

(19)



SUOMI - FINLAND

(FI)

PATENTTI- JA REKISTERIHALLITUS
PATENT- OCH REGISTERSTYRELSEN
FINNISH PATENT AND REGISTRATION OFFICE

(10) **FI 930435 A7**

(12) **JULKISEKSI TULLUT PATENTTIHAKEMUS
PATENTANSÖKAN SOM BLIVIT OFFENTLIG
PATENT APPLICATION MADE AVAILABLE TO THE
PUBLIC**

(21) Patentihakemus - Patentansökan - Patent application **930435**

(51) Kansainvälinen patenttiluokitus - Internationell patentklassifikation -
International patent classification
**G06F 9/44
G06F 15/82**

(22) Tekemispäivä - Ingivningsdag - Filing date **01.08.1991**

(23) Saapumispäivä - Ankomstdag - Reception date **01.02.1993**

(41) Tullut julkiseksi - Blivit offentlig - Available to the public **02.04.1993**

(43) Julkaisupäivä - Publiceringsdag - Publication date **13.06.2019**

(86) Kansainvälinen hakemus - **01.08.1991 PCT/SE1991/000517**
Internationell ansökan - International
application

(32) (33) (31) Etuoikeus - Prioritet - Priority

02.08.1990 SE 9002558

(71) Hakija - Sökande - Applicant

1 • Carlstedt Elektronik AB, Industrivägen 55 433 61 Partille, Sverige, SVERIGE, (SE)

(72) Keksijä - Uppfinnare - Inventor

1 • Carlstedt, Lars Gunnar, Sverige, SVERIGE, (SE)

(74) Asiamies - Ombud - Agent

Berggren Oy Ab, Antinkatu 3 C, 00100 Helsinki

(54) Keksinnön nimitys - Uppfinningens benämning - Title of the invention

Aktiivinen muistiväline redusointiprosessorissa

Aktivt minnesorgan i en reduktionsprocessor

Aktiivinen muistiväline redusointiprosessorissa - Aktivt minnesorgan i en reduktionsprocessor

5

Tietokoneet keksittiin 1940-luvulla. Sen jälkeen niitä on kehitetty mullistavalla vauhdilla. Siitä huolimatta nykyisissä tietokoneissa on lähes sama arkkitehtuuri kuin ensimmäisissä.

10

Enimmät parannukset on tehty laitteistopuolella. VLSI:n tulo ja litografian parannukset ovat mahdollistaneet yhden sirun tietokoneiden rakentamisen, joita vasta viisi vuotta sitten kutsuttiin supertietokoneiksi. Viivaleveys on kutistunut eksponentiaalisesti ja on nyt alle yhden mikrometrin. Kellotaajuudet samoin kuin aktiivisten transistoreiden lukumäärä ovat kasvaneet monta kertaluokkaa. Fyysiset rajoitukset tulevat todennäköisesti rajoittamaan viivaleveyden 0,2 mikrometriksi.

15

Tietokoneiden arkkitehtuurit eivät samalla ole parantuneet piin hyödyntämisen osalta. Päin vastoin, nopeuden lisäämiseksi useimmissa tietokoneissa on käytetty enemmän kuin optimaalinen määrä piitä.

20

Molemmat nämä tekijät tulevat pysäyttämään erillisten prosessorien nopeuden kehittymisen seuraavan viiden vuoden kuluessa. Käyttöön tulee rinnakkaisprosessoreita, joka kasvaneen mutkikuuden takia johtaa kasvaneisiin laitteistokustannuksiin, ja useimmilla ohjelmatyypeillä ohjelmointikustannusten huikeaan kasvuun.

25

30

Keskenään verrattuna laitteistokustannukset ovat kutistuneet, mutta uusien järjestelmien ohjelmointikustannukset ovat huomattavasti kasvaneet, ja ovat pian haitallisen korkealla tasolla.

35

Tietokone on erilaisten ohjelmisto- ja laitteistoyksiköiden mutkikas kokoonpano. Kehityksen eri paradigmat ja vaiheet ovat luoneet standardit - ad hoc ja laaditut -, jotka ovat hajautuneet

järjestelmään. Tästä epäyhdenmukaisuudesta johtuen on olemassa suuri määrä rajapintoja.

5 Kaikki nämä eri laatua ja tyyppiä olevat rajapinnat ja paradigmatt
ovat tehneet koneen käyttämisen vaikeaksi käyttäjälle tai ohjelmoijalle - se vaatii suuren määrän tietoa -, ja mutkikkaudesta johtuen ohjelmoija voi luoda piileviä virheitä.

10 Viime aikoina on kuitenkin syntynyt ns. redusointiprosessoreita. Redusointiprosessori suorittaa ohjelman, jolla on määrätty struktuuri aritmeettiset lausekkeet mukaanlukien, ja tätä struktuuria supistetaan useammassa redusointivaiheissa. Siten ohjelmaa ei suoriteta määrättyssä järjestyksessä, kuten toisen tyyppisissä tietokoneissa.

15 Eräitä vaikeuksia on esiintynyt redusointiprosessoreiden kehittämiseksi määrätyn rajallisen koon yli.

Ohjelmointikielten kehitys

20 Ensimmäisen elektronisen tietokoneen kehittäminen käynnisti useampien tämän tyyppiselle tietokoneelle soveltuvien ohjelmointikielten kehittämisen, kuten esimerkiksi FORTRAN, COBOL, Algol, BASIC, Pascal. Näitä kieliä on kutsuttu imperatiivisiksi
25 kieliksi, joita alla myös sanotaan tavanomaisiksi kieliksi, pääasiassa koska ne tavallisesti tuottavat ohjelmia, jotka muodostuvat tavanomaisella tietokoneella, ts. John von Neumann:in kehittämien periaatteiden mukaan suunnitellulla tietokoneella peräkkäin suoritettavien komentojen tai käskyjen
30 sekvenssistä. Kasvavat hankaluudet näiden kielten kanssa johtivat toisen kielisarjan kehittämiseen: LISP, ISWM, Scheme (LISP-murre), ML, Hope, SASL, jne. Näiden kielten kehittämisen pontimena oli ajatuksen yksinkertaisuus; mikään määrätty kone ei vaikuttanut rakenteeseen. Kesti jonkin aikaa, ennenkuin
35 funktionaaliset kielet alkoivat saada huomiota osakseen - eräänä syynä oli, että funktionaaliset kielet olivat suoritettaessa hitaita. Viimeisin kehitys on osoittanut, että suoritusnopeus

tavanomaisilla tietokoneilla voi eräissä tapauksissa olla lähes sama tai sama kuin tavanomaisilla (imperatiivisilla) kielillä tehdyillä ohjelmilla, vaikka näitä funktionaalisia ohjelmia ei ole tarkoitettu suoritettaviksi tämän tyyppisillä tietokoneilla.

5

Ohjelmistokriisi

Erittäin suuret ponnistukset funktionaalisten ohjelmien kehittämiseksi käynnistyivät imperatiivista tyyppiä olevien kielten kasvavan epämukavuuden johdosta. Noin vuonna 1970 alettiin puhua ohjelmistokriisistä. Ohjelmat tulivat yhä mutkikkaammiksi ja niissä oli paljon virheitä, niitä oli vaikea lukea, vaikea ymmärtää ja erityisen vaikea muuttaa. Eräänä syynä tähän on, että imperatiivisten ohjelmien ohjelmoinnin yksinkertaistamiseen kohdistuvat odotukset asetettiin liian korkealle - nämä ohjelmat eivät olleet niin korkealla tasolla, eli abstraktiotasolla, kuin miltä saattoi näyttää. Imperatiiviset kielet ovat edelleenkin varhaisiin tietokonekonsepteihin, von Neumannin tyyppiä oleviin tietokoneisiin sovitettuja, ja ohjelmointitaso on edelleen varsin lähellä konetasoa. Funktionaalisilla ohjelmointikielillä on monia ominaisuuksia, joilla lievennetään tavanomaisempien ohjelmointikielten eräitä puutteita.

Lisätietoa ja asian ymmärtämistä varten viitataan oppikirjaan "Functional Programming Using Standard ML", Åke Wikström, Prentice Hall 1987.

Keksinnön kaikkien tavoitteiden ja etujen ymmärtämiseksi on oleellista ymmärtää mitä funktionaalinen lähestymistapa tietokonalalla merkitsee. Erityisesti historiallisesti hallitsevaan imperatiiviseen lähestymistapaan verrattuna.

Ilmaisulla "funktionaalinen lähestymistapa" tarkoitetaan, että ohjelmat kirjoitetaan funktionaalisella kielellä ja talletetaan tietokoneelle ja suoritetaan tietokoneella, joka käsittää erityisesti näitä kieliä varten sopivan laitteiston. Samalla tavalla ilmaisu "imperatiivinen lähestymistapa" on tarkoitettu

merkitsemään, että ohjelmat kirjoitetaan funktionaalisella kielellä ja talletetaan tietokoneelle ja suoritetaan tietokoneella, joka käsittää imperatiivisia kieliä varten sopivan laitteiston.

5

On kuitenkin mahdollista tallettaa ja suorittaa funktionaalisella kielellä kirjoitettuja ohjelmia tavanomaisella tietokoneella. Myöskin päinvastainen on mahdollista - imperatiivisella kielellä kirjoitetut ohjelmat voidaan suorittaa tietokoneella, joka sopii funktionaalisella kielellä kirjoitetuille ohjelmille.

10

Funktionaalisten kielten ominaisuuksia

Funktionaalisella kielellä kirjoitettu ohjelma voidaan nähdä olioiden ominaisuuksien määritelmäjoukkona ja laskentasääntöinä. Määritelmät muodostavat selittävän osan ja laskentasäännöt, tai redusointi- tai uudelleenkirjoitussäännöt, toimintaosan, joita tietokone käyttää suorituksen aikana. Funktionaaliset kielet muodostavat tietokoneessa korkeamman tason rajapinnan, jonka avulla ohjelmioija voi abstraktioissa vapautua tietokoneen laitteisto-riippuvista yksityiskohdista. Funktionaalisten ohjelmien positiivisena sivuvaikutuksena ne usein ovat lyhyemmät ja helpommat ymmärtää kuin tavanomaiset imperatiiviset ohjelmat. Funktionaalisten ohjelmien eräänä suurimpana haittana on, että funktionaaliset ohjelmat on käännettävä tavanomaiselle kielelle, jotta ne voitaisiin suorittaa tavanomaisella tietokoneella. Tämä on tehty kääntäjillä tai tulkitsevilla ohjelmilla. On selvää, että eräitä funktionaalisen lähestymistavan etuja on haitannut seikka, ettei funktionaalisten ohjelmien tehtävien tallettamista ja tehokasta suorittamista varten ole ollut mitään ominaista laitteistoa.

15

20

25

30

35

Selittävän kielen ohjelmia varten olevaa tietojenkäsittelyjärjestelmää selitetään patenttijulkaisussa GB-A-2,227,583. Ohjelma käännetään pakettien kokoelmana, joista jokainen edustaa funktion soveltamista ainakin yhden argumentin osalta. Kun funktiota on sovellettava, käytetään erityisiä epätäydellisiä

5 pakettityyppejä. Näitä käsitellään täydellisen paketin kokoami-
 seksi, joka sisältää suoran viittauksen funktioon ja joka
 sisältää kaikki tätä funktiota varten vaaditut argumentit.
 Paketit talletetaan paikalliseen muistiin, joka on kytketty
 10 prosessoriin. Kun paketteja on käsiteltävä, pakettiajoitin
 välittää osoitteita pakettien käsittely-yksikölle, joka pääsee
 käsiksi paikallismuistissa oleviin paketteihin ja käsittelee
 niitä, esimerkiksi kirjoittamalla ne uudelleen tai muuttamalla
 tyyppiä ja tyyliä, sekä kirjoittamalla ne uudelleen ja muutta-
 15 malla muiden pakettien tyyppiä ja tyyliä, luomalla uusia
 paketteja, tai poistamalla paketteja. Rajapintaverkko muihin
 prosessoreihin sisältää tulosanomajonon.

15 Järjestelmämuistin ja allokaattorin kehitystä selitetään pa-
 tenttijulkaisussa US-4,598,361, joka liittyy tapaan jolla
 osoitetaan muistipaikkoja, joihin voidaan tallettaa redusoituja
 listoja. Allokaattori pyyhkii muistipaikkojen valittuja ryhmiä
 rinnakkain, nähdäkseen onko käyttämättömiä muistipaikkoja ole-
 massa, ja sijoittaa sitten näiden käyttämättömien muistipaikko-
 20 jen osoitteet jonoon redusointiprosessorilla käytettäväksi.
 Pyyhkiminen tehdään käyttäen muistin jokaisen muistipaikan
 merkkibittiiä. Merkkibitti on asetettu, jos muistipaikka on
 allokoitu käyttöä varten, ja muussa tapauksessa se on nollattu.
 Isäntäprosessori tukee datasiirron laajaa valikoimaa isäntätie-
 25 tokoneelle. Graafimanagerin ohjauslohko, joka vastaanottaa
 signaaleja isäntäprosessorilta, sisältää mikro-ohjelmia ja on
 yhteistyössä ohjausrekisterin kanssa.

MÄÄRITELMIÄ

30

Alla on luettelo tässä selityksessä käytetyistä ilmaisuista ja
 niille varatuista merkityksistä:

elementti (element)

35

osa jotain laajempaa datastruktuuria

lista (list)

elementtien järjestyksessä oleva jono, jolloin jokainen

elementti vuorostaan voi olla lista

liitetty lista (inserted list)

listan osa, joka on riittävän pieni talletettavaksi kokonaisuudessaan yhteen solukehikkoon. Mahdollistaa

5 mielivaltaisen pitkien listojen esittämisen

kehikko (closure)

hierarkkisesti strukturoitu kokonaisuus, joka määrittelee prosessin. Kaikilla kehikoilla on juuri, joka

yksiselitteisesti määrittelee kehikon. Redusointikoneessa

10 redusointi kohdistuu kehikoihin. Redusoinnit muuttavat

koneen kokonaistilaa

oliomuisti (object storage)

muisti, joka sisältää olioita tallettavia muistisoluja.

Esimerkiksi assosiatiivinen muisti

15 muistisolu (storage cell)

oliomuistin solu. Siihen talletetaan kehikko, joka voi

viitata toisiin solukehikoihin, jotka on talletettu

toisiin muistisoluihin

solukehikko (cell closure)

20 muistisolun sisältö

muistisolukenttä (storage cell field)

muistisolussa oleva kenttä

kehikkoelementti (closure element)

muistisolukenttään talletettu dataelementti

25 kehikkotunnistin (closure identifier)

kehikkosolun elementti, joka yksiselitteisesti osoittaa kehikon

kanoninen kehikko (canonical closure)

kehikko, jota ei enää voi redusoida, ts. solukehikko,

30 joka ei sisällä sellaisia kehikkotunnistimia, jotka

osoittavat johonkin toiseen solukehikkoon, jota vielä

voisi redusoida sillä tavalla, että tätä solukehikkoa

olisi vielä redusoitava

tavoite (goal)

35 suoritettava, ts. redusoitava kehikko

isä (father)

kehikko, jossa on ainakin yksi kehikkotunnistin

arvo/tunnistin-kentässä

poika (son)

toiseen kehikkoon kehikkotunnistimen avulla linkitetty

kehikko, jolloin tämä kehikkotunnistin osoittaa poikaa.

- 5 Poika voi myös olla isä. Isä voi myös olla poika. Pojalla voi olla useampia kuin yksi isä. Isällä voi olla useampia kuin yksi poika.

kehikon asema (closure position)

- 10 osoittaa, onko kehikko juuri tai solmu

juuri (root)

kehikkopuun ylin kehikko

solmu (node)

kehikkopuun sellainen solmu, joka ei ole juuri

- 15 missä (where)

muistisolu, joka sisältää kehikon aseman

tyyppi (type)

solukehikon tyyppikoodi, ts. bittikuvio, joka edustaa olion ominaisuutta, esim. käskykoodia

- 20 joutilas (lazy)

solukehikon elementti, joka osoittaa onko se suoritettavissa, myöhemmäksi siirretty arviointi, tai ei-aktiivinen

tunnistin (identifier)

- 25 erityistä lajia oleva kehikkoelementti, jota käytetään muistisoluun talletetun olion osoitteen merkitsemistä varten

ympäristö (environment)

olioita voidaan ryhmitellä antamalla niille sama ympäristö

- 30 arvo/tun. (value/des.)

kehikkoelementti, johon on talletettu joko yksinkertainen arvo, ts. välitön esitys, ei mitään (nothing), tai osoitus toiseen kehikkoon, ts. välillinen esitys

ydinsolu (core cell)

- 35 keksinnön mukainen strukturaalinen aritmeettinen yksikkö. Ydinsolu pystyy suorittamaan strukturaalista aritmetiikka, johon liittyy kehikoiden redusointia

numeerinen ALU (numeric ALU)

numeerinen aritmeettinen yksikkö, joka pystyy suorittamaan numeerisia ja loogisia perusoperaatioita.

Ydinsolu käyttää hyväksi numeerista ALU:a numeerisia

5 operaatioita varten

täysrekisteri (full register)

rekisteri, joka ulottuu ydinsolun kaikkien tasojen läpi ydinsana (core word)

ydinsolun täysrekisterin sisältö

10 rajoitettu rekisteri (limited register)

ydinsolun rekisteri, joka ulottuu rajallisen tasomäärän läpi ja joka on mitoitettu sisältämään

arvo/tunnistin-tyyppiä olevan kehikkosolun elementin elementtisana (element word)

15 sisältö rajoitetussa rekisterissä tai täysrekisterin osassa, jolla on sama laajuus kuin rajoitetulla rekisterillä

num-sana (num word)

elementtisanan osa, joka edustaa arvoa tai tunnistinta

20 merkitsin-sana (tag word)

elementtisanan osa, jossa on num-sanan esityksen ominaisuutta osoittava merkitsin

redusointi (reduction)

kehikon uudelleen kirjoittaminen/uudelleen

25 strukturoiminen käytetyn määrätyn ohjelmointikielen sääntöjen mukaisesti

käyttäytyminen (behaviour)

aikastrukturi, kuten sekvenssi tai portissa

yhdistettyjen sekvenssien vaihtoehto. Käytetään myös

30 degeneroidusti samaan tapaan kuin matemaattista termiä tyhjä joukko (empty set) joukko-opissa, niin että "1" voisi periaatteessa olla käyttäytyminen.

Keksinnön päätarkoituksena on aikaansaada tietojenkäsittelylaitte, jolla on selittävä ohjelmointikieli.

35

Keksinnön toisena tarkoituksena on aikaansaada redusointi-

prosessorina toimiva tietojenkäsittelylaite.

Keksinnön eräänä toisena tavoitteena on aikaansaada tietojenkäsittelylaite, jolla on yksinkertainen ohjelmointikieli, ja jossa
 5 ohjelmoija voi abstraktioissa vapautua kaikesta laitteistosta riippuvista yksityiskohdista, ja joka tuottaa ohjelmia, joiden kirjoittaminen vie vähemmän aikaa ja jotka sisältävät vähemmän virheitä.

10 Keksinnön eräänä toisena tavoitteena on aikaansaada tietojenkäsittelylaite, jossa sama kieli toimii ohjelmointikielenä, käyttöjärjestelmänä ja tietoliikenneprotokollana.

Keksinnön eräänä toisena tavoitteena on aikaansaada tietojenkäsittelylaite, joka pystyy suorittamaan funktionaalista kieltä,
 15 jonka sitovana menetelmänä on yhdistäminen.

Edellä olevat tarkoitukset on oleellisesti saavutettu muodostamalla tietojenkäsittelylaite, jossa:

20

a) muistiväline on aktiivinen, niin että muistisoluvälineiden joukossa jokaisella on tietoa, joka voi aiheuttaa operaation suorittamisen;

25

b) ainakin yksi porttiväline on kytketty aktiiviiseen muistivälineeseen, ja jossa vertailuvälineet ympäristövälineen jossakin porttivälineessä tarjoaman datan vertaamiseksi yhteen mainittuun muistisoluvälineeseen talletetun datan kanssa, jolloin datassa on mahdollisia määrittelemättömiä dataelementtejä tai
 30 osittain määriteltyjä dataelementtejä.

35

Se sisältää edullisesti välineet porttivälineeseen järjestetyn signaalisekvenssin vertaamiseksi ainakin yhteen aktiivisen muistivälineen muistisoluun talletetun sekvenssin kanssa, jolloin talletetussa sekvenssissä on mahdollisia määrittelemättömiä sekvenssielementtejä, sekä välineet vertailevan sekvenssin uudelleen kirjoittamiseksi tyhjään (nothing), joka ts. edustaa

ristiriitaa, jos vertailu tuottaa selvän eron, tai mainitun vertailevan sekvenssin uudelleen kirjoittamiseksi määrättyyn sekvenssiin.

- 5 Vertailuvälineet voisivat suorittaa ryhmien vertailua, jotka sisältävät ainakin kaksi elementtiä, joissa on ennalta määrätty lukumäärä listaelementtejä.

- 10 Signaalisekvenssi on esitetty järjestettynä ajan mukana muuttuvana näytteistettynä signaalina, jolla on yksilölliset näytteenottojaksot, jolloin signaalisekvenssi on elementtiryhmiensä lista, jolloin jokainen ryhmä sisältää kestoajan ja ainakin yhden signaalikvantiteetin tämän ajan kuluessa.

- 15 Voitaisiin järjestää väline yhden mainitun porttivälineen tahdistamiseksi ainakin yhteen toiseen mainituista porttivälineistä ja mainituista porteista jokaisen kestoajan puitteissa tulevien signaalikvantiteettien rinnakkaisen osoituksen aikaansaamiseksi. Tähän välineeseen voitaisiin johtaa eri porttien
20 signaalikvantiteetteja samalla hetkellä ja mahdollisesti verrata niitä toisiinsa.

- Muistisoluvälineet on edullisesti järjestetty struktuuriksi aktiivisessa muistissa, joka on sovitettu tietokoneohjelman
25 tallettamiseksi abstraktin syntaksin eksplisiittisen tai implisiittisen koodauksen muodossa, jolloin syntaksi lausekkeiden avulla kuvaa erilaisten abstraktien olioiden joukkoa, jolloin jokainen muistisoluväline pystyy tallettamaan ainakin osan yhdestä mainituista syntaksilausekkeista kerrallaan sopivan
30 data- ja/tai ohjelmastruktuurin muodossa.

- Jokaisella lausekelajilla voisi myös olla vastaava lausekelaji, joka ilmaisee, että se on ohjelmamuoto, jolloin kaikki ohjelma-
muodot ovat lausekkeita, jotka on sovitettu redusoitaviksi
35 itseensä ja siten pysymään paikallaan. Tämä mahdollistaa tulkitsimien ja kääntäjien kirjoittamisen kielten muita mieltävaltaisia lajeja varten.

Uudelleen kirjoittaminen tarkoittaa mainittujen muistisolujen kanssa tapahtuvaa vuorovaikutusta, mukaanlukien syntaksilausekkeiden uudelleenkirjoittamista, jolloin syntaksilausekkeiden uudelleenkirjoitus tapahtuu ennalta määrättyjen uudelleen kirjoittamisen sääntöjen mukaisesti.

Keksinnön mukainen tietojenkäsittelylaite on edullisesti tietokone, jossa selittävä ohjelma on toteutettu laitteistossa, jolloin ohjelmointikieli on myös konekieli. Keksinnön mukaisen tietojenkäsittelylaitteen langoitus voitaisiin kuitenkin, samoin kuin jokainen elektroninen laite nykyisin, simuloidan tavanomaista lajia olevassa tietokoneessa, joka on varustettu tulkilla tai kääntäjällä. Esimerkkejä sellaisista koneista ovat M68000-prosessoriin perustuvat, esimerkiksi Sun3. Keksinnön mukaan koneella on kuitenkin portti, johon on järjestetty yhdistämismahdollisuudet.

