

Теорија типова и унификација типова

Борисав Живановић

Увод

- ▶ Теорија типова је област математике и рачунарства која се бави формалним представљањем система типова
- ▶ Систем типова је формални (логички) систем правила који појмовима програмског језика додељује својство звано тип
 - ▶ Појам може да буде било шта, у зависности од програмског језика
 - ▶ Ми ћемо изучавати императивне програмске језике, те су наши појмови: литерали, изрази, функције, кориснички дефинисани типови
- ▶ Систем типова је настао као један од првих покушаја аутоматске провере исправности кода
- ▶ Да бисмо ово разумели, потребан је кратак осврт на историју програмских језика

Шта је рачунар?

*Рачунар је машина коју је могуће испрограмирати да изврши низ **аритметичких и логичких операција** (израчунавања) аутоматски.*

Шта рачунар заиста зна да ради?

- ▶ Језик рачунара: **скуп инструкција** (енгл. ISA, Instruction Set Architecture)
- ▶ Аритметичке операције: **add, sub, div, mul, ...**
- ▶ Померање података:
 - ▶ са улазног уређаја у меморију
 - ▶ из меморије на излазни уређај
 - ▶ са једне меморијске локације на другу
- ▶ Условно гранање: извршавање кода уколико је логички услов испуњен

Шта је програм?

*Рачунарски програм је **низ инструкција** садржаних у формату који рачунар може да **изврши**.*

Како рачунари омогућавају аутоматизацију процеса?

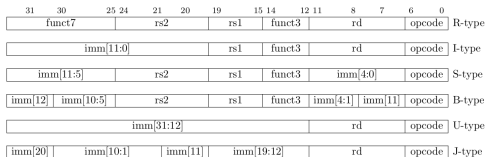
- ▶ Неопходно је да имамо формалну дефиницију процеса који желимо да аутоматизујемо - **морамо да дефинишемо алгоритам**
 - ▶ сама дефиниција мора бити формална, односно мора садржати прецизан опис корака
 - ▶ формат дефиниције не мора да буде формалан!
- ▶ Формалну дефиницију морамо изразити у формату који рачунар може да изврши - **морамо да имплементирамо алгоритам**
- ▶ У пракси, грешке у дизајну и имплементацији су честе - **морамо да тестирамо програм**

Како је могуће описати алгоритам?

- ▶ Очигледно је да је неопходно да формат буде разумљив рачунару
- ▶ Пожељно је да формат буде разумљив и људима
 - ▶ бржа имплементација, мање грешака, мање документације
- ▶ Још боље: аутоматска провера исправности програма
- ▶ Из овога је настала потреба за програмским језицима (и програмским преводиоцима)
- ▶ Програмски језици се класификују у 4 (по неким 5) генерација

I генерација

- ▶ Ручно уношење инструкција и података у бинарном формату
- ▶ Којим грешкама је ово подложно?



II генерација

- ▶ Инструкције су представљене својим симболичким називом
- ▶ Како побољшање ово представља?
- ▶ Који недостаци су и даље присутни?
- ▶ Које типове уочавамо?

nop	addi x0, x0, 0	No operation
li rd, immediate	<i>Myriad sequences</i>	Load immediate
mv rd, rs	addi rd, rs, 0	Copy register
not rd, rs	xori rd, rs, -1	One's complement
neg rd, rs	sub rd, x0, rs	Two's complement
negw rd, rs	subw rd, x0, rs	Two's complement word
sext.w rd, rs	addiw rd, rs, 0	Sign extend word
seqz rd, rs	sltiu rd, rs, 1	Set if = zero
snez rd, rs	sltu rd, x0, rs	Set if ≠ zero
sltz rd, rs	slt rd, rs, x0	Set if < zero
sgtz rd, rs	slt rd, x0, rs	Set if > zero
fmv.s rd, rs	fsgnj.s rd, rs, rs	Copy single-precision register
fabs.s rd, rs	fsgnjx.s rd, rs, rs	Single-precision absolute value
fneg.s rd, rs	fsgnjn.s rd, rs, rs	Single-precision negate
fmv.d rd, rs	fsgnj.d rd, rs, rs	Copy double-precision register
fabs.d rd, rs	fsgnjx.d rd, rs, rs	Double-precision absolute value
fneg.d rd, rs	fsgnjn.d rd, rs, rs	Double-precision negate
beqz rs, offset	beq rs, x0, offset	Branch if = zero
bnez rs, offset	bne rs, x0, offset	Branch if ≠ zero
blez rs, offset	bge x0, rs, offset	Branch if ≤ zero
bgez rs, offset	bge rs, x0, offset	Branch if ≥ zero
bltz rs, offset	blt rs, x0, offset	Branch if < zero
bgtz rs, offset	blt x0, rs, offset	Branch if > zero
bgt rs, rt, offset	blt rt, rs, offset	Branch if >
ble rs, rt, offset	bge rt, rs, offset	Branch if ≤
bgtu rs, rt, offset	bltu rt, rs, offset	Branch if ≥, unsigned
bleu rs, rt, offset	bgeu rt, rs, offset	Branch if ≤, unsigned

III генерација

- ▶ Структура програма слична стаблу
- ▶ Ограничен приступ меморији
- ▶ Ограничена слобода у условном гранању
- ▶ Подела на исказе и изразе

```
1  #include "myMult.h"
2
3  void myMult(const double a[12], const double b[20], double c[15])
4  {
5      int i0;
6      int i1;
7      int i2;
8      for (i0 = 0; i0 < 3; i0++) {
9          for (i1 = 0; i1 < 5; i1++) {
10             c[i0 + 3 * i1] = 0.0;
11             for (i2 = 0; i2 < 4; i2++) {
12                 c[i0 + 3 * i1] += a[i0 + 3 * i2] * b[i2 + (i1 << 2)];
13             }
14         }
15     }
16 }
```

Теорија и пракса

- ▶ На најнижем нивоу апстракције, тип представља бинарни формат и правила за његово тумачење
- ▶ Ограничење је потребно како би се извршавао искључиво код који уме да интерпретира садржај на исправан начин
- ▶ На вишем нивоу апстракције, тип представља скуп дозвољених вредности и дозвољених операција
- ▶ Ограничење је потребно како би се извршавале операције искључиво над семантички компатибилним ентитетима

Мутабилност I

- ▶ Мутабилност је појам који постоји у рачунарству, али не постоји у математици
- ▶ Математика познаје само вредности
- ▶ Вредности могу да припадају скуповима и променљиве могу да имају одређену вредност
- ▶ Вредности које припадају скупу су унапред задате дефиницијом скупа
- ▶ Међутим, природа **променљивости** саме вредности није дефинисана!
- ▶ Сматра се да је сама вредност **целовита** и **непроменљива**, док је могуће да променљива има различите вредности!

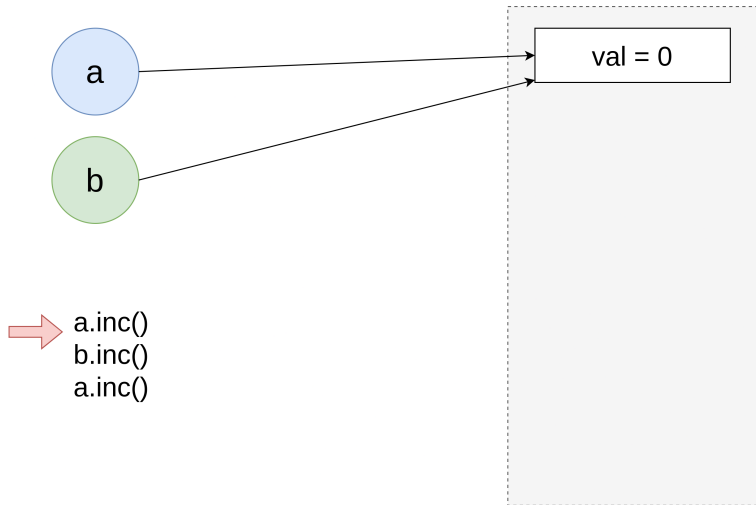
Мутабилност II

- ▶ Рачунарство такође познаје појам променљиве и вредности, али уводи и једно својство које сведочи о променљивости саме вредности
- ▶ Речником рачунарства, вредности у математици су имутабилне
- ▶ Више променљивих може да показује на исту вредност (показивачи), због чега измена саме вредности постаје видљива преко различитих променљивих!
- ▶ У рачунарству постоји ограничење задато хардвером (и у крајњој граници, законима физике) које захтева увођење оваквог својства

