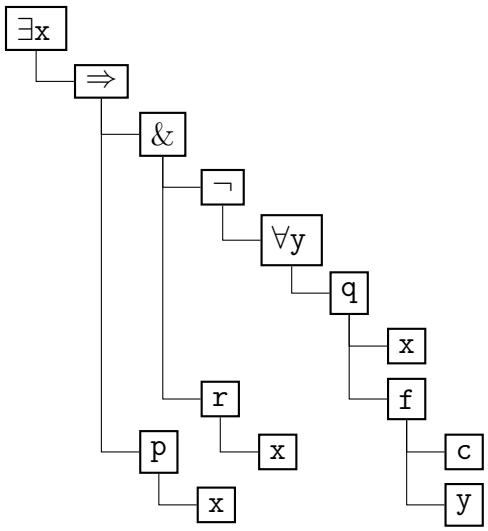
Задача 40. Да допуснем, че φ се удовлетворява от Γ' и Γ' е подмножество на Γ (няма да използваме, че Γ' е крайно). Това означава, че φ е изпълнима във всички структури, в които са верни елементите на Γ' (т.е. във всички модели на Γ').

Ако в някоя структура са верни елементите на Γ, то в нея ще са верни и елементите на Γ' (защото $\Gamma' \subseteq \Gamma$)) и значи в такава структура φ ще бъде изпълнима. По дефиниция това означава, че φ се удовлетворява от Γ.

Задача 41. Нека Δ е множество от частни случаи на Γ (няма да използваме, че Δ е крайно) и да допуснем, че някой частен случай ψ на φ се удовлетворява от Δ. За да докажем, че φ се удовлетворява от Γ, да изберем произволен модел $\mathbf{M}$ на Γ, т.е. структура, в която са тъждествено верни елементите на Γ. Трябва да докажем, че формулата φ е изпълнима в $\mathbf{M}$.

Тъй като частните случаи на тъждествено верни клаузи са тъждествено верни (вж. твърдение 4.24), то елементите на Δ са верни в $\mathbf{M}$. Щом ψ се удовлетворява от Δ и $\mathbf{M}$ е модел на Δ, то формулата ψ е изпълнима в $\mathbf{M}$. Всяка формула, която има изпълним частен случай, е изпълнима (вж. твърдение 4.23), то φ наистина е изпълнима в $\mathbf{M}$.

Задача 42. Вж. фиг. 15.



Фиг. 15. Синтактично дърво за формулата .
$\exists \mathtt{x}\,(\neg\forall \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x}))$

Задача 43. В следващия списък са подчертани подформулите:

$$\underline{\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}))}$$
$$\forall \mathtt{x}\,\underline{(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}))}$$
$$\forall \mathtt{x}\,(\underline{\mathtt{p}(\mathtt{x})} \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \underline{\exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x})})$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \underline{\exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))}\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\underline{\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))}\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \underline{\mathtt{r}(\mathtt{x})})$$

В следващия списък са подчертани термове, които също са са поддървета на синтактичното дърво от фигура 12, но не са подформули, защото са термове:

$$\forall \mathtt{x}\,(\mathtt{p}(\underline{\mathtt{x}}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\underline{\mathtt{x}},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\underline{\mathtt{f}(\mathtt{c},\mathtt{y})})\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\underline{\mathtt{c}},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\underline{\mathtt{y}}))\ \&\ \mathtt{r}(\mathtt{x}))$$
$$\forall \mathtt{x}\,(\mathtt{p}(\mathtt{x}) \Rightarrow \exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\underline{\mathtt{x}}))$$

Задача 44. Тук е подчертана областта на действие на първия квантор:

$$\underline{\forall \mathtt{x}\,(\forall \mathtt{x}\,\exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x},\mathtt{y}))}$$

Тук променливите на първия квантор са подчертани:

$$\forall \underline{\mathtt{x}}\,(\forall \mathtt{x}\,\exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))\ \&\ \mathtt{r}(\underline{\mathtt{x}}) \Rightarrow \mathtt{p}(\underline{\mathtt{x}},\mathtt{y}))$$

Тук е подчертана областта на действие на втория квантор:

$$\forall \mathtt{x}\,(\underline{\forall \mathtt{x}\,\exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))}\,\&\,\mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x},\mathtt{y}))$$