Suoritus redusoinnin avulla

Suoritettava ohjelma voidaan esittää kehikoiden suunnatulla graafilla, jossa ohjelman jokainen osa on esitetty kehikolla. Suorituksen aikana tämä kehikoiden suunnattu graafi redusoidaan asteittain käytetyn kielen redusointisääntöjen mukaisesti. Kun jäljellä ei enää ole mitään suoritettavia kehikoita, ohjelman suoritus on loppu. Kehikoiden suunnattua graafia voitaisiin pitää puurakenteena, jossa puun jokainen solmu on kehikko, ja jossa ylintä solmua sanotaan juureksi. Suoritus redusoinnin avulla toteutetaan sitten normaalisti redusoidulla puurakennetta alhaalta ylös, jolloin redusoidaan puun osia, jotka ovat kauimpana juuresta jatkaen ylöspäin kohti juurta. Tämän tapaista suoritusta kutsutaan tavallisesti tarpeen mukaiseksi suoritukseksi, ts. niin että ohjelman muiden osien suoritustuloksesta riippuvien osien suoritusta siirretään myöhemmäksi, kunnes tulos on käytettävissä.

Esillä olevan keksinnön sekä sen muiden tavoitteiden ja etujen täydellisemmän ymmärtämisen kannalta viitataan seuraavaan seli-

tykseen ja oheisiin piirustuksiin, joissa:

- kuvio 1 esittää kaaviollisesti esillä olevan keksinnön mukaisen tietojenkäsittelylaitteen kaaviollisen piirikaavion;
- 5 kuvio 2 on portissa olevan signaalisekvenssin kaavio;
- kuvio 3 esittää keksinnön mukaisen tietojenkäsittelylaitteen suoritusmuodon kaaviollisen lohkokaaavion;
- kuvio 4 esittää kaaviollisesti kuviossa 3 esitetyn suoritusmuodon oliomuistin eri kenttien käyttöä muistisolussa;
- 10 kuvio 5 esittää esimerkin siitä, mitan funktio voidaan tallettaa oliomuistin muistisoluihin;
- kuvio 6 esittää ympäristöön johtavan portin suoritusmuodon;
- kuvio 7 esittää ns. H-portin suoritusmuodon;
- kuviot 8A, 8B, 8C esittävät kuvion 3 tietojenkäsittelylaitteen suoritusmuodossa olevan strukturaalisen aritmeettisen yksikön eri lajisia rekistereitä;
- 15 kuvio 8D esittää strukturaalisen aritmeettisen yksikön suoritusmuodon;
- kuviot 9A - 9F esittävät strukturaalisen aritmeettisen yksikön erilaisia datamuistin muotoja;
- 20 kuviot 10A - 10H,
kuviot 11A - 11G,
kuviot 12A - 12H esittävät esimerkkejä keksinnön mukaisen tietojenkäsittelylaitteen toiminnasta.

25

Tietojenkäsittelylaite

- Kuviossa 1 kaaviollisesti esitetty keksinnön mukainen tietojenkäsittelylaite sisältää aktiivisen muistin U1, jota alla sanotaan aktiiviseksi oliomuistiksi. Se on aktiivinen, koska se sisältää muistisoluja, joista jokainen pystyy aikaansaamaan laskentaoperaation suorittamisen. Jokainen oliomuistisolu on edullisesti jaettu useammaksi muistikentäksi, ja assosiatiivinen haku oliomuistin läpi voidaan tehdä muistikenttien sisällön perusteella. Muistisolu voi tallettaa kehikon. Solukehikkoon voidaan tallettaa osoituksia muihin solukehikoihin kehikkoelementteinä. Funktio voidaan tallettaa kehikkopuuna, joka sisältää
- 30
- 35

useita toisiinsa osoittavia solukehikoita. Näitä ominaisuuksia selitetään alempana enemmän.

Oliomuisti muodostaa keksinnön mukaisen tietojenkäsittelylaitteen sisäisen käyttäytymisen. Oliomuistiin talletetaan eri paikkoihin, ts. eri muistisoluihin useita sisäisiä käyttäytymisiä PR1, PR2, ..., PRn jokaista porttia kohti. Joukko portteja U2 - U5 on kytketty aktiiviseen oliomuistiin U1. Tulosignaalit RWu2 - RWu5 portteihin U2 voitaisiin varustaa ilmaisimilla tai tiedolla, joka on tuotettu tietojenkäsittelylaitteen ulkopuolisilla laitteilla, ts. reaaliaikaisessa ympäristössä.

Käyttäytymiset

Käyttäytyminen on peruselementtien struktuuri, jotka voivat muuttua ajan mukana. Siten käyttäytyminen on aikastruktuuri, mutta se ei osoita minkä takia struktuuri muuttuu. Aikastruktuuria voidaan pitää yhdistetyn prosessin tilana. Keksinnön mukainen tietojenkäsittelylaite on erityisesti sovitettu selittävää lajia olevaa ohjelmointikieltä varten, josta alempana käytetään nimeä H. Sillä ei ole mitään tilan ja muutoksen eksplisiittistä määritelmää, kuten imperatiivisissa kielissä. Sen sijaan kaikki käyttäytymiset määritellään lausekkeiden avulla. Termi käyttäytyminen on siten ajallinen struktuuri, sekvenssi tai sekvenssien vaihtoehto. Sitä käytetään myös kaikkien mahdollisten arvojen degeneroituna muotona; vertaa matemaattisen joukko-opin tyhjän joukon käyttöön.

Portti muuntaa tulevan signaalin aktiivista oliomuistia varten sopivaan, esimerkiksi digitaalisen muotoon, ja järjestää sen muotoon, jota voidaan verrata aktiivisessa oliomuistissa olevaan tietoon. Tämä tieto, joka on paikallista kyseistä porttia varten, sisältää yhden tai useita vaihtoehtoisia käyttäytymisiä PR1, PR2, ..., PRn.

Portti Ui, jossa i on välillä 2 - 5 oleva luku, näkee osia ympäröivästä reaalisesta maailmasta käyttäytymisenä, joka muo-

dostuu arvojen päättömästä aikasekvenssistä. Sekvenssi voidaan tallettaa puurakenteen muodossa tietojenkäsittelylaitteeseen, kuten alempana selitetään. Kuviossa 2 esitetään esimerkki sellaisesta sekvenssistä. Näytteistettyjen signaalien sekvenssi voi olla ajallisesti muuttuva, ts. näytteenottojaksot voivat olla yksilöllisiä, kuten kuviossa 2 ilmenee, joko tietokoneohjelman valitsemia tai ympäristön antamia. Signaalisekvenssi on tässä muodostettu elementtiparien listana. Jokainen pari sisältää kestoajan ja signaalikvantiteetin tänä aikana. Aikajaksoa kohti voi myös olla enemmän kuin yksi signaalikvantiteetti. Tämä voitaisiin tehdä esimerkiksi sellaisella portilla, joka voi käsitellä useampia kuin yhtä signaalikvantiteettia aikajaksoa kohti, tai käyttämällä muutamia rinnakkaisia portteja, jotka voivat käsitellä yhtä signaalikvantiteettia kestoaikaa kohti. Tällöin portit tahdistetaan keskenään.

Käyttäytymisten yhdistäminen

Keksinnön mukaisesti portin kautta tapahtuva tulo ja lähtö ympäristöön tehdään yhdistämällä sen kummallakin puolella tarjolla olevat prosessit, ts. tietojenkäsittelylaitteen sisäiset käyttäytymiset ja sen ympäristön ulkoiset käyttäytymiset. Funktionaalisella kielellä H kirjoitettu ohjelma, joka on järjestetty aktiiviseen oliomuistiin, luo paikalliset käyttäytymiset. Reaalimaailman käyttäytyminen kuvaa yhden reaaliaikaisen protokollan. Siten tietojenkäsittelylaitteen ympäristöä päin olevat portit rekisteröivät tietojenkäsittelylaitteen näkemän todellisen käyttäytymisen. Se yhdistetään tietokoneessa kuvatun protokollan kanssa. Alempana selitetään miten tämä tehdään keksinnön mukaisessa tietojenkäsittelylaitteessa. Sekä tietojenkäsittelylaitteen sisäiset käyttäytymiset että ulkoiset käyttäytymiset mallinetaan siten samalla tavalla, jolla aikaansaadaan erittäin elegantti toimintatapa, jonka ansiosta tietojenkäsittelylaite on erittäin sopiva reaaliaikaisten operaatioiden suorittamiseksi.

Siten tietojenkäsittelylaite sisältää ainakin yhden vertailu-

mahdollisuuden ympäristöä kohti olevaan porttiin järjestetyn signaalisekvenssin vertaamiseksi aktiivisen oliomuistin ainakin yhteen, useimmissa tapauksissa useampaan muistisolun talletetun sekvenssin kanssa. Yhdistämisoperaation (unifying) yhteydessä poistetaan kaikki sellaiset vaihtoehtoiset sisäiset käyttäytymiset, jotka eivät ole yhdenmukaisia reaaliympäristön käyttäytymisen kanssa. Korrektilla ohjelmalla tulisi olla vain yksi vaihtoehtoinen käyttäytyminen jäljellä yhdistämisoperaation jälkeen, ja tämän käyttäytymisen on yhdyttävä kyseessä olevassa portissa olevaan käyttäytymiseen. Ellei mikään sisäisistä käyttäytymisistä vastaa kyseessä olevan portin reaaliympäristön käyttäytymistä, yhdistämisoperaatio tuottaa tuloksena nothing, jota pidetään ohjelmointivirheenä.

15 H-PORTTI

Edellä oleva tulo/lähtö-portin selitys voitaisiin yleistää, niin että se pystyy käsittelemään muiden kuin during-struktuurien tuloja ja lähtöjä. Tässä tapauksessa voitaisiin suorittaa bittikuvioiden yhdistäminen sen sijaan, että yhdistetään sisäinen käyttäytyminen ulkoiseen käyttäytymiseen, kestoajkoja ja signaalikvantiteetteja yhdistämällä. Tulosignaali voisi olla digitaalinen. Lisäksi tuleva signaali voisi edustaa datastruktuureja tai ohjelmastruktuureja, kuten H-kielisiä ohjelmia. Voitaisiin esimerkiksi yhdistää (unify) portin muistisolun muistisolukenttä portin tulossa oleviin bittikuvioihin.

Esimerkki: sisäinen struktuuri `apply($1 $2)` voitaisiin yhdistää tulossa oleviin bittikuvioihin siten, että lopullinen yhdistetty sisäinen struktuuri on muotoa: `apply(+ list(1 2))`, joka on kuvioissa 11A ja 11B esitetty struktuuri. Tässä \$1 ja \$2 edustavat mitä tahansa bittikuviota ("any bitpattern"). \$1 yhdistetään bittikuvioon, joka edustaa operaation "+" käskykoodia, ja \$2 yhdistetään bittikuvioon, joka edustaa suuretta "list(1 2)".

Ohjelma voitaisiin ladata oliomuistiin H-portin kautta, joka

hyväksyy H-kielistä koodia. H-porttia voitaisiin myös käyttää prosessorien välillä, jolloin ne voisivat välittää ohjelmia tai dataa tai molempia. Siirretty ohjelmakoodi voitaisiin aluksi merkitä dataksi sen välittömän suorituksen välttämiseksi.

- 5 H-porttia voidaan myös käyttää rajapintana keksinnön mukaisen tietojenkäsittelylaitteen ja muuntuyppisen tietojenkäsittelylaitteen välillä, joka voi olla aikaisemmin tunnettua ja siten tavanomaista lajia.

10 Oliomuisti

Kuviossa 3 esitetyn tietojenkäsittelylaitteen suoritusmuodossa oliomuisti 1 sisältää joukon muistisoluja 2, jotka kukin on jaettu useammaksi muistikentäksi, kuten alempana selitetään.

- 15 Koska oliomuistin tarkka rakenne ei ole esillä olevan keksinnön osa, sitä ei selitetä yksityiskohtaisesti. Oliomuistiksi 1 sopivaa oliomuistia selitetään rinnakkaisessa patenttihakemuksessamme nro PCT/SE91/00513.

20 Strukturaalinen aritmeettinen yksikkö

Oliomuistin jokainen muistisolu voi tallettaa solukehikon, joka sisältää useita kehikoelementtejä. Oliomuisti 1 on kytketty strukturaaliseen aritmeettiseen yksikköön 3 muistiväyläjärjestelyn $BUS_1 - BUS_N$ kautta, joka on riittävän leveä muistisolun jokaisen bittisolun kytkemiseksi yksikköön 3. Jokaisen muistisolun jokainen bitti on kytketty kaikkien muiden muistisolujen vastaaviin bitteihin väyläjärjestelyn kautta. Strukturaalinen aritmeettinen yksikkö 3 voi olla määrätty osa oliomuistia 1, johon redusoitavat kehikot siirretään väliaikaisesti redusointiprosessin aikana. Kuitenkin on myös mahdollista, että yksiköllä 3 on erityinen rakenne, jota havainnollistetaan alempana, ja joka mahdollistaa nopeat ja välittömät redusointioperaatiot. Koska yksikön 3 tarkka rakenne ei ole esillä olevan keksinnön

25

30

35

osa, sitä ei selitetä yksityiskohtaisesti. Edullisesti yksiköksi 3 järjestettävää yksikköä selitetään rinnakkaisessa patenttihakemuksessamme nro PCT/SE91/00514.

Kuviossa 3 esitetty tietojenkäsittelylaite on rakentuu muistiväyläjärjestelyn $BUS_1 - BUS_N$ ympärille, niin että käytännöllisesti katsoen kaikilla dataa käsittelevillä laitteilla on siihen suora pääsy. Tällä tavalla yksikköä 3 ja portteja voidaan pitää

5 aktiiviseen oliomuistiin kuuluvina, ts. siinä olevina määrättyinä soluina.

Keskeinen ohjausyksikkö

10 Keskeinen ohjausyksikkö CU ohjaa sekä oliomuistia että yksikköä ?
 3. Yksikköön 3 on myös kytketty useita keskeisen ohjausyksikön CU ohjaamia portteja 4, 5, 6, 11. Kaksi porteista, 4 ja 5, on esitetty yhdistävinä portteina, jotka on kytketty ympäristöihin. Jokaiseen porttiin on kytketty ulkoinen laite 7 ja vastaavasti

15 8, kuten esimerkiksi anturi tai tietojenkäsittelylaitteen ohjaama ja/tai tietojenkäsittelylaitetta ohjaava laite. Portti 6 on H-portti, joka on kytketty toiseen tietojenkäsittelylaitteeseen 9, joka tuottaa tietojenkäsittelylaitteeseen liitettäviä ohjelmia. Portti 11 on kytketty toiseen tietojenkäsittely-

20 laitteeseen 12, joka on esimerkiksi samaa lajia kuin tietojenkäsittelylaitteet 1 - 6, 10, 11, CU, ja jonka kanssa se on tietoliikenneyhteydessä. Useita kuvion 3 esitetyn lajisia tietojenkäsittelylaitteita voidaan suoraan kytkeä toisiinsa muistiväyläjäjärjestelyjensä ja jokaisessa laitteessa olevan

25 väliaikaisen muistikehikon (ei esitetty) välityksellä, niin että aikaansaadaan moniportainen rinnakkais-tietojenkäsittelylaite. Tällä tavalla voidaan säilyttää järjestelmänlaajuiset hakumahdollisuudet. Eräissä sovellutuksissa on kuitenkin tarkoituksen-

30 mukaista kytkeä kaksi tai kolme tietojenkäsittelylaitetta eli rinnakkaista monilaskentalaitetta H-porttien 11 kautta, niin että jokainen tietojenkäsittelylaite toimii erillisenä tietojenkäsittelylaitteena ja että laitteet lähettävät keskenään ohjelmia ja/tai dataa erillistä käsittelyä varten. H-porttia 11 voidaan myös käyttää rajapintana toista lajia olevaan prosessoriin päin. Kun H-portti lähettää ohjelman, lähetys tapahtuu

35 samalla tavalla kuin datasiirroissa, ja tietojenkäsittelylaite käsittelee sitä samalla tavalla.

Numeerinen ALU

Yksikköön 3 voidaan myös liittää lisälaitte 10 määrättyjen, erikoistyyppisten operaatioiden suorittamiseksi. Esimerkkinä
 5 sellaisesta laitteesta on numeerinen aritmeettinen yksikkö, jota alempana sanotaan numeeriseksi ALU:ksi. Lisälaitetta 10 voidaan kuitenkin myös käyttää vertailu- tai vastaaviin tarkoituksiin. Numeerinen ALU voisi olla lähes mikä tahansa alalla tavallinen
 10 ALU, mutta erityisesti keksinnön mukaista tietojenkäsittelylaitetta varten soveltuvaa numeerista ALU:a on selitetty rinnakkaisessa PCT-hakemuksessamme nro PCT/SE91/00515.

Vertailu

15 Yksikön 3 sisältävällä aktiivisella muistilla on ainakin yksi vertailumahdollisuus johonkin porttiin, esim porttiin 4, järjestetyn signaalisekvenssin vertaamiseksi ainakin yhteen toiseen porttiin, esim. porttiin 6, järjestetyn ainakin yhden signaalisekvenssin kanssa. Tämä vertailumahdollisuus voidaan
 20 järjestää välittömästi muistiin tai porttien läheisyyteen järjestämällä kehikko, jossa on arvoja, kuten aikavälipareja, joilla kulloinkin on kestoajaksi ja arvo signaalikvantiteetin muodossa. Voidaan kuitenkin tehdä myös pelkkä vertailu lisäyksikössä 10. Myös kun kahta ulkoista käyttäytymistä verrataan
 25 toisiinsa, vertailusekvenssit kirjoitetaan uudelleen nothing:ksi, jos vertailu tuottaa eron, ts. ympäristöön johtavilla porteilla olevat kestoajat tai signaalikvantiteetit eivät ole samanlaiset. Muussa tapauksessa verratut sekvenssit kirjoitetaan uudelleen samaan määritettyyn sekvenssiin.

30

Siten aktiivisella oliomuistilla 1, 3 on rakenne, joka pystyy tallettamaan suorittamaan abstraktin syntaksin eksplisiittisen tai implisiittisen koodauksen muodossa olevan tietokoneohjelman. Syntaksilla kuvataan joukko erilaisia abstrakteja olioita
 35 lausekkeiden avulla. Jokainen muistisolmu pystyy kerrallaan tallettamaan ainakin syntaksilausekkeen osan sopivan dataja/tai ohjelmastruktuurin muodossa. Syntaksia selitetään alla

yksityiskohtaisemmin.

Oliomuistin mahdollisuudet

5 Oliomuistilla 1 on oleellisesti enemmän älykkyyttä kuin tavallisel-
lisella RAM-tyyppisellä muistilla. Se on assosiatiivinen, joka
mahdollistaa useampien palvelujen aikaansaamisen kuin pelkäs-
tään tavallisen RAM-tyyppisen muistin aikaansaamat "lue" ja
"kirjoita".

10

Kuten edellä mainittiin, oliomuisti on jaettu muistisoluihin 2,
joista jokainen sisältää useita muistikenttiä. Aikaansaadut
palvelut ovat korkealla tasolla. On esimerkiksi mahdollista,
että käyttämällä yhtä ainoata muistikäskyä löydetään määrätyn
15 dataelementin kaikki esiintymispaikat, riippumatta siitä millä
kohtaa yksilöllistä muistisolua se on, ja kirjoitetaan uudelleen
uusi arvo löydettyyn määrättyyn dataelementtiin globaalisesti,
ts. oliomuistin kokonaisuuden puitteissa. Koska oliomuisti on
assosiatiivinen, tämä uudelleen kirjoitusoperaatio voitaisiin
20 tehdä kahdessa fyysisessä muistijaksossa, sitä koskevien muis-
tisolujen lukumäärästä riippumatta.

Muistisolu

25 Kuviossa 4 esitetään kaaviollisesti muistisolun 2 suoritusmuoto.
Siihen voidaan tallettaa kahden tyyppisiä elementtejä ja se
sisältää muistikenttiä, jotka on erityisesti sovitettu talle-
tettaviin elementteihin. Näille kentille on kuviossa 4 annettu
samat nimet kuin niihin talletettavilla elementeillä.

30

Jokainen muistisolu on sovitettu tallettamaan ainakin joitakin
seuraavan luettelon ominaisuuksia:

35 merkkejä jotka osoittavat, onko mainitun solun lauseke
redusoitava;
merkkejä jotka osoittavat, kun lauseke on puun elin,
lausekkeen ominaisuuksia;

merkkejä jotka osoittavat miten lauseke on luotu;
 merkkejä jotka osoittavat muodostuuko lauseke
 toistettujen käyttäytymisten joukosta;
 merkkejä jotka osoittavat, onko lauseke vain osa
 5 sellaista listaa, jolla on toiseen muistisoluun
 talletettuja muita listaelimiä.

Elementtien ensimmäinen laji, jota alla sanotaan attribuuttiele-
 menteiksi, kuvaavat muistisolun erilaisia tiloja. Eräs tämän
 10 lajinen elementti on LAZY, joka osoittaa onko solu joutilas,
 jolloin solun muuta osaa pidetään passiivisena tietona; exec,
 ts. suoritettavassa tilassa; tai wait, ts. solun arviointia on
 siirretty myöhemmäksi ja se odottaa tulosta ennen kuin se voidaan
 suorittaa. Toinen attribuuttielementti on TYPE, joka sisältää
 15 tyyppikoodin (par, seq, apply, list, unify, jne).

Elementtien toinen laji, jota alla sanotaan olioelementeiksi,
 kuvaa tunnistinta, ympäristöä tai arvoa. Näitä ovat IDENTIFIER
 (tunnistin), ENVIRONMENT (ympäristö), VALUE/DES (arvo/tun).
 20 Nämä elementit on sovitettu talletettaviksi ydinrekisterien
 osiin, jotka on järjestetty tasoihin HEAD (pää) ja NUM (num)
 (ks. kuviot 7B, 7C). Jokainen näistä elementeistä sisältää
 elementtisanan, joka on vuorostaan jaettu num-sanaksi (num-word)
 ja merkitsin-sanaksi (tag word).

25

Merkitsin-sana

Toisen lajin kehikkoelementeillä on merkitsin-sana, joka osoit-
 taa num-sanan ominaisuuden. Merkitsimiä on kahta lajia, välil-
 30 liset merkitsimet, ts. merkitsimet joita käytetään tunnistimille
 ja ympäristölle, sekä välittömät merkitsimet, ts. merkitsimet
 joita käytetään yksinkertaisille arvoille ja vastaaville.

Esimerkkejä välillisistä merkitsimistä ovat cls, canon, ja open.
 35 Jos merkitsin-sana on cls, se tarkoittaa että num-sana edustaa
 kehikkoa, joka saattaa olla redusoitavissa. Jos merkitsin-sana
 on canon, se tarkoittaa että tunnistinkenttä edustaa kehikkoa,

jota ei enää voi redusoida. Jos merkitsin-sana on open, se tarkoittaa että tunnistinkenttä edustaa kehikkoa, joka on liitetty lista.