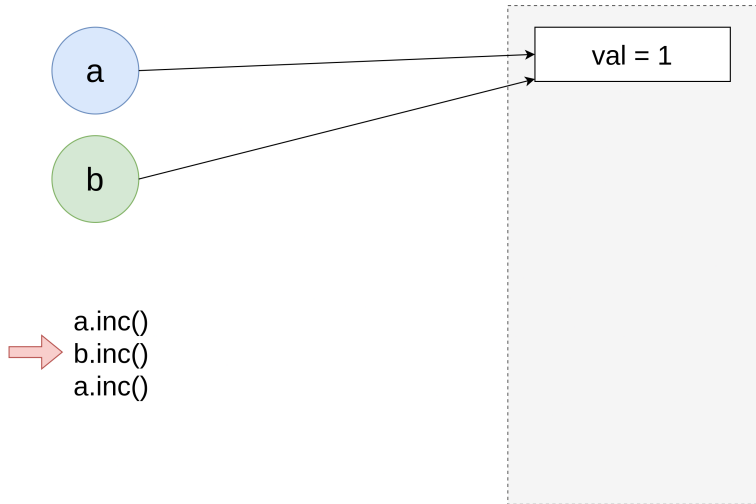
Мутабилност III

- ▶ Прости типови заузимају мало простора, због чега је њихово складиштење на стеку и у регистрима једноставно
- ▶ Креирање копија вредности простих типова је једноставно и брзо
- ▶ Сложени типови заузимају далеко више простора и најчешће се складиште на хипу
- ▶ Стање мутабилних објеката је могуће мењати и након креирања
- ▶ Мутабилност побољшава перформансе, али уноси непредвидивост
 - ▶ имате ли идеју како?
- ▶ Мутабилност утиче на дизајн система типова!
 - ▶ више речи о овоме нешто касније

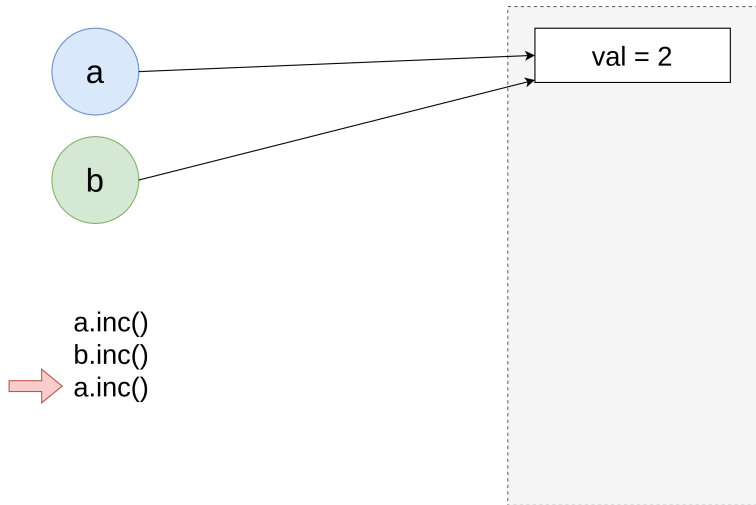
Мутабилни објекти



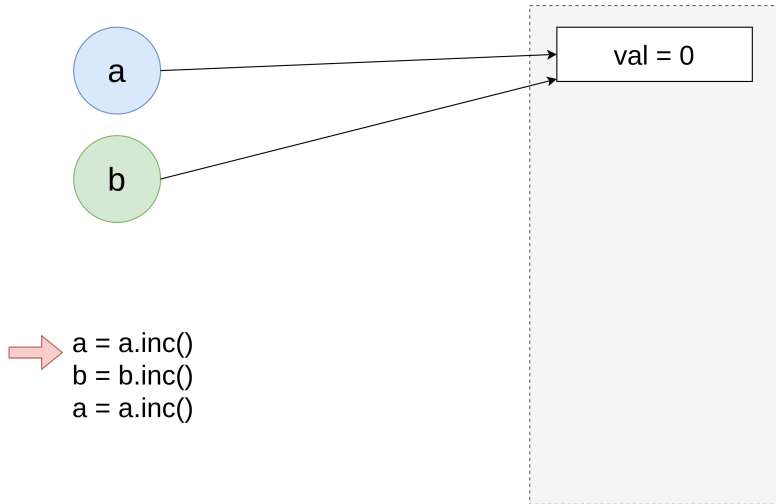
Мутабилни објекти



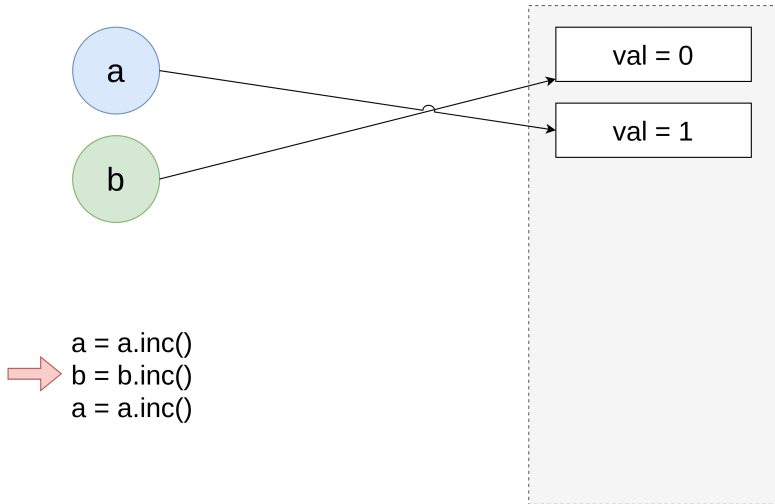
Мутабилни објекти



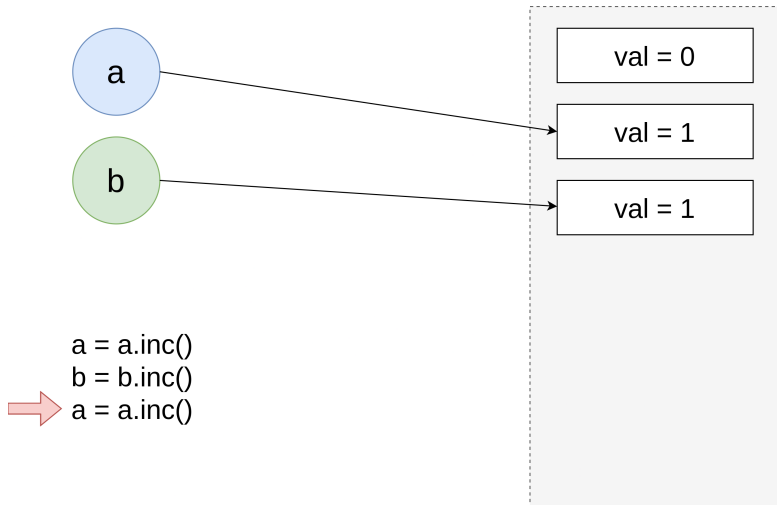
Имутабилни објекти



Имутабилни објекти



Имутабилни објекти



Теорија типова и теорија скупова

- ▶ Једнакост типова осигурава исправност програма
- ▶ Да ли отежава писање програма?
 - ▶ у стварном свету, уочавамо сличност између различитих појмова и облика
 - ▶ некада су ти појмови довољно слични да можемо да занемаримо разлике
 - ▶ пример: потребан нам је аутомобил, али нас не занима произвођач
- ▶ Тип је појам сродан скупу
- ▶ Ако постоје подскупови, да ли постоје и подтипови?
- ▶ Шта описују подскупови, а шта би описивали подтипови?
- ▶ Релација подтипа је слична релацији подскупа!

Конвертибилност типова (релација подтипа)

- ▶ Кажемо да је $A \leq B$ уколико је A конвертибилно у B
 - ▶ $A \leq B$ (**рефлексивност**)
 - ▶ $A \leq B \wedge B \leq C \Rightarrow A \leq C$ (**транзитивност**)
 - ▶ $A \leq B \wedge B \leq A \Rightarrow A = B$ (**антисиметричност**)
- ▶ Релација подтипа је релација парцијалног поретка!

