Registrite allokeerimine

Taavet Kikas

2004

Ülevaade

- Väärtusi võib hoida mälus või registrites.
  - Mälu on suur kuid aeglane.
  - Registrid on kiired kuid neid on vähe.
- Eesmärk on vähendada mälupöördumiste arvu.
- Millised muutujad paigutada registritesse?
- Üldine ülesanne - muuta m registriga programm n registriga programmiks.

Ülevaade

- Kaks peamist lähenemisviisi:
  - Registrite allokeerimine kasutusloendite abil.
  - Registrite allokeerimine graafi värvimise teel.

Elusad muutujad

- Öeldakse, et muutuja on punktis *p* elus kui pärast punkti *p* kasutatakse teda enne kui ta (üle) defineeritakse

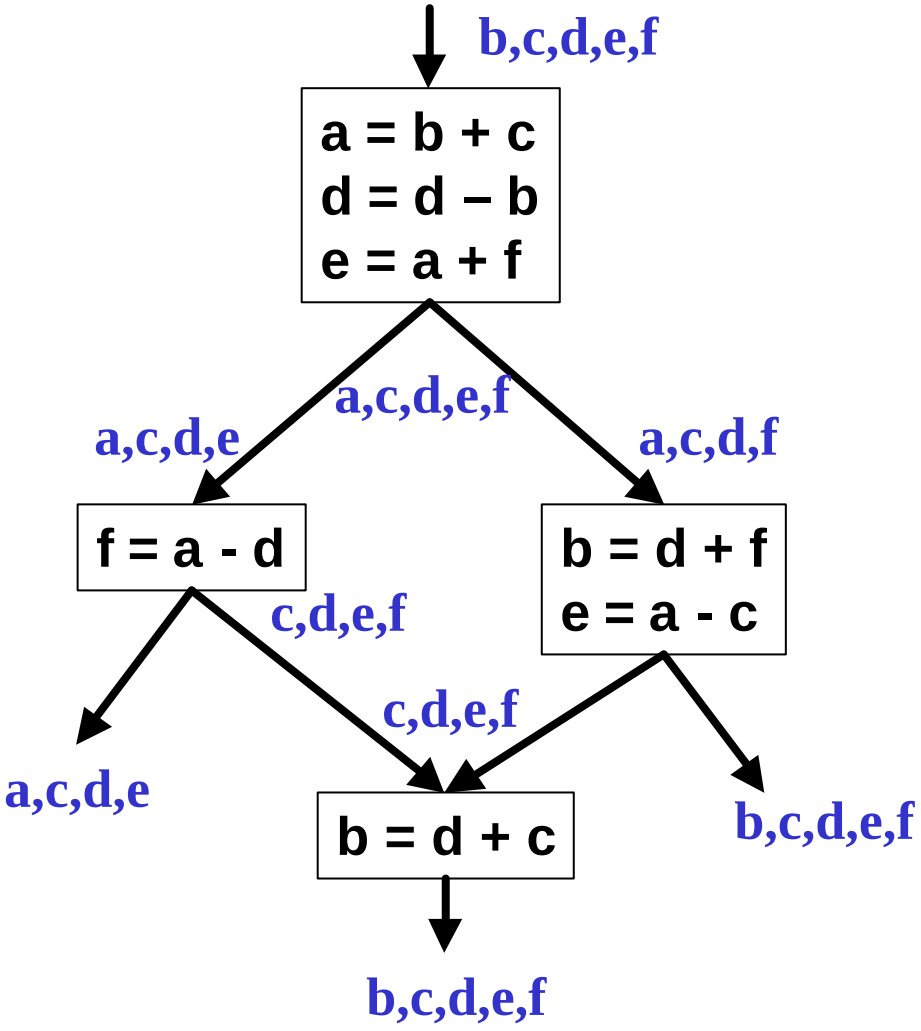
t1 = z-2	**t1,x,y,z**
t2 = x+t1	**t2,y,z**
t3 = z-2	**t2,t3,y**
t4 = uminus y	**t2,t3,t4**
t5 = t3 pow	**t4 t2,t5**
t6 = t2 <= t5	**t6**
b = t6	

Kasutusloendid

- Kasutusloendi abil hinnatakse tulu, mis saadakse väärtuse paigutamisest registrisse.
- Iga baasploki (*basic block*) B ja muutuja x korral säästame
  - 1 ühiku x igalt kasutuselt, mis eelneb x definitsioonile
  - 2 ühikut kui x on bloki lõpuks, kus ta defineeriti, veel elus
- Kogutulu

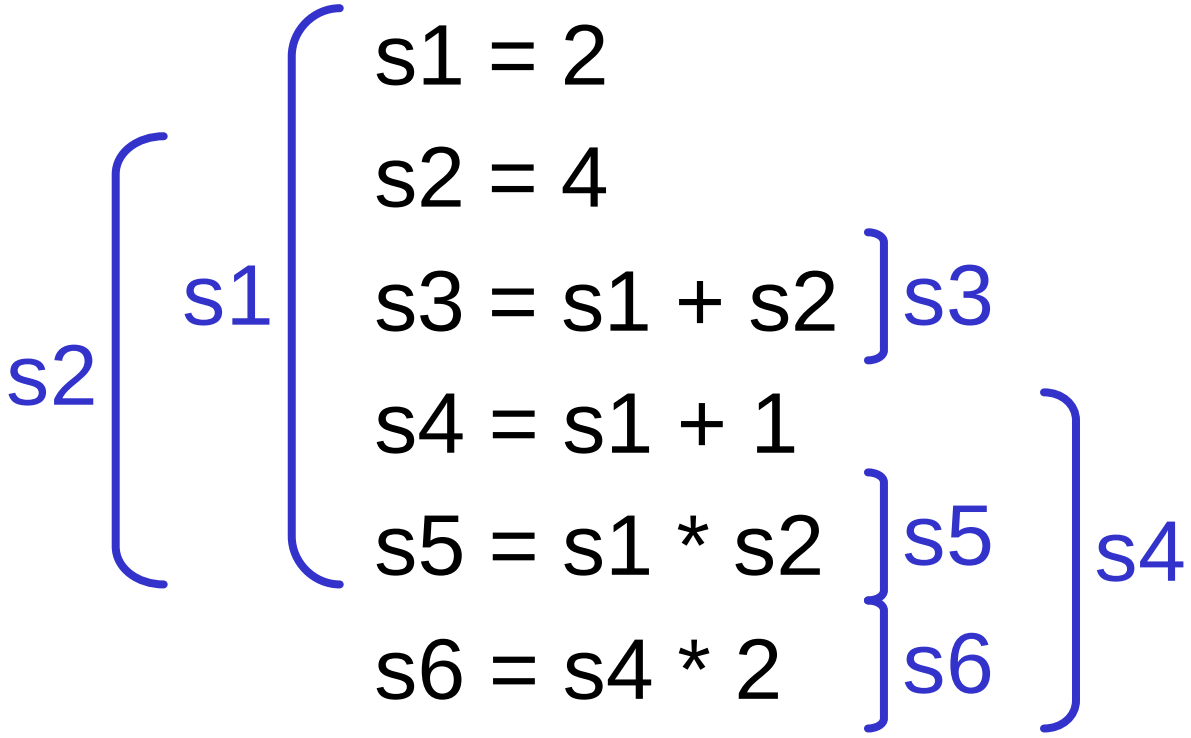
$$\textbf{benefit(x)} = ? \ \textbf{(use(x, B) + 2*live(x,B))}$$

Kasutusloendid

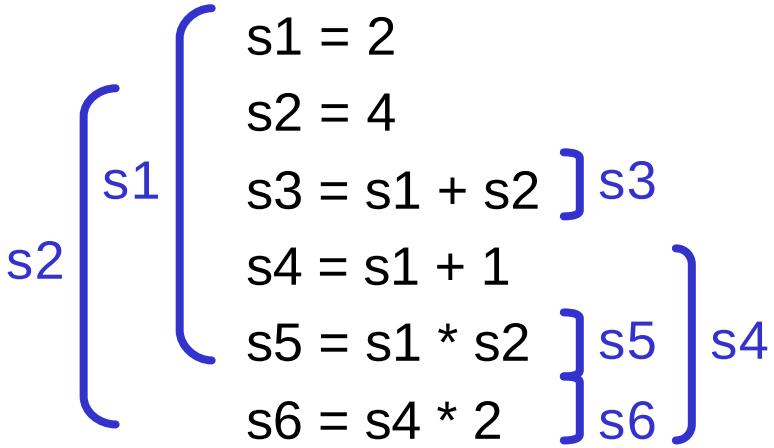


	kasut	elus	kasu
a	2	1	4
b	2	2	6
c	3	0	3
d	4	1	6
e	0	2	4
f	2	1	4

Väärtuste eluajad (*live ranges*)



Interferents

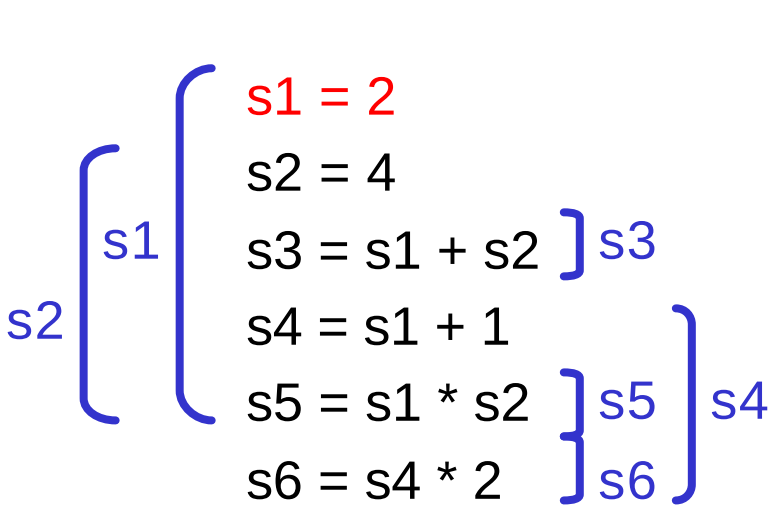


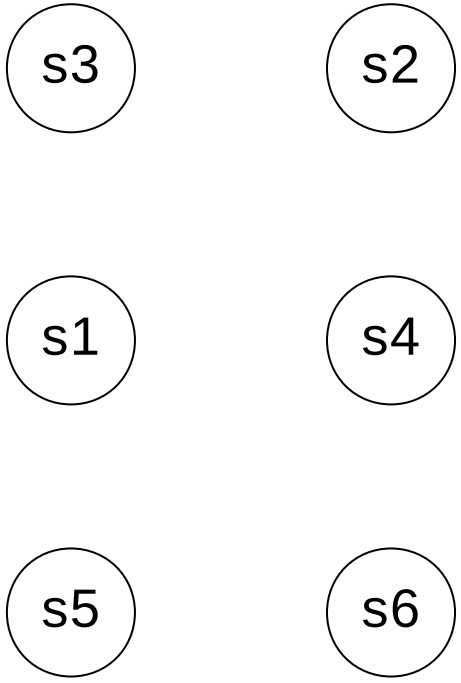
- Ühte ja samasse registrisse ei saa paigutada kahte erinevat väärtust kui neid on vaja samaaegselt kasutada
- Öeldakse, et väärtused interfereeruvad
- Väärtusi, mida soovitakse registritesse paigutada, nimetatakse kandidaatideks