Тук променливите на втория квантор са подчертани:

$$\forall \mathtt{x}\,(\forall \underline{\mathtt{x}}\,\exists \mathtt{y}\,\mathtt{q}(\underline{\mathtt{x}},\mathtt{f}(\mathtt{c},\mathtt{y}))\,\&\,\mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x},\mathtt{y}))$$

Тук е подчертана областта на действие на третия квантор:

$$\forall \mathtt{x}\,(\forall \mathtt{x}\,\underline{\exists \mathtt{y}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\mathtt{y}))}\,\&\,\mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x},\mathtt{y}))$$

Тук променливите на третия квантор са подчертани:

$$\forall \mathtt{x}\,(\forall \mathtt{x}\,\exists \underline{\mathtt{y}}\,\mathtt{q}(\mathtt{x},\mathtt{f}(\mathtt{c},\underline{\mathtt{y}}))\,\&\,\mathtt{r}(\mathtt{x}) \Rightarrow \mathtt{p}(\mathtt{x},\mathtt{y}))$$

Последната променлива y е единствената свободна променлива, а всички други променливи са свързани. Множеството от свободните променливи на тази формула е {y}.

Задача 45. **(а)** Да. Можем да използваме следните еднократни преименувания:

$$\forall \mathtt{x}\,\forall \mathtt{y}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$$
$$\forall \mathtt{x}\,\forall \mathtt{z}\,\forall \mathtt{z}\,\mathtt{p}(\mathtt{x},\mathtt{z})$$
$$\forall \mathtt{x}\,\forall \mathtt{z}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$$

(б) Да. Можем да използваме следните еднократни преименувания:

$$\forall \mathtt{x}\,\forall \mathtt{y}\,\forall \mathtt{y}\,\mathtt{p}(\mathtt{x},\mathtt{y})$$
$$\forall \mathtt{x}\,\forall \mathtt{y}\,\forall \mathtt{z}\,\mathtt{p}(\mathtt{x},\mathtt{z})$$
$$\forall \mathtt{x}\,\forall \mathtt{t}\,\forall \mathtt{z}\,\mathtt{p}(\mathtt{x},\mathtt{z})$$
$$\forall \mathtt{y}\,\forall \mathtt{t}\,\forall \mathtt{z}\,\mathtt{p}(\mathtt{y},\mathtt{z})$$
$$\forall \mathtt{y}\,\forall \mathtt{x}\,\forall \mathtt{z}\,\mathtt{p}(\mathtt{y},\mathtt{z})$$
$$\forall \mathtt{y}\,\forall \mathtt{x}\,\forall \mathtt{x}\,\mathtt{p}(\mathtt{y},\mathtt{x})$$

(в) Не. В първата от формулите първият аргумент на p се свързва от първия квантор, а във втората формула същият аргумент се свързва от втория квантор. Преименуванията на свързани променливи не могат да доведат до подобни изменения.

Задача 46. Тъй като субституцията не променя променливата y, ще обясним какво се случва само с променливите x.

(а) Първата променлива x е свързана и затова не я пипаме. Втората променлива x можем да заменим без проблеми. Отговор: $\forall \mathtt{x}\,\mathtt{p}(\mathtt{x},\mathtt{y}) \vee \mathtt{q}(\mathtt{f}(\mathtt{y}))$.

(б) Ако просто заменим променливата x с f(y) ще получим грешка от втори вид, защото променливата y в f(y) ще се свърже с квантора ∀y . Затова най-напред трябва да преименуваме свързаната променлива y например на z и след това да заменим x с f(y). Отговор: $\forall \mathtt{z}\,\mathtt{p}(\mathtt{f}(\mathtt{y}),\mathtt{z}) \vee \mathtt{q}(\mathtt{y})$. Забележете, че свободната променлива y не се преименува!

Задача 47. Да, ако $\mathtt{x} = \mathtt{y}$ и $\mu \neq \nu$.

Приложение B

Задължителни неща

На изпита по логическо програмиране се очаква всички студенти да са научили нещата от списъка, даден по-долу. Познаването само на тези неща не гарантира получаването висока оценка. Не е нужно да се помнят условията на задачите, включени в този списък, но трябва да може да се решава всяка задача с подобно условие.