Шта одређује конвертибилност типова? I

- ▶ Правила која дефинишу конвертибилност типова су одлука дизајнера система типова
- ▶ Главни водич је тип A мора да садржи све вредности које подржава B као и да приликом имплицитне конверзије не долази до губитка података
 - ▶ Релативан појам: скуп целих бројева је подскуп скупа реалних бројева у математици, док је у програмирању могућ губитак приликом претварања целобројне вредности у вредност са покретним зарезом
 - ▶ Неки програмски језици ово игноришу, док други ово сматрају за грешку и захтевају експлицитну конверзију целобројног типа
- ▶ Додатно: мутабилност не сме да изазове грешке приликом извршавања програма

Шта одређује конвертибилност типова? II

- ▶ За просте типове, конвертибилност је дефинисана правилима система типова
- ▶ За сложене типове, конвертибилност је релацијом између сложених типова (коју задаје корисник) и/или у односу на садржај (правила дефинише систем типова)
 - ▶ више речи о овоме нешто касније

Закључивање типова

- ▶ До сада смо разумели појам типа, система типова и релације подтипа
- ▶ Како можемо да стечено знање употребимо за решавање полазног проблема: одређивање исправности израза?
- ▶ Као и сваки формални систем, и систем типова се састоји од аксиома и правила
- ▶ Идеја: типови простих израза (литерали и променљиве) су познати (аксиоми), а тип сложеног израза је могуће закључити уколико су подизрази одговарајућих типова (правила)

Правила

- ▶ Бинарни изрази:
 - ▶ оба подизраза морају да имају заједнички тип у који су конвертибилни како би операција била могућа
 - ▶ резултат бинарне аритметичке операције је заједничког типа
 - ▶ резултат бинарне логичке или релацине операције је булова вредности
- ▶ Позив функције:
 - ▶ евалуација позива функције враћа вредност типа повратног типа функције
 - ▶ шта је још потребно да би позив био могућ?

Сложени типови

- ▶ До сада смо разумели просте типове као и њихову примену
- ▶ Уочавамо потребу за креирањем сложених типова
 - ▶ једноставан пример: желимо обраду над скупом простих типова
 - ▶ напреднији пример: желимо да ентитете из стварног света представимо у програмима, уз задржавање правила за аутоматску проверу исправности
 - ▶ додатно: постоји потреба да ентитете програмског језика (попут функција) опишемо типом, како би могли да их обрађујемо на исти начин као и корисничке типове
- ▶ Како бисмо могли да креирамо овакве типове?

Конструктор типа

- ▶ Конструктор типа омогућава креирање новог типа користећи претходно дефинисане типове
- ▶ Подсетник: систем типова дефинише основне типове
- ▶ Додатно: систем типова дефинише конструкторе типова
- ▶ Омогућено је произвољно комбиновање типова без обзира на контекст
 - ▶ аксиоми и правила система типова омогућавају проверу исправности употребе у односу на релацију подтипа
- ▶ Релација подтипа је дефинисана само за типове који су инстанцирани од сродних конструктора типова
 - ▶ физика: није могуће поредити килограме и километре
 - ▶ народски речено: не треба да се мешају бабе и жабе

- ▶ Низови су најједноставнији пример сложеног типа
- ▶ У пракси, честа је потреба за обрадом колекције података
- ▶ Желимо да спречимо складиштење произвољних вредности како би омогућили униформну обраду
- ▶ Да ли је услов за униформну обраду једнакост типова ($=$) или релација подтипа (\leq)?
- ▶ Можемо ли да упоредимо два типа низова?

Низови II

- ▶ У низ $t : T[]$ можемо да ускладишtimo вредност $x : X$ уколико $X \leq T$
 - ▶ T представља горњу границу типа вредности у низу!
- ▶ У променљиву $y : A[]$ можемо да ускладишtimo низ $x : B[]$ уколико је $B \leq A$
 - ▶ тип низа је коваријантан у односу на тип T
 - ▶ да ли морамо да водимо рачуна и о дужини низа?

Коваријантност типова

Сложени тип $A\langle T \rangle$ је коваријантан у односу на тип параметра T уколико важи $A\langle X \rangle \leq A\langle Y \rangle$ за $X \leq Y$

Варијантност и мутабилност I

- ▶ Имутабилни низови су очигледно коваријантни у односу на T
- ▶ Проблем: шта се дешава уколико су низови мутабилни?

Варијантност и мутабилност II

$$A \leq B \leq C$$

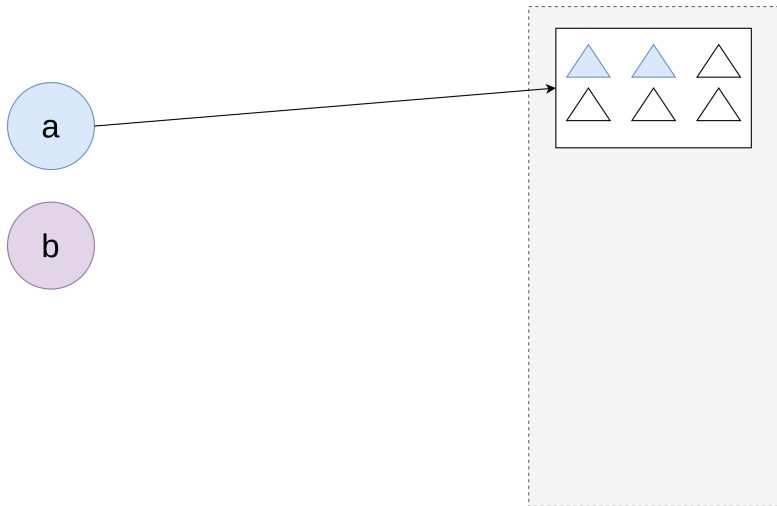
$$a : A[] = [A, A, A]$$

$$b : B[] = a$$

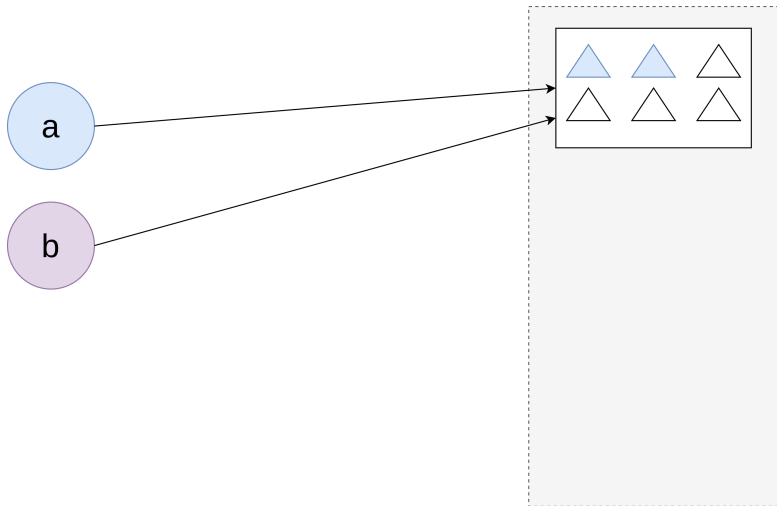
$$b.add(B)$$

Грешка: није могуће сачувати вредност
У објекту типа A јер не важи $B \leq A$!

