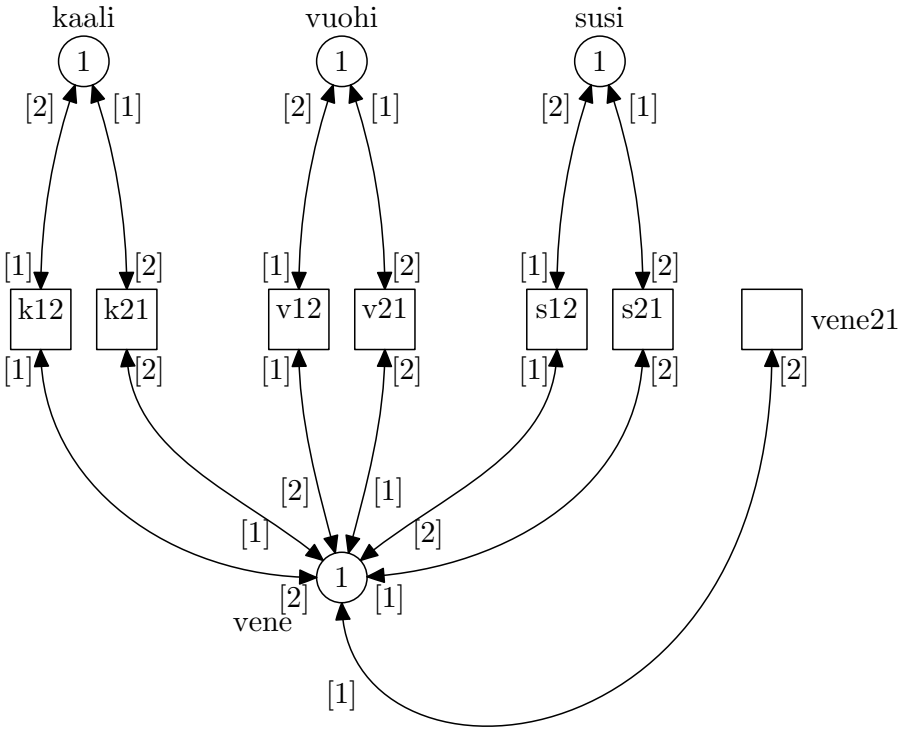
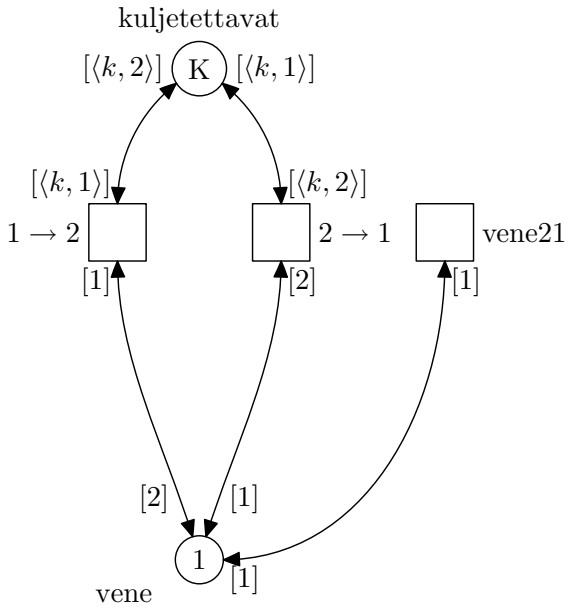
1. Seuraavassa on esitetty kaali-vuohi-susi-ongelman mallittava korkean tason verkko, joka saadaan laskostamalla kuljetettavia ja venettä kuvaavat paikat. Kaikkien paikkojen tietotyyppi on $T = \{1, 2\}$.