Interferentsi graaf

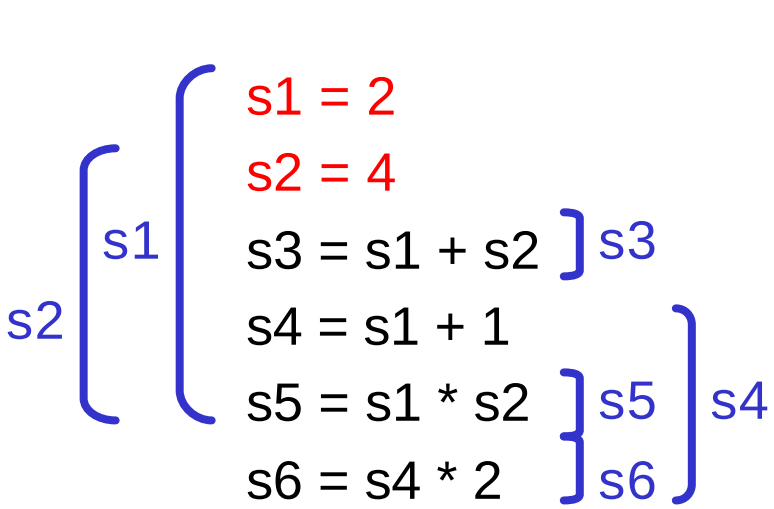
- Interferentside kirjeldamiseks saab koostada interferentside graafi
- Interferentsi graafiks nimetatakse suunamata graafi, kus iga tipp tähistab kandidaati või registrit
- Interferentsi graafis eksisteerib serv kahe tipu vahel parajasti siis kui tippudele vastavad kandidaadid interfereeruvad või kui kandidaati ei saa paigutada konkreetsesse registrisse.
- Registrid interfereeruvad üksteisega