При твърденията, които трябва да се знаят без доказателство, от студентите се очаква да се сещат сами за съответното твърдение, ако то е нужно за решаването на някоя задача. Да разгледаме например следната задача:

> Нека структурата $\mathbf{M}$ е с универсум множеството на естествените числа, а оценките v и w в $\mathbf{M}$ са такива, че $v(\mathtt{x}) = 5$, $v(\mathtt{y}) = 8$, $v(\mathtt{z}) = 3$, $w(\mathtt{x}) = 7$, $w(\mathtt{y}) = 8$, $w(\mathtt{z}) = 9$. Да се докаже, че стойността на формулата
>
> $$\forall \mathtt{x}\, \mathtt{p}(\mathtt{x}, \mathtt{y}) \Rightarrow \exists \mathtt{z}\, \forall \mathtt{x}\, (\neg \mathtt{q}(\mathtt{z}, \mathtt{x}, \mathtt{y}) \,\&\, \mathtt{p}(\mathtt{x}, \mathtt{y}))$$
>
> в $\mathbf{M}$ при оценка v е еквивалентна на стойността ѝ в $\mathbf{M}$ при оценка w.

От студентите се очаква да могат да дадат приблизително следното решение:

Променливата $\mathtt{y}$ е единствената свободна променлива в тази формула. Тъй като $v(\mathtt{y}) = w(\mathtt{y}) = 8$, то исканата еквивалентност следва от следното твърдение:

Твърдение. *Нека v и v' са оценки в структура $\mathbf{M}$. Ако v и v' съвпадат за всички свободни променливи на формулата φ, то $\mathbf{M} \vDash \varphi[v]$ е еквивалентно на $\mathbf{M} \vDash \varphi[v']$.*

Не е нужно твърденията да могат да се цитират точно по начина, по който са били дадени на лекциите или са формулирани в записките. Разбира се, твърденията, които се цитират трябва да са верни, а ако има съществени разлики между цитираното твърдение, и твърдението, формулирано в записките, тогава се очаква при поискване формулираното твърдение да може да се докаже.

- дефиниция 2.4
- уговорка 2.1
- задача 3
- задача 4
- задача 8
- дефиниция 3.2 б)
- уговорка 3.5
- задача 16
- дефиниция 3.2 в)
- дефиниция 3.10
- дефиниция 3.14
- дефиниция 3.15
- дефиниция 3.16
- дефиниция 3.20 а)
- задача 21
- дефиниция 3.26
- задачи 22 и 23
- твърдение 3.27 с доказателство
- дефиниция 3.28
- задача 17
- дефиниция 3.30

– дефиниция 3.32
– задача 26
– дефиниция 3.34
– дефиниции 4.2 а) и 4.2 б)
– дефиниция 4.5
– задача 29
– коментар 4.13
– теореми 4.27, 4.39 и 4.40 без доказателство
– следствие 4.20 с доказателство
– твърдение 4.23 с доказателство
– твърдение 4.24 с доказателство
– дефиниция 4.29
– дефиниция 4.30
– следствие 4.33 без доказателство
– задача 37
– дефиниция 4.41
– дефиниция 4.42
– дефиниция 4.44
– дефиниция 4.46
– задача 39
– дефиниция 4.49
– теореми 4.50 и 4.53 без доказателство
– таблица 2 от раздел 5.1
– задача 42
– дефиниция 5.5
– дефиниция 5.6 и коментар 5.7
– задача 43
– дефиниция 5.12, примери 5.9, 5.10, 5.11 и задача 44
– дефиниция 5.14 в)
– твърдения 5.15 и 5.16 и лема 5.17 без доказателство
– коментари 5.18 и 5.19
– задача 46

– дефиниция 5.31
– дефиниция 5.32
– дефиниции 5.33 а), 5.33 б) и 5.33 в)
– дефиниция 5.33 г) и твърдение 5.34 без доказателство
– твърдение 5.36 без доказателство
– твърдение 5.39 без доказателство
– твърдение 5.40 без доказателство
– дефиниция 5.56
– твърдение 5.57 с доказателство
– дефиниция 5.61
– твърдение 5.62 с доказателство
– твърдение 5.64 с доказателство
– твърдение 5.66 без доказателство
– дефиниция 5.79
– твърдение 5.80 с доказателство
– твърдение 6.2 с доказателство
– дефиниция 6.3
– твърдение 6.7 с доказателство
– дефиниция 6.8 б)
– твърдение 6.9 без доказателство
– дефиниция 6.10 б)
– твърдение 6.11 с доказателство
– дефиниция 6.12
– теорема 6.14 без доказателство
– теорема за пълнота на либералната резолюция без доказателство