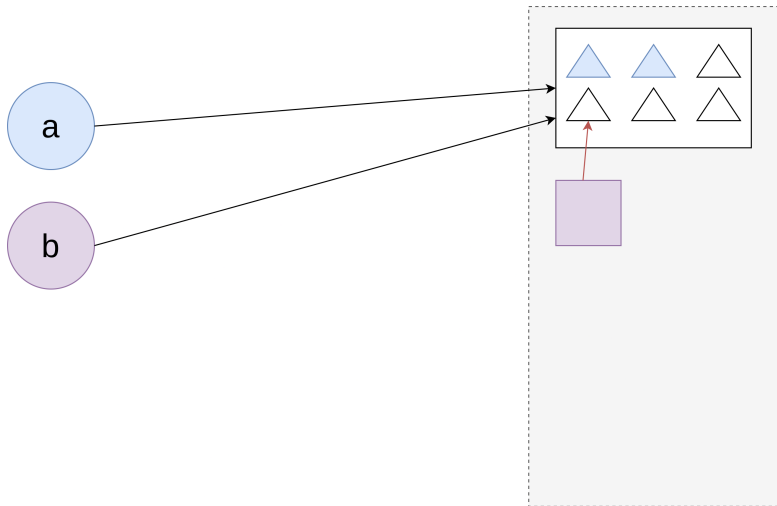
Варијантност и мутабилност III



Варијантност и мутабилност IV



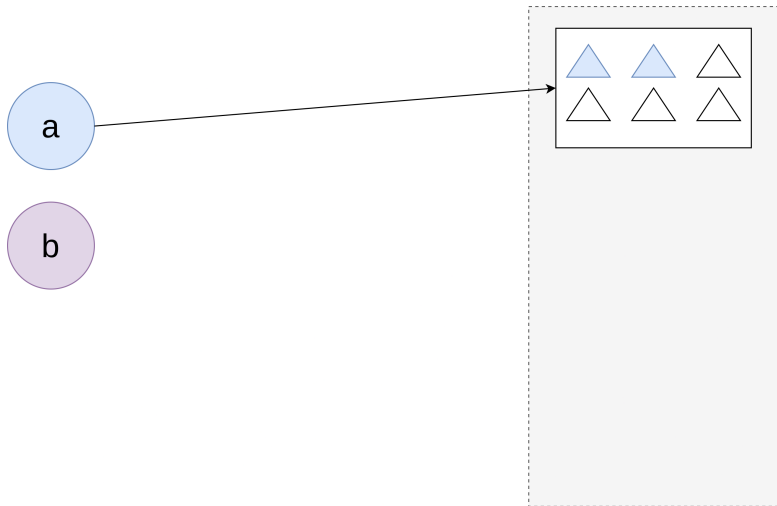
Варијантност и мутабилност V



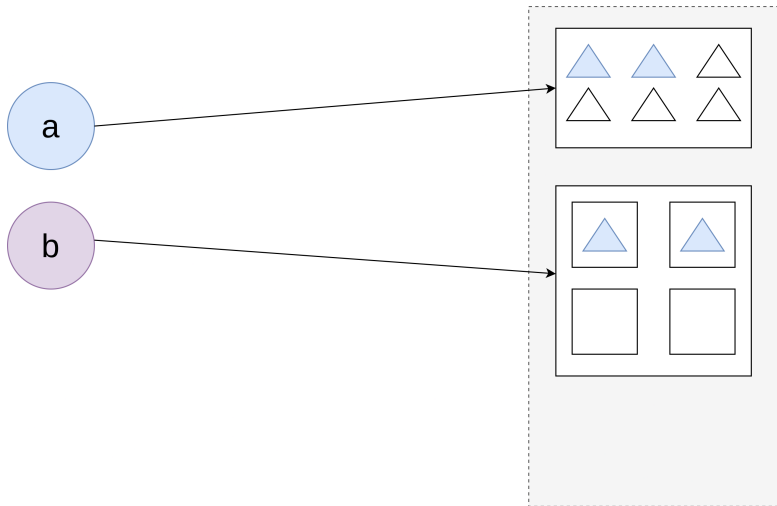
Варијантност и мутабилност VI

- ▶ Дошло је до грешке приликом извршавања кода
- ▶ Систем типова је то морао да спречи!
- ▶ Могућа решења:
 - ▶ обавезно копирање низа (имутабилност)
 - ▶ инваријантност у односу на T

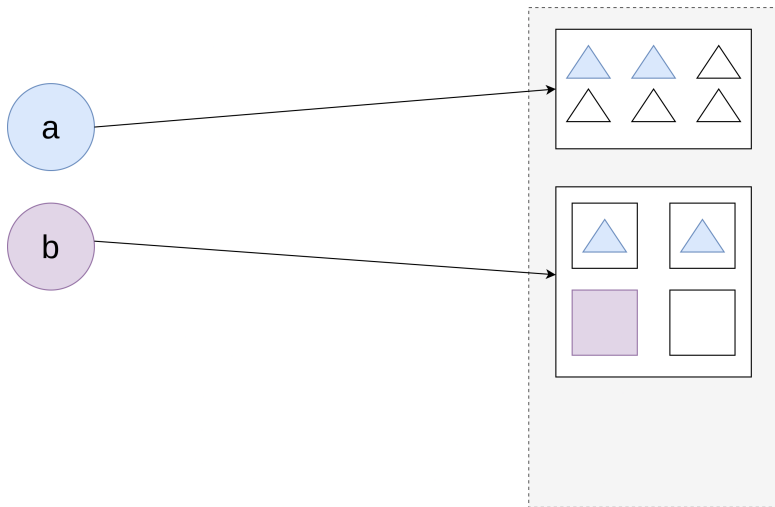
Варијантност и мутабилност VII



Варијантност и мутабилност VIII



Варијантност и мутабилност IX



Структуре

- ▶ Структура садржи именоване вредности (поља) чији тип може бити произвољан тип, укључујући и саму структуру из дефиниције (рекурзија је дозвољена)
- ▶ Које услове би требало да задовољи структура $s : B$ како би важило $B \leq A$?
- ▶ Проблем: мутабилност
 - ▶ мутабилне структуре такође морају да буду инваријантне у односу на типове поља
 - ▶ имутабилне структуре могу да буду варијантне
- ▶ Потребно је обратити пажњу на називе поља и њихове типове
- ▶ Мишљења о томе како би ту пажњу требало обратити су подељена

Номинални системи типова

У номиналном систему типова, релације између сложених типова су задате приликом конструкције типова употребом ознака типова.

Структурални системи типова

У структуралном систему типова, релације између сложених типова се одређују приликом употребе типа, поређењем садржаја очекиваног и употребљеног типа.

Структуре (наставак) I

- ▶ Интуиција: уколико је вредност типа B могуће сачувати у $a : A$ уколико важи $B \leq A$, да ли је правило могуће генерализовати и применити на поља структуре?
- ▶ Структуре су коваријантне у односу на тип појединачних поља
 - ▶ поредак важи искључиво за иста поља (поља истог имена)
- ▶ Иста општа правила важе и у номиналним и у структуралним системима типова
- ▶ Различит је начин на који се провера спроводи

Структуре (наставак) II

- ▶ У номиналним системима типова, приликом конструкције типова се проверава да ли поља задовољавају релације у складу са осталим релацијама подтипа које је корисник задао
- ▶ У структуралним системима типова, приликом употребе типа у одређеном контексту се проверава да ли поља задовољавају релације у складу са релацијом између доступног и траженог типа која би требало да буде задовољена
- ▶ У пракси, системи типова често комбинују елементе оба приступа

Функције I

- ▶ Тип функције је сложени тип који се састоји од типа параметера и типа повратне вредности
- ▶ Не треба мешати тип функције и тип повратне вредности функције!
- ▶ У језицима у којима функције представљају *грађане првог реда*, функције је могуће чувати у променљивама и вратити као тип израза
- ▶ Како можемо да дефинишемо релацију поретка?
- ▶ Да ли је тип функције боље посматрати и описивати номиналним или структуралним приступом?

Функције II

- ▶ Интуиција: враћање вредности је једнако додели, типови функција су коваријантни у односу на тип повратне вредности
- ▶ Да ли на исти начин можемо да посматрамо и типове аргумената?
- ▶ Може ли интуиција да нас превари?

Функције III

- ▶ Прослеђивање аргумената (конкретне вредности које се додељују параметрима) приликом позива функције је такође једнако додели вредности
- ▶ Нека су A и B типови функција, а a и b променљиве
- ▶ Нека су типови свих параметара B подтипови параметара A
- ▶ Покушајмо да B доделимо у a и извршимо позив функције

Функције IV

X

$Y \leq X$

$Z \leq X$

$A : (p : X) \rightarrow X$

$B : (p : Y) \rightarrow Z$

$a : A = \text{some_}B_func$

$a(\text{new } Z)$

Грешка!

$\text{some_}B_func$ очекује $type(p) \leq Y$, али A дозвољава $type(p) \leq X$!

Тип Z не задовољава услов задат од стране $\text{some_}B_func$!