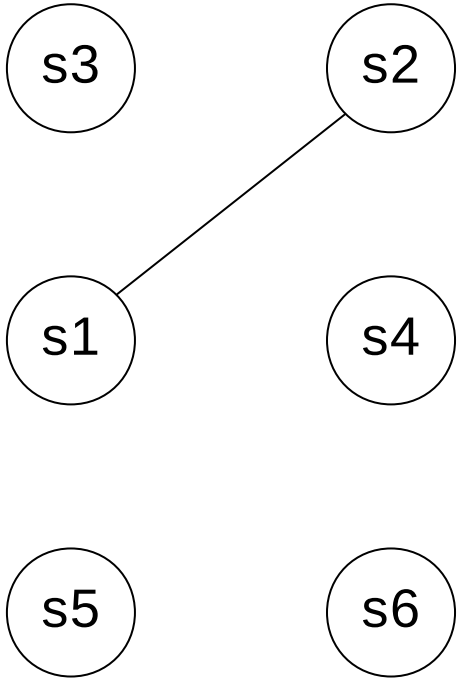
Interferentsi graaf



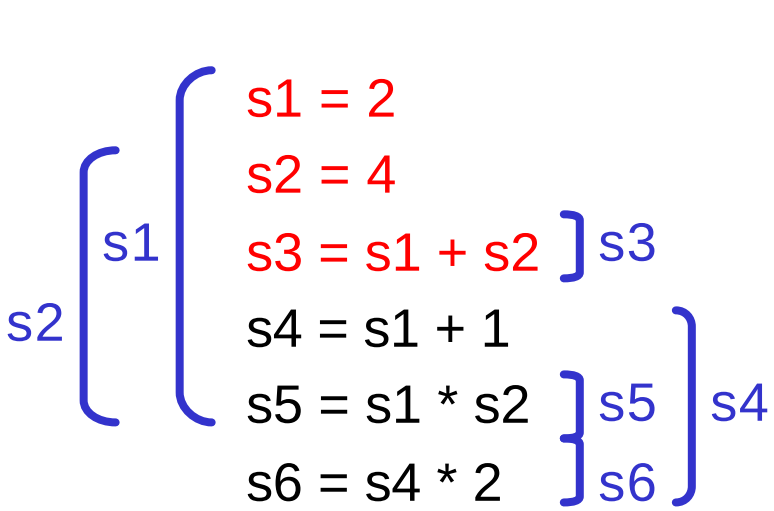


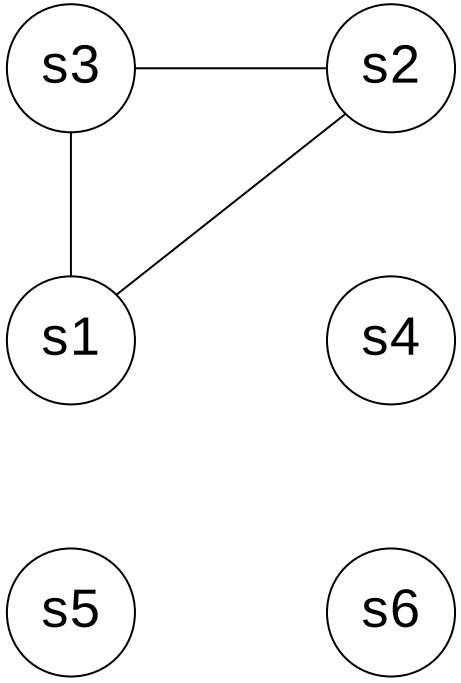
Interferentsi graaf



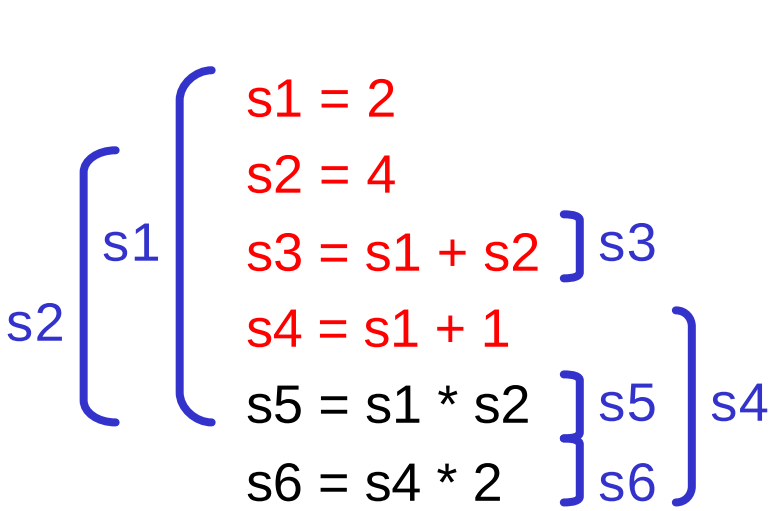


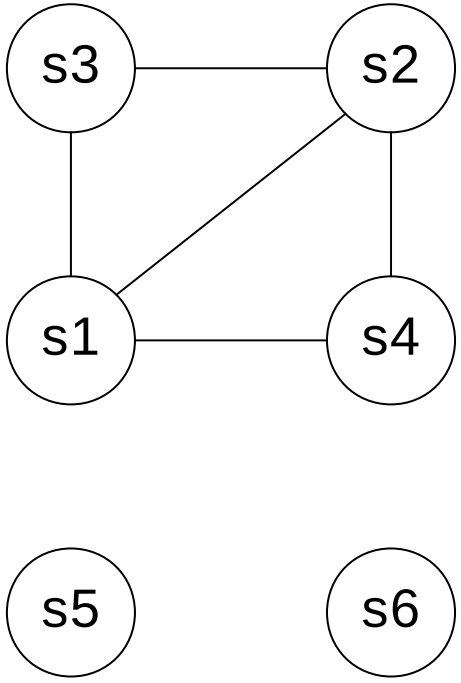
Interferentsi graaf



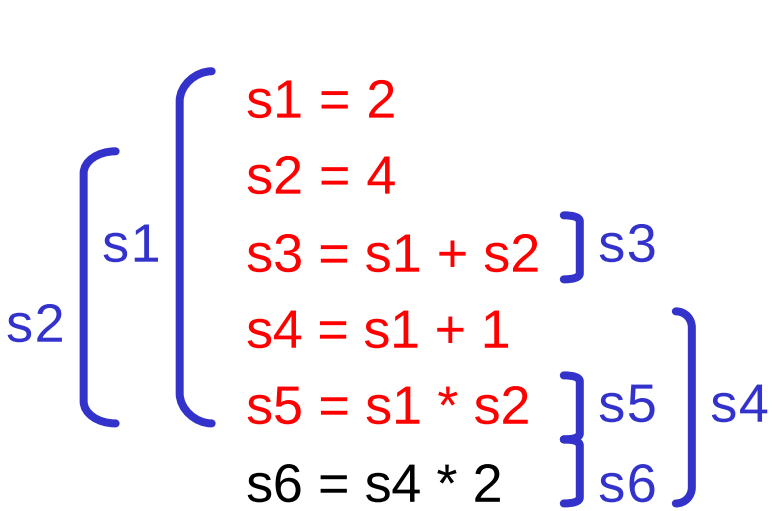


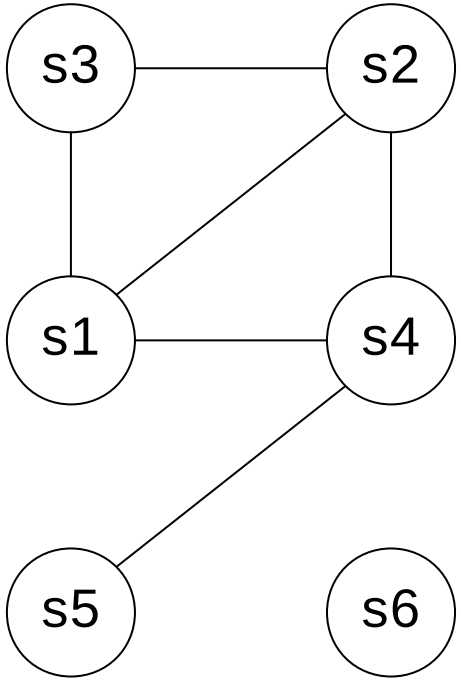
Interferentsi graaf



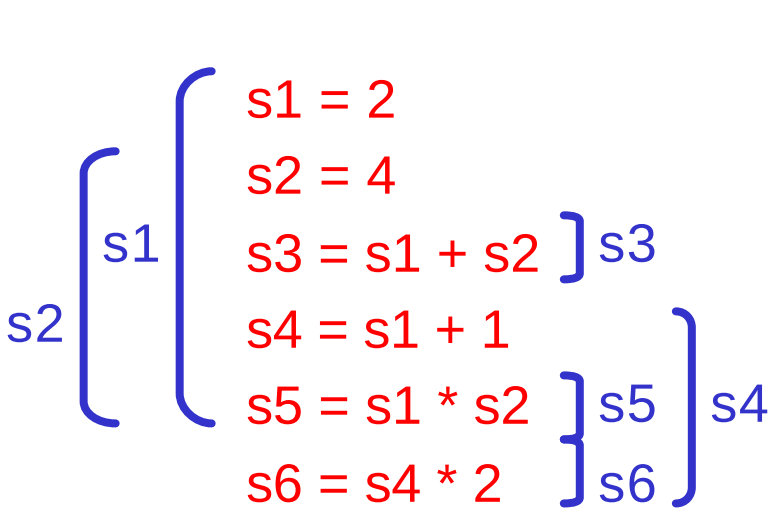


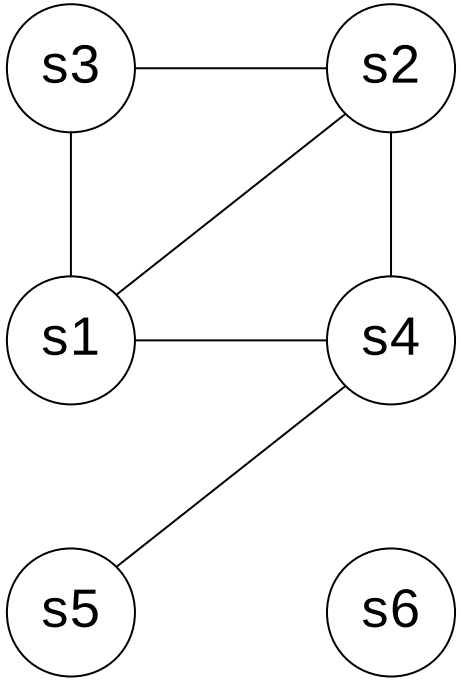
Interferentsi graaf





Interferentsi graaf

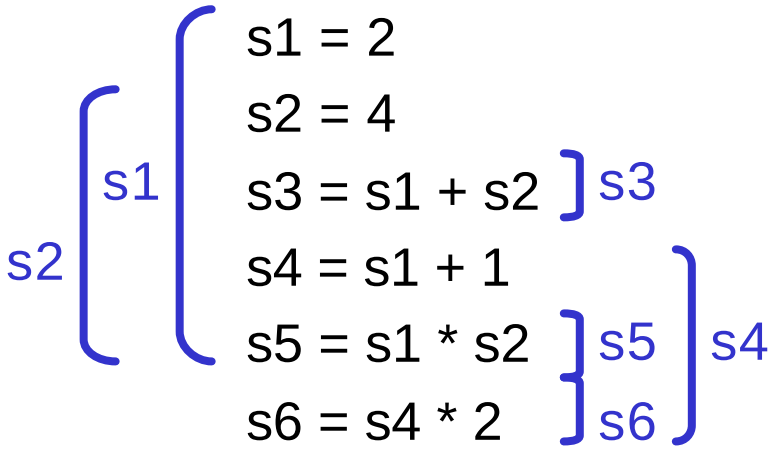


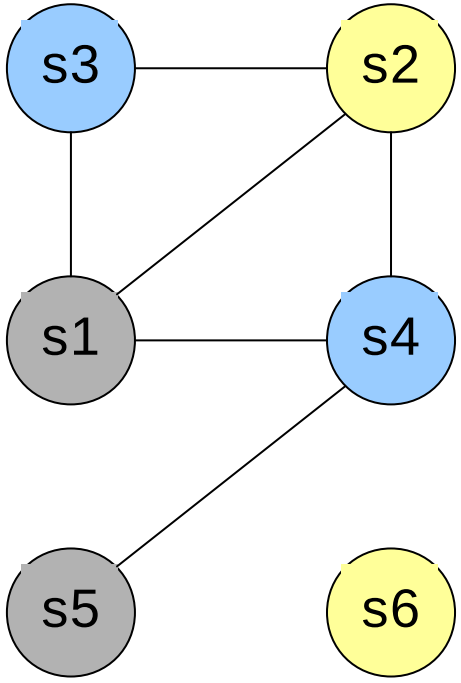


Interferentsi graafi värvimine

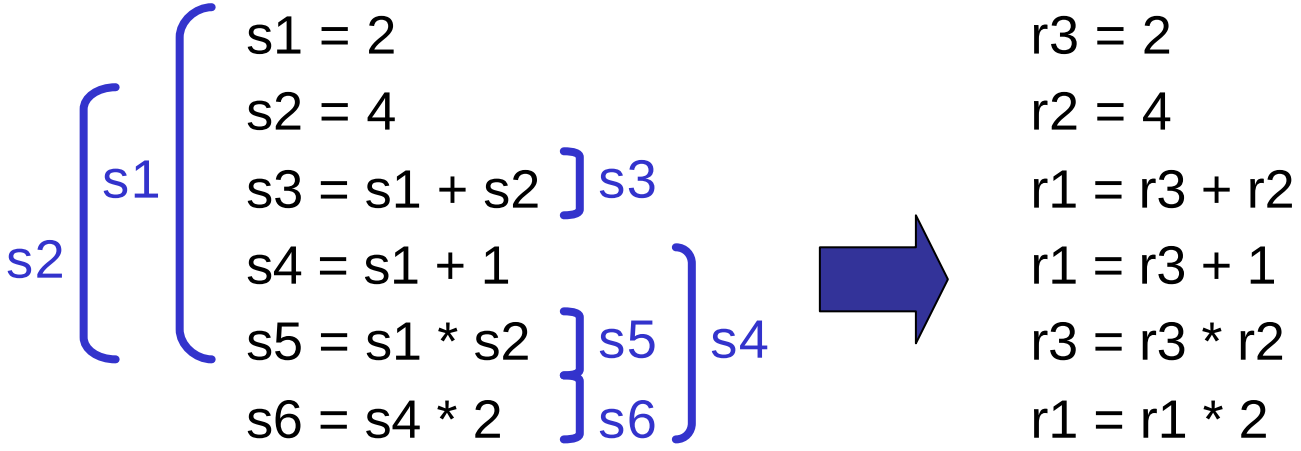
- Registrite allokeerimiseks seame igale graafi tipule vastavusse registri, selliselt et kaarega ühendatud tipud ei kuulu ühte ja samasse registrisse
- Tegu on graafi värvimise probleemiga: kas on võimalik värvida graafi k värviga nii, et suvalised kaks kaarega ühendatud tippu ei ole sama värvi?
- Graafe, mis on sellisel viisil k värviga värvitavad, nimetatakse k-aluselisteks

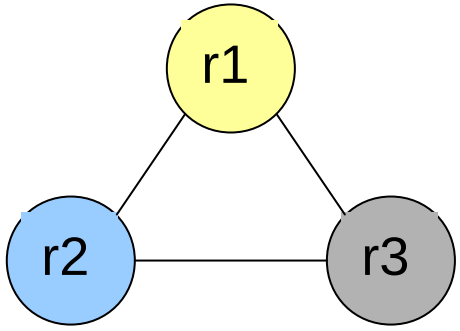
Interferentsi graafi värvimine





Interferentsi graafi värvimine





Interferentsi graafi värvimine

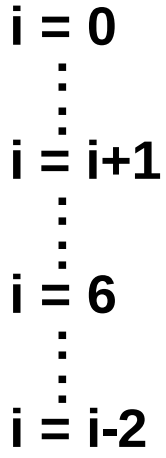
- Kui interferentsi graaf ei ole k värviga värvitav, ei saa kõiki kandidaate registritesse paigutada.
- Osasid kandidaate tuleb mälus hoida.
- Seda nimetatakse *spilling*'uks.
- Meie eesmärk on valida vähim arv *spilling*'u kandidaate, et graaf muutuks k värviga värvitavaks.

Registrite allokeerimise algoritm

1. Identifitseeri kandidaadid
2. Konstrueeri interferentsi graaf
3. Värvi interferentsi graaf
4. Kui graaf on k värviga värvitav, siis allokeeri registrid ja lõpeta
5. Vali *spilling*'u kandidaadid
6. Kirjuta kood ümber
7. Mine 2. punkti juurde

Kandidaatide identifitseerimine

- Muutujanimede kasutamine kandidaatidena ei ole mõistlik - muutujanimi seotakse konkreetse registriga, samas võidakse sama nimega muutujat mitu korda defineerida ning üks ja sama nimi viitab siis tegelikult erinevatele väärtustele.
- Näiteks.