- 5 Esimerkkejä välittömistä merkitsimistä ovat discr, cont, unused ja nothing. Jos merkitsin-sana on discr, se tarkoittaa että num-sana on kokonaisluku. Jos merkitsin-sana on cont, se tarkoittaa että num-sana on liukupilkkuarvo. Jos merkitsin-sana on unused, se tarkoittaa että num-sana on vailla merkitystä. Jos
10 merkitsin-sana on nothing, se tarkoittaa että num-sana ei edusta mitään, ts. nothing-merkityn kentän sisältävän kehikon yhdistelmä on aina nothing.

Tunnistimet ja ympäristö

- 15 Jos muistisolun tunnistinkenttä sisältää tunnistinelementin, niin tässä muistisolussa oleva prosessin tila voitaisiin siirtää strukturaaliseen aritmeettiseen yksikköön 3, jota alla sanotaan ydinsoluksi. Jokainen muistisolukenttä arvo/tun. (VALUE/DES)
20 voi sisältää tunnistimen, joka osoittaa toiseen solukehikkoon, aikaansaaden siten linkin tähän toiseen solukehikkoon. Ympäristökentät voisivat sisältää tunnistimen, joka osoittaa juurikehikon kehikkoverkon osassa, ts. kehikkopuussa, joka muodostaa kehikon ympäristön. Ympäristökentällä voisi kuitenkin olla
25 muutakin käyttöä. Ympäristöjä voitaisiin käyttää struktuurin luojan jäljittämiseen tallettamalla luojan tunnistin kaikkien luotujen solukehikoiden ympäristöihin. Esimerkiksi kaikki solukehikot alipuussa, jossa kaikkien samalla nimellä varustettujen symbolien on tarkoitettava samaa asiaa, voitaisiin ryhmitellä
30 antamalla niille sama ympäristö. Tällä tavalla koko struktuuriin on pääsy puun eräästä kehikosta, juuresta, ainoastaan yhdessä operaatiossa.

- 35 Tunnistinfunktiota voitaisiin pitää suorana linkkinä isästä poikaan, ts. kehikkoelementti osoittaa yksilöllisesti solukehikon. Assosiatiiivista lajia olevalla oliomuistilla varustetun koneen käyttäytyminen esitetään siten kehikoihin suunnattuna

graafina.

Jos siten kehikon ympäristö on annettu, tässä ympäristössä oleva juurikehikko voidaan löytää. Ympäristön juurikehikko voidaan
 5 varustaa erityisellä merkillä (esim. "1") sen muistisolun kentässä WHERE. Ympäristön solmukehikko voidaan varustaa toisella merkillä (esim. "0") kentässä WHERE.

Esimerkki

10

Kuviossa 5 on esitetty esimerkki muistisolusta, johon on talletettu lauseke

```
id1 = list(par(1 2 3) par(4 5 6))
```

15

joka on kahden rinnakkaisarvon yhdistelmien lista. Ensimmäisen rinnakkaisen yhdistelmän par(1 2 3) tunnistimena on id₂, ja toisen rinnakkaisen yhdistelmän par(4 5 6) tunnistimena on id₃. Solukehikon sisältävä juuren muistisolun, jolla puussa on
 20 tunnistin id₁, saa merkitsimen cls, ja sillä on merkintä exec LAZY-kentässä, asetettu "1" on WHERE-kentässä, merkintä list TYPE-kentässä, ja tunnistimet id₂ ja id₃ kahdessa ensimmäisessä value/des.-kentässä. Näiden kenttien merkitsimet on merkitty canon, koska näiden kenttien sisältö on välillistä ja linkitetty
 25 kanonisiin kehikkosoluihin. Solmu-muistisolun, joka sisältää tunnistimella id₂ varustetun solukehikon, sisältää arvon "0" WHERE-kentässä, sillä on merkintä par TYPE-kentässä, ja diskreetit arvon 1, 2 ja 3 on talletettu kolmeen ensimmäiseen value/des-kenttään. Näiden kenttien merkitsimet on tämän johdosta merkitty discr. Solmu-muistisolun, joka sisältää tunnistimella id₃ varustetun solukehikon, sisältää arvon "0" WHERE-kentässä, sillä on merkintä par TYPE-kentässä, ja diskreetit arvon 4, 5 ja 6 on talletettu kolmeen ensimmäiseen value/des-kenttään. Näiden kenttien merkitsimet on tämän johdosta myös merkitty
 30 discr.
 35

Portti ulkopuolisen maailman rajapintana

Kuten edellä mainittiin, keksinnön mukainen käsittelylaite on varustettu yhdellä tai useammalla portilla datan syöttämiseksi
 5 laitteeseen ja siitä tulostamista varten. Jokainen portti on yksikkö, joka aikaansaa tietoliikennettä käsittelylaitetta ympäröivän ulkomaailman kanssa. Portti on edullisesti liitetty oliomuistissa 1 (ks. kuvio 3) olevaan muuntavaan rajapintaan (ei esitetty). Porttia voidaan pitää oliomuistin jatkeena, koska se
 10 sisältää edullisesti neljä muistisolua, jotka toimivat samalla tavalla kuin oliomuistin solut 2 (ks. kuvio 4) ja ovat yhteensopivia niiden kanssa. Portti suorittaa tulo/lähtö-operaatiot yhdistämisen avulla.

15 Sisäinen käyttäytyminen

Oliomuisti tarjoaa prosessorin sisäisen käyttäytymisen, yhden käyttäytymistavan jokaista järjestettyä porttia kohden. Käyttäytyminen on aikastruktuurin semanttinen merkitys, joka voi-
 20 taisiin esittää kehikkostruktuurina. Aikastruktuuria voitaisiin pitää prosessin tilana, joka muodostuu arvojen päättymättömästä aikasekvenssistä.

Ulkoinen käyttäytyminen

25

Tulosignaalit portille (porteille) voitaisiin järjestää ilmaisimilla tai laitteilla, jotka ovat prosessorin ulkopuolella, ts. reaali-
 maailman ympäristössä. Portti muuntaa tulevat signaalit digitaaliseen muotoon, joka on sovitettu oliomuistin
 30 formaattiin, ts. kehikon esitysmuotoon. Tämä kehikon esitysmuoto on aikastruktuuri, ts. käyttäytyminen, verrattavissa prosessorin sisäiseen käyttäytymiseen. Siten prosessorin kannalta tätä aikastruktuuria voidaan pitää ulkoisena käyttäytymisenä.

35 Portin kautta syöttö ja tulostus tehdään yhdistämällä prosesseja, joita on sen kummallakin puolella, ts. prosessorin sisäisiä käyttäytymisiä ja sen ympäristön ulkopuolisia käyttäytymisiä.

Sekä sisäiset että ulkoiset käyttäytymiset mallinnetaan samalla tavalla, joka muodostaa erittäin elegantin toimintatavan, ja joka tekee prosessorin erittäin sopivaksi reaaliaikaisten operaatioiden suorittamista varten.

5

Kuviossa 2 on esitetty esimerkki ulkoisesta käyttäytymisestä, ts. aikasekvenssistä. Näytteistettyjen signaalien sekvenssiä voitaisiin muuttaa ajallisesti, ts. näytteenottojaksot voivat olla erilaisia, kuten kuvioista 2 ilmenee. Signaalisekvenssit on tässä muodostettu elementtiparien listana. Jokainen pari sisältää kestoajan ja signaalikvantiteetin tänä aikana. Signaalikvantiteetit on merkitty q_i ja kestoajat on merkitty t_i , jolloin i on välillä 1...6 oleva luku.

10

Ympäristöön johtava portti, kuten esim. kuvion 3 portti 4, voitaisiin esimerkiksi liittää ilmaisimeen 7. Portin rajapinta voisi käsitellä kaikenlaaisia signaaleja yleisestä näkökohdasta. Sellaiset rajapintatyyppit ovat kuitenkin mutkikkaita, ja aiheuttavat usein häiriöitä järjestelmässä. Tämän johdosta edullisena pidettävällä rajapinnalla tulisi olla seuraavat ominaisuudet:

20

1. Signaali mitataan määrättyinä hetkinä. Mitattu signaali katsotaan vakioksi, seuraavaan mittaukseen saakka.

25

2. Aika voitaisiin ilmoittaa joko portin määrittämänä kiinteänä jaksena, tai koneen ohjelman määrittämänä kiinteänä jaksena, tai määrättyinä aikaväleinä, jotka määritetään ohjelman tai ulkoisten kellosignaalien avulla.

3. Mitattu signaali tuodaan koneelle digitaalisessa muodossa.

30

4. Digitoitu signaali on joko Boolean signaali tai koodattu digitaalisen muotoon, joka voisi osoittaa kokonaislukua tai liukupilkkuarvoa.

35

Keksinnön mukainen prosessori ei ota vastaan tai tulosta arvoja tavanomaisten prosessorien tavoin. Keksinnön mukaisessa prosessorissa porttia pidetään prosessorin ohjelman ja prosessorin ympäristön välisenä rajapintana. Tietojenkäsittelylaite on

sovitettu funktionaaliseen kieleen H. Rajapinta on sellainen, että järjestetään tapahtumasarja, joka ilmenee sekä sen toisella puolella olevana signaalina että prosessoriohjelman H-strukturina sen toisella puolella. Ympäristö asettaa osansa fyysisestä kuvasta ja prosessori oman osansa, ja tämä tehdään niin, että nämä kaksi ovat saman asian malleja, jolloin tietojenkäsittely-

5 laite muodostaa reaaliajasta kuvan, joka on niin lähellä eksaktia reaaliaikaa kuin mahdollista.

10 Ohjelma on talletettu aktiiviseen oliomuistiin ja luo sisäiset käyttäytymiset. Ohjelman suoritus perustuu siihen, että sekä tietojenkäsittelylaitteen ohjelma että ympäristö näkevät saman reaaliaikaisen tapahtumasarjan rajapinnassa.

15 Ohjelma kuvaa rajapinnan arvojen ja kestoajien parien päättymättömänä pitkänä listana, ts. jokainen signaalin arvo kestää kestoajan verran. Siten rajapinta kuvaa kestoajan ja arvon jokaisen aikavälin aikana, johon sisältyy signaaliarvon ja kestoajan pari. Kaikki sellaiset parit on järjestetty sekvenssiksi, ja portti muuntaa reaaliaikaisen käyttäytymisen samantyyppiseksi listaksi.

20 Syöttäminen tietojenkäsittelylaitteeseen ja tulostaminen siitä tehdään yhdistämällä (unification). Tämä tarkoittaa sitä, että portin kummallakin puolella kesto aika ja arvo jokaisessa aikavälissä ovat samat.

Ohjelma voisi määritellä määrätyn signaalikvantiteetin tai se voisi jättää signaalikvantiteetin määrittelemättä käyttämällä määrättyä symbolia, esimerkiksi \$, joka merkitsee kaikkia mahdollisia arvoja.

Oikeassa oleva ohjelmalla tulisi olla vain yksi mahdollinen käyttäytymismahdollisuus jäljellä yhdistämisoperaation jälkeen, ja tämän käyttäytymisen on oltava sama kuin kyseessä olevassa portissa. Ellei mikään sisäisistä käyttäytymismahdollisuuksista vastaisi kyseessä olevan portin reaaliaikaista ympäristöä,

yhdistämisoperaatio tuottaa tuloksena nothing (ei mitään), jota pidetään ohjelmointivirheenä, koska nothing edustaa ristiriitaa. Ohjelmointivirheenä pidetään myös tilannetta, jossa yhdistämisoperaation jälkeen jäljellä on enemmän kuin yksi käyttäytymismahdollisuus.

Kun sisäinen käyttäytyminen määrittelee erityisen signaaliarvon, jota myös sanotaan signaalikvantiteetiksi, tämä määrätty signaaliarvo tulostetaan ympäristöön. Jos sisäinen käyttäytyminen jättää signaaliarvon määrittelemättä, ympäristöstä syötetään määrätty signaali.

Kun sisäinen käyttäytyminen määrittelee aikavälin erityisen kestoajan, tällöin ympäristö sallii mielivaltaisen kestoajan pituuden. Jos sisäinen käyttäytyminen jättää kestoajan määrittelemättä, ympäristö määrittää kestoajan.

Siten mekanismi on symmetrinen:

1. Aikaväli, jossa on lauseke during, sisältää kestoajan ja signaalikvantiteetin.
2. Aikaväliä selitetään ohjelmassa muodolla: during kesto aika signaalikvantiteetti.
3. Ympäristö voi selittää sellaisen aikavälin eri tavoin, kuten edellä selitettiin.
4. Portti määrittelee kummallakin puolellaan olevat signaalikvantiteetit. Tämä merkitsee sitä, että signaalikvantiteetin on oltava määritetty. Tietojenkäsittelylaitteessa oleva ohjelma asettaa kestoajan tai signaalikvantiteetin portin tietojenkäsittelylaitetta kohti olevalle puolelle. Määrätyn kestoajan ja/tai aikana tehdään tulostus tai signaalikvantiteetti syötetään portin ympäristöä kohti olevan puolen johtimille.

Ympäristöön johtavan portin suoritusmuoto

Kuviossa 6 esitetään portin eräs suoritusmuoto aikavälien syöttöä ja/tai tulostusta varten. Portti on kytketty ensimmäisen

5 strukturaaliseen aritmeettiseen yksikköön 3 tai suoraan oliomuistiin 1 väyläjärjestelyn DU kautta. Väyläjärjestelyn DU jokainen väylä edustaa väylää, jossa on esimerkiksi 38 lankaa. Porttia ohjataan keskeisellä yksiköllä CU. Portti sisältää tunnistinrekisterin P_{ID}. Rekisteriin P_{ID} talletettua tunnistinta

10 voidaan käyttää tunnistimena oliomuistissa 1, ts. sitä voidaan käyttää linkittämään portti oliomuistin solukehikoihin. Tätä tunnistinta käytetään vain sisäisesti prosessorissa. Portti sisältää myös tunnistinrekisterin CE_{ID}, jota voidaan käyttää samalla tavalla kuin rekisteriä P_{ID}. Portti sisältää myös joukon

15 muistisoluja CE_{DU}, CE_{LAST}, CE_{NEXT}, CE_{ID}, jotka kaikki rakenteeltaan muistuttavat oliomuistin muistisoluja ja toimivat niiden tavoin.

Jokainen muistisolu voisi tallettaa kehikon, jossa on ainakin

20 kolme muistikenttää, joihin voidaan tallettaa ainakin kolme kehiikkoelementtiä, esimerkiksi during, kesto aika, ja signaali-kvantiteetti.

Ainakin kaksi viimeksimainittua kehiikkoelementtiä voitaisiin

25 tallettaa muistikenttiin, jotka ovat value/designation-lajia. Luseke during voidaan järjestää TYPE-kenttään. Koska portti aina aikaansaa during-operaation, tämä TYPE-kenttä voitaisiin jättää pois.

Kuviossa 6 esitetyssä suoritusmuodossa during on kuitenkin

30 tyyppiä 'apply ('@), joka on sijoitettu tyyppikenttään yhdessä ennalta määrätyn funktionimen kanssa (during-koodi). Funktionimi talletetaan portin muistisolujen ensimmäiseen value/designation (arvo/tunnistin) -kenttään, kuten kaikki funktionimet

35 (esim. + - * /), jotka kuuluvat tyyppiin 'apply. During-koodi voisi olla tunnistin, joka merkitsee oliomuistiin talletettua funktion määritelmää.

Tunnistin id voidaan myös tallettaa value/designation-kenttään. Tämä tunnistin yhdistää porttistruktuurin oliomuistin during-struktuuriin.

5 Ellei tarvita tunnistimen sisältävää kenttää, tarvitaan itse asiassa vain kolme value/designation-kenttää kuvion 6 esittä-
mässä suoritusmuodossa. Voitaisiin kuitenkin järjestää neljäs
value/designation-kenttä, joka on merkitty unused, niin että
10 porttirekisterit saadaan yhteensopiviksi tietojenkäsittelylait-
teen muiden muistisolujen kanssa. On kuitenkin myös mahdollista,
että neljänteen value/designation-rekisteriin kytketään ylimää-
räinen laite arvo sisään/ulos -toimintaa (VALUE IN/OUT) varten.
Tämä tekee portista arvojen ryhmän portin, ts. kahden signaali-
kvantiteetin kestoaikaa ja lajia varten.

15

Portin jokaisella muistisolulla voisi siten myös olla attribuut-
tikenttiä. Anakin viimeksi mainitut muistisolut voisivat olla
keskenään yhdistettyjä väylillä W_{01} , W_{11} , W_{T1} , W_{V1} , W_{E1} , W_{T2} , W_{V2} ,
20 W_{E2} , jolloin jokaisella väylällä on esimerkiksi 38 lankaa jotka
on kytketty bittikohtaisesti jokaisen rekisterin bittisoluihin.
Indeksi I tarkoittaa väylää, joka voidaan kytkeä rekistereiden
during-koodikenttiin, indeksi T tarkoittaa väylää, joka voidaan
kytkä rekistereiden kestoajan value/designation-kenttiin, in-
deksi V tarkoittaa väylää, joka voidaan kytkeä rekistereiden
25 signaalikvantiteetin value/designation-kenttiin, indeksi E tar-
koittaa väylää, joka voidaan kytkeä rekistereiden ylimääräisiin
value/designation-kenttiin. Indeksi 2 tarkoittaa väylää, joka
on kytketty ympäristöön päin, ja indeksi 1 väylää, joka on
kytketty muuntavaan rajapintaan (ei esitetty).

30

On huomattava, että tyyppikenttä, joka sisältää suuren '0', ja
ensimmäinen value/designation-kenttä, joka sisältää during-koo-
din, käytännössä jätetään pois, koska niiden sisältämä tieto
aina on sama, ja siten tämä voitaisiin toteuttaa pelkästään
35 keskeisessä ohjausyksikössä CU. Tämän vuoksi voidaan käyttää
porttia, jonka muistisolut sisältävät vain kaksi value/designa-
tion-kenttää, yhden kestoaikaa ja yhden signaalikvantiteettia

varten (ei esitetty).

Portin esitetyssä suoritusmuodossa muistisolut tallettavat edellisen kehikon, käsillä olevan kehikon, seuraavan kehikon, ja tulevien kehikoiden tunnuksen. Muistisoluilla voisi myös olla ennalta määrätyt paikat ja siten koko niiden sisältö voitaisiin siirtää niiden välillä. Sopiva rakenne jokaista rekisterien rekisterisolua varten voi olla sama kuin strukturaalisessa aritmeettisessä yksikössä 3, joka esitetään rinnakkaisessa PCT-hakemuksissamme nro PCT/SE91/00512 ja PCT/SE91/00513.

On kuitenkin myös mahdollista järjestää ensiksi mainitut muistisolut siten, että ne vaihtamalla nimityksiään voisivat vaihtaa paikkaa keskenään, joka tehdään keskeisen ohjausyksikön CU ohjaamana.

Portti voitaisiin varustaa eri lukumäärällä muistisoluja, esim. yhdellä muistisolulla.

20 Aikalaskuri ja aikavertailija

Portti sisältää aikalaskurin T_{COUNT}. Sitä ohjataan keskeisestä ohjausyksiköstä CU kellosignaalin C_{CLOCK} ja se mittaa aikaa. Se voidaan nollata joko ulkoisella nollaussignaalilla tai ohjausyksikön CU nollaussignaalilla. Aikalaskuri T_{COUNT} on kytketty ensimmäiseen valitsimeen SEL₁, jota ohjataan keskeisestä ohjausyksiköstä CU joko kytkemään aikalaskurin T_{COUNT} lähtö suoraan väylälle W_{T2}, joka on kytkettävissä muistisolujen aikamuistikenttiin, tai aikavertailijan T_{COMP} ensimmäiseen tuloon. Aikavertailijan T_{COMP} toinen tulo on kytketty väylään W_{T2}. Aikavertailijan lähtö siirretään myös ohjausyksikköön CU.

Portti sisältää myös ulkoisen arvon tulo- ja lähtöporttipiikin VALUE IN/OUT. Se on kytketty lähtösignaalin osalta muuntimeen CONV_{OUT}, esim. digitaali/analogia-muuntimeen. Se on myös kytketty tulosignaalin osalta muuntimeen CONV_{IN}, esim. analogia/digitaali-muuntimeen, joka ottaa näytteitä tulosignaalista. Näyt-

teenottoajat voitaisiin määrätä aikavertailijan T_{COMP} lähdöllä tai aikalaskurin T_{COUNT} nollaussignaaleilla (ei esitetty). Väylään W_{V2} kytketty valitsin SEL_2 , joka on ohjattavasti kytkettävissä muistisolujen CE_{DU} , CE_{LAST} , CE_{NEXT} signaalikvantiteettirekistereihin, valitsee keskeisen ohjausyksikön CU ohjaamana syötetäänkö signaalikvantiteetti ulos portista tai sisään päin porttiin. Ohjausyksikkö CU lukee ensin muistisolun rekisterien sisällön, mukaanlukien ympäristöön kytkettävät rekisterit, jotta se näkisi sisältääkö se erityisen symbolin \$, joka tarkoittaa, että rekisterin arvo on mielivaltainen ja voidaan vaihtaa mihin tahansa mahdolliseen arvoon. Sitten ohjausyksikkö CU vastaavasti ohjaa rekisteriin kytkettyä porttivalitsinta.

On myös mahdollista tehdä mutkikkaampi portti. Esimerkiksi signaalikvantiteetit $VALUE$ IN/OUT:a varten voidaan muuntaa muuntaviin piireihin $CONV_{IN}$ talletetulla kaavalla tuloarvon osalta, ja $CONV_{OUT}$ lähtöarvon osalta. Muunnosfunktio voisi esimerkiksi olla joukko integraaleja, joista jokainen koskee signaalikvantiteetin signaaliarvojen tai vastaavien eri luokkaa.

Voidaan myös ottaa jokaisen kestoajan puitteissa kaksi arvoa, esimerkiksi yksim arvo kestoajan alussa ja toisen sen lopussa, ja tallettaa niistä toinen suoraan value/designation-kenttään ja syöttää nämä kaksi mitattua arvoa ja kesto aika kaltevuuden laskevaan laitteeseen (ei esitetty). Kaltevuuden laskevan laitteen lähtö voitaisiin sitten tallettaa esimerkiksi value/designation-lisäkenttään (ei esitetty), ts. $VALUE$ IN/OUT:n lisävalitsimella.

30 Signaalikvantiteetin ulos syöttäminen

Ulos syötettävä tieto on tarjolla muistisolussa CE_{DU} . Siten solussa on tieto

during kesto aika signaalikvantiteetti.

Tämän rekisterin signaalikvantiteetti on tunnettu ja se on siten

syötettävä ulos. Signaalikvantiteetti noudetaan muistisolun signaalikvantiteetin rekisteristä ja syötetään ulos valitsimen SEL₂ kautta, joka toimii ajuriportaana, sekä muuntimen CONV_{out} kautta.

5

Signaalikvantiteetin sisään syöttäminen

Muistisolussa CE_{DU} oleva tieto on

10

during kesto aika \$.

Tässä rekisterissä oleva signaalikvantiteetti on siten tuntematon, ja rekisteri valmistetaan signaalikvantiteetin vastaanotamista varten. Signaalikvantiteetti noudetaan portin signaalikvantiteetin tulo/lähdöstä, digitoidaan AD-muuntimessa CONV_{IN}, ja syötetään signaalikvantiteetin rekisteriin muistisolussa CE_{DU}.

15

Kesto aika asetetaan prosessorin ohjelmalla

20

Muistisolussa CE_{DU} oleva tieto on

during kesto aika signaalikvantiteetti.

25

Siten kesto aika on tunnettu. Keskeinen ohjausyksikkö CU on nollannut aikalaskurin T_{COUNT} kesto ajan alussa. Jatkuvasti askellettua laskurin T_{COUNT} aikaa verrataan aikavertailijalla T_{COMP} aikaan, joka on muistisolun CE_{DU} aikarekisterissä. Kesto aika päättyy, kun nämä ajat ovat yhtä suuret.

