

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?

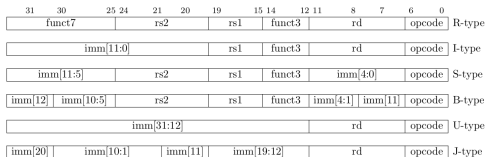
- ▶ Неопходно је да имамо формалну дефиницију процеса који желимо да аутоматизујемо - **морамо да дефинишемо алгоритам**
 - ▶ сама дефиниција мора бити формална, односно мора садржати прецизан опис корака
 - ▶ формат дефиниције не мора да буде формалан!
- ▶ Формалну дефиницију морамо изразити у формату 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 }
```

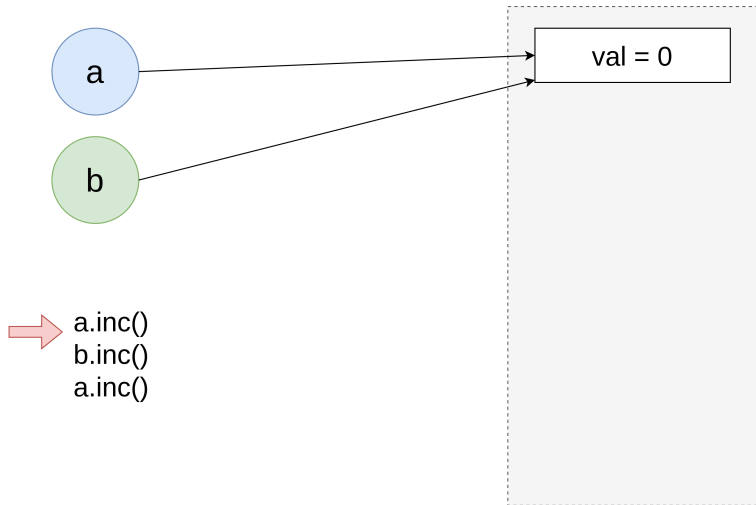
Теорија и пракса

- ▶ 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

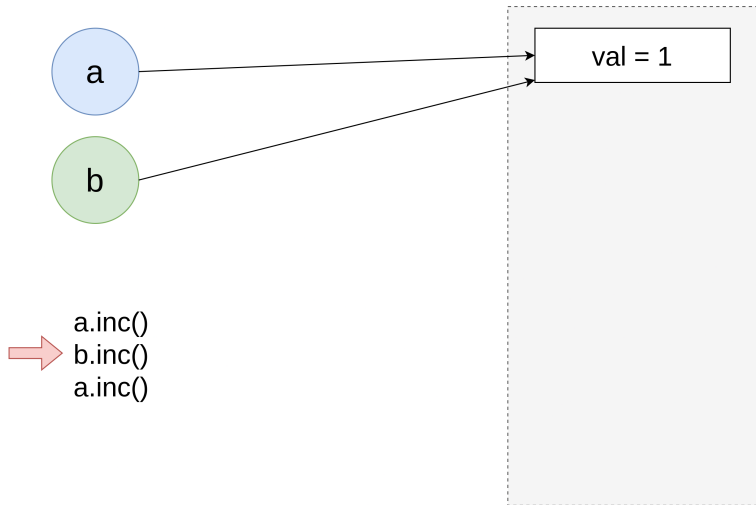
Мутабилност