Функције V

- ▶ Интуиција (други покушај): како би спречили прослеђивање типа са којим функција не може да ради, неопходно је да аргументи подтипа функције буду у \geq релацији у односу на аргументе надтипа функције
- ▶ Важи правило које је супротно од коваријантности

Контраваријантност типова

Сложени тип $A\langle T \rangle$ је контраваријантан у односу на тип параметра T уколико важи $A\langle X \rangle \leq A\langle Y \rangle$ за $X \geq Y$

Функције (други покушај)

 X $Y \leq X$ $Z \leq X$ $Q \leq Y$ $A : (p : Y) \rightarrow X$ $B : (p : X) \rightarrow Z$ $a : A = \text{some_}B_func$ $a(\text{new } Q)$

Релација подтипа је транзитивна!

За сваки тип T за који важи $T \leq A$ такође важи $T \leq B$ уколико $A \leq B$ (односно $B \geq A$).

Функције (закључак)

- ▶ Функције су коваријантне у односу на тип повратне вредности и контраваријантне у односу на типове параметара
- ▶ Вредност функције је заправо програмски код који она садржи
- ▶ Како ово није могуће у току извршавања програма, можемо функције да сматрамо имутабилним
- ▶ У теорији, можемо да слободно користимо варијантност у релацији подтипа
- ▶ У пракси, програмски језици често сматрају функције инваријантним у односу на типове параметара
- ▶ Овиме се олакшава имплементација, а исправност се не крши јер је инваријантност *строжа* од варијантности!
- ▶ Доста програмских језика погрешно сматра функције коваријантним у односу на типове параметара!

Генерички типови

- ▶ До сада смо баратали искучиво са унапред познатим типовима
- ▶ Релација подтипа нам је давала одређену слободу да не морамо да знамо све детаље о коришћеним типовима
- ▶ Генеричко програмирање омогућава опис алгоритама који раде над типовима који су накнадно дефинисани
- ▶ Најчешћа примена: опште структуре података морају да омогуће складиштење свих корисничких типова, уз правило да се у инстанци генеричке колекције не мешају типови који нису компатибилни
 - ▶ због чега релација подскупа није употребљива за ову проверу?

Конструкција типова (наставак)

- ▶ Уводи се ниво индирекције у конструкцији типова
- ▶ Дефиниција типа креира апстрактни тип
- ▶ Конструктор генеричког типа садржи параметре типова
- ▶ Корисник дефинише ограничења над параметрима (у виду релације подтипа)
- ▶ Додатно: корисник дефинише варијантност у односу на тип параметра
- ▶ Сви до сада наведени сложени типови могу да буду генерички типови!

Унификација типова I

- ▶ Приликом креирања конкретног типа из апстрактног, корисник може да проследи произвољан тип
- ▶ Потребно је проверити да ли типови задовољавају ограничења која задаје апстрактни тип
- ▶ Додатно: потребно је доделити конкретне типове параметара

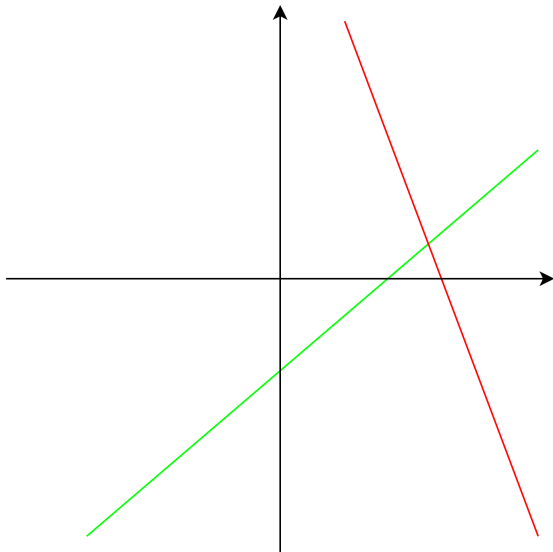
Унификација типова II

- ▶ Уколико систем типова не подржава релацију подтипа, поступак је једнак решавању система једначина
- ▶ Уколико је систем одређен, унификација је успешна
- ▶ Уколико је систем неодређен или немогућ, унификација је неуспешна

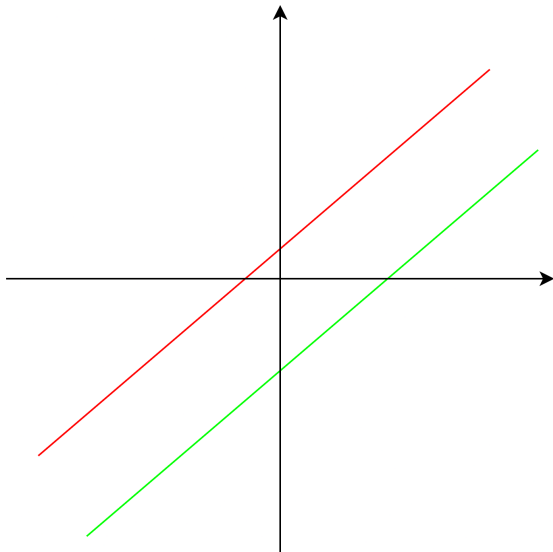
Унификација типова III

- ▶ Систем типова дефинише под којим условима је могуће унификовати два типа
 - ▶ подсетник: номинални и структурални системи различито посматрају једнакост типова и релацију подтипа
 - ▶ додатни подсетник: бабе и жабе
- ▶ Опште правило:
 - ▶ два проста типа је могуће унификовати уколико су једнаки
 - ▶ два сложена типа је могуће унификовати уколико су једнаки и уколико је могуће унификовати све типове од којих се састоје
- ▶ Поступак се примењује док не остану искључиво једначине познатих простих типова и параметара и простих типова
- ▶ Методом замене решавамо зависности између параметара

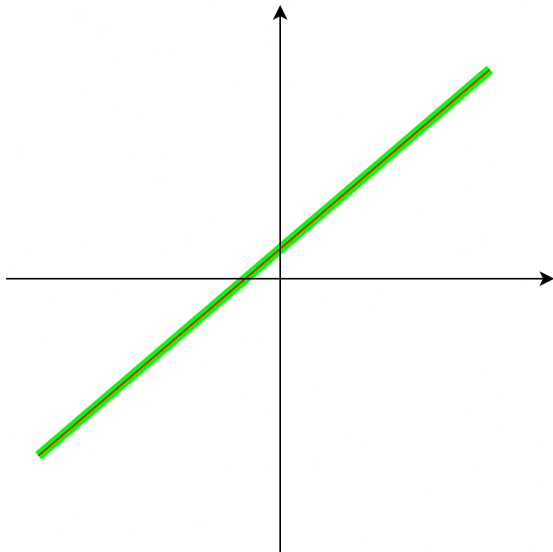
Одређен систем једначина



Немогућ систем једначина



Неодређен систем једначина



Унификација типова (поступак) I

A, B, C

$List<T>$

$Map<K, V>$

$func : (a : T1, b : T2, c : List<T1>, d : List<T2>) \rightarrow$
 $Map<T1, T2>$

$a = A$

$b = B$

$c = List<A>$

$d = List$

$x = func(a, b, c, d)$

$type(x) = ?$

Унификација типова (поступак) II

$$A = T1$$

$$B = T2$$

$$List<A> = List<T1>$$

$$List = List<T2>$$

Унификација типова (поступак) III

$$T1 = A$$

$$T2 = B$$

$$List = List$$

$$A = T1$$

$$List = List$$

$$B = T2$$

Унификација типова (поступак) IV

$$T1 = A$$

$$T2 = B$$

Систем је одређен, унификација је успешна!

Унификација типова (поступак) V

A, B, C

$List<T>$

$Map<K, V>$

$func : (a : T1, b : T2, c : List<T1>, d : List<T2>) \rightarrow$
 $Map<T1, T2>$

$a = A$

$b = B$

$c = List<A>$

$d = List<A>$

$x = func(a, b, c, d)$

$type(x) = ?$

Унификација типова (поступак) VI

$$A = T1$$

$$B = T2$$

$$List<A> = List<T1>$$

$$List<A> = List<T2>$$

Унификација типова (поступак) VII

$$T1 = A$$

$$T2 = B$$

$$List = List$$

$$A = T1$$

$$List = List$$

$$A = T2$$

Унификација типова (поступак) VIII

$$T1 = A$$

$$T2 = B, T2 = A$$

Систем је немогућ, унификација је неуспешна!