30

Muistisolu CE_{LAST} on tyhjennetty ja muistisolu CE_{NEXT} on täytetty tämän kesto ajan kuluessa. Seuraavaan jaksoon siirryttäessä muistisolun CE_{DU} sisältö siirretään muistisoluun CE_{LAST}, ja muistisolun CE_{NEXT} signaalikvantiteetti siirretään muistisoluun CE_{DU}.

35

Kestoaika asetetaan ympäristöstä

Muistisolussa CE_{DU} oleva tieto on

5 during \$ signaalikvantiteetti

Siten kesto aika on tuntematon ja ympäristö määrittää sen. Ulkoinen ohjaus on nollannut aikalaskurin T_{COUNT} kestoajan alussa. Jatkuvasti askellettu laskurin T_{COUNT} aika syötetään
10 muistisolun CE_{DU} aikarekisteriin. Kestoaika päättyy, kun ympäristö tuottaa seuraavan ulkoisen nollauksen. Siten ulkoinen nollaussignaali ohjaa valitsimen SEL₁ katkaisemaan yhteyden aikalaskuriin, joko välittömästi tai aikalaskurin nolla-signaalilla.

15

Tämän kestoajan kuluessa muistisolu CE_{LAST} on tyhjennetty ja muistisolu CE_{NEXT} on täytetty. Seuraavaan jaksoon siirryttäessä muistisolun CE_{DU} sisältö siirretään muistisoluun CE_{LAST}, ja muistisolun CE_{NEXT} signaalikvantiteetti siirretään muistisoluun
20 CE_{DU}.

Yksinkertainen esimerkki

Portin toiminnan havainnollistamiseksi alla esitetään pieni
25 esimerkki. Oletetaan, että halutaan ilmaista toinen kahdesta erilaisesta sekvenssistä, jolloin ensimmäisessä sekvenssissä on määritetty joka toinen signaalikvantiteetti, ts. arvolla 17, ja kaikki kestoajat on määritetty, esim. 1 sekunti, ja jolloin toisessa sekvenssissä on määritetty kaikki signaalikvantiteetit, esim. 1, 2, 3, 4, jne, mutta kestoaikaa ei ole määritetty.
30 Käytettävälle portille annetaan nimi port₁. Kehikon esitystapa:

unify(port₁ "sisäinen käyttäytyminen"),

35 jossa "sisäinen käyttäytyminen" on alt-operaatio, joka merkitsee vaihtoehtoista sekvenssiä. ts. suurta alla olevaa lajia olevaa datastruktuuria:

```
alt(seq(during(1s 17)during(1s $)during(1s 17)....)
     seq(during($ 1)during($ 2)during($ 3)....))
```

Jos nyt tulossa oleva signaalisekvenssi voidaan yhdistää (unify) edellä olevaan struktuuriin, niin tulo hyväksytään ja asiaan-
 kuuluva toimenpide voidaan toteuttaa. Hyväksytyn sekvenssin eräänä esimerkkinä on ((1s 17)(1s 0)(1s 17)(1s 99)(1s 17)), joka sekvenssi voidaan yhdistää sisäisen käyttäytymisen edellä annettuun sekvenssiin. Toinen hyväksyttävä sekvenssi on ((2s 1)(4s 2)(1s 3)), joka yhdistyy sisäisen käyttäytymisen toiseen sekvenssiin. Esimerkki sekvenssistä, jota ei voida hyväksyä, on ((1s 2)....), koska signaalikvantiteettia, ts. 2, ei voi yhdistää sisäisen käyttäytymisen kumpaankaan signaalikvantiteettiin.

15 H-portin suoritusmuoto

Kuviossa 7 esitetyn H-portin suoritusmuoto on tarkoitettu liittämään toisiinsa kaksi erillistä keksinnön mukaista tietojenkäsittelylaitetta. Ensimmäiseen tietojenkäsittelylaitteeseen kuuluva H-portin puolikas sisältää tyyppikentän TYPE₁₁, neljä value/designation-kenttää V/D₁₁, V/D₁₂, V/D₁₃, V/D₁₄, ja tunnistinrekisterin ID_{P11}. Toiseen tietojenkäsittelylaitteeseen kuuluva H-portin puolikas sisältää tyyppikentän TYPE₂₁, neljä value/designation-kenttää V/D₂₁, V/D₂₂, V/D₂₃, V/D₂₄, ja tunnistinrekisterin ID_{P21}. Kuviossa 7 esitettyssä suoritusmuodossa kummankin porttipuolikkaan tyyppikentät ja value/designation-kentät on liitetty toisiinsa kulloisellakin väylällä. Siirto tehdään sarjaväylillä.

Koska mahdollisimman harvojen fyysisten liitäntöjen käyttäminen on edullista, paras toimintatapa on siirtää dataa porttipuolikaiden välillä ainoastaan yhdellä sarnaväylällä. Sellainen siirto tekee kuitenkin keskeisen ohjausyksikön CU sisäiset ohjauspiirit mutkikkaammiksi kuin jos siirto tehtäisiin käyttämällä yhtä siirtoväylää muistikenttää kohti. Kuviossa 7 ei ole esitetty rajapintoja rinnakkaisdatan muuntamiseksi sarjamuotoon ja päinvastoin, eikä niitä selitetä, koska niiden uskotaan olevan

alan ammattilaiselle varsin tuttuja.

H-portin molemmat puolikkaat on kytketty ohjausyksikköön CU₁ ja vastaavasti CU₂, joka ohjaa tietojenkäsittelylaitetta, jonka
 5 osana se on. Toinen ohjausyksikkö CU₁ tuottaa tahdistussignaalin SYNC, joka tahdistaa toisen ohjausyksikön CU₂. Kun on tehtävä siirto, H-portin toinen puolikas sisällyttää erityisen symbolin \$ kaikkiin rekisreihinsä, paitsi tunnistinrekisteriin, joka osoitetaan sillä ohjausyksiköllä, joka sitten vastaanottaa
 10 toisen puolikkaan rekistereiden sisällön.

Kun ohjelman suuri datapaketti on siirrettävä tietojenkäsittelylaitteesta toiseen H-portin kautta, tunnistimet on muutettava vastaanottavassa tietojenkäsittelylaitteessa, niin että ne
 15 sovitetaan tunnistinlukujen sisäiseen järjestykseen, jotta vältettäisiin samojen numeroiden kaksinkertainen käyttö. Tämän vuoksi siirretään puu joko sen juuresta kohti lehtiä, jota pidetään edullisimpana, tai lehdistä kohti juurta, samalla kun siirrettyjen kehikoiden jokainen tunnistin vaihdetaan uudeksi,
 20 ja ne talletetaan vastaavalla tavalla oliomuistiin.

H-kielen käyttäytymiset

Ohjelmointikieli H käyttää käyttäytymistä, jossa on kolme
 25 näkökohtaa muodossa

p t a

jossa p, t ja a ovat kolme erilaista mallinnusmuuttujaa.
 30 Parametrit p, t ja a mallintavat rinnakkaisuutta, peräkkäistä aikaa, ja vastaavasti vaihtoehtoisia toimenpiteitä. Niitä voitaisiin pitää indekseinä, joilla merkitään erilaisia rinnakkaisia toimenpiteitä, erilaisia tapahtumia ja vastaavasti vaihtoehtoisia käyttäytymisiä. Nämä kolme näkökohtaa ovat
 35 hyödyllisiä sovellettaessa useita käyttäytymislajeja, jotka kuvataa imperatiivisten kielten erityisillä rakenteilla. Yhdessä ne muodosta H-kielen käyttäytymisen keskeisen ajatuksen, ts.

ytimen. Keksinnön mukaisen tietojenkäsittelylaitteen rakenne on varustettu sovitettu käsittelemään käyttäytymisiä, joilla on nämä kolme näkökohtaa. On huomattava, että eräissä sovellutuksissa mallinnusmuuttuja a voisi olla enemmän kuin yksi arvo, ts. useita vaihtoehtoisia tapahtumia voisi tapahtua saman ajan puitteissa.

Aritmeettisen yksikön (ydinsolun) rakenne

10 Kuten edellä mainittiin, redusointioperaatiot suoritetaan edullisesti erityisessä strukturaalisessa aritmeettisessä yksikössä 3, johon rajallinen solukehikoiden puu siirretään, kun on suoritettava redusointi.

15 Seuraavassa esitetään strukturaalisen aritmeettisen yksikön 3 (ydinsolun) rakenteen suoritusmuodon lyhyt selitys, eräitä esimerkkejä siitä, miten oliomuistiin talletetuilla lausekkeilla ja arvoilla on vastineensa ydinsolun 3 rekistereissä, sekä oliomuistin 1 ja ydinsolun välistä vuorovaikutusta.

20

Ydinsolun rekisterit

Rekisterit, joita voitaisiin käyttää ydinsolun suoritusmuodossa, on esitetty kuvioissa 8B - 8C, ja eräässä ydinsolun suoritusmuodossa käytettävien rekisterien rakenne on esitetty kuviossa 8D.

30 Kuviossa 8A on esitetty rekisteri. Piirustus on tarkoitettu havainnollistamaan sitä, että rekisteri muodostuu rekisterisoluista, jolloin jokainen solu pystyy tallettamaan yhden bitin verran tietoa. Tapa, jolla rekisteri on piirretty, on tarkoitettu havainnollistamaan sitä, että rekisteri ulottuu ydinsolun eri tasojen läpi, jolloin jokainen rekisterisolu sijaitsee yhdessä tasossa.

35

Kuvio 8B esittää rekisterin, joka ulottuu ydinsolun kaikkien tasojen läpi, ts. täysrekisterin. Tämän lajinen rekisteri voi

sisältää tunnistimen tai arvon niissä rekisterisoluissa, jotka sijaitsevat ydinsolun NUM- ja HEAD-tasoissa. Se voi myös sisältää edellä selitetyn tilan rekisterisoluissa, jotka sijaitsevat ydinsolun tasoissa TYPE, WHERE, LAZY ja CLOS/SIMPLE.

5

Kuvio 4C esittää rekisterin, joka ulottuu vain ydinsolun NUM- ja HEAD-tasojen läpi, ts. rajoitetun rekisterin.

10

Kuvio 8D esittää rekisterien mahdollisen rakenteen ydinsolun eräässä suoritusmuodossa. Ydinsolussa on perusrekisterit edullisesti järjestetty neliöksi, jota kutsutaan perusrekisterimatriisiksi. Perusrekistereillä on toista sivua pitkin päärivit, ja näitä sanotaan päärekistereiksi. Perusrekistereiden sarakkeita, joissa on alhaalla yksi päärekisteri, sanotaan apurekistereiksi. Ydinsolu voitaisiin myös varustaa tunnistinrekisterillä ja ympäristörekisterillä. Lisärekisterien sarake on sijoitettu perusrekisterimatriisiin sivulle.

15

20

Ydinsolun eräässä suoritusmuodossa kaikki apurekisterit voisivat olla kuviossa 8C esitettyä muotoa, ts. rajoitettuja rekistereitä, ja muut kuvion 8D rekisterit voisivat olla kuviossa 8B esitetyn lajisia rekistereitä, ts. täysrekistereitä.

25

Ydinsolun laitteistorakenteesta esitetään yksityiskohtaisempi selitys rinnakkaisessa patenttihakemuksessamme nro PCT/SE92/00514. Datat erilaisten talletusmuotojen osalta esitetään lyhyt selitys kuvioihin 9A - 9F viitaten, ja eräitä esimerkkejä sen toiminnasta esitetään kuvioihin 10A - 10H, 11A - 11G ja 12A - 12G viitaten.

30

Yksinkertainen arvo

35

Kuten kuviossa 9A esitetään, redusoinnin tuloksena oleva yksinkertainen arvo 25 esiintyy päärekisterien määrättyssä rekisterissä. Tämä tulos voisi olla eräs osa solukehikkoa.

Yksitasoinen rakenne

Tavoite (goal) on ydinsoluun ladattu jotta se redusoitaisiin. Kuten kuviossa 9B esitetään, vain yhden tason sisältävä tavoite, 5 tyypillisesti kehikko ilman viittauksia muihin solukehikoihin, talletetaan päärekistereihin. Esimerkki esittää yksinkertaisen numeerisen operaation, ts. arvojen 1, 2 ja 3 yhteenlaskun. Numeerinen käsky (+) on talletettu ensimmäiseen päärekisteriin, ja käsiteltävät elementit on talletettu muihin päärekistereihin.

10

Kaksitasoinen rakenne

Kuten kuviossa 9C esitetään, kaksitasoisen rakenteen sisältävä puun juurilista, joka on isä, ja joka on talletettu vaakasuorassa 15 päärekistereihin, ja listat, jotka ovat poikia, pystysuorassa perusrekistereihin. Tässä esimerkissä talletetaan listamuodossa oleva struktuuri ((1 2) (3 4)) perusrekisterimatriisiin. Juurilista, ts. 1 ja 3 jotka ovat alilistojen ensimmäiset elementit, on talletettu päärekistereihin, ja poika-listat, ts. 20 (1 2) ja (3 4) on talletettu pystysuorassa apurekistereihin. Alla esitetään esimerkki, jossa tämän lajista talletusta käytetään; ks. kuviot 9A - 9H.

Kolmitasoinen rakenne

25

Kuten kuviossa 9E esitetään, kolmitasoisen rakenteen sisältävän tavoitepuun juuri on talletettu johonkin lisärekisteriin, ja sen ainoa poika on talletettu päärekistereihin. Kuviossa 9D tavoitepuun juuri, joka on käsky Transpose (Tr), on talletettu yhteen 30 lisärekisteriin ja sen poika, joka on lista (id_1 , id_2 , id_7) on talletettu päärekistereihin. Tämän listan jokainen elementti on vuorostaan isä, jolla on poikia. Kuviossa 9E nämä pojat on ladattu pystysuorassa perusrekistereihin, jolloin id_1 on vaihdettu sen osoittamaksi listaksi, ts. (1 2 3), ja jolloin id_2 on 35 vaihdettu sen osoittamaksi listaksi, ts. (11 12 13), ja jolloin id_7 on vaihdettu sen osoittamaksi listaksi, ts. (21, 22 23).

Liukuhihna-toimintamuoto

Kuten kuviossa 9F esitetään, liukuhihna-toimintamuodossa talletettu puu on ladattu niin, että tavoitelista on päärekistereissä ja tavoitteen isä apurekistereissä, ja sen käskyt ja käsiteltävät elementit on talletettu kummankin lajisiin rekistereihin. Liukuhihna-toimintamuotoa käytetään edullisesti redusointaessa numeerisia lausekkeita. Eräänä etuna on, että välitulokset voidaan tilapäisesti tallettaa ydinsoluun oliomuistin sijasta.

Siten tavoitepuun juurilista edullisesti talletetaan eri paikkoihin ydinsolun rekistereissä, puurakenteen tasosta ja suoritettavasta operaatiosta riippuen.

15 H-kielen rakenteet ja tähän liittyvä laitteisto

Keksinnön mukaiseen tietojenkäsittelylaitteeseen erityisesti soveltuva kieltä sanotaan H-kieleksi. Kompleksisuuden ongelman perinteisinä ratkaisuinä on ollut mutkikkuuden nostaminen, piilottamalla kaikki mutkikkaat asiat ja tekemällä yksinkertaiset näkyviksi. Keksinnön mukainen tietojenkäsittelylaite käyttää päinvastaista menetelmää. Kaikki tarpeettomat ohjelmistotasot on poistettu. H-kieltä käytetään koneessa kaikkeen: konekieltä, ohjelmointikieltä, käyttöjärjestelmää ja tietoliikenneprotokollaa varten. Kuitenkin, kuten kaikissa elektronisissa laitteissa nykyisin, myös keksinnön mukaisen tietojenkäsittelylaitteen langoitusta voitaisiin kohtuullisessa määrin simuloida tietokoneessa, joka on varustettu tulkilla eli kääntäjällä. Esimerkki sellaisesta koneesta on Sun 3. Koneessa on kuitenkin oltava portti, johon on järjestetty yhdistämismahdollisuudet.

Tavanomaisten loogisten ohjelmointikielten semanttisina pääominaisuuksina ovat resoluutio ja yhdistäminen. Keksinnön mukainen tietojenkäsittelylaite, joka on varustettu H-kielellä, käyttää redusointikielessä mallien sovittamista resoluutiota varten, ja vapaita muuttujia ja mallien sovittamista yhdistä-

mistä varten. H-kielen olioilla, jotka on talletettu muistisolujen kenttiin, on ominaisuudet: rinnakkainen olemassaolo, peräkkäinen olemassaolo, ja vaihtoehtoinen olemassaolo.

- 5 Tavoitepuun juuri on redusoitavaa lajia oleva kehikko, kuten unify, alt, apply, ja määrätyssä mielessä myös par ja seq, koska nämä joskus voidaan redusoida tyhjään (nothing). Funktiosovel-
- 10 lutuksessa apply (jolla myös on nimi @) ensimmäinen elementti on käsky (+ - * / during), joka välittömästi tulkitaan
- laitteistolla, tai välillisesti funktion määritelmänä käytetyn
- kehikon osoittava tunnistin, ja muut elementit ovat käskyn/funk-
- tiomääritelmän argumentteja. Kuten edellä mainittiin, keksinnön
- mukainen tietojenkäsittelylaite on sovitettu käsittelemään
- määrättyä selittävää H-nimistä kieltä, joka määritellään abst-
- 15 raktin syntaksin ja semantiikan avulla.

Syntaksi kuvaa erilaisten abstraktien olioiden joukkoa lausek-

keiden avulla. Voitaisiin käyttää seuraavia peruslausekkeita:

- 20 port, nothing, alt(list), par(list), seq(list),
unify(list), during(list), cont(v), period(v),

- jossa (list) on määritelty dataelementtien ($e_1, \dots, e_n$) muodossa
- oliomuistin solussa (soluissa), joka sisältää lausekkeen johon
- 25 se kuuluu, ja v on reaaliluku.

- Koska jokaisella muistisolulla keksinnön langoitetussa sovellu-
- tuksessa on rajoitettu lukumäärä value/designation-kenttiä,
- voidaan lista tallettaa useampaan muistisoluuun. On huomattava,
- 30 että jos oliomuistin rakennetta simuloidaan tavallista tyyppiä
- olevan tietokoneen ohjelmalla, niin simuloidulla muistisolulla
- voisi olla vaihteleva määrä muistikenttiä.

- Port
- 35 lauseke edustaa fyysistä porttia,
ts. during-sekvenssejä, ja se on erityistä
tyyppiä oleva välillinen elementti.

	Nothing	on erityistä lajia oleva arvo, joka edustaa ristiriitaa.
5	Alt	osoittaa, että tämän lausekkeen sisältävä muistisolun sisältää listattuja elementtejä, joita pidetään vaihtoehtoisina elementteinä.
10	Par	osoittaa, että tämän lausekkeen sisältävä muistisolun sisältää listattuja elementtejä, joita pidetään rinnakkaisina elementteinä.
15	Seq	osoittaa, että tämän lausekkeen sisältävä muistisolun sisältää listattuja elementtejä, joita pidetään peräkkäisinä elementteinä.
	Unify	osoittaa, että tämän lausekkeen sisältävä muistisolun sisältää listattuja elementtejä, joita pidetään yhdistettävänä elementteinä.
20	During tai '@(..)	osoittaa että tämän lausekkeen sisältävä muistisolun sisältää listattuja elementtejä, jotka ilmaisevat kestoajan, jonka aikana jotain tulee tapahtumaan.
25	cont	merkitsee tilasuuretta, jossa on ääretön lukumäärä äärettömän pieniä käyttäytymisiä pitkin etäisyyttä, jonka pituus on v.
30	period	merkitsee kestoaikaa, jossa on joukko äärettömän lyhyitä käyttäytymisiä jaksossa, jonka pituus on v.

Listan elementit talletetaan arvoina value/designation-kenttiin.

35

Muistisolujen kanssa vuorovaikutuksessa olevaan keskeiseen ohjauslaitteeseen CU on sisäänrakennettu uudelleenkirjoittami-

sen toiminne. Siten ohjauslaite toteuttaa uudelleenkirjoittamisen säännöt. Se kirjoittaa mainitut syntaksilausekkeet uudelleen ennalta määrättyjen uudellenkirjoittamisen sääntöjen mukaisesti.

5

Keskeinen ohjauslaite on esitetty Boolean porttimatriisipiiri, joka on kytketty lukemaan talletettuja lausekkeita, ja aikaansaamaan uudellenkirjoittamisen operaatiot sisäänrakennettujen sääntöjen mukaisesti. Kerrallaan on luettava ainoastaan osa puusta. Sellaisten Boolean porttimatriisipiirien rakennetta, jotka pystyvät reagoimaan yhdistettyihin tuloihin yhdistettyjen lähtöjen tuottamiseksi sisäänrakennetun protokollan mukaisesti, on selitetty julkaisussa Carver Mead ja Lynn Conway: "Introduction to VLSI Systems", Addison Wesley Publishing Company, 1980.

15

Uudelleekirjoittamisen säännöt

Uudelleenkirjoittamisen säännöt ovat seuraavat:

20

alt

alt()

nothing

alt(e)

e

alt(e₁...nothing...e_n)

alt(e₁...e_n) (ts. arvo
nothing jätetään pois)

25

alt(e₁...alt(c₁...c_k)...e_n)

alt(e₁...c₁...c_k...e_n)

par

par(e₁...nothing...e_n)

nothing, jos jokin arvoista
e₁...e_n on nothing

30

par(e₁...alt(c₁...c_k)...e_n)alt(par(e₁...c₁...e_n)...par(e₁...
c_k...e_n))

par(e₁...e_n)

par(e₁...e_n), muussa
tapauksessa

35

seq

seq(e₁...nothing...e_n)

nothing, jos jokin arvoista
e₁...e_n on nothing

	<u>seq</u> (e1... <u>alt</u> (c1...ck)...en)	<u>alt</u> (<u>seq</u> (e1...c1...en)... <u>seq</u> (e1...ck...en)
	<u>seq</u> (e1...en)	<u>seq</u> (e1...en), muussa tapauksessa
5	<u>unify</u>	
	<u>unify</u> ()	nothing
	<u>unify</u> (e)	e
	<u>unify</u> (e1...en)	e1, jos kaikki arvot e1...en ovat samoja
10	<u>unify</u> (e1...en)	nothing, jos kaikki arvot e1...en ovat erilaisia
	<u>unify</u> (e1 e2 e3...en)	<u>unify</u> (<u>unify</u> (e1 e2) e3...en)
	<u>unify</u> (e1...en)	nothing, jos jokin arvoista e1...en on nothing
15	<u>unify</u> (e1... <u>alt</u> (c1...ck)...en)	<u>alt</u> (<u>unify</u> (e1...c1...en)... <u>unify</u> (e1...ck...en))

Kun tehtävänä on oliomuistiin talletetun puustruktuurin redusointi, ainakin osa puustruktuurista siirretään yksikköön 3.

20 Puustruktuurien yksikössä 3 tapahtuvan käsittelyn eri mahdollisuuksista esitetään alla lisää kuvion 3 järjestelyn erityistä suorituserityistä varten.

Ohjausyksikkö CU lukee nyt yksikköön 3 talletettujen solukehik-

25 koiden TYPE-kentät, ja kirjoittaa uudelleen, ts. järjestää uudelleen solukehikoiden sisällöt yksikössä 3, edellä mainittujen uudelleenkirjoittamisen sääntöjen mukaisesti, ja siirtää uudelleenkirjoitetun tuloksen takaisin talletettavaksi oliomuistiin 1. Jos kirjoitettu tulos pelkistetään arvoksi tai suureeksi

30 nothing, ohjausyksikkö CU suorittaa järjestelmänlaajuisen haun oliomuistissa vaihtaakseen redusoidun puustruktuurin isiä osoittavien muistikenttien sisällöt redusoinnin tuloksena olevaksi arvoksi.