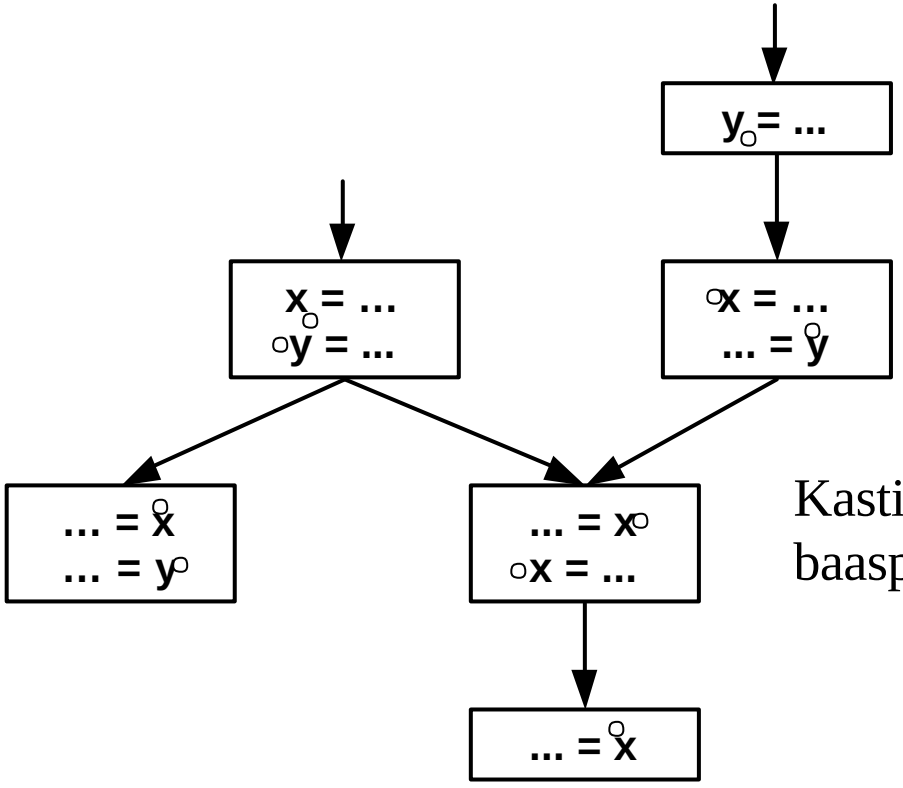


Võrgud

- Võtame kasutusele võrgu (*web*) mõiste
  - muutuja definitsioon ja kõik selle kasutused asuvad samas võrgus
  - kõik muutuja definitsioonid, millest jõutakse sama kasutuseni, asuvad samas võrgus

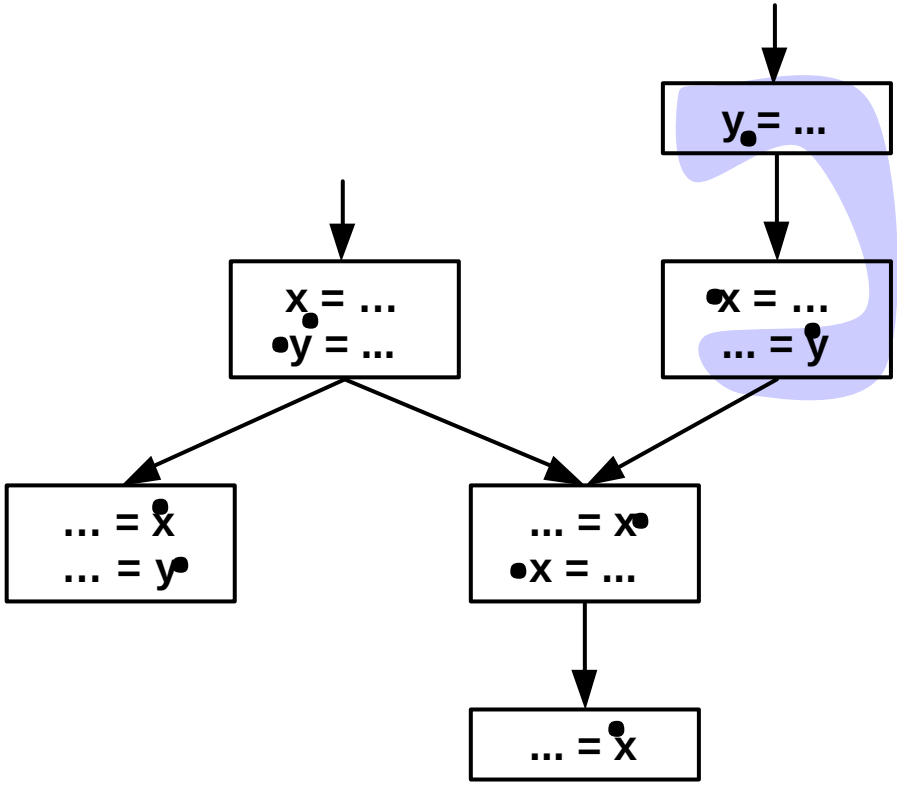
  (Ei ole sama, mis graafiteooria võrk.)
- Sama muutujanimi erinevates võrkudes viitab erinevatele väärtustele.