- ▶ 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

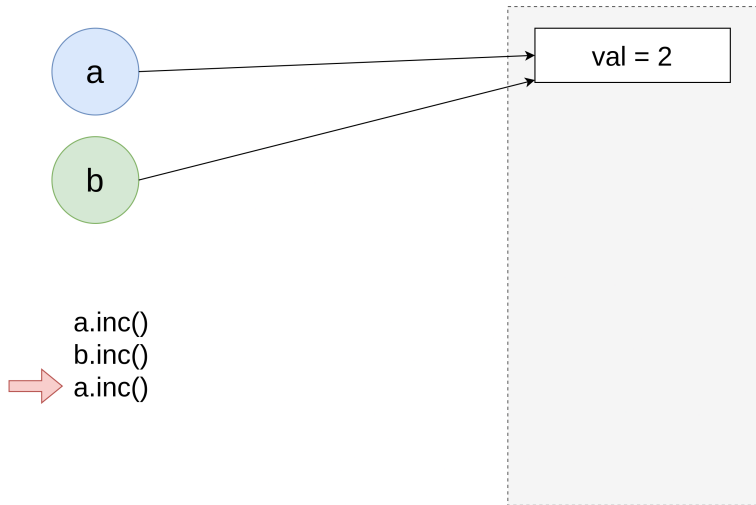
Мутабилни објекти



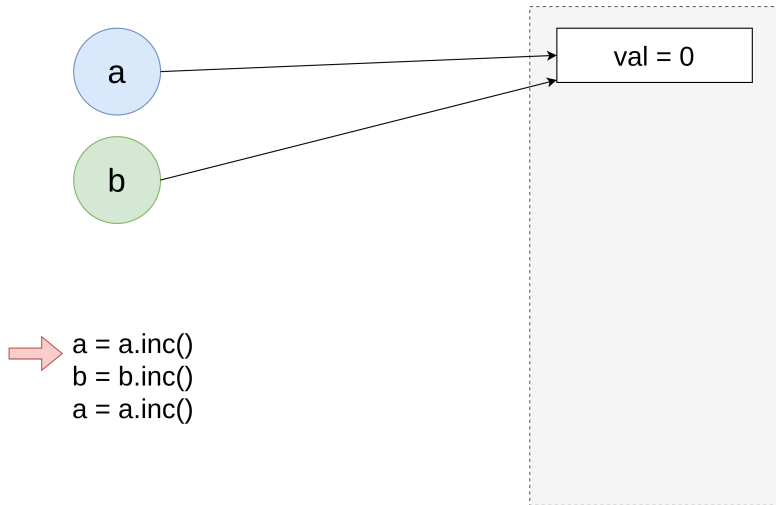
Мутабилни објекти



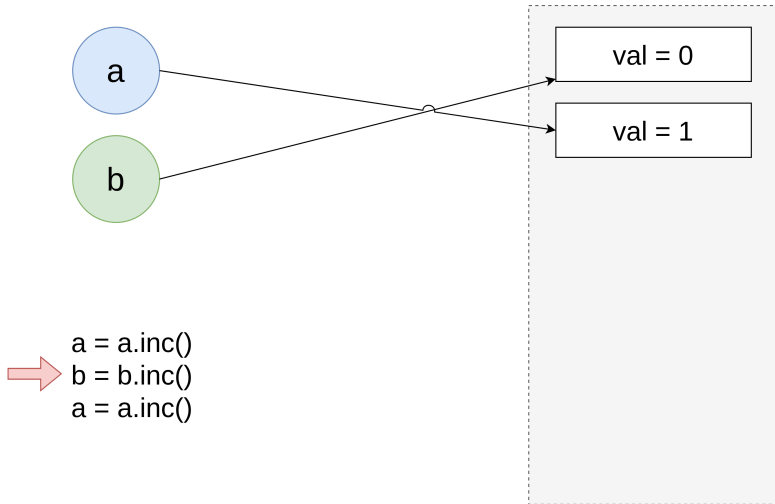
Мутабилни објекти



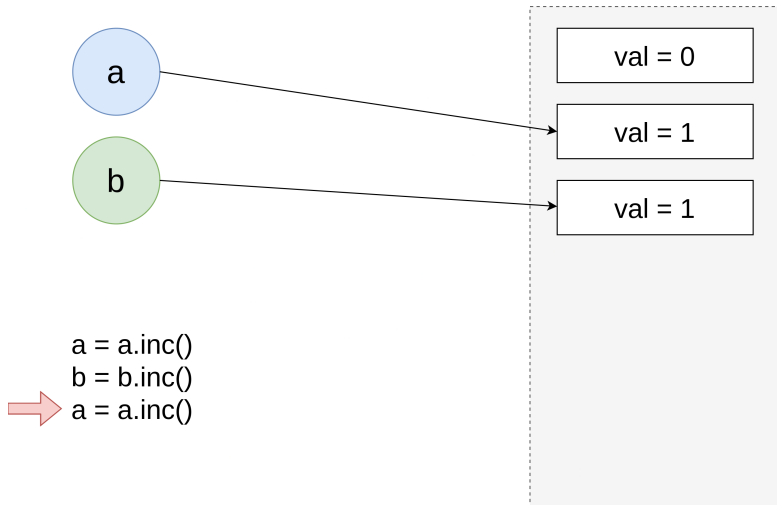
Имутабилни објекти



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 А мора да садржи све вредности које подржава Б 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

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
- ▶ Како бисмо могли да креирамо овакве типове?

- ▶ 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 ()?
- ▶ Можемо ли да упоредимо два типа низова?