35 Seuraavassa esitetään näitä syntaksin sääntöjä käyttävän toimintatavan esimerkkejä. Alla olevissa esimerkeissä esittämättä olevat syntaksisäännöt ovat hyvin samanlaisia kuin esitetyt, ja

alan ammattilainen ymmärtää ne helposti alla olevien esimerkkien yhteydestä.

Esimerkki 1

5

Kuvioissa 10A - 10H esitetty ensimmäinen esimerkki on rinnakkaisarvojen yhdistäminen, jotka esitetään redusoitavissa olevana kehikkona:

10 unify(par(1 par(1) 3) par(1 par(1) 2))

joka on redusoitavissa oleva kehikko, jossa on tehtävä useampia rinnakkaisia yhdistämissiä. Tämä redusoitavissa oleva kehikko on kirjoitettava uudelleen yhdistämisten rinnakkaisrakenteeksi.

15

Kuvio 10A esittää redusoitavissa olevan kehikon alussa. Kuvio 10B esittää miten tämä redusoitavissa oleva kehikko talletetaan oliomuistiin. Kuvioon 10A on merkitty ne muistisolut, joihin redusoitavissa olevan kehikon eri osat on talletettu. Elementtikehikoiden ja solukehikoiden väliset linkit on merkitty kuvioon 10B. Tunnistimella id₁ varustetulla solukehikolla on merkitsin cls ja tyyppikentässä tyyppikoodi unify, ja tunnistimilla id₂, id₃, id₄ varustetuilla solukehikoilla on tyyppikenttissään tyyppikoodi par. Tunnistimella id₁ varustettu solukehikko sisältää ensimmäiseksi kaksi value/designation-kehikkoelementtiä, jotka osoittavat solukehikkoihin, joilla on tunnistimet id₂ ja id₄. Näillä solukehikoilla on merkitsimenä canon. Tunnistimella id₂ varustettu solukehikko sisältää ensimmäisen ja kolmannen value/designation-kehikkoelementin, joissa on diskreetit arvot merkitsimellä discr, ja toisena arvo/tunnistin -kehikkoelementin, jonka osoittaa solukehikkoon, jolla on tunnistin id₃ ja siten merkitsin canon. Tunnistimella id₃ varustettu solukehikko sisältää ensimmäisen value/designation-kehikkoelementin, jossa on kokonaisluku ja siten merkitsin discr. Tunnistimella id₄ varustettu solukehikko sisältää ensimmäisen ja kolmannen value/designation-kehikkoelementit, joissa on diskreetit arvot merkitsimellä discr, ja toisen value/designation-

nation-kehikkoelementin, jonka osoittaa solukehikkoon, jolla on tunnistin d_3 ja siten merkitsin canon.

5 Kuten kuviossa 10C on esitetty, tunnistimella id_1 varustetun solukehikon sisältö ladataan ensimmäisessä operaatiovaiheessa ensin ydinsoluun, joka sijoittaa sen tunnistimen tunnistinrekisteriin id_1 :nä sisältäen kehikon tyyppikoodin unify, ja value/designation-elementit tavoitteena päärekistereihin.

10 Kuten kuviossa 10D esitetään, tunnistimilla id_2 ja id_4 varustetut pojat ladataan pystysuorassa perusrekistereihin, niin että niiden ensimmäisen value/designation-elementin sisältö sijoitetaan sen tunnistimella merkittyyn päärekisteriin ja muut sen value/designation-elementit rekistereihin pystysuoraan sarakkeeseen sen yläpuolelle. Jokaisen tällaisen pojan tyyppikoodi par ladataan myös päärekisteriin. Tyyppikoodi ladataan rekisterisoluhiin, jotka sijaitsevat TYPE-tasoissa.

20 Kuten kuviossa 10E esitetään, perusrekisterien sisältöjä siirretään (transpose) 90° , niin että perusrekisterien ensimmäisen pystysarakkeen sisältö sijoitetaan päärekistereihin ja toinen pystysarake sijoitetaan perusrekisterin riville, joka on päärekisterien rinnalla. Tunnistinrekisteriin ja päärekistereihin järjestetyt tyyppikoodit par ja unify vaihdetaan keskenään, 25 jonka ohjausyksikkö tekee automaattisesti. Nyt perusrekisterit sisältävät isän, jolla on kolme sarakkeisiin sijoitettua poikaa.

Jokainen poika ladataan nyt takaisin oliomuistiin käyttäen käskyä make. Vastauksena oliomuistista tulevat talletettujen 30 poikien tunnistimet, jotka talletetaan päärekistereihin. On huomattava, että ohjausyksikkö CU ilmaisee sisällöt erityisesti tasoissa CLOS/SIMPLE - TYPE olevissa rekistereissä, ja tuottaa käskyt, ts. se ohjaa kytkimiä ja portteja löytämänsä informaation perusteella. Pojat on nimetty järjestyksessä id_1 :stä alkaen, ja 35 jo varattuja nimiä ei käytetä. Nimien järjestyksellä ei kuitenkaan ole merkitystä, ja voisi siten olla mielivaltaisen.

Kuten kuviossa 10F esitetään, ensimmäinen poika saa tunnistimen `id2`, toinen poika joka sisältää tunnistimen `id3` sisältämät elementtikehikot, saa tunnistimen `id4`, ja kolmas poika saa tunnistimen `id5`. Tunnistimilla `id2`, `id4`, `id5` varustettuihin solukehikoihin linkitettyt elementtikehikot omaava isä säilyttää tunnistimen `id1`, ja talletetaan sitten oliomuistiin.

Kuvio 10G esittää muistisolut, joihin on talletettu redusoitavissa oleva kehikko

```
par(unify(1 1) unify(par(1 par(1)) unify(2 3)))
```

Redusoitavissa oleva kehikko on esitetty kuviossa 10H. Kuviot 10G ja 10H on esitetty samalla tavalla kuin kuviot 10A ja 10B ja ovat siten itsestään selviä.

Kuviossa 10G on myös esitetty, että solukehikoille, joille on annettu tyyppikoodi unify, on myös annettu merkintä exec kentässä LAZY, ja tunnistimella `id1` varustetulle solukehikolle on annettu merkintä wait, joka tarkoittaa, että exec-merkitty solukehikko tulisi suorittaa ennen tunnistimella `id1` varustettua solukehikkoa niiden sisällön redusoimiseksi arvoiksi.

Kuviossa 10H oleva kehikko voitaisiin myöhemmin ladata takaisin ydinsoluun jatkokäsittelyä varten. Esimerkiksi tunnistimella `id2` varustettu solukehikko saa arvon 1, koska sen value/designation-elementtien arvot 1 ja 1 ovat samat, ja tunnistimella `id5` varustettu solukehikko johtaa tulokseen nothing, koska arvot 2 ja 3 sen arvo/tunnus-elementeissä eivät ole samat.

Jokainen yhdistäminen voidaan edullisessa suoritusmuodossa tehdä numeerisessa ALU:ssa, joka vertaa arvoja komparaattoreilla ja johtaa vertailutuloksen ohjausyksikölle CU. Sen jälkeen ohjausyksikkö vastaavasti käyttää Boolean porttimatriisipiiriään tiedon järjestämiseksi ydinsolun ensimmäiseen päärekisteriin. Kun redusointi on johtanut kanoniseen merkintään tai yksinkertaiseen arvoon tai arvoon nothing (ei mitään), niin se jaetaan

järjestelmänlaajuisesti oliomuistin kaikkiin muistikenttiin siten, että redusoidun kehikon jokainen välillinen merkintä korvataan arvon välittömällä merkinnällä.

5 Esimerkki 2

Tämä esimerkki on laitteistkäsky list expansion, joka tarkoittaa, että solukehikko sisältää siihen liitetyn listan. Tämän tyyppinen käsky on muiden redusointien lisävaihe.

10

Kone redusoi esimerkinomaisen käskyn, nimeltään `ex.type`, joka voisi olla minkä tahansa lajinen käsky, joka sisältää arvoja ja listan, ja joka on muotoa

15 `ex.type(1 list(2 3 list(4 5 6))7)`

Muoto on esitetty kuviossa 11A ja sen solukehikko kuviossa 11B. Kuviot 11A ja 11B on merkitty samalla tavalla kuin kuviot 10A ja 10B, ja ovat siten itsestään selviä.

20

Kuten kuviossa 11C on esitetty, tunnistimella `id1` varustettu solukehikko ladataan ydinrekisterin päärekistereihin, joilla on sen tunnistin ja tyyppikoodi tunnistinrekisterissä. Koska toisen päärekisterin sisältö on merkitty välillisellä elementillä open,
 25 niin siihen linkitetty solukehikko `id2` ladataan poikana pystysuorassa perusrekisteriin, kuten kuviosta 11D ilmenee.

25

Laitteisto-käsky list expand siirtää sitten kolmannessa rekisterissä olevan diskreetin arvon 7 tunnistimen `id4` vieressä
 30 olevaan kohtaan kolmannessa perussarakkeessa, ja siirtää toisen päärekisterin yläpuolella toisessa sarakkeessa olevan listan osan kolmanteen sarakkeeseen sijoittaen alimman elementin (arvon 3) kolmanteen päärekisteriin antaen sille tyyppikoodin list ks. kuvio 11E). Koska toisen päärekisterin sisältönä on diskreetti
 35 arvo, sillä on merkitsin discr.

35

Sitten tehdään uusi listan laajennus sijoittamalla päärekisterin

yläpuolella olevan kolmannen sarakkeen sisältö neljänteen sarakkeeseen ja antamalla sille tyyppin list. Kolmannen päärekisterin sisältönä on diskreetti arvo ja sillä on merkitsin discr, kuten ilmenee kuviosta 11F.

5

Sitten neljännen sarakkeen lista talletetaan oliomuistiin käyttämällä käskyä make. Se talletetaan tunnistimella id2 varustettuun muistisoluuun, koska siitä on tullut joutilas (idle) ja tunnistimen id2 lähde lähetetään takaisin ydinsoluun tallettavaksi neljänteen päärekisteriin, kuten on esitetty kuviossa 11G.

10

Sen jälkeen tehdään käskyn ex.type jokin toisen tyyppinen redusointi ennenkuin redusoinnin tulos ladataan takaisin oliomuistiin.

15

KÄYTÄNNÖN ESIMERKKI - ENSIMMÄINEN LÄHESTYMISTAPA

Kuvioon 3 palaten, keksinnön mukaisesti portin toiminnan perusideana on, että nähdään ympäristön 7, 8 ja aktiivisen oliomuistin 1, 3 keskinäinen käyttäytyminen yhdistämisenä, jonka tulisi johtaa siihen, että signaalikvantiteetit jokaisen kestoajan kuluessa saavat vain yhden yhteisen määritellyn arvon.

20

Toimintatavan ymmärtämiseksi esitetään seuraava esimerkki:

25

Tuloportilla ja lähtöportilla varustetun koneen tulisi tuottaa lähtöön tulevien signaalikvantiteettien neliö. Koneeseen talletettava ohjelma on tällöin:

30

```
unify(par(portin portout) "konekäyttäytyminen"),
```

jossa "konekäyttäytyminen" on alt-operaatio, joka tarkoittaa kaikkia tulo- ja lähtöparien ajateltavissa olevia sekvenssejä, ts. seuraavaa lajia olevaa suurta datastruktuuria:

35

```

    alt(seq(par(during(1s 1)during(1s 1))
              (par(during(1s 2)during(1s 4))
               (par(during(1s 3)during(1s 9))
                ...))
5      (seq(par(during(1s 1)during(1s 1))
              (par(during(1s 3)during(1s 9))
               (par(during(1s 2)during(1s 4))
                ...))
10     (seq(par(during(1s 2)during(1s 4))
              (par(during(1s 1)during(1s 1))
               (par(during(1s 3)during(1s 9))
                ...))
15     (seq(par(during(1s 2)during(1s 4))
              (par(during(1s 3)during(1s 9))
               (par(during(1s 1)during(1s 1))
                ...))
20     (seq(par(during(1s 3)during(1s 9))
              (par(during(1s 1)during(1s 1))
               (par(during(1s 2)during(1s 4))
                ...)) )

```

25 Kun tuloportti alkaa tuottaa arvojaan, ydinsolussa tehdyt
 redusointioperaatiot poistavat peräkkäin kaikki vaihtoehtoisess
 operaatiossa alt olevat haarat, joista tulee nothing, ts. ne
 haarat jotka eivät sovi.

Seuraavia redusointisääntöjä suoritetaan ydinsolussa:

Alla olevaa muotoa oleva lauseke

```

30  unify(par(port1...portn)
      seq(par(during(q11 t1) ... during(q1n t1))
      ...
      par(during(qk1 tk) ... during(qkn tk)
      ....)
35  uudelleenkirjoitetaan muotoon
      nothing

```

tai lausekkeeksi

```

      seq(par(during(q11 t1) ... during(q1n t1))
      ...
5      (par(during(qk1 tk) ... during(qkn tk)
      ....)

```

ympäristöstä tulevan signaalin ja lausekkeen vertailusta riip-
 puen. Jokaisen seq-operaation operaatiossa par olevat operaatiot
 10 during käyttävät saman ajan. Tarvitaan tahdistus, jotta kaikki
during-operaatiot käynnistyisivät ja pysähtyisivät samalla
 hetkellä.

Edellä olevan lausekkeen uudelleenkirjoittamisen vaihtoehtona
 15 on sen uudelleenkirjoittaminen muotoon:

nothing

tai lausekkeeksi

```

20      seq(during(par(q11 ... q1n) t1)
      ...
      (during(par(qk1 ... qkn) tk)
      ....)

```

25 ympäristöstä tulevan signaalin ja lausekkeen vertailusta riip-
 puen.

Käyttämällä edellä olevaa ensimmäistä uudelleenkirjoittamisen
 30 sääntöä annetussa esimerkissä, saadaan seuraava redusointisään-
 tö:

```

      unify(par(e1 ... en) par(h1 ... hn))
      ->>      par(unify e1 h1) ... unify en hn)

```

35

ja vastaava pätee seq-operaatiolle. Heti kun tuloarvo ei yhdisty,
 ts. ole sama kuin koneessa oleva vastaava arvo, tuloksena on

nothing. Tämä etenee koko matkan alt-operaatioon saakka, ja koko haara poistetaan alt-lausekkeesta.

Yhteydet ulkomaailmaan

5

Kun portti suunnitellaan sellaiseksi, että arvot voidaan siirtää prosessoriin tai prosessorista riippuen siitä, onko ulkoinen signaali määritelty vai ei tai onko konearvo määritelty vai ei (jolloin sillä on arvo \$), nähdään nerokkuus siinä ettei kone ole rajoitettu edellä selitettyyn yksinkertaiseen tuloportti- ja lähtöporttijärjestelmään. Siten voidaan kirjoittaa ohjelma, jossa kone vuorotellen kirjoittaa porttiin ja lukee sitä. Seuraava ohjelma on esimerkkinä tästä:

10

15 unify(port "konekäyttäytyminen"),

jossa konekäyttäytyminen on:

20

```

alt(seq(during(ls 1)during(ls 1)during(ls 2)during(ls 4)
      seq(during(ls 1)during(ls 1)during(ls 3)during(ls 9)
      seq(during(ls 2)during(ls 4)during(ls 1)during(ls 1)
      seq(during(ls 3)during(ls 9)during(ls 2)during(ls 4)
      jne.

```

ts.

25

```

(seq(during input)(during output)(during input)(during output)

```

Tässä konekäyttäytymisessä ympäristö antaa arvon ja kone vastaa tulostamalla sen neliön seuraavan aikavälin aikana. Tällä menetelmällä voidaan täydellisesti hallita sitä, miten data virtaa yksinkertaisella tavalla.

30

Yksikkö 3 on yhteydessä portteihin ja se on sovitettu tallettamaan portista saatava tulo sekvenssinä oliomuistiin. Se järjestetään lausekkeena, joka on muotoa:

35

```

unify(port seq(during t1 q1) .... seq(during tk qk) ...)

```

Kun tämä lauseke redusoidaan, keskeisen ohjausyksikön CU protokolla on sellainen, että se uudelleenkirjoittaa tämän lausekkeen muodoksi `nothing`, tai uudeksi lausekkeeksi

5 `seq(during newti newqi) ... seq(during newtk newqk) ...)`

riippuen ympäristöstä tulevan signaalin ja lausekkeen vertailusta. Lausekkeessa `newti` on arvoparien sekvenssin `i`:nnen aikavälin uusi kesto aika ja `newqi` uusi signaalikvantiteetti (ks. kuvio 2).

Jos portissa olevalla signaalilla on kuviossa 2 esitetty muoto, edellä oleva `unify`-lauseke uudelleenkirjoitetaan muodoksi `nothing` yksikössä 3, jos signaalin arvo `qi` on erilainen kuin `unify`-lausekkeen antama suure `qi`, tai jos signaalin arvo `ti` on erilainen kuin vastaava `unify`-lausekkeessa annettu arvo. Muutoin `unify`-lauseke uudelleenkirjoitetaan `seq`-lausekkeeksi. Kuten edellä mainittiin, `unify`-lausekkeeseen voitaisiin järjestää erityinen symboli `$`, joka osoittaa että arvo voi olla mielivalta-

On lausekkeessa olevat arvot `ti` tai `qi` ovat `$`, kun `i` on mielivaltainen kokonaisluku, niin vastaavat `newti` tai `newqi` uudelleenkirjoitetaan mitatuilla suureilla `ti` tai `qi`.

Vastaavasti, jos ympäristöstä tuleva signaali on määrittelemätön, vaikka `unify`-lausekkeen suureet `ti` ja `qi` ovat määrättyjä arvoja, niin signaalinvoimakkuus annetaan porttiin (ks. kuvio 6) digitaali/analogia-muuntimella `CONVOUT` suureena `qi` aikana `ti`, samalla kun `seq`-lausekkeen suureet `newti` ja `newqi` ovat samat kuin `unify`-lausekkeen suureet `ti` ja `qi`.

Muita H-kielen rakenteita

35 Kaikki mahdolliset käyttäytymiset voidaan kuvata edellä selitetyn järjestelmän mukaisesti. Valitettavasti ohjelmat voivat olla äärettömän suuria. Sellaisen tilanteen hallitsemiseksi voidaan

käyttää muita kielirakenteita. Tärkeimmät niistä ovat symb, lambda ja apply.

H-kielen hyödyllinen syntaksi voisi sisältää seuraavat syntaksilausekkeet:

```

5      cont(v), delta, period(v), deltat, par(list), seq(list),
      'alt(list), 'con(list), 'pri(list), 'lambda(list),
      'hide(list), 'symb(list), 'unify(list), 'set(list),
10     'apply(list), alt(list), nothing, con(list), pri(list),
      unify(list), lambda(list), hide(list), set(list),
      apply(list), symb(list), discr(n), pulses(n),

```

joissa syntaksilausekkeet on järjestetty muistisolun tyyppikentissä oleviksi operaattoreiksi koodatussa muodossa.

Siten jokainen lauseke on operaattori, joka on talletettu lausekkeen oliomuistin soluun, ja (list):n jokainen elementti on lista, joka osoittaa käyttäytymistä, jolloin n on kokonaisluku ja v reaaliluku. Jokainen elementti voidaan tallettaa muistikenttään muistisolussa, joka sisältää mainitun lausekkeen, johon lista kuuluu.

Lauseke cont(v), jossa v on reaaliluku ja merkitsee avaruuskvantiteettia, jolla on ääretön määrä äärettömän pieniä käyttäytymisiä pitkin etäisyyttä, jonka pituus on v. Lauseke period(v), jossa v on reaaliluku ja merkitsee kestoaikaa, jolla on ääretön määrä äärettömän lyhyitä käyttäytymisiä jakson aikana, jonka pituus on v.

Lauseke delta merkitsee käyttäytymistä, jolla on atomaarinen ulottuvuus avaruudessa ja epämääräinen kesto aika. Lauseke deltat merkitsee käyttäytymistä, jolla on atomaarinen ulottuvuus ajassa ja epämääräinen ulottuvuus avaruudessa.

Kvantiteettien ilmaiseminen koostumuksina

Kokonaisluvun ulottuvuus avaruudessa voitaisiin koodata muodossa par/delta seuraavalla tavalla:

```

5      0=par()
        1=par(delta)
        2=par(delta delta)
        ...
10     5=par(delta delta delta delta delta)

jne.

```

Kokonaisluvun ulottuvuus ajassa voitaisiin koodata muodossa
15 par/deltat samalla tavalla.

Siten kokonaislukua ($\text{discr}(n)$) pidetään eräänlaisten atomaaristen yksiköiden delta laakeana koostumuksena.

20 Kvantiteettien eräs hyvin tärkeä käytötapa on elementteinä during-operaatiossa. Tässä operaatiossa kvantiteetteja käytetään osoittamaan sekä aikaa että avaruutta. Tämän johdosta käytetään delta:n aikavastaavuutta deltat.

25 delta on atomaarinen avaruudessa ja epämääräinen ajassa
deltat on epämääräinen avaruudessa ja atomaarinen ajassa

Edellä olevat lausekkeet talletetaan oliomuistiin samalla tavalla kuin syntaksilausekkeet.

30 Ohjelmat "kielenä"; lausekkeet, joissa on '

Kone redusoi kaikki syötetyt lausekkeet lähes välittömästi niiden arvoiksi. Esimerkiksi lauseke

35 apply(+ par(2 4))

voidaan välittömästi redusoida arvoksi 6. Tämän vuoksi ohjelmaan pääsy on mahdoton, kun se kerran on asennettu. Sitä ei voi murtaa osiin eikä osilla voi tehdä laskutoimituksia.

- 5 Sellaisten toimenpiteiden mahdollistamiseksi otetaan käyttöön peruskielen passiivinen vastike. Näitä sanotaan ohjelmamuodoiksi. Ne redusoituvat itseensä (ne ovat kanonisia) ja pysyvät siten paikallaan. Siten ne voidaan murtaa osiin.
- 10 Funktio plusrevarg voidaan määritellä seuraavalla tavalla:

```
unify(symb(plusrevarg)lambda(par(y x) 'apply(+ par(x y)))
```

- joka annettulla, edellä olevaa ohjelmaa vastaavalla ohjelmamuodolla
- 15

```
'apply(+ par(2 4)
```

- antaa plus-joukon argumentin käänteisessä järjestyksessä,
- 20 par(4 2).

- Talletetun ohjelmamuodon muuntamiseksi vastaavaksi ohjelmaksi käytetään funktiota eval. Siten ohjelma kopioidaan ja "täplät" poistetaan. Sitten ohjelma luonnollisella tavalla redusoituu arvokseen.
- 25

```
apply(symb(eval)'apply(+ par(2 4)))=apply(+ par(2 4))=6
```

- Ohjelmakoostumuksen purkaminen yksinkertaistaa ohjelmilla "järkeilyyn", ja tämän ansiosta tulkitsimien ja kääntäjien kirjoittaminen muita minkä tahansa lajista muita kieliä varten käy yksinkertaisella tavalla käyttäen konetta, jonka peruskielenä on H-kieli.
- 30

- 35 Yleisenä sääntönä kaikilla yhdistävillä rakenteilla (esim. delta, par) on toinen muoto, jota edeltää ' leksikaalisena elementtinä (esim. 'delta', 'par'), joka määrittelee muodon

ohjelmamuodoksi. Tässä on tärkeätä huomata, että vain varsinainen yhdistävä operaatio on ohjelmamuodossa, ei koko puu. Koko puun määrittelemiseksi ohjelmamuodoksi on kaikki yhdistävät operaatiot merkittävä '-merkillä.