Võrgud

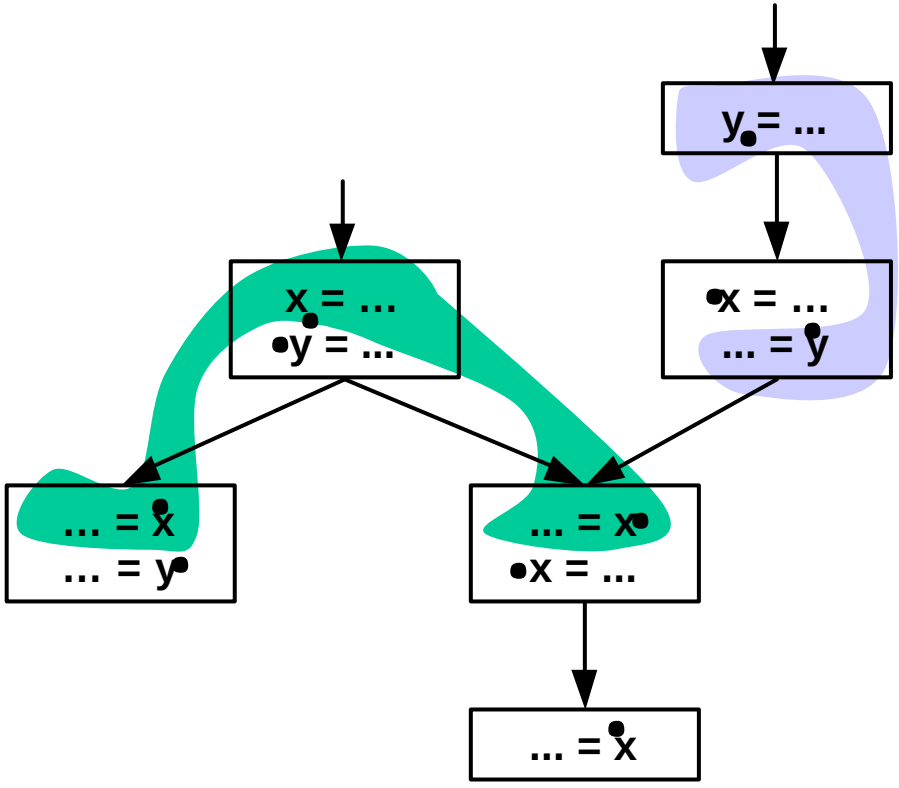


Kastid tähistavad programmi baasplokke

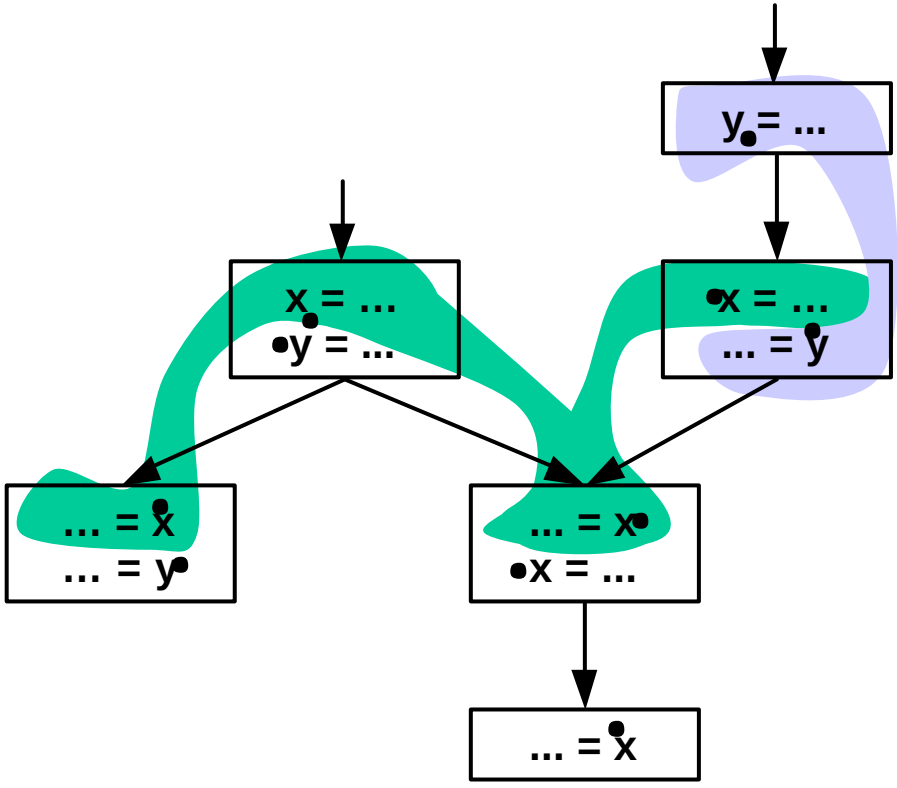
Võrgud



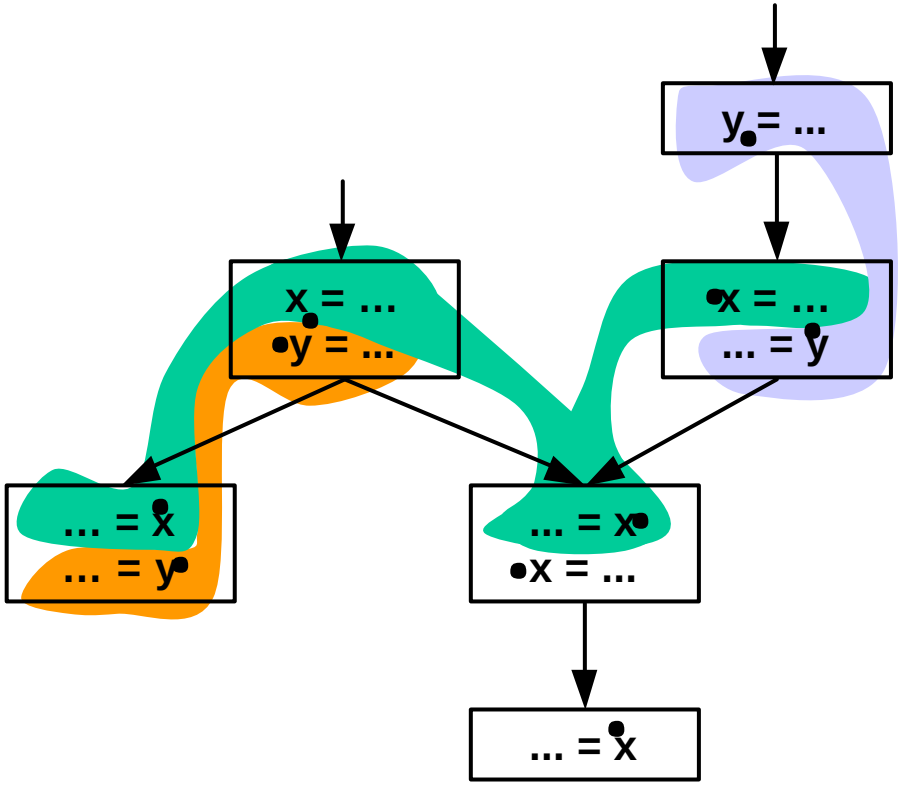
Võrgud



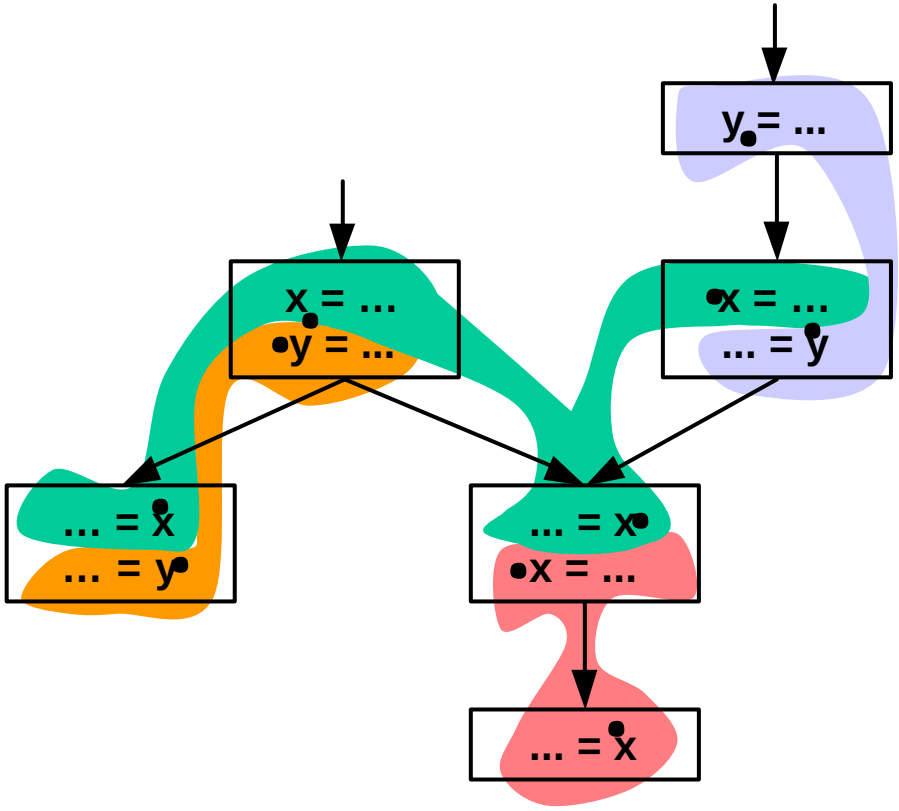
Võrgud



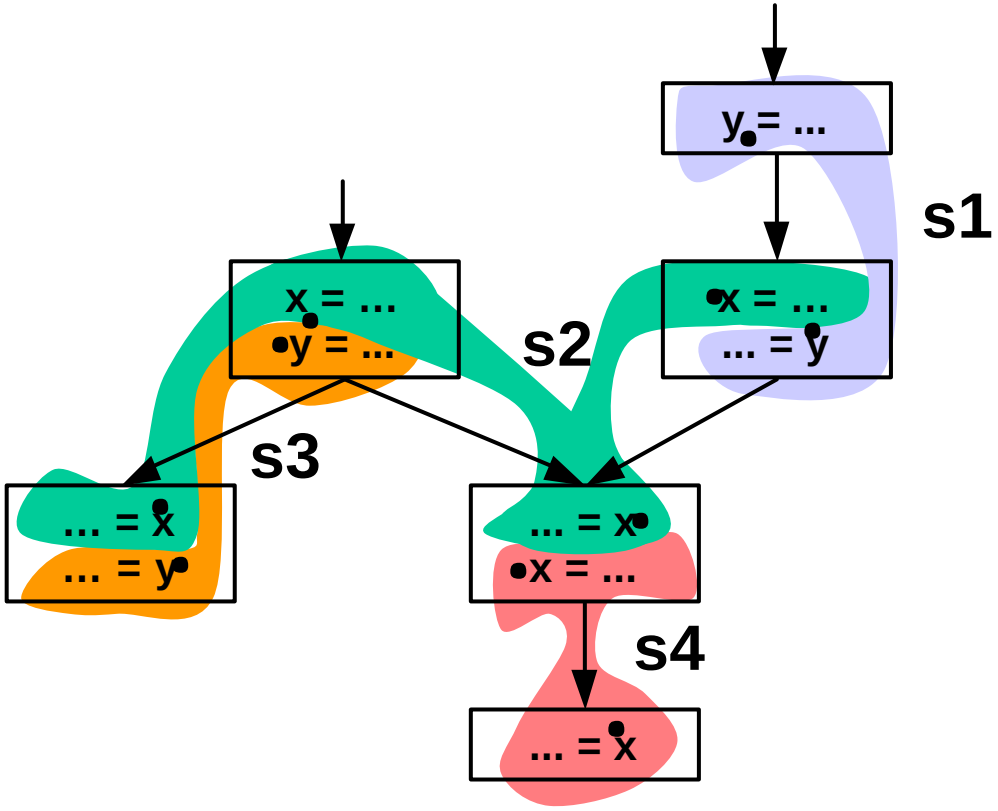
Võrgud



Võrgud



Võrgud

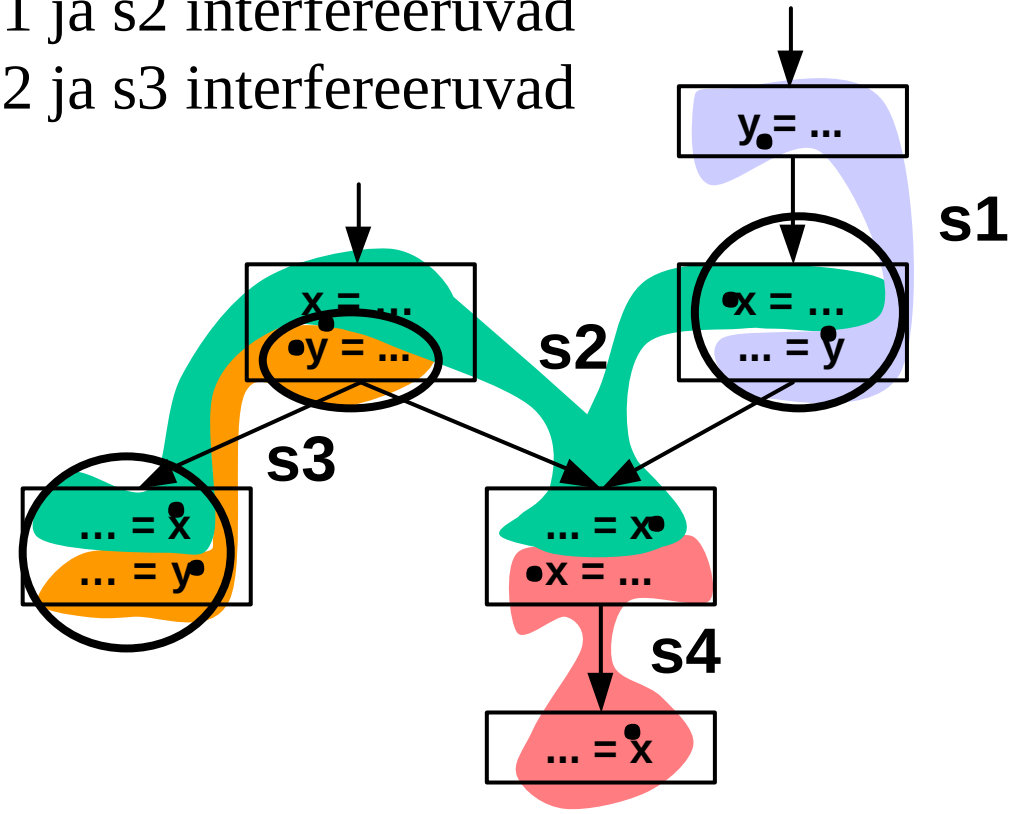


Võrkude eluajad ja interferents

- Öeldakse, et baasplokkide (*basic blocks*) hulk S on kumer (*convex*) kui suvaliste baasplokkide a ja b korral, mis kuuluvad hulka S, kuulub suvaline baasplokk c, mis asub teel plokist a plokki b, samuti hulka S.
- Võrgu eluajaks nimetatakse kõiki võrgu definitsioone ja kasutusi sisaldavate instruktsioonide minimaalset kumerat hulka.
- Teisisõnu, võrgu eluaeg on kõikide baasplokkide hulk, kus võrk on “elus”.
- Kaks võrku interfereeruvad kui nende eluajad kattuvad.
- Kattuvate eluaegadega võrgud tuleb paigutada erinevatesse registritesse.

Võrkude eluajad ja interferents

Võrgud s1 ja s2 interfereeruvad
Võrgud s2 ja s3 interfereeruvad



Graafi värvimine

- Kas on võimalik värvida graafi k värviga nii, et suvalised kaks kaarega ühendatud tippu ei ole sama värvi?
- Värvimise ülesanne on rohkem kui kolme tipu korral NP-täielik.
- Eksisteerivad head heuristilised meetodid.

Graafi värvimine

- Olgu antud graaf G=(V,E) ja selle graafi tipp v, nii et deg(v) < k. Graaf G on värvitav k värviga parajasti siis kui graaf G'=(V\{v},E') on värvitav k värviga.
- Seega, lisades k värviga värvitavale graafile G' tipu v, nii et deg(v) < k, saame tulemuseks k värviga värvitava graafi.