Библиография

> Нека се занимаем сега със списъка на автори, какъвто има в други книги и какъвто липсва във вашата. Лек за това се намира много лесно, ще трябва само да намерите книга, която да съдържа пълен списък с автори от първата до последната буква на азбуката, както сам вие казахте. Същия този азбучен списък поставете във вашата книга и не се тревожете никак дори ако се открие измамата, поради това, че вие не сте имали никаква нужда да черпите знания от тези автори. Все ще се намери някой наивен читател, който да повярва, че сте използвали всички тези автори за вашата проста и обикновена история. Накрая, ако този дълъг поменик от автори не послужи за друго, то поне ще даде тежест на вашата книга. Още повече че никой не ще седне да проверява дали наистина сте чели, или не тези автори, защото от това няма да му стане нито по-топло, нито по-студено.
>
> Мигел де Сервантес Сааведра [24]

[1] John Barnes. *Spark: The Proven Approach to High Integrity Software*. Altran Praxis, http://www.altran.co.uk, UK, 2012.

[2] J.A. Bergstra and J.V. Tucker. A characterisation of computable data types by means of a finite, equational specification method. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 1980.

[3] J.A. Bergstra and J.V. Tucker. The completeness of the algebraic specification methods for computable data types. *Information and Control*, 54(3):186 – 200, 1982.

[4] Garrett Birkhoff. On the structure of abstract algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 31:433–454, 10 1935.

[5] Chin-Liang Chang and Richard C. T. Lee. *Symbolic logic and mechanical theorem proving.* Computer science classics. Academic Press, 1973.

[6] H. B. Curry and R. Feys. *Combinatory Logic, Volume I.* North-Holland, 1958. Second printing 1968.

[7] P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM J. Res. Dev.*, 4(1):28–35, January 1960.

[8] L. L. Ivanov. *Algebraic recursion theory.* Ellis Horwood series in mathematics and its applications: Statistics and operational research. E. Horwood, 1986.

[9] Robert A. Kowalski. Predicate logic as programming language. In *IFIP Congress*, pages 569–574, 1974.

[10] Michael G. Main and David B. Benson. Free semiring-representations and nondeterminism. *Journal of Computer and System Sciences*, 30(3):318 – 328, 1985.

[11] Conor McBride. Faking it simulating dependent types in haskell. *J. Funct. Program.*, 12(5):375–392, July 2002.

[12] I. McGilchrist. *The Master and His Emissary: The Divided Brain and the Making of the Western World.* Yale University Press, 2009.

[13] Dauda Odunayo Olanloye. An expert system for diagnosing faults in motorcycle. *International Journal of Engineering and Applied Sciences*, 5(06), 2014.

[14] John C. Reynolds. *Theories of programming languages.* Cambridge University Press, 1998.

[15] Dorothy Sayers. The psychology of advertising. *The Spectator*, pages 896–898, 1937.

[16] Dimiter Skordev. *Computability in combinatory spaces. An algebraic generalization of abstract first order computability.* Kluwer Academic Publishers, Dordrecht-Boston-London, 1992.

[17] Ivan Soskov. On the computational power of the logic programs. In P. Petkov, editor, *Heyting'88 (Varna)*, pages 117–137. Plenum Press, 1990.

[18] Leon Sterling and Ehud Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT Press, Cambridge, MA, USA, 1986.

[19] Robert D. Tennent. *Semantics of programming languages*. Prentice Hall International Series in Computer Science. Prentice Hall, 1991.

[20] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, October 1976.

[21] Eliezer Yudkowsky. Cognitive biases potentially affecting judgment of global risks. In Nick Bostrom and Milan Cirkovic, editors, *Global Catastrophic Risks*. Oxford University Press, August 2006.

[22] Анатолий Иванович Мальцев. Несколько замечаний о квазимногообразиях алгебраических систем. *Алгебра и логика (семинар) 5*, 3:3–9, 1966.

[23] Анатолий Иванович Мальцев. *Алгебраические системы*. Современная алгебра. Наука, 1970.

[24] Мигел де Сервантес Сааведра. *Знаменитият идалго Дон Кихот де ла Манча*. Библиотека за ученика. Държавно издателство „Отечество“, София, 1986. Четвърто съкратено издание. Превод на Тодор Нейков.

[25] Димитър Скордев. *Комбинаторные пространства и рекурсивность в них*. Изд. на БАН, 1980.

[26] Иван Сосков. Пример полного по Московакису базиса, которы не является программного полным. In А. П. Ершов, editor, *Математическая теория и практика систем программного обеспечения (Новосибирск)*. ВЦ СОАН, 1982.

Именен указател

Предметен указател