5

Jos ohjelmamuodossa (varustettu ':lla) olevan listan sisältönä on nothing, lauseke muunnetaan nothing:ksi.

Ohjelmien muodostaminen

10

Edellä selitettiin määritelmien tekemistä varten tarvittavia mekanismeja, kuten esim. `sqr:n` määritelmää varten. Pienistä määritelmistä koostuvan ohjelman muodostamista varten tarvitaan kuitenkin mukava mekanismi. Halutaan rakentaa ohjelma päälauserkkeena, jolla on joukko lisämääritelmiä. Tämä tehdään rakenteen `con(list)` avulla.

15

Con

20

Rakenne `con` sisältää näkyvän arvon `v` ja useita näkymättömiä arvoja `ei`, jolloin `i` on kokonaisluku välillä 1 - `n`.

`con(v e1 e2 e3...en)`

25

Tämän rakenteen arvo on sama kuin ensimmäisellä elementillä `v`. Jos lista kuitenkin on tyhjä, tai jos jokin listan elementeistä on nothing, niin `con`-rakenteen arvo on nothing. Listan kaikki elementit ovat samassa asemassa. Siten näkymättömiä elementtejä voidaan käyttää rajoituksina, joko sitomalla arvot vapaisiin muuttujiin tai yhdistämällä arvot nothing:ksi.

30

Uudelleenkirjoittamisen säännöt `con`-rakennetta varten ovat

35

<code>con()</code>	-> nothing
<code>con(e₁...nothing...e_n)</code>	-> nothing
<code>con(e₁...e_n)</code>	-> e ₁

Kuten kaikilla muillakin säännöillä, tämä sääntö on myös sisäänrakennettu aktiivisen muistin 1, 3 ja keskeisen ohjausyksikön CU välisenä vuorovaikutuksena.

5 Pri

Mitään edellä kuvatuista rakenteista ei voi käyttää osoittamaan sitä, että jokin on pätevää, jos jokin toinen ei ole pätevä (on nothing). Koska "negatiivinen" ilmaisutapa joskus on paljon
10 kompaktimpi kuin "positiivinen" ilmaisutapa, on rakenne pri lisätty "negatiivista" ilmaisutapaa varten.

Rakenteen pri redusointisäännöt ovat sellaiset, että pri-rakenteen arvo on ensimmäinen elementti, joka on muu kuin nothing.
15 Ellei sellaista elementtiä ole, pri on nothing. Siten saadaan:

$$\text{pri}(\text{nothing nothing } 9 \text{ nothing } 7 \text{ } 8) = 9$$

Rakenne pri(e1 e2) voidaan tulkita seuraavalla tavalla: ellei
20 e1 ole pätevä, niin silloin on käytettävä e2:ta.

Rakenne pri muistuttaa rakennetta alt. Se on käyttäytymisjoukko, mutta pri:ssä käyttäytymiset ovat järjestyksessä oleva joukko. Ensimmäinen käyttäytyminen on listan ensimmäisenä, toinen
25 toisena, jne.

On tärkeätä huomata, että rakenteella pri ei ole mitään yhteistä käsitteellä prioriteetti, jota käytetään tavanomaisissa kielissä ja koneissa prosessien järjestämiseen. Sen sijaan sitä
30 käytetään toteuttamaan negatio puuttumisen johdosta.

Uudelleenkirjoittamisen säännöt pri-rakennetta varten ovat

35	<u>pri</u> ()	-> nothing
	<u>pri</u> (nothing ₁ ...nothing _n)	-> nothing
	<u>pri</u> (nothing ₁ ...nothing _{k-1} e _k ...e _n)	-> e _k

Kuten kaikilla muillakin säännöillä, tämä sääntö on myös sisäänrakennettu aktiivisen muistin 1, 3 ja keskeisen ohjausyksikön CU välisenä vuorovaikutuksena.

5 Ulottuvuuden (scope) säännöt

Ulottuvuus on teksti-ympäristö, jonka puitteissa saman symbolin kaikki esiintymiskerrat edustavat samaa semanttista kokonaisuutta. Ulottuvuuden sääntö määrää jos, ja sitten miten, määrätty rakenne tuottaa uuden ulottuvuuden. H-kielessä on seuraavat säännöt ulottuvuutta varten:

1. Rakenteet `par`, `seq`, `alt`, `@`, ja `unify` eivät tuota uutta ulottuvuutta. Tämä tarkoittaa, että kaikki symbolin esiintymiskerrat lausekkeessa tarkoittavat samaa kokonaisuutta.
2. Rakenne `lambda`(korvaava malli) tuottaa uuden alempitasoisen ulottuvuuden. Kaikki symbolien esiintymiskerrat, joita ei mainita ylempitasoisessa ulottuvuudessa, kuuluvat uuteen ulottuvuuteen.
3. Rakenne `hide`(lauseke) tuottaa täysin uuden ulottuvuuden. Mikään sääntörakenteen `hide` puitteissa olevan symbolin esiintymiskerta ei edusta samaa symbolia kuin `hide`-rakennetta ympäröivässä sääntörakenteessa.

25 Hide

Tuotettaessa laajoja ohjelmia on käytettävä samoja symboleja eri merkityksissä. Tämä voidaan aikaansaada käyttämällä rakennetta (`hide`), joka tuottaa eristettyjä ulottuvuuksia.

30

Piilottamisen (`hide`) erittäinen tärkeätä käsitettä käytetään pääasiassa olion käytön ja kyseisen olion määritelmän eristämiseksi. Keksinnön mukaisen tietojenkäsittelylaitteen käyttäjä voi löytää tämän ohjelmapaketista, kirjastosta ja sovellutusten käsitteistä.

35

H-kieli määrittelee piilottamisrakenteen `hide(e)`. Se määritte-

lee käyttäytymisen. Rakenteen sisäpuolella ja ulkopuolella olevat vapaat muuttujat ovat erillään, ts. ne eivät näe toisiaan. H-kielen tämä ominaisuus poikkeaa muun tapaisista kielistä, jotka yleensä antavat ulospäin datatyyppin ja useita symbolinimiä. Sellaisia nimiä ei voi käyttää paketin sisäpuolella. Tämä tarkoittaa sitä, että yleispätevä symbolipiiri on todella yleispätevä kaikille paketeille, ja näkyy niiden sisäpuolelle täydellisenä tai rajoitettuna joukkona.

10 Tyypillinen ohjelmapaketti määritellään seuraavalla tavalla:

```
lambda(a_pkg $ $).hide(...paikallinen sovellutus...);
```

15 Se määrittelee säännön, jonka malli alkaa ilmaisulla a_pkg, ja loppuosa on määrittelemätön. Tämä on pelkästään säännön määritelmä, jossa säännön korvauksen paikalliset symbolit ovat tuntemattomia.

20 Kirjasto on H-kielen käsitteenä paketti, jossa käytetään parametria, jolla valitaan yksi tai useampia sääntöjä. Sitä voidaan soveltaa muodossa:

```
hide(
  alt(lambda "sääntö a".a-spec
25    lambda "sääntö b".b-spec
    ...
    lambda "sääntö n".n-spec))
```

30 Kirjasto on vaihtoehto, joka sisältää sääntöjä. Mallia "sääntö a" käytetään ohjelman a spec valitsemiseksi. Se voidaan määritellä mielivaltaisena rakenteena, mutta yleensä se on abstraktio tai sääntö.

35 Kuten edellä osoitettiin, H-kielessä on elegantti menetelmä tarvittavien ohjelmistorakenteiden käsitteiden määrittelyä varten. Kielen ei tarvitse muodostaa uusia kielirakenteita tätä varten.

On huomattava, että `hide` vaikuttaa vain ulottuvuuden (`scope`) käsittelyyn.

lambda

5

Lauseke `lambda(es ep1...epn)` on sääntö, joka osoittaa että elementti `es` korvaa elementit `ep1...epn`; jolloin `er` on sääntö `lambda(es ep1...epn)` ja mainittu lauseke `apply(er earg1...earg2)` on sovellutus, joka redusointioperaatioissa korvataan `es`:llä, jos jokainen `epi` yhdistyy vastaavaan suureeseen `eargi`, tai muuten se korvataan suureella `nothing`.

10

apply

15 Seuraavassa esitetään esimerkki siitä, miten sääntöä `apply` sovelletaan.

Esimerkki 3

20 Suoritettavana on numeerinen käsky. Numeerinen käsky voi olla `+`, `-`, `*`, `/`, `mod`, jne. Käskyn jälkeen seuraavat argumentit. Tässä esimerkissä tehtävänä on listan lukujen yhteenlasku. Kone redusoi käskyn `apply (application, soveltaminen)`, jonka muoto on

25

```
apply(+ list(1 2))
```

30

Sovellutus on esitetty kuviossa 12A ja sen solukehikot kuviossa 12B. Kuviot 12A ja 12B on merkitty samalla tavalla kuin kuviot 10A ja 10B, ja siten ne ovat itsestään selvät.

35

Kuten kuviossa 12C on esitetty, tunnistimella `id1` varustettu solukehikko ladataan ydinrekisterin päärekistereihin, joilla on sen tunnistin ja tyyppikoodi tunnistinrekisterissä. Käskyksi on merkitty numeerinen käsky `(+)`. Koska toisen päärekisterin sisältö on merkitty välillisellä elementillä `open`, niin siihen linkitetty solukehikko ladataan poikana pystysuorassa perusre-

kisteriin, kuten kuviosta 12D ilmenee.

Sitten tehdään listan laajennus (list expansion), jolloin toisessa päärekisterissä oleva diskreetti arvo merkitään discr,
 5 ja tyyppikoodikentässä listan laajennusarvo 2 merkitään list. Tämä tehdään, koska kone tekee saman operaation riippumatta siitä onko tunnistimella id₂ varustetussa listassa kaksi, kolme tai neljä elementtiä. Koska uudessa listassa on vain yksi elementti, kone korvaa merkinnän list osoituksella siitä, että päärekisteri
 10 sisältää discr-arvon, kuten ilmenee kuviosta 12F.

Sitten päärekisteri sisältää käskymerkin (+) ja kaksi diskreettiä arvoa, ja tämä saattaa ohjausyksikön välillisesti, tai esimerkiksi oliomuistin käskyt tallettavaan ei-kirjoitettavaan
 15 osaan talletettua käskyä koskevan tiedon avulla, ohjaamaan numeerisen ALU:n suorittamaan käskyn (yhteenlaskun) ja tuottamaan numeerisen operaation tuloksen kanonisena arvona ensimmäiseen päärekisteriin, kuten on esitetty kuviossa 12G. On huomattava, että merkintä apply tyyppikoodikentässä on merkintä
 20 siitä, että suoritettavana on funktion soveltaminen. Tuloksena oleva arvo, tässä tapauksessa yksinkertainen arvo 3, jaetaan sitten yleisesti tunnistimen id₁ jokaisen esiintymisen korvaamiseksi tällä arvolla.

25 Uudelleenkirjoitussäännöt

Uudelleenkirjoitussäännöt määrittelevät samansisältöiset lausekkeet. Näihin sääntöihin liittyy uudelleenkirjoitussuunta. Tuloksena on yksinkertaisempi lauseke, kun noudatetaan uudelleenkirjoittamissuuntaa. Kun lauseketta ei voi kirjoittaa
 30 uudelleen yksinkertaisemmaksi lausekkeeksi, se on kanoninen. Siten keksinnön mukaisessa tietojenkäsittelylaitteessa käytetään uudelleenkirjoitussääntöjä lausekkeiden kirjoittamiseksi uudelleen kanonisempiin muotoihin.

35

Edellä esitettyjen uudelleenkirjoittamisen sääntöjen lisäksi keskeisessä ohjausyksikössä CU toteutetaan myös seuraavat

uudelleenkirjoitussäännöt:

type($e_1 \dots \text{nothing} \dots e_n$) $\rightarrow$ nothing, jossa type edustaa lauseketta, joka on edellä olevassa listassa lähtien lausekkeesta par(list) lausekkeeseen unify(list) saakka, ja $e_1 \dots e_n$ ovat listaelementtejä.

type($e_1 \dots \text{alt}(c_1 \dots c_n) \dots e_n$) $\rightarrow$
alt(type($e_1 \dots c_1 \dots e_n$) $\dots$ type($e_1 \dots c_k \dots e_n$))

jossa type edustaa lauseketta, joka on edellä olevassa listassa lähtien lausekkeesta par(list) lausekkeeseen unify(list) saakka (lukuunottamatta alt(list)), ja e_1, e_n, c_1, c_k ovat listaelementtejä.

set() $\rightarrow$ nothing

set(a) $\rightarrow$ 'alt(a), jos a on lauseke, joka on edellä olevassa listassa lähtien lausekkeesta par(list) lausekkeeseen 'apply(list) saakka.

set(alt($e_1 \dots e_n$)) $\rightarrow$ 'alt($e_1 \dots e_n$), jossa $e_1 \dots e_n$ ovat elementtejä vailla mitään määrättyä järjestystä, ja $e_a \dots e_m$ ovat lajitellussa järjestyksessä.

set($e_1 \dots e_n$) $\rightarrow$ set(con($e_1 \dots e_n$))

lambda(alt($e_1 \dots e_n$)) $\rightarrow$ set(alt($e_1 \dots e_n$))

lambda($e_1 \dots e_n$) $\rightarrow$ par($e_1 \dots e_n$)

hide(alt($e_1 \dots e_n$)) $\rightarrow$ set(alt($e_1 \dots e_n$))

hide($e_1 \dots e_n$) $\rightarrow$ par($e_1 \dots e_n$)

	<u>apply()</u>	-> <u>nothing</u>
	<u>apply(e)</u>	-> <u>eval_{meta}(e)</u>
5	<u>apply('alt(b₁...b_n)a₂...a_m)</u>	-> <u>apply(eval_{meta}'alt(b₁...b_n)a₂...a_m)</u>
10	<u>apply(par(b₁...b_n)a₂...a_m)</u>	-> <u>nothing</u> , jos kaikki suureet b _i ja a _i ovat lausekkeita, jotka ovat edellä olevassa listassa lähtien lausekkeesta <u>par</u> (list) lausekkeeseen ' <u>apply</u> (list) saakka, jos n#m tai mikä tahansa a _i #b _i , kun i>1.
15	<u>apply(par(b₁...b_n)a₂...a_m)</u>	-> b ₁ , jos kaikki suureet b _i ja a _i ovat lausekkeita, jotka ovat edellä olevassa listassa lähtien lausekkeesta <u>par</u> (list) lausekkeeseen ' <u>apply</u> (list) saakka, jos n=m tai jokainen a _i =b _i , kun i>1.
20	<u>apply(seq(b₁...b_n)a₂...a_m)</u>	-> <u>nothing</u> , jos kaikki suureet b _i ja a _i ovat lausekkeita, jotka ovat edellä olevassa listassa lähtien lausekkeesta <u>par</u> (list) lausekkeeseen ' <u>apply</u> (list) saakka, jos n#m tai mikä tahansa a _i #b _i , kun i<1.
25	<u>apply(seq(b₁...b_n)a₂...a_m)</u>	-> b ₁ , jos kaikki suureet b _i ja a _i ovat lausekkeita, jotka ovat edellä olevassa listassa lähtien lausekkeesta <u>par</u> (list) lausekkeeseen ' <u>apply</u> (list) saakka, jos n#m tai mikä tahansa a _i #b _i , kun i<1.
30	<u>apply(seq(b₁...b_n)a₂...a_m)</u>	-> b ₁ , jos kaikki suureet b _i ja a _i ovat lausekkeita, jotka ovat edellä olevassa listassa lähtien lausekkeesta <u>par</u> (list) lausekkeeseen ' <u>apply</u> (list) saakka, jos n=m tai jokainen a _i =b _i , kun i>1.
35		

Funktio eval_{meta} muuntaa ohjelmamuodossa olevat lausekkeet (jotka ts. on varustettu merkillä ') arvomuotoon (ts. ilman '). Merkki ' poistetaan vain ylimmästä struktuurista.

5 eval_{meta}'type(a) -> type(a), jossa type edustaa lauseketta, joka on edellä olevassa listassa lähtien lausekkeesta par(list) lausekkeeseen unify(list) saakka, lukuunottamatta alt(list).

10

symb

15 Lauseke symb(s) sisältää elementin s, joka kanoniseksi lausekkeeksi redusoituna on tunnistin, jolloin mainittu elementti voidaan tallettaa mainitun oliomuistin eri muistisoluihin kuuluvaa muistikenttään, jolloin mainitut muistisolut on määriteltä järjestettäväksi ulottuvuuksiksi, ja jolloin kaikki tunnistimet kulloisenkin ulottuvuuden puitteissa osoittavat samaa käyttäytymistä.

20

Discr, pulses

25 Lauseke discr(n), jossa n on kokonaisluku, osoittaa rinnakkaista käyttäytymistä, jolla on n atomaarista käyttäytymistä. Lauseke pulses(n), jossa n on kokonaisluku, osoittaa n:n atomaarisen käyttäytymisen aikaketjua.

Seuraavan uudelleenkirjoitussäännöt pätevät:

30 par(delta₁...delta_n) -> discr(n)
 seq(deltat₁...deltat_n) -> pulses(n)

KÄYTÄNNÖN ESIMERKKI - TOINEN LÄHESTYMISTAPA

35 Esimerkkien antamiseksi laajennetun syntaksin käyttömahdollisuuksista, voitaisiin ohjelma, jolla samassa portissa vaihtoehdoisesti luetaan ja kirjoitetaan, toteuttaa tiiviimmässä muo-

dossa. Ensin merkintä square (sqr) diskreeteillä arvoilla 1, 2 ja 3 määritellään seuraavalla lyhyellä ohjelmalla:

```
unify(symb(sqr)alt(lambda(1 1)lambda(4 2)lambda(9 3))
```

5

joka on talletettu oliomuistin 1 kehikkojen puuhun. Siten "konekäyttäytyminen" voidaan kirjoittaa ainoastaan yhtenä seq-operaationa:

```
10  seq(during(1s x)during(1s  apply(symb(sqr( x)
    (during(1s x)during(1s  apply(symb(sqr( x)...
```

Suorituksen aikana tämä seq-operaatio uudelleenkirjoitetaan edellä selitetyksi alt-operaatioksi. Myös tämä ohjelma on yhtä
15 pitkä kuin tulodata. Eräs elegantti tapa tämän käsittelemiseksi on järjestää joku määrä funktioita ja vakiovaihtoehtoja laitteistona, tai ohjelmistona, joka ladataan yhdessä kyseessä olevan ohjelman kanssa.

20 Esimerkki tästä on kertolasku (x), ja tämän vakio-operaation avulla voidaan arvon neliö kirjoittaa yksinkertaisena ohjelmana:

```
unify(symb(sqr) lamda(symb(a) apply(x par(symb(a)
                                                    symb(a))))
```

25

Tämä ohjelma pystyy neliöimään kaikki laitteiston määrittelemät luvut.

Tätä sääntöä voidaan käyttää pienten rekursiivisten ohjelmien
30 kirjoittamiseksi, jotka uudelleenkirjoittamisen aikana lavenne-
taan redusointisääntöjen mukaisesti suureksi alt-rakenteeksi,
joka mainittiin edellä. Laitteisto on sellainen, että tarpeeton
lavennus vältetään. Tämä tehdään edullisesti käyttämällä rin-
nakkaisessa patenttihakemuksessamme nro PCT/SE91/00514 selite-
35 tyn muotoisella ydinsolulla.

Kuvattuja uudelleenkirjoitussääntöjä toteuttamaan pystyvän tie-

tojenkäsittelylaitteen käyttäytyminen, joka osoittaa selittävää lähestymistapaa käyttämällä käyttäytymisten yhdistämistä, on määritelty määrittelemällä järjestys kuluvaan ajan puitteissa. Ajallinen selitys muistuttaa jonkin verran merkintäkieltä.

5

Ohjelmien saattaminen luettavammiksi

Abstrakti syntaksi on varsin yksinkertainen. Tämä ei valitettavasti tee siitä kovinkaan luettavaa. Ohjelmien saattamiseksi
10 luettavammiksi, otetaan käyttöön mutkikkaampi syntaksi, koska ihminen ei halua liian suppeata mutkikkuutta.

Tämä muunnos voitaisiin tehdä useammalla eri tavalla. Kuitenkin on luonnollista, että otetaan käyttöön tekstiin perustuva ja
15 graafinen muoto.

Virhe

Virheilmoitus aikaansaadaan, kun laitteisto havaitsee ettei arvo
20 ole oikea, esimerkiksi tarkistamalla laskettujen arvojen pariteettibitit, tai vastaavalla tavalla. Tällöin käytetään lauseketta backup.

Lausekkeen backup redusointisäännöt ovat sellaiset, että
25 backup:n arvo on ensimmäinen arvo, joka on erilainen kuin fault (virhe). Ellei sellaista elementtiä ole, backup on nothing.

Lausekkeen backup uudelleenkirjoitussäännöt ovat:

30 backup() -> fault
 backup(fault₁...fault_n) -> fault
 backup(fault₁...fault_{k-1} e_k...e_n) -> e_k

Kuten muidenkin sääntöjen osalta myös nämä säännöt ovat
35 sisäänrakennettuja aktiivisen muistin 1, 3 ja keskeisen ohjausyksikön välisenä vuorovaikutuksena.

Useita kielen esitystapoja

Kuten edellisestä käy ilmi, H-kiielellä on useampia esitystapoja. Kielessä on ydin Habs, joka on abstrakti kieli, jolle käytetään
 5 esitystapojen perhettä. Kielen pääasialliset esitystavat ovat:

Habstämä kielen esitystapa on muodollista määritelmää lukuunottamatta periaatteellisin. Se on täysin selittävä. Sillä ei
 10 kuitenkaan ole käyttäjän luettavissa olevaa syntaksia.

Hfaulttämä kielen esitystapa on Habs laajennettuna rakenteilla tietojenkäsittelylaitteen laitteistovikojen mallintamiseksi. Kielen tämä esitystapa ei ole selittävä; sen sijaan tuloksena on stokastinen arvo. Tämä esitystapa on kielen esitystavan juuri,
 15 johon kielen kaikki muut esitystavat perustuvat. Sillä ei ole käyttäjän luettavissa olevaa syntaksia.

Husertämä kielen esitystapa on Hfault laajennettuna rakennetuilla joukoilla kokonaisluku- ja reaalitylukuaritmetiikkaa varten sekä
 20 numeerista ja strukturaalista aritmetiikkaa varten. Myös ääretömät vaihtoehdot on sisällytetty. Kielen esitystavalla Huser ei ole mitään käyttäjän luettavissa olevaa syntaksia.

Hasciion kielen esitystapa, joka käyttää tavallisia ASCII-merkkien ja etumerkkien esitystapaa koko syntaksia varten, lukuunottamatta tavallisimpia lausekkeita, joille käytetään infiksi-
 25 syntaksia.

Vaikka keksintöä on selitetty määrättyihin suoritusmuotoihin viitaten, alan ammattilaiset ymmärtävät, että siihen voi tehdä
 30 lukuisia muutoksia ja sen elementtejä voi korvata ekvivalenteilla poikkeamatta keksinnön varsinaisesta hengestä ja suoja-alasta. Lisäksi voidaan tehdä muunnelmia poikkeamatta keksinnön oleellisista opetuksista.