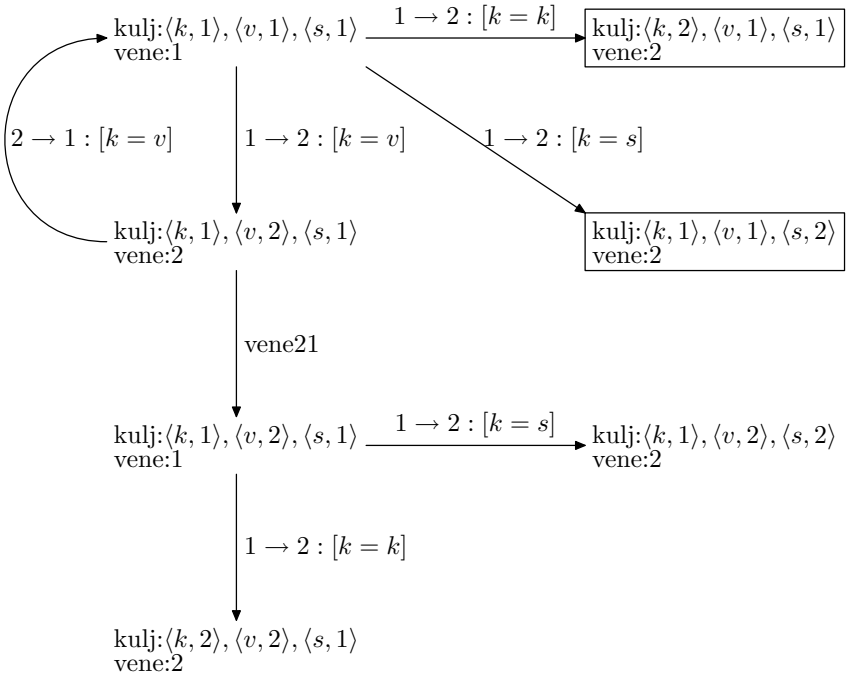


Seuraavassa kuvassa verkkoa on laskostettu lisää. Kaikki kuljetettavia kuvaavat paikat on yhdistetty, mutta venettä kuvaava paikka on vielä erillään. Merkitään kuljetettavien joukkoa $K = \{kaali, vuohi, susi\}$. Paikan kuljetettavat tietotyyppi voidaan nyt esittää $D = K \times T$.

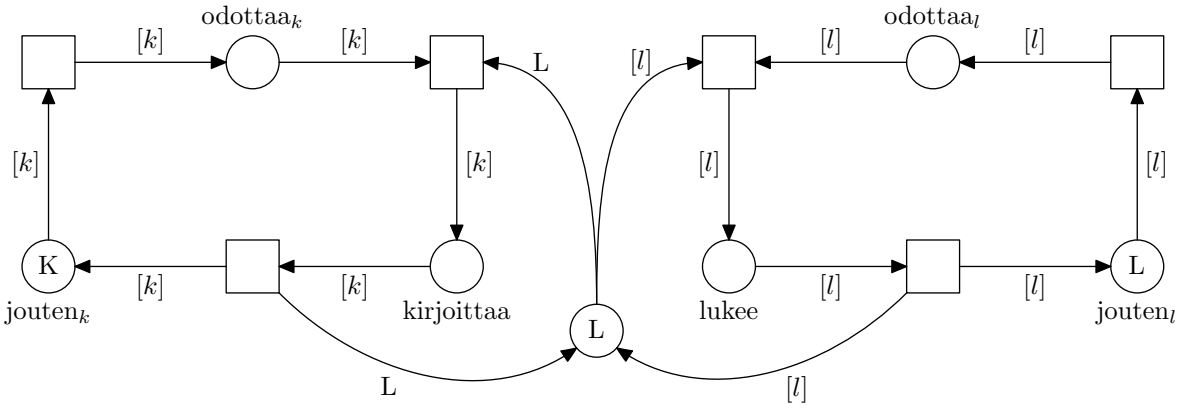
Siirtymät $1 \to 2$ ja $2 \to 1$ voitaisiin vielä yhdistää. Myös paikka vene voitaisiin yhdistää paikkaan kuljetettavat. Tällöin yhdistetyn siirtymän syötekaaren kaarilausekkeeksi tulisi $[\langle k, x\rangle, \langle vene, x\rangle]$ ja ulosmenevälle kaarelle tulisi $[\langle k, x \oplus 1\rangle, \langle vene, x \oplus 1\rangle]$, missä $x \oplus 1 = (x \bmod 2) + 1$.



Verkon saavutettavuusgraafi generoidaan periaatteessa samoin kuin matalan tason verkoillekin, mutta siirtymiä laukaistaessa on muistettava ottaa huomioon muuttujien sidonnat. Yksi korkean tason siirtymä voi siis laueta yhdestä merkinnästä usealla eri muuttujasidonnalla. Kuvassa on esitetty saavutettavuusgraafin generoinnin 3 ensimmäistä askelta. Tilansäästön vuoksi kuljetettavien nimistä käytetään vain ensimmäistä kirjainta. Kehystetyt tilat ovat hylättyjä, joten saavutettavuusgraafin generointia ei jatketa niistä eteenpäin.



2. Ongelman ratkaisu on esitetty alla. Merkitään lukijoiden joukkoa $L = [l_1, l_2, \ldots, l_m]$ ja kirjoittajien joukkoa $K = [k_1, k_2, \ldots, k_n]$. Paikkojen jouten$_l$, odottaa$_l$ ja lukee tietotyyppi on $D_l = \{l_1, l_2, \ldots, l_m\}$. Paikkojen jouten$_k$, odottaa$_k$ ja kirjoittaa tietotyyppi on $D_k = \{k_1, k_2, \ldots, l_n\}$. Paikan, joka säätelee lukemista ja kirjoittamista, tietotyyppi on $D_l = \{l_1, l_2, \ldots, l_m\}$.