Низови II

- ▶ У низ $T[]$ можемо да ускладишtimo вредност x : X уколико $X \leq T$
 - ▶ T представља горњу границу типа вредности у низу!
- ▶ У променљиву u : $A[]$ 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

$$A \leq B \leq C$$

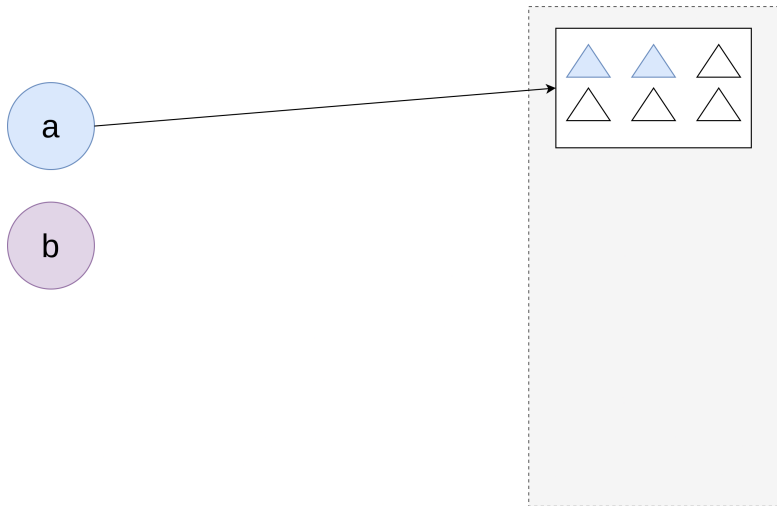
$$a : A[] = [A, A, A]$$

$$b : B[] = a$$

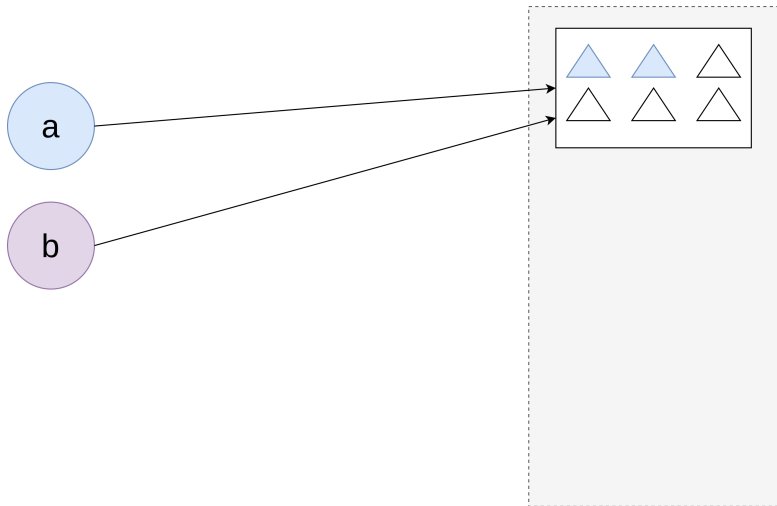
$$b.add(B)$$

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

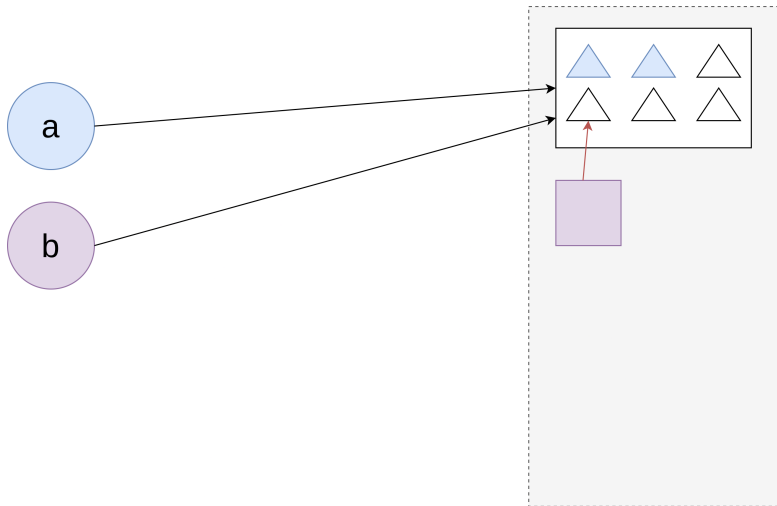
Варијантност и мутабилност III