Унификација типова (поступак) IX

A, B, C

$List<T>$

$Map<K, V>$

$func : (a : T1, b : T2, c : List<T1>, d : List<T2>) \rightarrow$
 $Map<T1, T2>$

$a = A$

$b = B$

$c = List<A>$

$d = Map<A, B>$

$x = func(a, b, c, d)$

$type(x) = ?$

Унификација типова (поступак) X

$$A = T1$$

$$B = T2$$

$$List<A> = List<T1>$$

$$Map<A, B> = List<T2>$$

Унификација типова (поступак) XI

$$T1 = A$$

$$T2 = B$$

$$List = List$$

$$A = T1$$

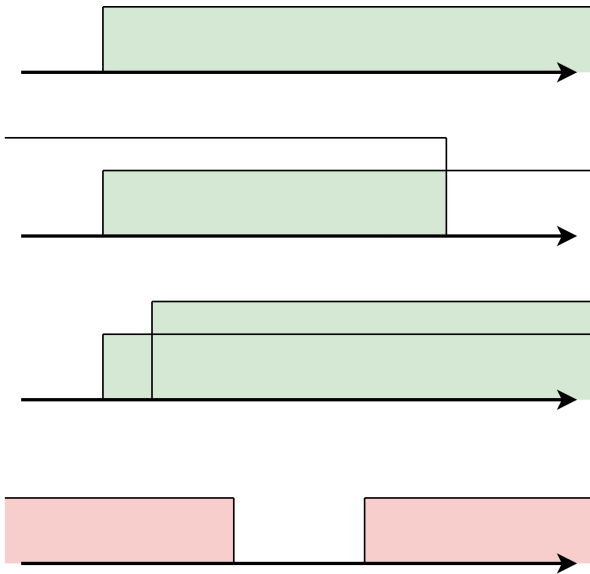
$$Map = List$$

Унификција два различита сложена типа није могућа!

Унификација и релација подтипа

- ▶ Уколико систем типова подржава релацију подтипа, поступак унификације је једнак решавању система неједначина
- ▶ Решење има доњу и горњу границу
- ▶ Исправна су сва решења која су у интервалу!
- ▶ У пракси, бира се доња или горња граница јер резултат унификације мора да буде јединствено решење

Системи неједначина



Унификација типова (наставак) I

- ▶ Поступак можемо да посматрамо као генерализовану верзију претходног поступка
- ▶ Општа правила:
 - ▶ два проста типа је могуће унификовати уколико пружени тип задовољава релацију у односу на тражени тип у датом контексту употребе (\leq , \geq , $=$)
 - ▶ два сложена типа је могуће унификовати уколико пружени тип задовољава релацију у односу на тражени тип у датом контексту употребе и уколико је могуће унификовати све типове од којих се састоје
- ▶ Додатно: решење постоји уколико је интервал правило одређен

Унификација типова (наставак) II

- ▶ Подсетник: релација подтипа (\leq, \geq) је антисиметрична, а релација једнакости ($=$) симетрична!
- ▶ Због тога је потребно разликовати пружени и тражени тип
- ▶ Додела вредности у променљиву захтева \leq зависност
- ▶ Прослеђивање вредности генеричком параметру захтева $\leq, \geq, =$ у зависности од варијантности
- ▶ Правила за свођење израза који садрже сложене типове у изразе који садрже просте типове и параметре су приближно иста
- ▶ Додатно: потребно је обратити пажњу на варијатност типова у односу на одређени параметар јер утиче на смер релације подтипа!

Унификација типова (наставак) III

$List<T>$, коваријантно у односу на T
 $Map<K, V>$, инваријантно у односу на K ,
коваријантно у односу на V

$ArrayList \leq List$

$LinkedList \leq List$

$HashMap \leq Map$

$OrderedHashMap \leq Map$

Q, P, R

$Q \leq P \leq R$

K, L, M

$K \leq L \leq M$

Унификација типова (наставак) IV

$$A(p_1 : List<X>, p_2 : Map<X, Y>) \rightarrow Y$$
$$a_1 = LinkedList<Q>$$
$$a_2 = HashMap<Q, P>$$
$$res = A(a_1, a_2)$$
$$X, Y = ?$$

Унификација типова (наставак) V

$$type(a_1) \leq type(p_1)$$

$$type(a_2) \leq type(p_2)$$

Прослеђивање аргумента функцији захтева да пружени тип буде подтип траженог типа (\leq).

Унификација типова (наставак) VI

$$\mathit{LinkedList}\langle Q \rangle \leq \mathit{List}\langle X \rangle$$

$$\mathit{HashMap}\langle Q, P \rangle \leq \mathit{Map}\langle X, Y \rangle$$

Унификација типова (наставак) VII

$$\textit{LinkedList} \leq \textit{List}$$

$$Q \leq X$$

$$\textit{HashMap} \leq \textit{Map}$$

$$Q = X$$

$$P \leq Y$$

Приликом свођења на простији облик, сложени типови су задржали релацију подтипа (\leq).

Код параметара који су коваријантни, услов је да задовољавају релацију подтипа (\leq).

Код параметара који су инваријантни, услов је да задовољавају релацију једнакости ($=$).

Унификација типова (наставак) VIII

$$Q \leq X$$

$$Q = X$$

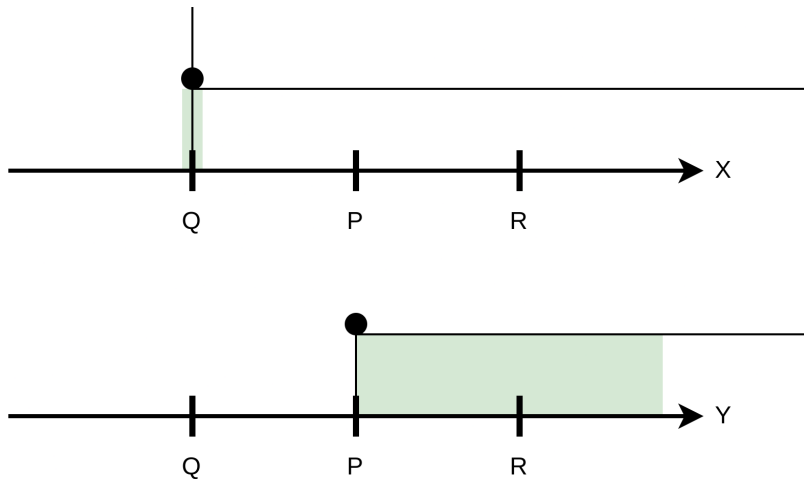
$$P \leq Y$$

Унификација је успешна!

Познати типови су задовољили задата ограничења, а интервал решења непознатих типова је исправан.

Из интервала по потреби бирамо горњу или доњу границу (задаје се као додатно ограничење).

Унификација типова (наставак) IX



Унификација типова (наставак) X

$$A(p_1 : List<X>, p_2 : Map<X, Y>) \rightarrow Y$$

$$a_1 = LinkedList<Q>$$

$$a_2 = HashMap<P, R>$$

$$res = A(a_1, a_2)$$

$$X, Y = ?$$

Унификација типова (наставак) XI

$$type(a_1) \leq type(p_1)$$

$$type(a_2) \leq type(p_2)$$

Унификација типова (наставак) XII

$$\mathit{LinkedList}\langle Q \rangle \leq \mathit{List}\langle X \rangle$$

$$\mathit{HashMap}\langle P, R \rangle \leq \mathit{Map}\langle X, Y \rangle$$

Унификација типова (наставак) XIII

$$\textit{LinkedList} \leq \textit{List}$$

$$Q \leq X$$

$$\textit{HashMap} \leq \textit{Map}$$

$$P = X$$

$$R \leq Y$$

Унификација типова (наставак) XIV

$$Q \leq X$$

$$P = X$$

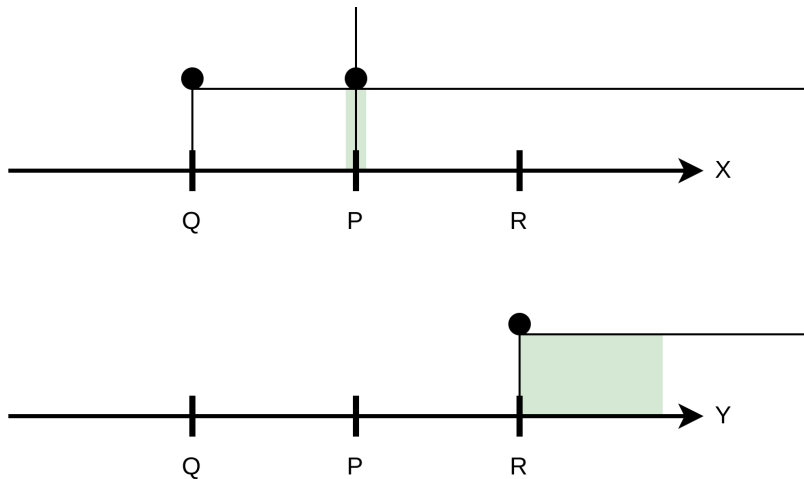
$$R \leq Y$$

Унификација је успешна!