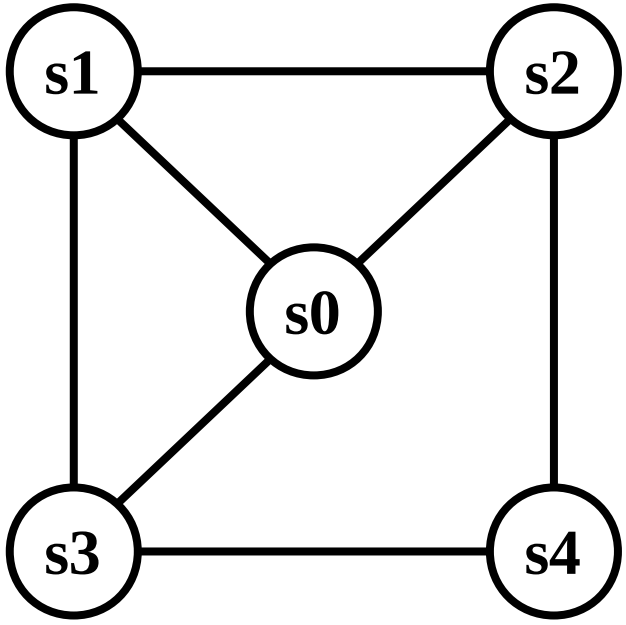
Värvimise algoritm

Graafi värvimiseks k värviga

1. Eemalda graafist ükshaaval tipud, mille aste on väiksem kui k ning pane magasini
2. Kui graaf on tühi, siis alusta värvimist:
   - Võta magasinist tipp
   - Omista tipule tema naabertippudest erinev värv (kuna tipu aste on väiksem kui k, siis peab selline värv leiduma)
   - Korda 2. punkti kuni magasin on tühi