Варијантност и мутабилност IV



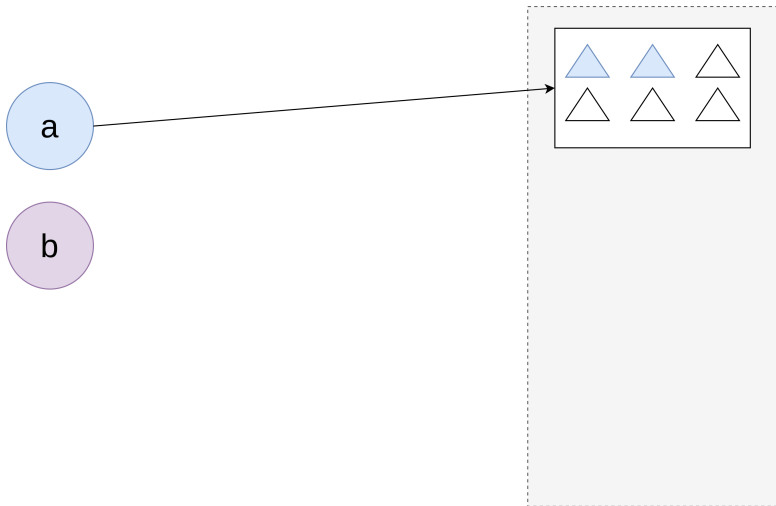
Варијантност и мутабилност V



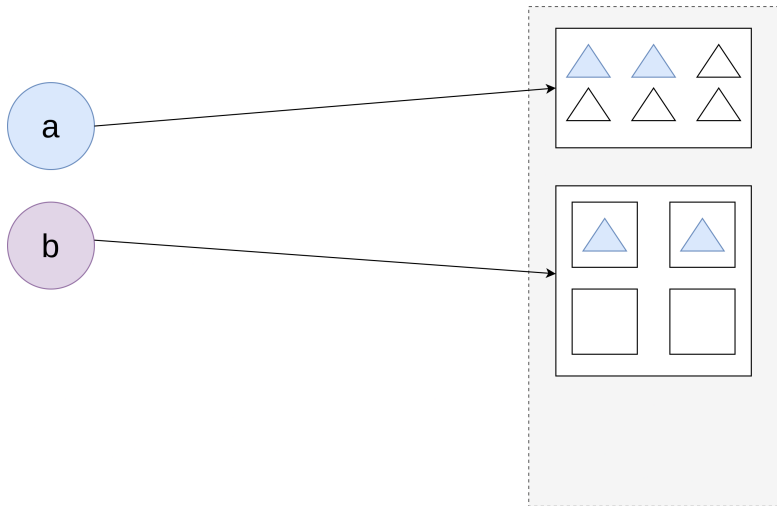
Варијантност и мутабилност VI

- ▶ 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

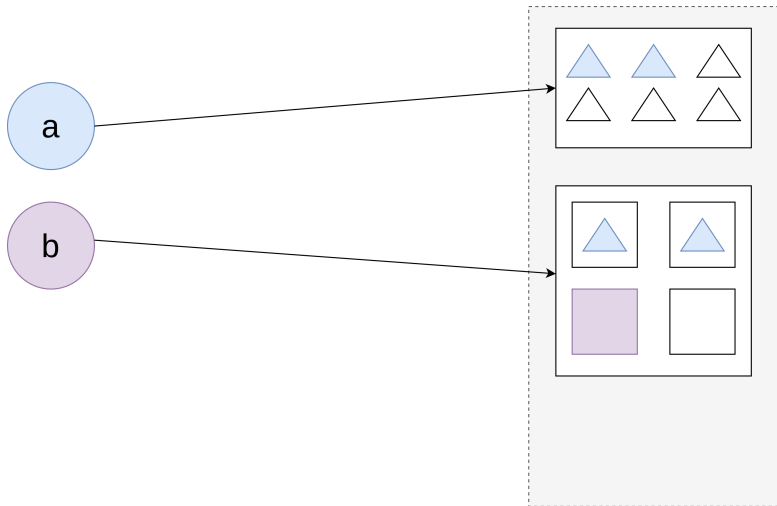
Варијантност и мутабилност VII



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 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 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

- ▶ 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)

..