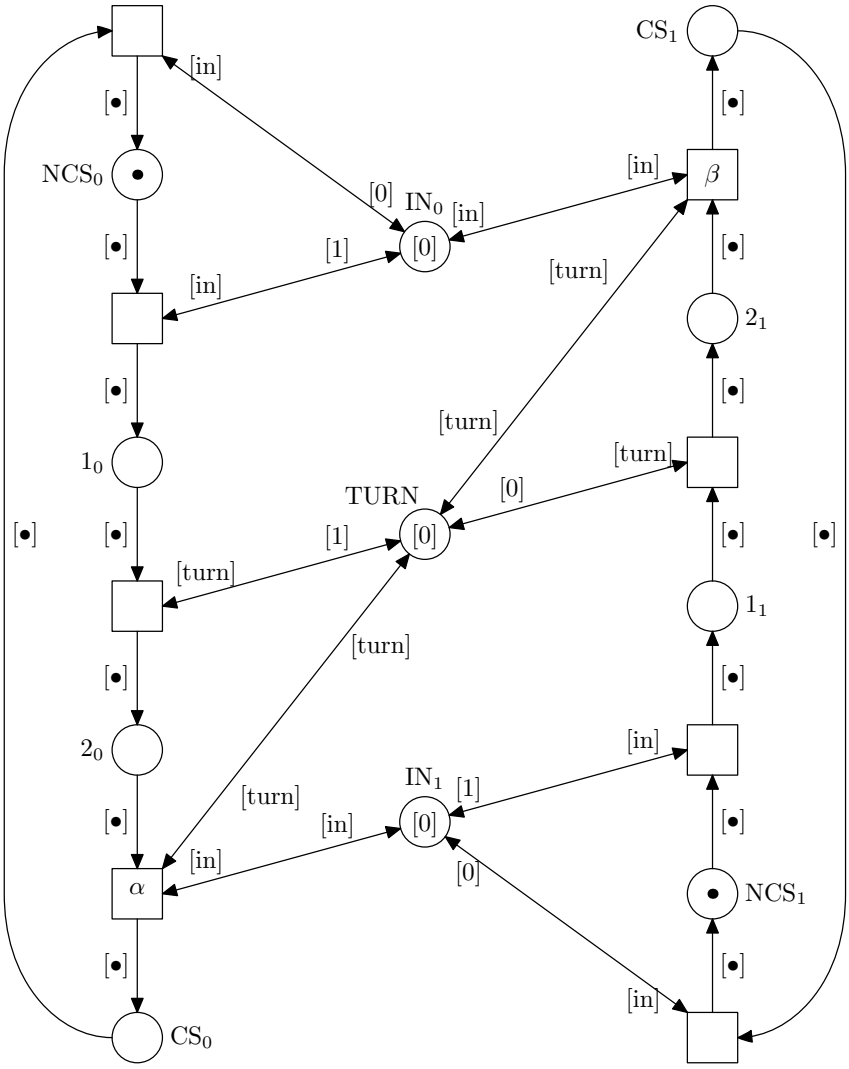
3. Algebrallinen verkko, joka mallintaa Petersonin algoritmia, on kuvattu alla. Laukeamisehto α on $\text{in} = 0 \vee \text{turn} = 0$ ja ehto β on $\text{in} = 0 \vee \text{turn} = 1$.

   Mallin oikeellinen toiminta voitaisiin tarkastaa fakta-siirtymällä, jonka esipaikoiksi tulisi CS_0 ja CS_1.

   Jos käytetään reject-kaavaa, sopiva kaava olisi esimerkiksi

```
reject place cs0 equals is token_t {} &&
       place cs1 equals is token_t {}
```

   Edellä on oletettu, että musta merkki (•) on määritelty seuraavasti:
   `typedef struct {} token_t;`



4. Seuraavassa kuvassa on esitetty yksinkertaistettu huoltoasemaa mallintava paikka-siirtymä-järjestelmä. Kuvaan on myös lisätty kohdan b) muutokset, eli paikan “space at pump free” alkumerkintä on kasvatettu kahdesta neljään, ja paikan “cashier free” alkumerkintä on kasvatettu yhdestä kahteen. Paikassa “pump free” on kaksi merkkiä, sillä se kuvaa nyt molempia huoltoaseman pumppuja.

   Malli voidaan muuntaa äärellistilaiseksi lisäämällä siihen paikka, joka rajoittaa kunkin paikan suurimman mahdollisen merkinnän johonkin äärelliseen lukuun n. Paikka tulee siirtymän “car leaves filling station” jälkipaikaksi ja siirtymän “car enters filling station” esipaikaksi. Ko. paikka siis rajoittaa siirtymän “car enters filling station” laukeamisia. Jos mallinnetaan n:n auton varikon tankkauspaikkaa, voidaan rajoittavaan paikkaan laittaa alkumerkinnäksi yhtä monta merkkiä kuin varikolla on autoja.