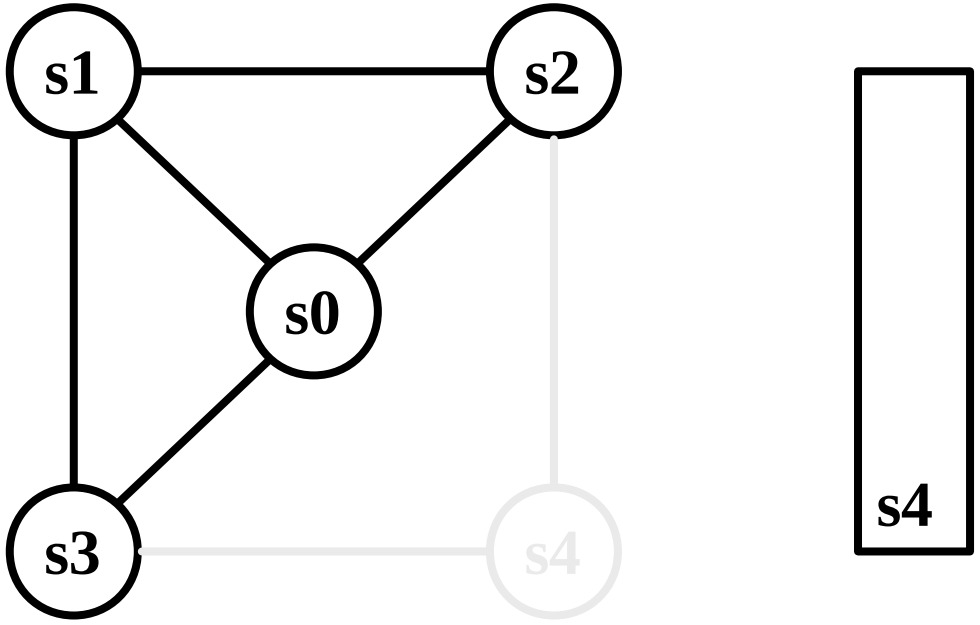
Värvimise algoritm

k = 4



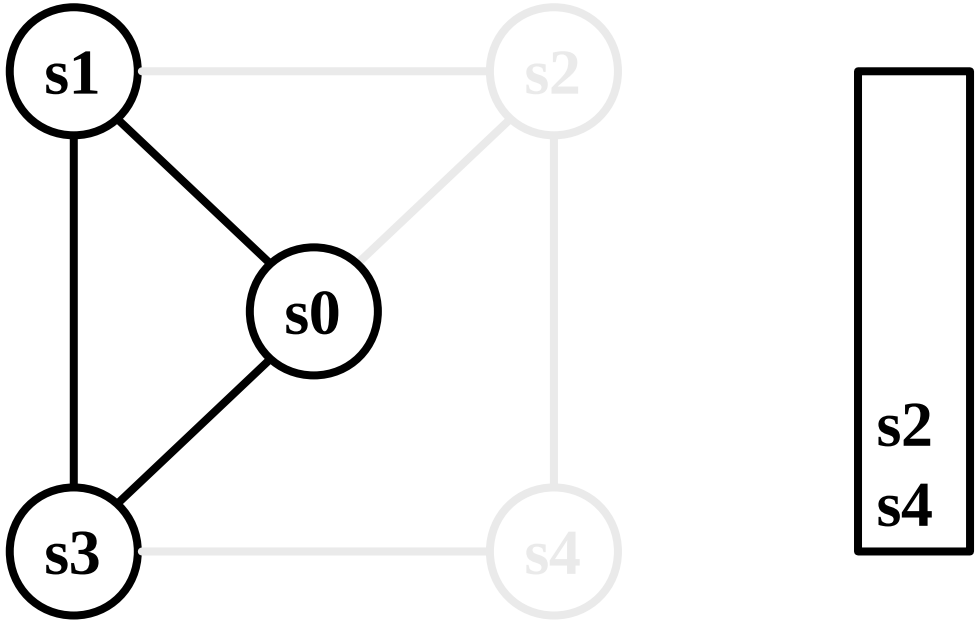
Värvimise algoritm

k = 4



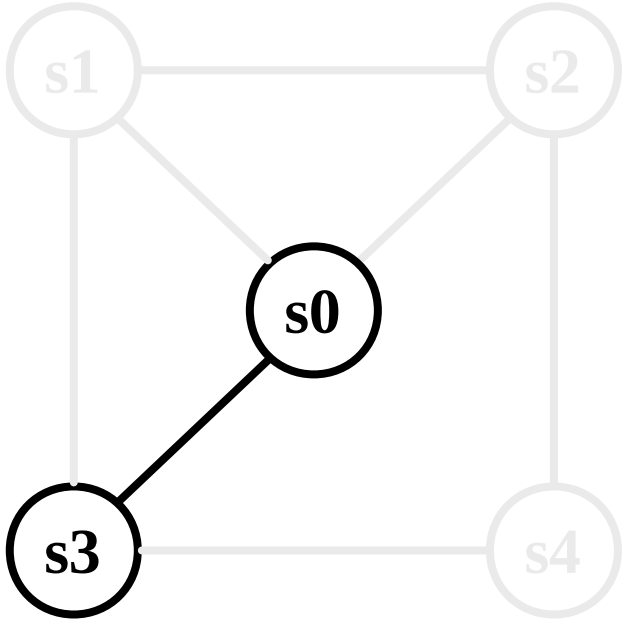
Värvimise algoritm

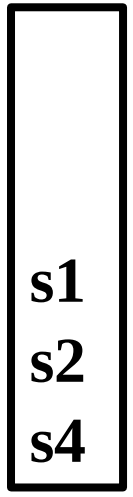
k = 4



Värvimise algoritm

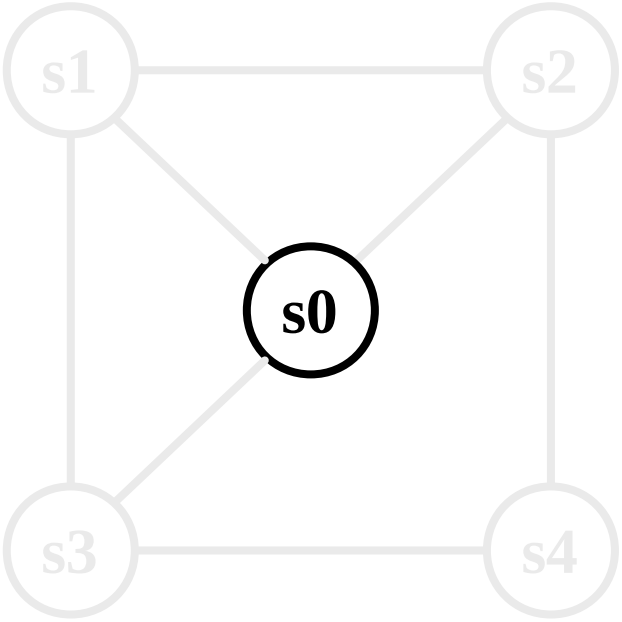
k = 4





Värvimise algoritm

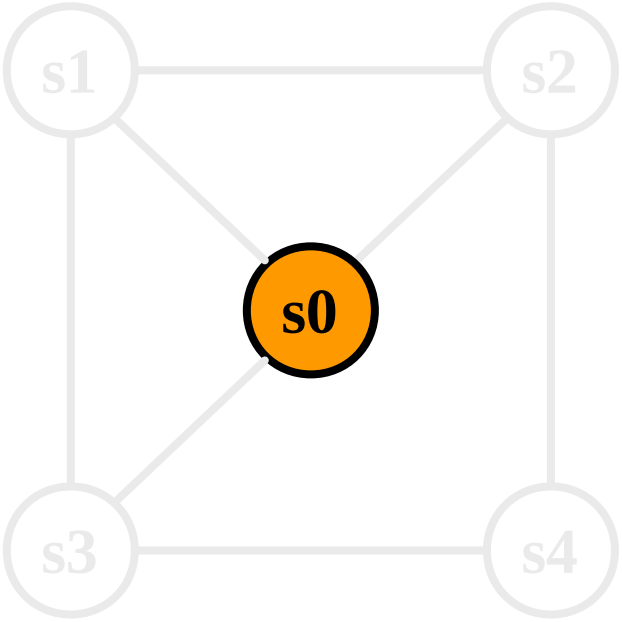
k = 4

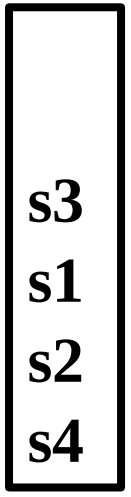


s3
s1
s2
s4

Värvimise algoritm

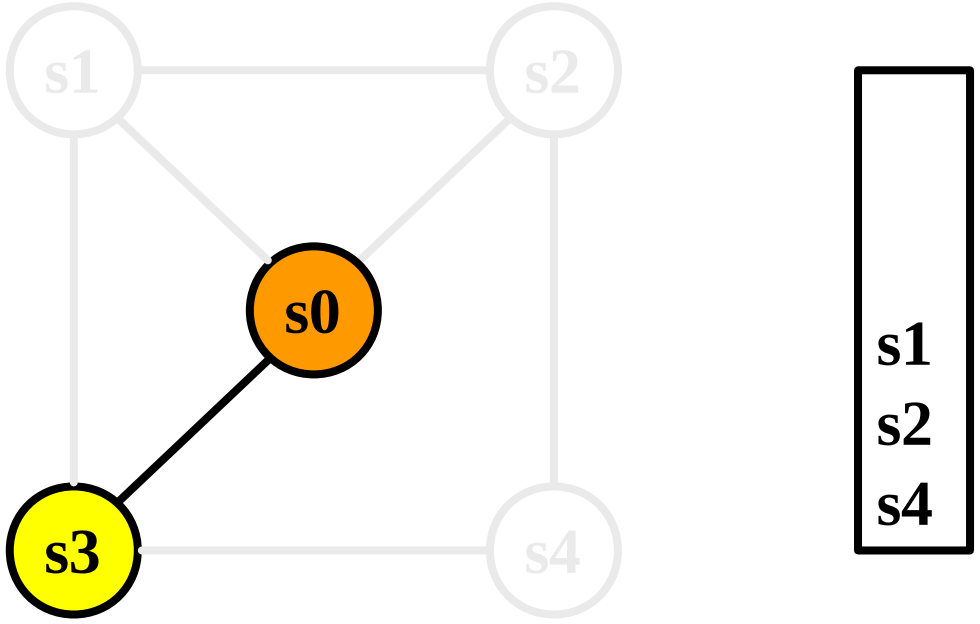
k = 4





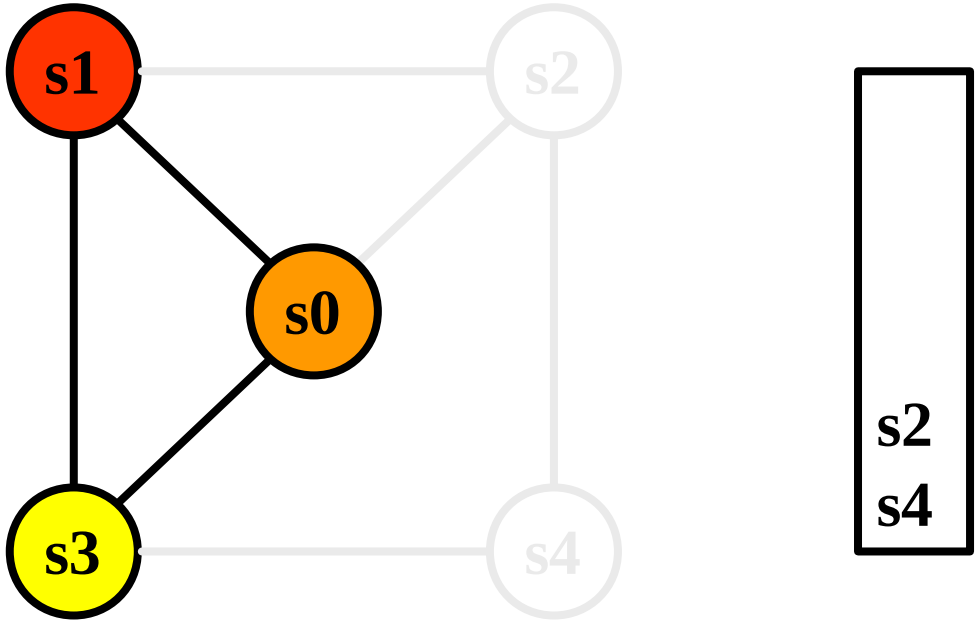
Värvimise algoritm

k = 4



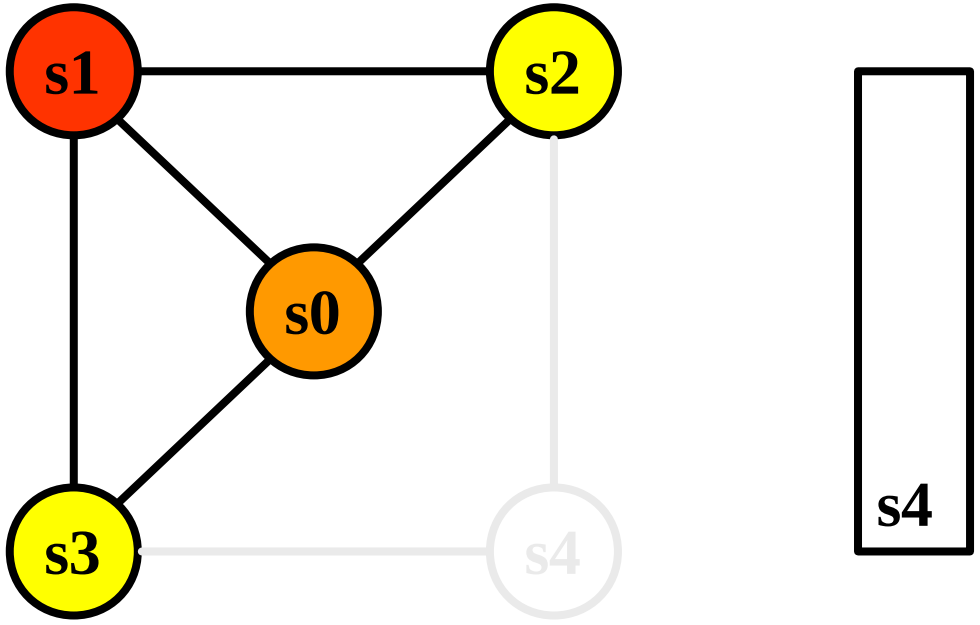
Värvimise algoritm

k = 4



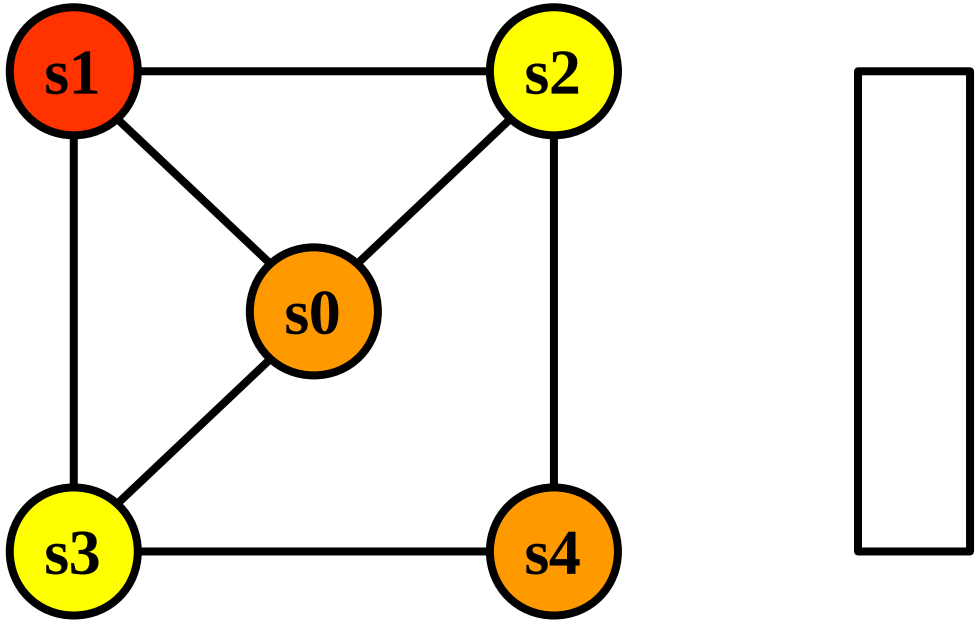
Värvimise algoritm

k = 4



Värvimise algoritm

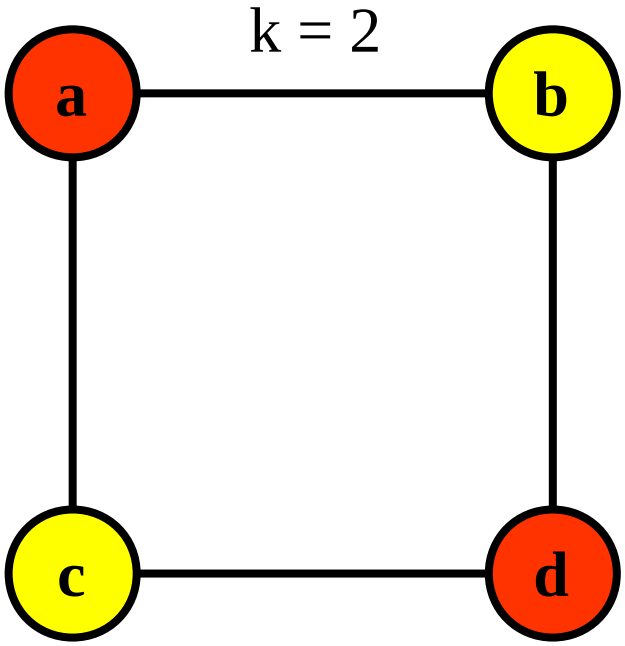
k = 4

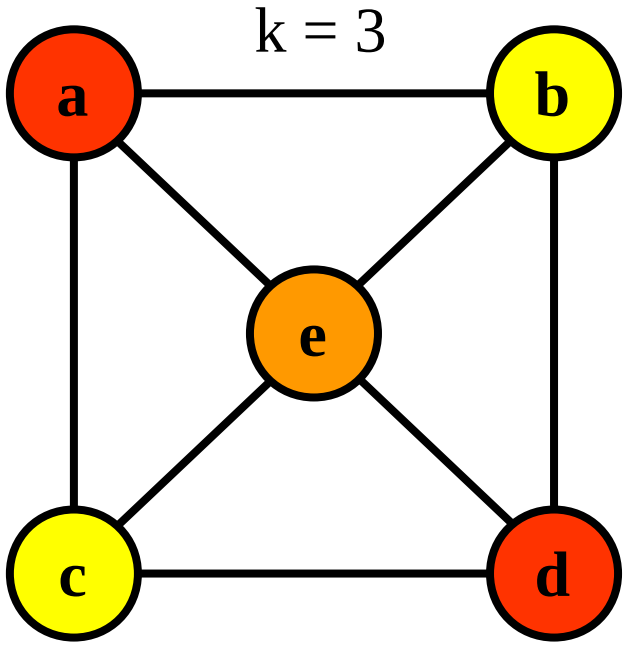


Õnnestus värvida vähem kui 4 värviga

Värvimise algoritm

Kumbki järgnevatest graafidest pole eelmise algoritmi põhjal värvitavad. Ometi leidub moodus nende värvimiseks.