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

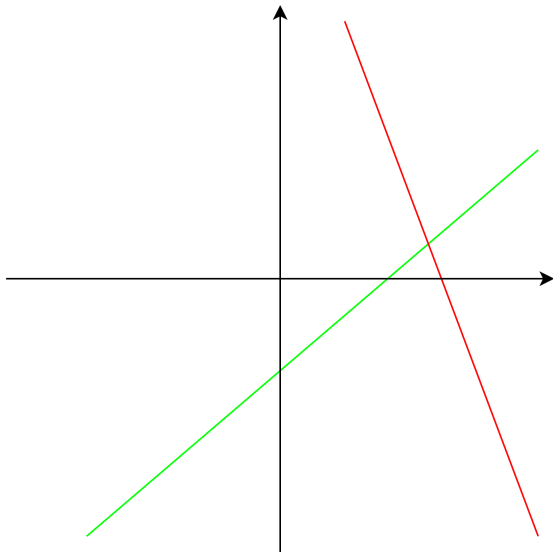
Унификација типова II

- ▶ 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

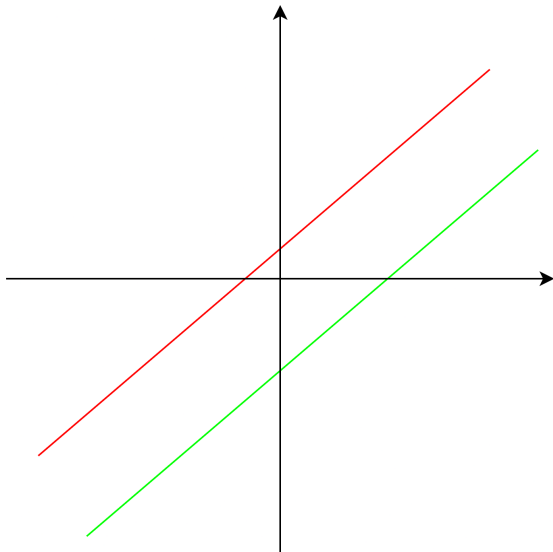
Унификација типова III

- ▶ 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

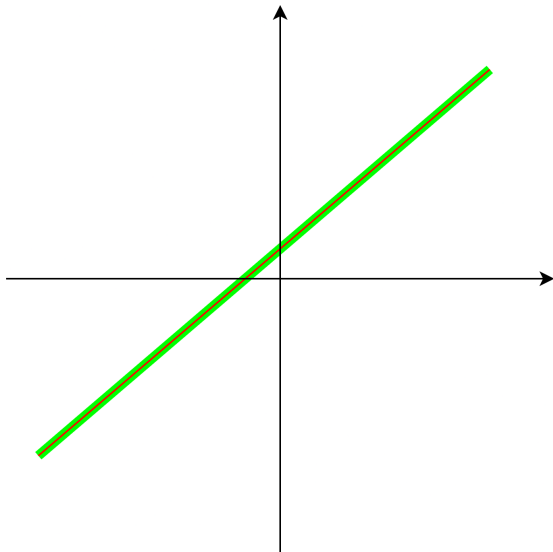
Одређен систем једначина



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

$$T1 = A$$

$$T2 = B$$

$$List = List$$

$$A = T1$$

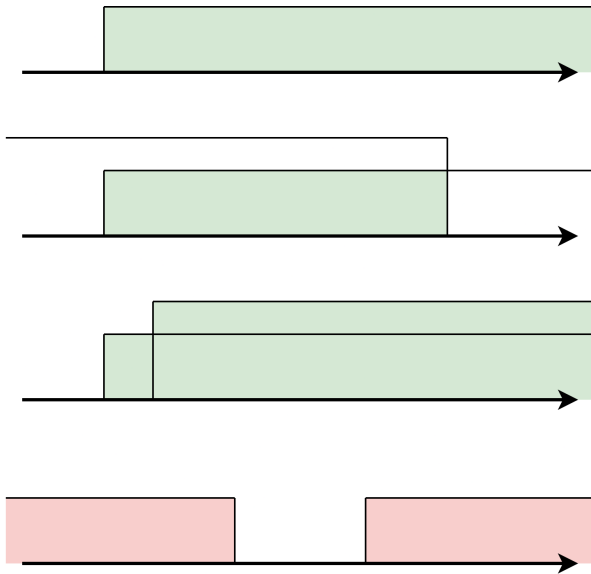
$$Map = List$$

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 $A \leq B$
 - ▶ два сложена типа је могуће унификовати уколико $A \leq B$ и уколико је могуће унификовати све типове од којих се састоје
- ▶ Додатно: решење постоји уколико је интервал правило одређен
- ▶ 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!