Patenttivaatimukset

1. Redusointityyppinen tietojenkäsittelylaite muistivälineeseen (Cu, 1) talletettujen ohjelmien arvioimiseksi, joka sisältää
 5 useita muistisoluvälineitä (2), jolloin tietojenkäsittelylaite myös sisältää ainakin yhden porttivälineen (4, 5, 6, 11) ja ainakin yhden ympäristövälineen (7, 8, 9, 12), joka on kytketty mainittuun ainakin yhteen porttivälineeseen, tunnettu siitä, että

10 a) muistiväline (Cu, 1) on aktiivinen, niin että muistisoluvälineiden (2) joukossa jokaisella on tietoa, joka voi aiheuttaa operaation suorittamisen;

b) mainittu ainakin yksi porttiväline (4, 5, 6, 11) on kytketty aktiiviseen muistivälineeseen (Cu, 1), ja jossa vertailuvälineet
 15 (SEL₁, SEL₂, Cu₁, Cu₁) ympäristövälineen jossakin porttivälineessä tarjoaman datan vertaamiseksi yhteen mainittuun muistisoluvälineeseen (2) talletetun datan kanssa, joka muistisoludata vertailuhetkellä on järjestetty mainitun porttivälineen toiselle puolelle, ja

20 c) jolloin mainitun porttivälineen toisella puolella olevassa datassa on mahdollisia määrittämättömiä dataelementtejä (\$) tai osittain määriteltyjä dataelementtejä (alt).

2. Patenttivaatimuksen 1 mukainen tietojenkäsittelylaite, tunnettu siitä, että mainitut porttivälineet on sovitettu vertaamaan signaalisekvenssiä, ja sisältää välineet (Cu, 3; Cu₁, Cu₂) vertailevan sekvenssin uudelleenkirjoittamiseksi johonkin, joka edustaa puutteellisuutta, jos vertailu tuottaa selkeän eron, tai muutoin mainitun vertailevan sekvenssin uudelleen kirjoittamiseksi määrättyyn sekvenssiin.
 25
 30

3. Patenttivaatimuksen 1 tai 2 mukainen tietojenkäsittelylaite, tunnettu siitä, että vertailuvälineet (SEL₁, SEL₂, Cu₁, Cu₂) on sovitettu vertailemaan ryhmiä, jotka sisältävät ainakin kaksi elementtiä (V_{DU}, t_{DU}), joissa on ennalta määrätty lukumäärä listaelementtejä.
 35

4. Patenttivaatimuksen 2 mukainen tietojenkäsittelylaite, **tunnettu** välineistä (7, 8, 9; TCOMP, SEL1, Tcount, SEL2, CONV_{IN}, CONV_{OUT}) signaalisekvenssin sovittamiseksi ajan mukana muuttuvaksi näytteistetyksi signaaliksi, jolla on yksilölliset näytteenottojaksot, jolloin signaalisekvenssi on elementtiryhmiensä lista, jolloin jokainen ryhmä sisältää kestoajan (t_1 , t_2 , jne) ja ainakin yhden signaalikvantiteetin (q_1 , q_2 , jne) tämän ajan kuluessa.

5. Patenttivaatimuksen 4 mukainen tietojenkäsittelylaite, **tunnettu** siitä, että jokaisen vertailuvälineelle esitetyn ryhmän listaelementtien ennalta määrätty lukumäärä on kaksi, jotka on järjestetty pariaksi, jolloin jokainen pari sisältää kestoajan ja signaalikvantiteetin yhdistelmän (t_1 , q_1 ; t_2 , q_2 ; jne).

6. Jonkin edellisen patenttivaatimuksen 1 - 5 mukainen tietojenkäsittelylaite, **tunnettu** välineistä (Cu_1 , sync., Cu_2) yhden porttivälineen tahdistamiseksi ainakin yhden toisen porttivälineen kanssa sekä porteista tulevien signaalikvantiteettien rinnakkaista osoittamista varten jokaisen kestoajan aikana.

7. Jonkin edellisen patenttivaatimuksen 1 - 6 mukainen tietojenkäsittelylaite, **tunnettu** siitä, että muistisoluvälineet on järjestetty struktuuriksi aktiivisessa muistissa, joka on sovitettu tietokoneohjelman tallettamiseksi abstraktin syntaksin eksplisiittisen tai implisiittisen koodauksen muodossa, jolloin syntaksi lausekkeiden avulla kuvaa erilaisten abstraktien olioiden joukkoa, jolloin jokainen muistisoluväline pystyy tallettamaan ainakin osan yhdestä mainituista syntaksilausekkeista kerrallaan sopivan data- ja/tai ohjelmastruktuurin muodossa.

8. Patenttivaatimuksen 7 mukainen tietojenkäsittelylaite, **tunnettu** siitä, että muistisoluvälineessä syntaksin jokaisella lausekelajilla on myös syntaksin vastaava lausekelaji, joka ilmaisee, että se on ohjelmamuoto, jolloin kaikki ohjelmamuodot ovat lausekkeitä, jotka on sovitettu redusoitaviksi itseensä ja

siten pysymään paikallaan.

9. Patenttivaatimuksen 8 mukainen tietojenkäsittelylaite, **tunnettu** uudelleenkirjoittamisvälineistä (CU), jotka ovat vuorovaikutuksessa syntaksilausekkeet sisältävien muistisolujen kanssa syntaksilausekkeiden uudelleenkirjoittamiseksi ennalta määrättyjen uudelleen kirjoittamisen sääntöjen mukaisesti.

10. Patenttivaatimuksen 9 mukainen tietojenkäsittelylaite, **tunnettu** siitä, että syntaksi sisältää ainakin joitakin seuraavista syntaksin peruslausekkeista:

port, nothing, alt(list), par(list), seq(list), unify(list), during(list), cont(v), period(v), jossa (list) on dataelementtien ($e_1, \dots, e_n$) lista, joista jokainen on talletettu yhden muistisolun muistikenttään, joka muistisolu sisältää lausekkeen johon se kuuluu, jos muistisolussa on riittävästi tilaa varten, ja muutoin talletettu ainakin sellaiseen muistisoluun, josta on linkki lausekkeen sisältävään muistisoluun, ja jossa v on reaaliluku, ja jossa list lausekkeessa during(list) voi sisältää seuraavat elementit: cont(v), period(v) ja mielivaltaisen arvon (\$), ja jossa lauseke port edustaa jotakin fyysisistä porteista ja on erityinen välillisen elementin laji, nothing on erityistä lajia oleva arvo, joka edustaa ristiriitaa, alt tarkoittaa, että tämän lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään vaihtoehtoisina elementteinä, par että tämän lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään rinnakkaisina elementteinä, seq että tämän lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään peräkkäisinä elementteinä, unify että tämän lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään yhdistävinä elementteinä, during että tämän lausekkeen sisältävä muistisolu sisältää kestoajan ja signaalikvantiteetin sisältävän parin, cont(v) merkitsee avaruuskvantiteettia, jolla on ääretön lukumäärä äärettömän pieniä käyttäytymisiä pitkin etäisyyttä, jonka pituus on v, ja period

merkitsee kestoaikaa, jolla on ääretön lukumäärä äärettömän lyhyitä käyttäytymisiä kestoajaltaan v olevan jakson aikana.

11. Patenttivaatimuksen 9 ja 10 yhdistelmän mukainen tietojenkäsittelylaite, tunnettu siitä, että uudelleenkirjoitussääntöjä ovat ainakin muutamat alla olevista valitut:

```

nothing          -> nothing
alt(e)           -> e
10  alt(e1...nothing...en)
      -> alt(e1...en) (ts. arvo nothing jätetään
      pois)
      alt(e1...alt(c1...ck)...en) -> alt(e1...c1...ck...en)
      par(e1...nothing...en)
15      -> nothing, jos jokin arvoista e1...en
      on nothing
      par(e1...alt(c1...ck)...en)
      -> alt(par(e1...c1...en)...par(e1...ck...en))
      par(e1...en) -> par(e1...en), muussa tapauksessa
20  seq(e1...en)  -> nothing, jos jokin arvoista e1...en
      on nothing
      seq(e1...alt(c1...ck)...en)
      -> alt(seq(e1...c1...en)...seq(e1...ck...en))
      seq(e1...en) -> seq(e1...en), muussa tapauksessa
25  unify()       -> nothing
      unify(e)    -> e
      unify(e1 e2) -> e2, jos e1=e2
      unify(e1 e2) -> nothing, jos e1 erisuuri kuin e2
      unify(e1 e2 e3...en)
30      -> unify(unify(e1 e2) e3...en)
      unify(e1...en) -> nothing, jos jokin arvoista e1...en
      on nothing
      unify(e1...alt(c1...ck)...en)
      -> alt(unify(e1...c1...en)...unify(e1...ck...en))
35  )

```

12. Patenttivaatimuksen 8 mukainen tietojenkäsittelylaite, tunnettu siitä, että syntaksi sisältää ainakin yhden syntaksi-

lausekkeen, joka on valittu seuraavasta joukosta:

cont(v), period(v), deltat, par(list), seq(list),
'alt(list), 'con(list), 'pri(list), 'lambda(list),
'hide(list), 'symb(list), 'unify(list), 'set(list),
5 'apply(list), alt(list), nothing, con(list), pri(list),
unify(list), lambda(list), hide(list), set(list),
apply(list), symb(list), discr(n), pulses(n),

jossa jokainen syntaksilauseke on lausekkeena oliomuistisoluun talletettu operaattori ja jolloin jokainen (list):n elementti
10 on käyttäytymistä osoittava listaelementti, jolloin n on kokonaisluku ja v reaalityluku, jolloin jokainen elementti voidaan tallettaa muistisolun muistikenttään, joka sisältää lausekkeen, johon lista kuuluu; jolloin lauseke cont(v) merkitsee avaruuskvantiteettia, jolla on ääretön lukumäärä äärettömän pieniä
15 käyttäytymisiä pitkin etäisyyttä, jonka pituus on v; jolloin lauseke period(v) merkitsee kestoaikaa, jolla on ääretön lukumäärä äärettömän pieniä käyttäytymisiä jakson aikana, jonka pituus on v; jolloin lauseke delta merkitsee käyttäytymistä jolla on atomaarinen ulottuvuus avaruudessa ja epämääräinen ulottuvuus
20 ajassa; jolloin lauseke deltat merkitsee käyttäytymistä jolla on atomaarinen ulottuvuus ajassa ja epämääräinen ulottuvuus avaruudessa; jolloin lauseke par(list) merkitsee sitä, että tämän lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään rinnakkaisina elementteinä; jolloin lauseke seq(list) että tämän lausekkeen sisältävä
25 muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään peräkkäisinä elementteinä; jolloin kaikki leksikaalisen elementin ' sisältävät lausekkeet määrittävät muodon ohjelmamuodoksi; jolloin lauseke alt(list) merkitsee, että tämän
30 lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään vaihtoehtoisina elementteinä; jolloin lauseke nothing on erityistä lajia oleva arvo, joka edustaa ristiriitaa; jolloin lauseke con(list) sisältää listan, jossa on ainakin yksi nolla, mutta myös useita käyttäytymiselementtejä, ja on identtinen listassaan olevan ensimmäisen elementin kanssa, jos listan kaikki elementit ovat kanonisia eivätkä nothing; jolloin lauseke pri(list) sisältää listan,

jossa on ainakin yksi nolla, mutta myös useita käyttäytymiselementtejä, ja on identtinen listassaan olevan ensimmäisen kanonisen elementin kanssa joka ei ole nothing; jolloin lauseke unify(list) tarkoittaa, että tämän lausekkeen sisältävä muistisolu sisältää listatattuja diskreettejä elementtejä joita pidetään yhdistävinä elementteinä; jolloin lauseke lambda(*e*₁...*e*_{*n*}) on sääntö, joka osoittaa että elementti *e*_s korvataan elementeillä *e*₁...*e*_{*n*}; jolloin lauseketta apply(list) käytetään numeerista aritmetiikkaa varten, jossa on funktiokäsky (+ - * / , jne) ensimmäisenä elementtinään, jolloin muut elementit ovat argumentteja; jos *e*_r on sääntö lambda(*e*_s *e*₁...*e*_{*n*}) ja lauseke apply(*e*_r *e*_{arg1}...*e*_{arg2}) on sovellutus, joka redusointioperaatiossa korvataan *e*_s:llä, jos jokainen *e*_{*i*} yhdistyy vastaavaan suureeseen *e*_{argi}, tai muuten se korvataan suureella nothing; jolloin lauseke hide(list) merkitsee käyttäytymistä, jossa rakenteen sisä- ja ulkopuolella käytetyt täydet muuttujat ovat toisistaan erotettuja, ts. toistensa näkymättömissä; jolloin lauseke set(list) tarkoittaa, että listan sisältö siirretään ohjelmamuotoon; jolloin lauseke symb(*s*) sisältää elementin *s*, joka kanoniseksi lausekkeeksi redusoituna on tunnistin, jolloin elementillä *s* on mahdollisuus olla talletettuna oliomuistin eri muistisoluihin kuuluviin muistikenttiin, jolloin muistisolut on määritelty järjestettäväksi ulottuvuuksiksi, jolloin kaikki saman ulottuvuuden puitteissa olevat tunnistimet *s* merkitsevät samaa käyttäytymistä; jolloin lauseke discr(*n*), jossa *n* on kokonaisluku, tarkoittaa avaruuslaatua, jolla on lukumäärä *n* atomaarista käyttäytymistä; jolloin lauseke pulses(*n*), jossa *n* on kokonaisluku, tarkoittaa *n* atomaarisen käyttäytymisen kesto-aikaa.

13. Patenttivaatimuksen 11 ja 12 mukainen tietojenkäsittelylaitte, **tunnettu** siitä, että uudellen kirjoittamisen sääntöjä on laajennettu seuraavilla säännöillä:

type(*e*₁...nothing...*e*_{*n*})

-> nothing, jossa type edustaa lauseketta, joka on edellä olevassa listassa lähtien

lausekkeesta par(list) lausekkeeseen
'apply(list) saakka, ja $e_1 \dots e_n$ ovat l
istaelementtejä.

type($e_1 \dots \text{alt}(c_1 \dots c_n) \dots e_n$)

5 -> alt(type($e_1 \dots c_1 \dots e_n$)...type($e_1 \dots c_k \dots e_n$))
jossa type edustaa lauseketta, joka on
edellä olevassa listassa lähtien
lausekkeesta par(list) lausekkeeseen
unify(list) saakka (lukuunottamatta
10 alt(list)), ja e_1, e_n, c_1, c_k ovat
listaelementtejä.

set()

-> nothing

set(a)

-> 'alt(a), jos a on lauseke, joka on
edellä olevassa listassa lähtien

15 lausekkeesta par(list) lausekkeeseen
'apply(list) saakka.

set(alt($e_1 \dots e_n$)) -> 'alt($e_1 \dots e_n$), jossa $e_1 \dots e_n$ ovat
elementtejä vailla mitään määrättyä
järjestystä, ja $e_a \dots e_m$ ovat lajitellussa
20 järjestyksessä.

set($e_1 \dots e_n$) -> set(con($e_1 \dots e_n$))

lambda(alt($e_1 \dots e_n$)) -> set(alt($e_1 \dots e_n$))

lambda($e_1 \dots e_n$) -> par($e_1 \dots e_n$)

hide(alt($e_1 \dots e_n$)) -> set(alt($e_1 \dots e_n$))

25 hide($e_1 \dots e_n$) -> par($e_1 \dots e_n$)

apply() -> nothing

apply(e) -> eval_{meta}(e)

apply('alt($b_1 \dots b_n$) $a_2 \dots a_m$)

-> apply(eval_{meta}'alt($b_1 \dots b_n$) $a_2 \dots a_m$)

30 apply(par($b_1 \dots b_n$) $a_2 \dots a_m$)

-> nothing, jos kaikki suureet b_i ja a_i
ovat lausekkeita, jotka ovat edellä olevassa
listassa lähtien lausekkeesta par(list)
lausekkeeseen 'apply(list) saakka, ja
35 $n \neq m$ tai mikä tahansa $a_i \neq b_i$, kun $i > 1$.

apply(par($b_1 \dots b_n$) $a_2 \dots a_m$)

-> b_1 , jos kaikki suureet b_i ja a_i ovat

lausekkeita, jotka ovat edellä olevassa listassa lähtien lausekkeesta par(list) lausekkeeseen 'apply(list) saakka, j
a $n=m$ tai jokainen $a_i=b_i$, kun $i>1$.

5 apply(seq($b_1 \dots b_n$) $a_2 \dots a_m$)
 -> nothing, jos kaikki suureet b_i ja a_i
 ovat lausekkeita, jotka ovat edellä
 olevassa listassa lähtien lausekkeesta
 par(list) lausekkeeseen 'apply(list)
10 saakka, ja $n \neq m$ tai mikä tahansa $a_i \neq b_i$,
 kun $i < 1$.

apply(seq($b_1 \dots b_n$) $a_2 \dots a_m$)
 -> b_1 , jos kaikki suureet b_i ja a_i ovat
 lausekkeita, jotka ovat edellä olevassa
15 listassa lähtien lausekkeesta par(list)
 lausekkeeseen 'apply(list) saakka,
 jos $n=m$ tai jokainen $a_i=b_i$, kun $i>1$.

Funktio eval_{meta} muuntaa ohjelmamuodossa olevat lausekkeet (jotka
ts. on varustettu merkillä ') arvomuotoon (ts. ilman ').

20 Merkki ' poistetaan vain ylimmästä struktuurista.
 eval_{meta}'type(a) -> type(a), jossa type edustaa lauseketta,
 joka on edellä olevassa listassa lähtien
 lausekkeesta par(list) lausekkeeseen
 unify(list) saakka (lukuunottamatta
25 alt(list)).

con() -> nothing
 con($e_1 \dots \text{nothing} \dots e_n$) -> nothing
 con($e_1 \dots e_n$) -> e_n
 pri() -> nothing
30 pri($\text{nothing}_1 \dots \text{nothing}_n$)
 -> nothing
 pri($\text{nothing}_1 \dots \text{nothing}_{k-1} e_k \dots e_n$)
 -> e_k
 par($\text{delta}_1 \dots \text{delta}_n$) -> discr(n)
35 seq($\text{deltat}_1 \dots \text{deltat}_n$) -> pulses(n)

14. Jonkin edellisen patenttivaatimuksen 1 - 13 mukainen

tietojenkäsittelylaite, tunnettu siitä, että jokainen muistisolu on sovitettu talletettavaan ainakin joitakin seuraavassa luettelossa esitetyjä ominaisuuksia:

merkkejä, jotka osoittavat onko solussa oleva lauseke redusoitava;

merkkejä, jotka lausekkeen ollessa puun elin, osoittavat lausekkeen ominaisuuksia;

merkkejä, jotka osoittavat miten lauseke on luotu;

merkkejä, jotka osoittavat käsittääkö lauseke luvun tai toistetun käyttäytymisen;

merkkejä, jotka osoittavat onko lauseke vain osana listaa, jolla on muita listan elimiä talletettuna toiseen muistisoluun.

15. Jonkin edellisen patenttivaatimuksen 1 - 14 mukainen tietojenkäsittelylaite, tunnettu siitä, että se oliomuistin muistisoluvälineessä sisältää välineet tietokoneohjelman tallettamisen mahdollistamiseksi abstraktin syntaksin eksplisiittisen tai implisiittisen koodauksen muodossa, ja tämän syntaksin kautta redusointisääntöjen semantiikan.

16. Patenttivaatimuksen 15 mukainen tietojenkäsittelylaite, tunnettu siitä, että semantiikka sisältää joukon uudelleen kirjoittamisen sääntöjä, jolloin jokainen sellainen sääntö sisältää lausekkeen ja ryhmän arvoja tarkoittaen että ryhmän jokaista arvoa pidetään ekvivalenttisena ryhmän toisen arvon kanssa ja sen kanssa vaihdettavissa.

17. Jonkin edellisen patenttivaatimuksen 1 - 17 mukainen tietojenkäsittelylaite, tunnettu välineistä jatkuvien virhetarkistusten suorittamista varten sekä varmistustoiminnasta seuraavien uudelleen kirjoittamisen sääntöjen mukaisesti:

backup() -> fault

backup(fault₁...fault_n) -> fault

backup(fault₁...fault_{k-1} ek...en) -> ek

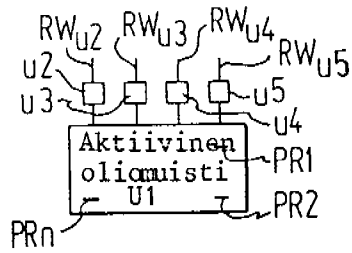


FIG.1

Signaalin
voimakkuus

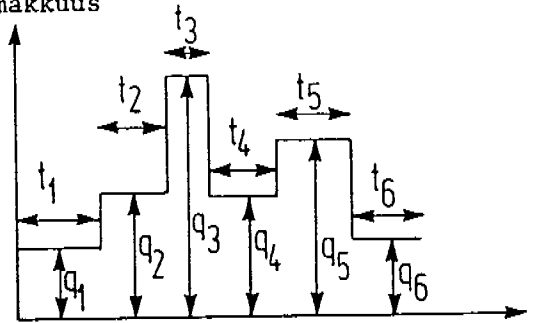


FIG.2

Aika

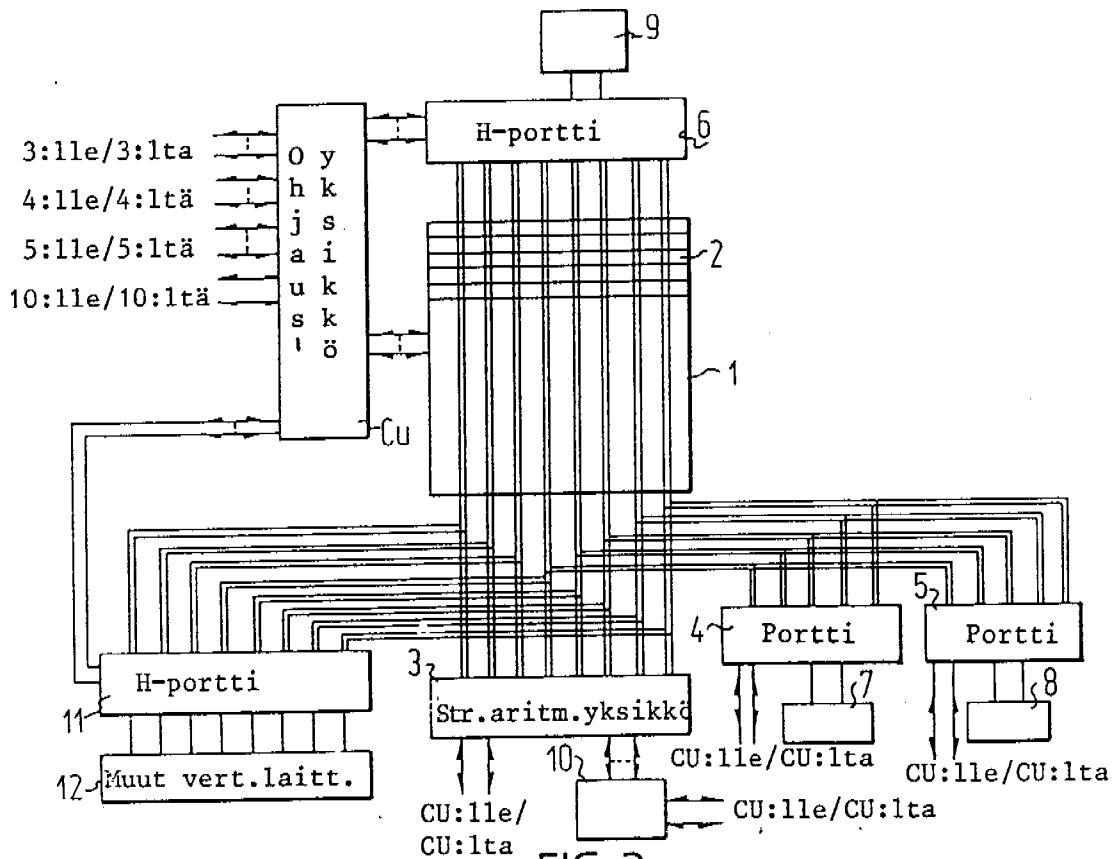


FIG.3

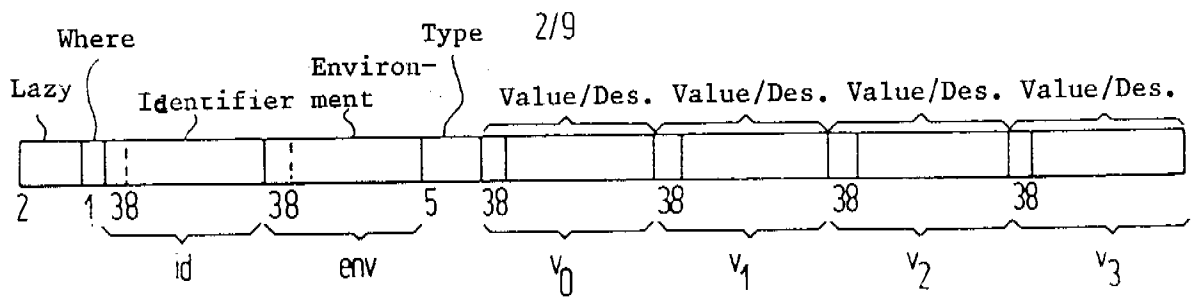


FIG.4

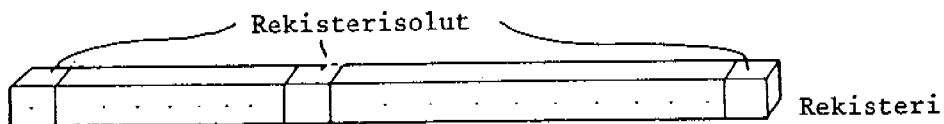
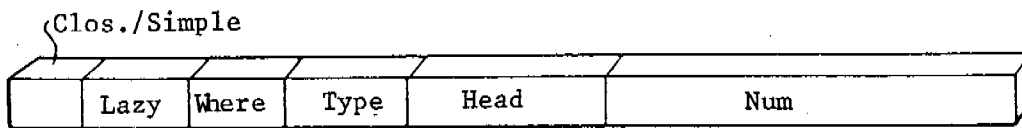
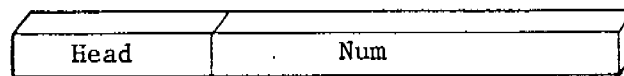