Värvimise algoritm nr. 2

1. Eemalda graafist ükshaaval tipud, mille aste on väiksem kui k ning pane magasini
2. Kui ühegi allesjäänud tipu aste pole väiksem kui k, siis vali tipp *spilling*'uks. Eemalda tipp ja lisa magasini.
3. Kui graaf on tühi, siis alusta värvimist:
   - Võta magasinist tipp
   - Omista tipule tema naabertippudest erinev värv
   - Kui 2. punkti ei rakendatud, siis selline värv leidub
   - Vastasel korral võib selline värv leiduda või mitte
   - Korda 2. punkti kuni magasin on tühi

Kui erinevat värvi ei leidu

1. Kui tipu värvimiseks ei leidu naabertippudest erinevat värvi, siis
   - vali võrk, mis pannakse registri asemel mällu
   - või jaota võrk alamvõrkudeks
2. Proovi graafi uuesti värvida.
3. Korda 1. ja 2. punkti seni, kuni värvimine õnnestub.

Spilling

- Millist võrku mällu “spillida”?
  - Sellist, millelele vastavava tipu v aste deg(v)=k
  - Sellist, mille *spilling*’u maksumus on kõige väiksem
- *Spilling*’u maksumuseks nimetatakse mällu kirjutamise ja mälust lugemisega kaasnevaid lisakulutusi.

Spilling'u maksumus

- *Spilling*'u maksumuse määravad sooritatavad mällu kirjutamise ja mälust lugemise operatsioonid.
- Reaalselt ei saa sellisel viisil *spilling*'u maksumust hinnata, sest
  - me ei tea milliseid harusid pidi programm edasi liigub
  - me ei tea tsüklite kordamise arvu
  - maksumus võib sõltuda programmi sisendist

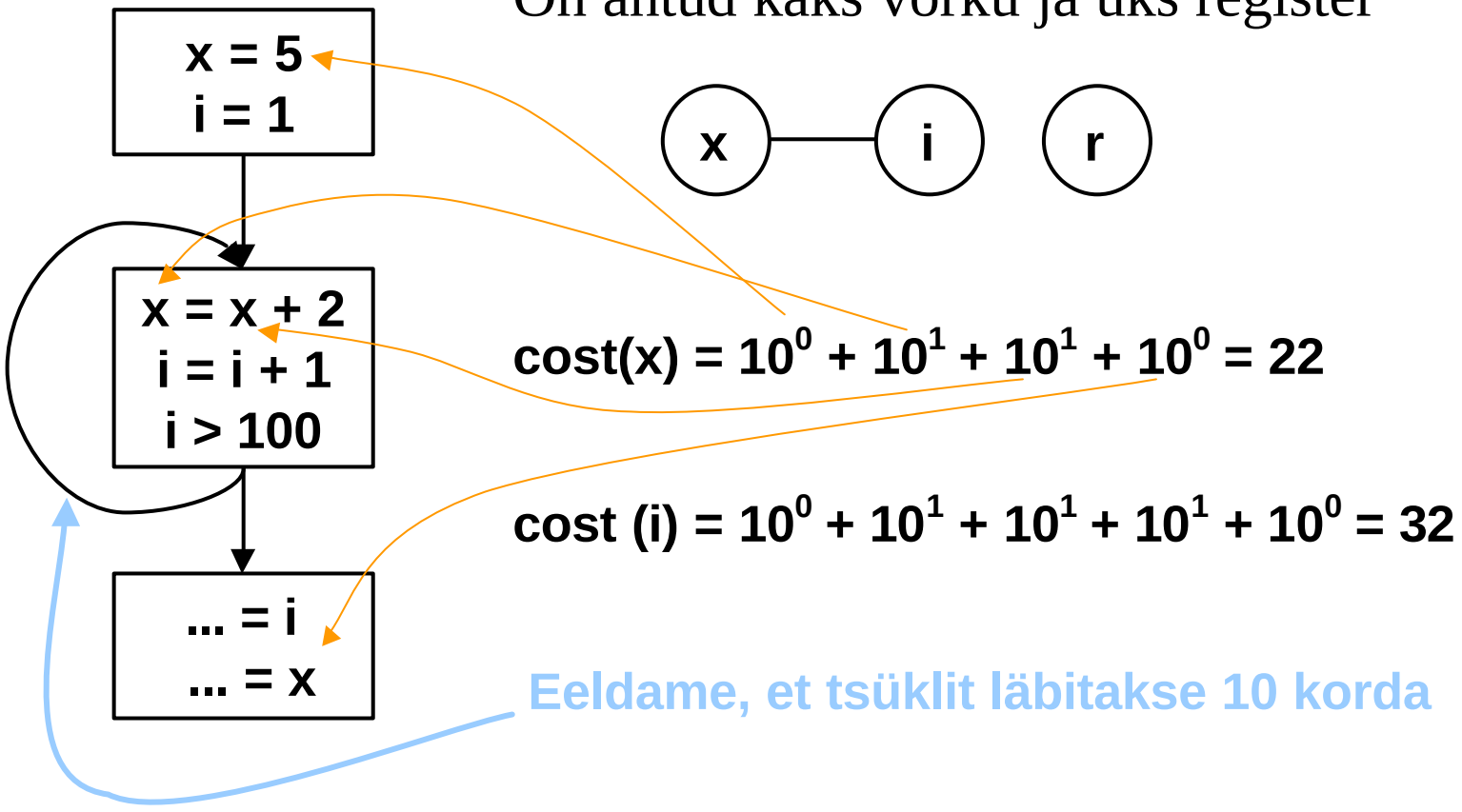
Spilling'u maksumus

- Anname tipu w *spilling*'u maksumuse ligikaudse hinnangu.

$$\mathbf{cost(w) = cost_{def} * \sum_{def\ ?\ w} 10^{depth(def)} + cost_{use} * \sum_{def\ ?\ w} 10^{depth(use)}}$$

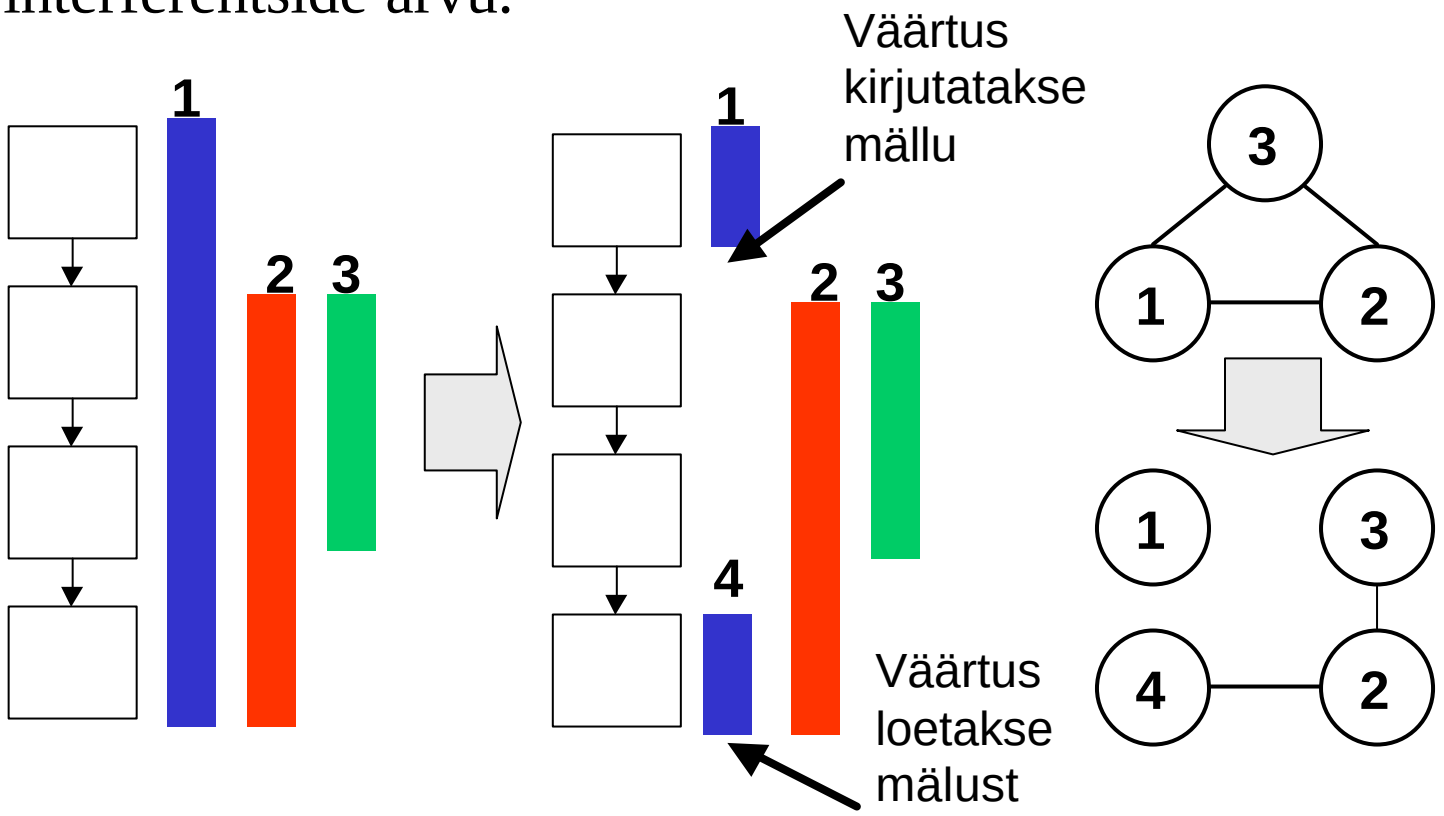
- $\sum_{def\ ?\ w} 10^{depth(def)}$ on hinnang definitsioonide esinemissagedusele, eeldades, et tsükleid korratakse keskeltläbi 10 korda.
- depth(x) näitab kui sügavale on instruktsioon pesastatud.
- Eelistamaks suure astmega tippe, võib tulemust jagada tipu astmega.

Spilling'u maksumus



Alamvõrkudeks jaotamine

- Võrgu jaotamine alamvõrkudeks võimaldab vähendada interferentside arvu.



Edasine optimeerimine

- Elimineeri kopeerimised ühest registrist teise.