Уочавамо да иако су два различита типа прослеђења на месту на којем је параметар X , гледамо који тип задовољава оба ограничења.

Тако параметар X постаје јединствено одређен тип!

Унификација типова (наставак) XV



Унификација типова (наставак) XVI

$$A(p_1 : List<X>, p_2 : Map<X, Y>) \rightarrow Y$$
$$a_1 = LinkedList<Q>$$
$$a_2 = ArrayList<K>$$
$$res = A(a_1, a_2)$$
$$X, Y = ?$$

Унификација типова (наставак) XVII

$$type(a_1) \leq type(p_1)$$

$$type(a_2) \leq type(p_2)$$

Унификација типова (наставак) XVIII

$LinkedList<Q> \leq List<X>$

$ArrayList<K> \leq Map<X, Y>$

Унификација типова (наставак) XIX

$LinkedList \leq List$

$Q \leq X$

$ArrayList \leq Map$

Дошло је до грешке!

Није могуће унификовати два сложена типа чији конструктор типова није сродан!

Унификација типова (наставак) XX

$$A(p_1 : List<X>, p_2 : Map<X, Y>) \rightarrow Y$$
$$a_1 = LinkedList<Q>$$
$$a_2 = HashMap<K, P>$$
$$res = A(a_1, a_2)$$
$$X, Y = ?$$

Унификација типова (наставак) XXI

$$type(a_1) \leq type(p_1)$$

$$type(a_2) \leq type(p_2)$$

Унификација типова (наставак) XXII

$$\textit{LinkedList}\langle Q \rangle \leq \textit{List}\langle X \rangle$$

$$\textit{HashMap}\langle K, P \rangle \leq \textit{Map}\langle X, Y \rangle$$

Унификација типова (наставак) XXIII

$$LinkedList \leq List$$

$$Q \leq X$$

$$HashMap \leq Map$$

$$K = X$$

$$P \leq Y$$

Унификација типова (наставак) XXIV

$$Q \leq X$$

$$K = X$$

$$R \leq Y$$

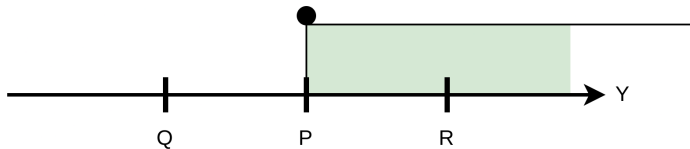
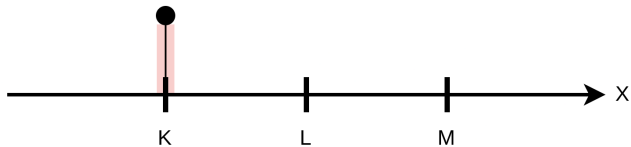
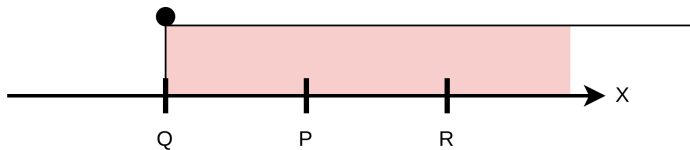
Унификација је неуспешна!

Не постоји тип који задовољава ограничења која се захтевају од параметра X .

Интервал решења X није исправан јер типови Q и K нису у никаквој релацији.

Доња граница интервала мора да буде мања или једнака горњој граници како би интервал био исправан.

Унификација типова (наставак) XXV



Имплементација I

- ▶ Приказани модел тражења интервала решења преко преклапања решења на оси је погодан за разумевање од стране људи
- ▶ Могуће га је имплементирати као алгоритам, али приказани облик није најпогоднији
- ▶ Уместо накнадног тражења интервала из ограничења, једноставније је интервал тражити у лету
- ▶ Подсетник: решење има горњу и доњу границу
- ▶ Ограничење које смо уврстили у интервал можемо да уклонимо из скупа нерешених ограничења
- ▶ Унификација је успешна уколико су сва ограничења разрешена успешно

Имплементација II

- ▶ Нека је X непознат тип, а $[lower, upper]$ интервал решења
- ▶ Интервал је исправан уколико важи $lower \leq upper$
- ▶ Горња или доња граница могу да недостају
- ▶ У решавању бројчаних неједначина крајње границе интервала су $+\infty$ и $-\infty$
- ▶ У неједначинама типова, обиласком хијерархије типова се проналазе доња и горња граница у складу са релацијом подтипа

Имплементација III

- ▶ Ограничење $X \geq T$:
 - ▶ како би интервал био исправан, мора да важи $T \leq upper$
 - ▶ уколико $lower$ нема вредност, доња граница добија нову вредност $lower = T$
 - ▶ уколико $T \leq lower$, доња граница задржава стару вредност
 - ▶ уколико $T \geq lower$, доња граница добија нову вредност $lower = T$

Имплементација IV



Имплементација V



Имплементација VI



Имплементација VII



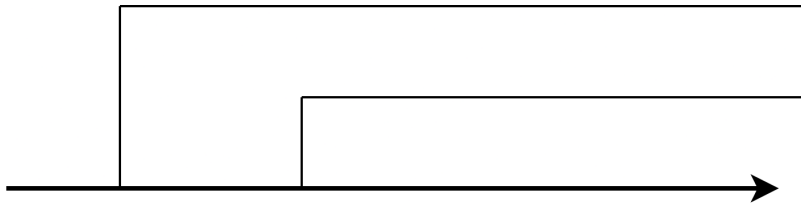
Имплементација VIII



Имплементација IX



Имплементација X



Имплементација XI



Имплементација XII



Имплементација XIII

- ▶ Ограничење $X \leq T$:
 - ▶ како би интервал био исправан, мора да важи $T \geq lower$
 - ▶ уколико $upper$ нема вредност, горња граница добија нову вредност $upper = T$
 - ▶ уколико $T \geq upper$, горња граница задржава стару вредност
 - ▶ уколико $T \leq upper$, горња граница добија нову вредност $upper = T$

Имплементација XIV



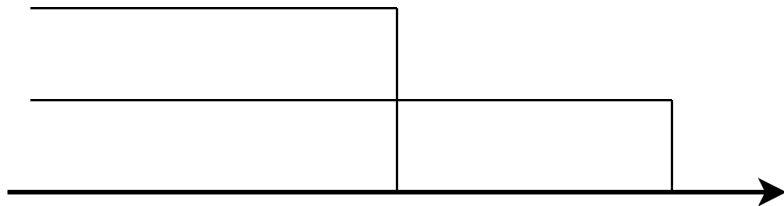
Имплементација XV



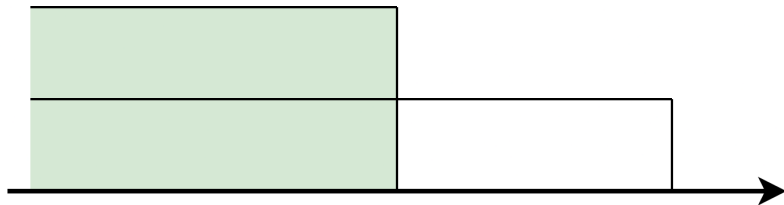
Имплементација XVI



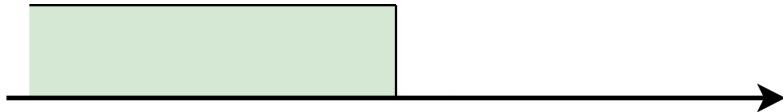
Имплементација XVII



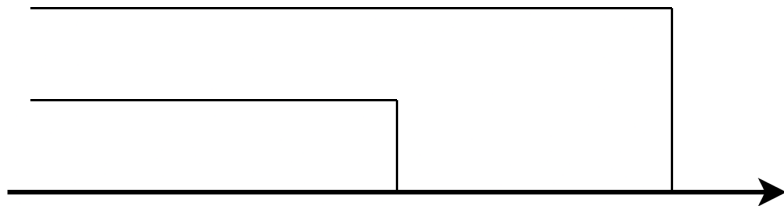
Имплементација XVIII



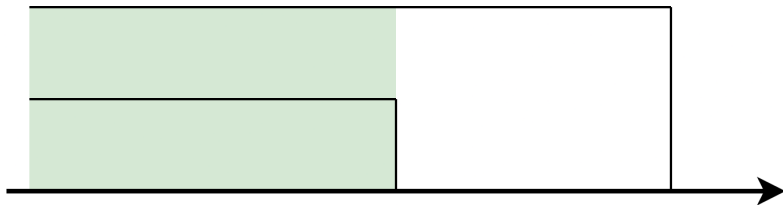
Имплементација XIX



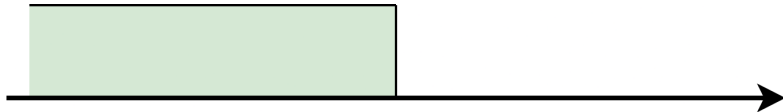
Имплементација XX



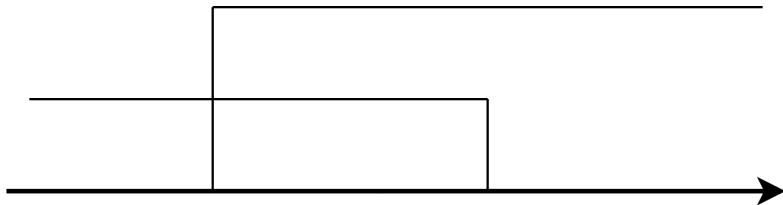
Имплементација XXI



Имплементација XXII



Имплементација XXIII



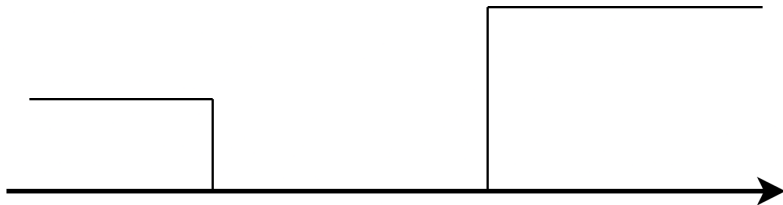
Имплементација XXIV



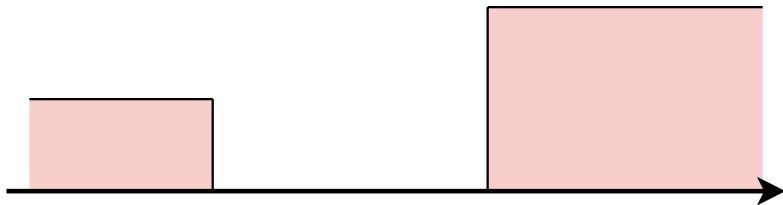
Имплементација XXV



Имплементација XXVI



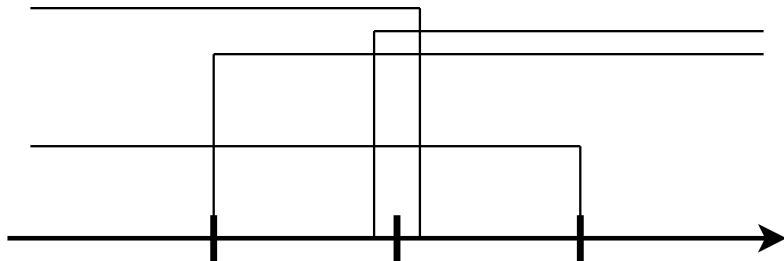
Имплементација XXVII



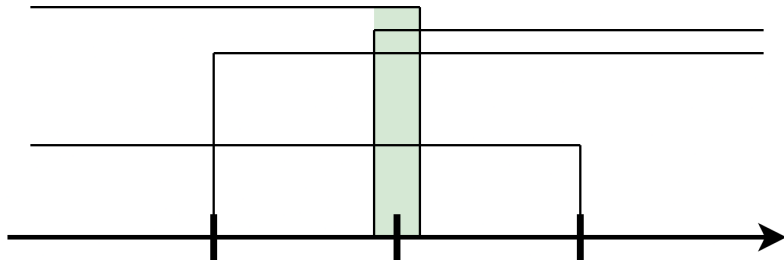
Имплементација XXVIII

- ▶ Проблем: како уврстити ограничење $=$?
- ▶ Подсетник: релација подтипа је антисиметрична $(A \leq B \wedge B \leq A \Rightarrow A = B)!$
- ▶ Уместо ограничења $A = B$ уводимо пар ограничења: $A \leq B$ и $B \leq A$
- ▶ У пракси, ово се ради у једном кораку
- ▶ Интервал тражимо по правилима дефинисаним за ограничења \leq и \geq

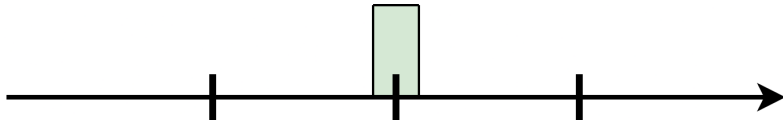
Имплементација XXIX



Имплементација XXX



Имплементација XXXI



Закључак I

- ▶ Као покушај налажења заједничког језика разумљивог и рачуанрима и људима, настали су виши програмски језици
- ▶ Додатна замисао је била аутоматска провера исправности програма
- ▶ Једнакост између очекиваног и обезбеђеног типа је била довољан услов да се спречи неисправна употреба вредности у програму
- ▶ Како појмови у природи показују сличност, замисао је била представити ту сличност кроз релацију подтипа

Закључак II

- ▶ Сложени типови су неопходни како би се описале појаве из природе, као и појаве из света рачунарства
- ▶ Релација подтипа код сложених типова зависи од конструктора типова и од садржаја самих типова
- ▶ За неке типове је природно да релација између садржаних типова директно одговара релацији између сложених типова (коваријантност), док је за одређене типове она супротна (контраваријантност)
- ▶ За разлику од математике, рачунарство познаје појам мутабилности
- ▶ Иако типови природно показују варијантност, мутабилност је често значајно ограничава

Закључак III

- ▶ Релација подтипа не обезбеђује довољну флексибилност за типове попут листа и мапа
- ▶ Генерички типови омогућавају да корисник при инстанцирању типа зада прецизнија ограничења типа
- ▶ Код генеричких типова, потребно је обавити поступак унификације како би одредили вредности непознатих параметара
- ▶ Уколико систем типова подржава само једнакост типова, поступак је једнак решавању система једначина
- ▶ Уколико систем типова подржава релацију, поступак је једнак решавању система неједначина

Литература I

- ▶ Type Checking (Part 1), Keith Schwarz
<https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/lectures/09/Slides09.pdf>
- ▶ Type Checking (Part 2), Keith Schwarz
<https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/lectures/10/Slides10.pdf>
- ▶ Everything You Always Wanted to Know About Type Inference
- And a Little Bit More, Robert Griesemer
<https://go.dev/blog/type-inference>
- ▶ Unification in Chalk (part 1), Niko Matsakis
<https://smallcultfollowing.com/babysteps/blog/2017/03/25/unification-in-chalk-part-1/>

Литература II

- ▶ Type unification rules (The Go Programming Language Specification) https://tip.golang.org/ref/spec#Type_unification_rules
- ▶ Kotlin type constraints (Kotlin language specification), Marat Akhin & Mikhail Belyaev <https://kotlinlang.org/spec/kotlin-type-constraints.html>
- ▶ Type inference (Rust Compiler Development Guide) <https://rustc-dev-guide.rust-lang.org/type-inference.html>