FIG.8A



Täysrekisteri

FIG. 8B



Rajoit.rekisteri

FIG. 8C

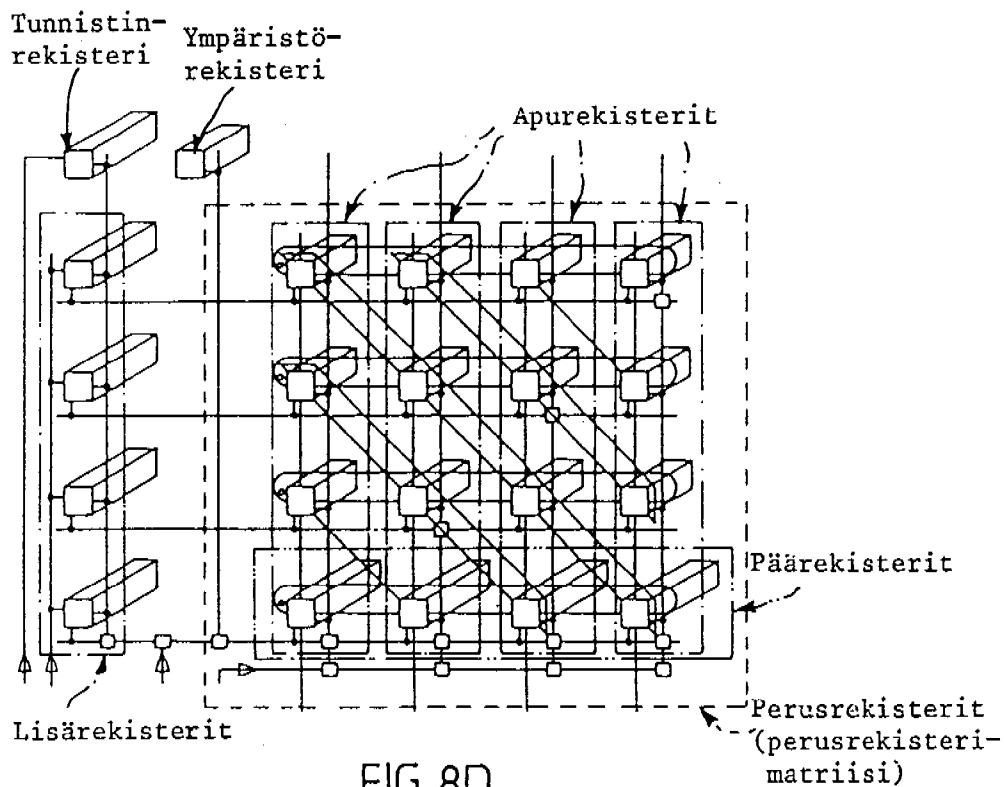


FIG.8D

id_2 id_3
 $\text{id}_1 = \text{list}(\text{par}(1\ 2\ 3)\text{par}(4\ 5\ 6))$

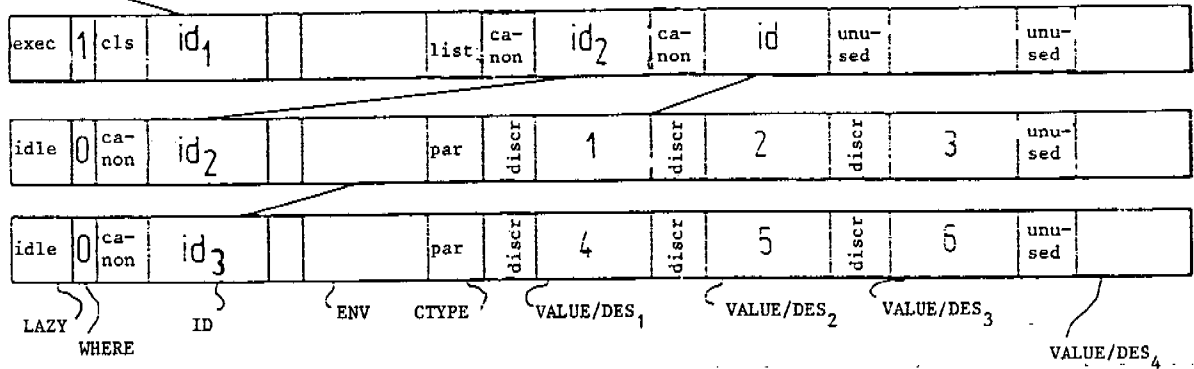


FIG. 5

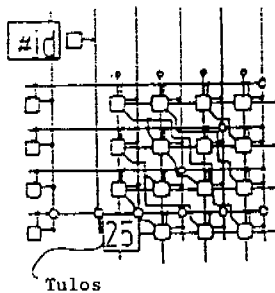
YKSINKERTAINEN ARVO
TAI KONETUNNISTIN

FIG. 9A

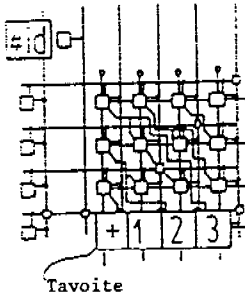
YKSITASOINEN
RAKENNE

FIG. 9B

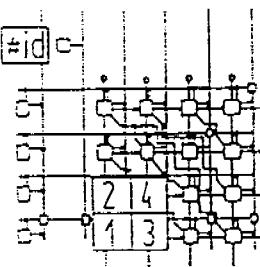
KAKSITASOINEN
RAKENNE

FIG. 9C

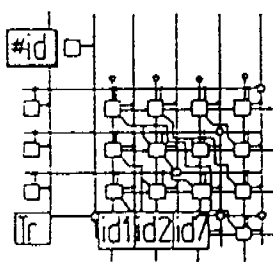
KAKSITASOINEN
RAKENNE

FIG. 9D

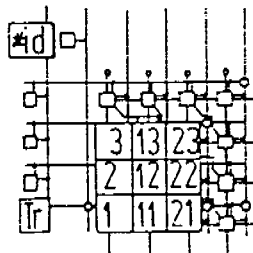
KOLMITASOINEN
RAKENNE

FIG. 9E

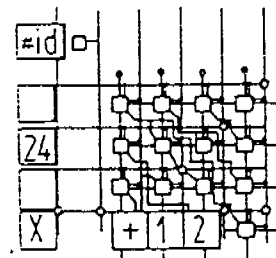
LIUKUHIHNA-
TOIMINTATILA

FIG. 9F

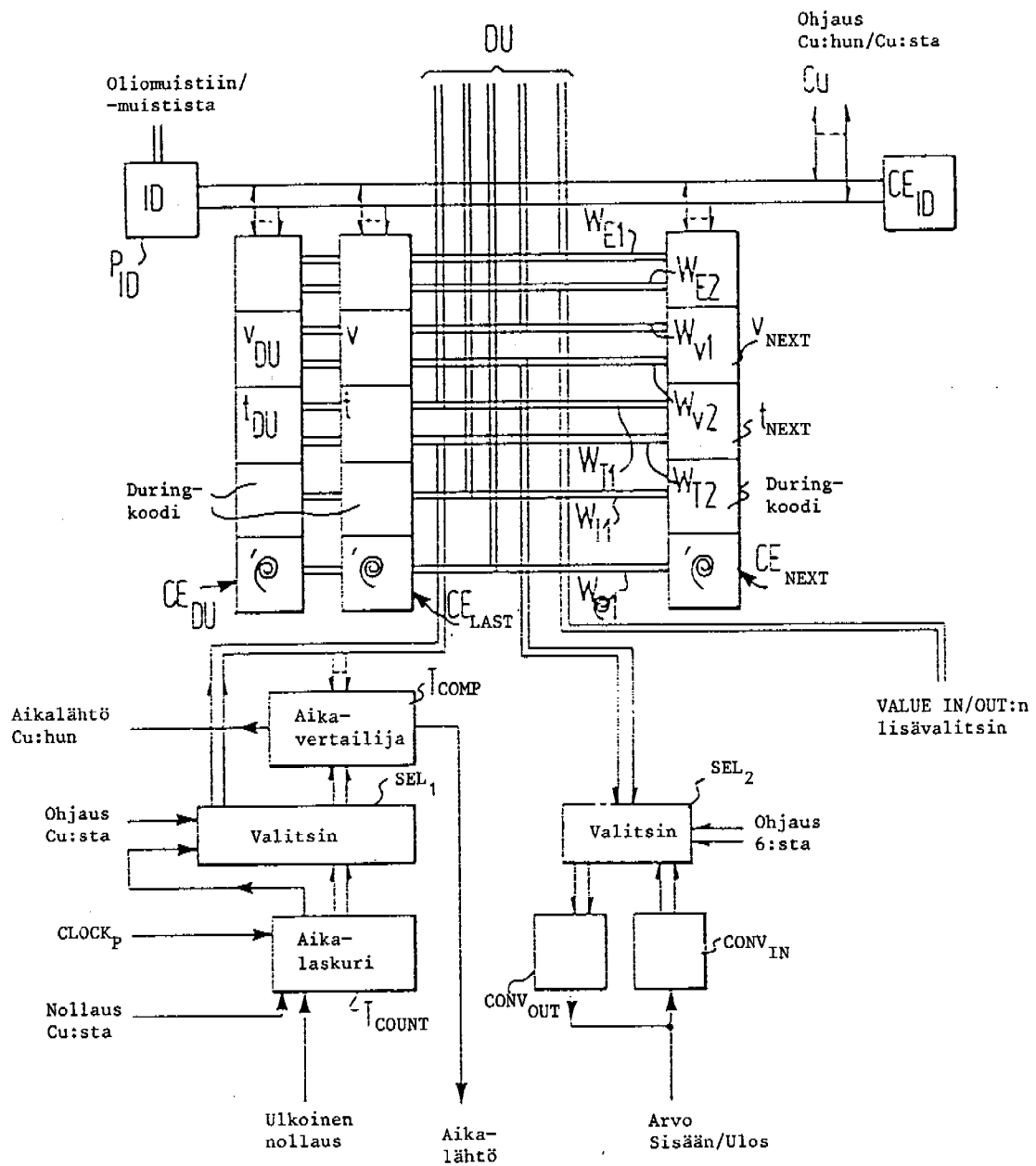


FIG. 6

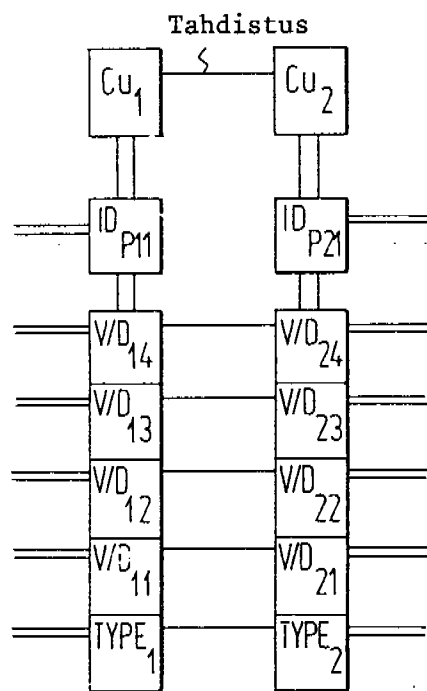


FIG.7

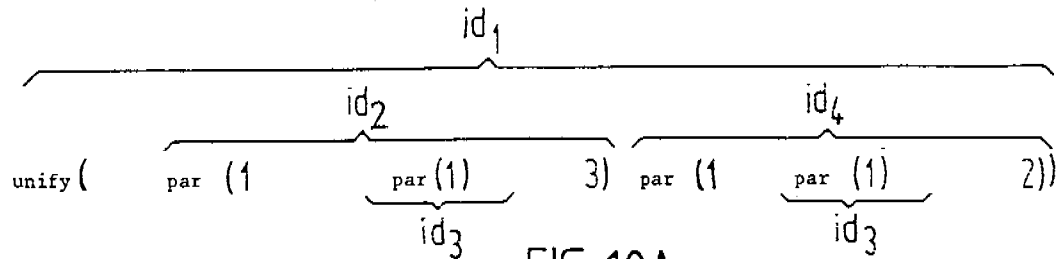


FIG. 10A

WHERE

exec	0	cls	id ₁			uni- fy	ca- non	id ₂	ca- non	id ₄	unu- sed		unu- sed	
idle	0	ca- non	id ₂			par	discr	1	ca- non	id ₃	discr	3	unu- sed	
idle	0	ca- non	id ₃			par	discr	1	unu- sed		unu- sed		unu- sed	
idle	0	ca- non	id ₄			par	discr	1	ca- non	id ₃	discr	2	unu- sed	

FIG. 10B

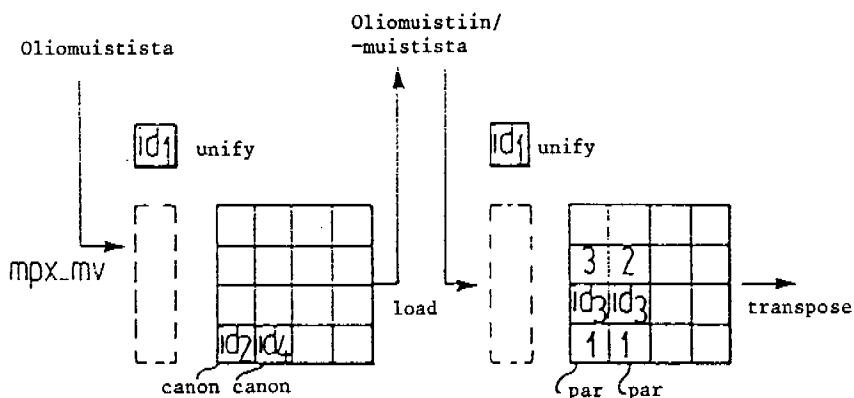


FIG. 10C

FIG. 10D

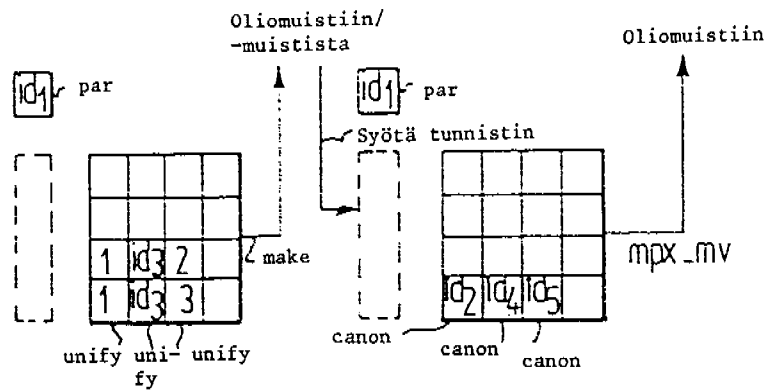


FIG. 10E

FIG. 10F

wait	-	cls	id ₁		par	cls	id ₂	cls	id ₄	cls	id ₅	un-	
exec	-	cls	id ₂		uni-	discr	1	discr	1	un-		un-	
-	-	ca-	id ₃		par	discr	1	un-		un-		un-	
exec	-	cls	id ₄		uni-	ca-	id ₃	ca-	id ₃	un-		un-	
exec	-	cls	id ₅		uni-	discr	2	discr	3	un-		un-	

FIG. 10G

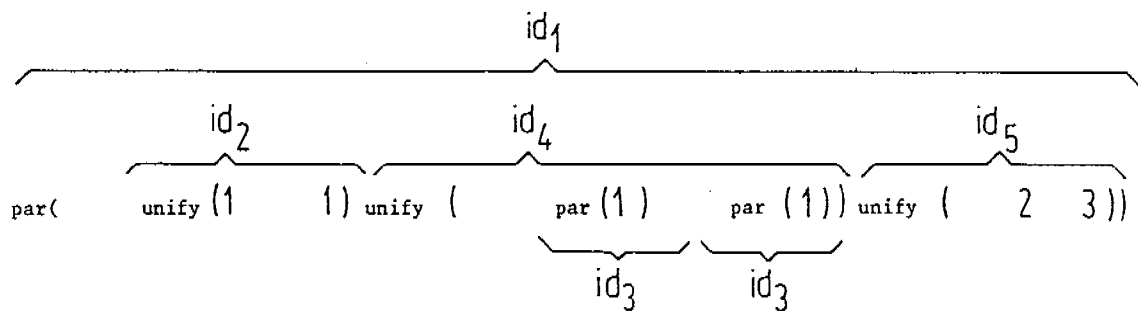


FIG. 10H

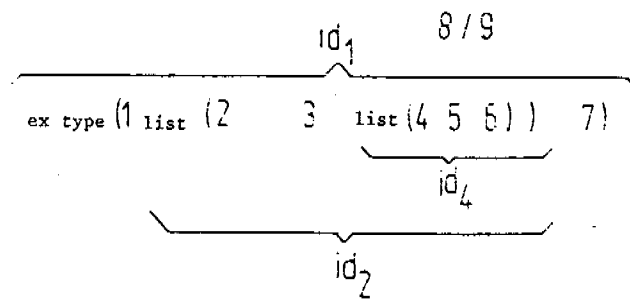


FIG. 11A

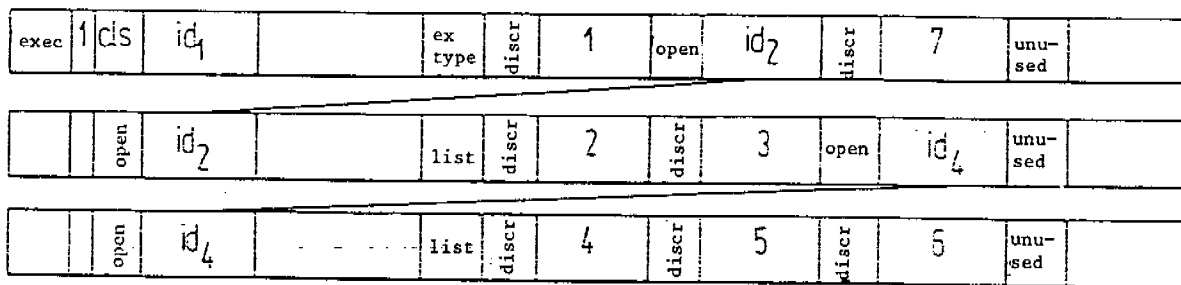


FIG. 11B

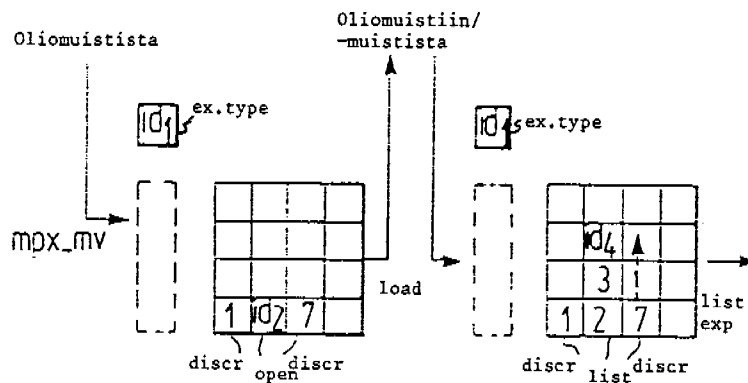


FIG. 11C

FIG. 11D

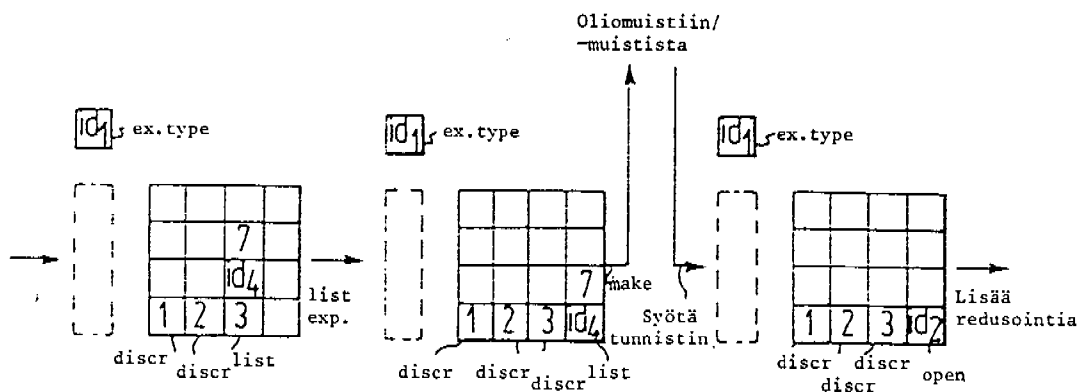


FIG. 11E

FIG. 11F

FIG. 11G

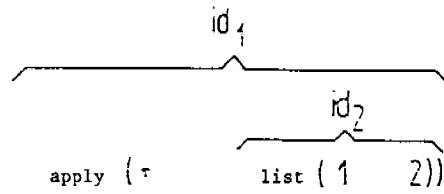


FIG. 12A

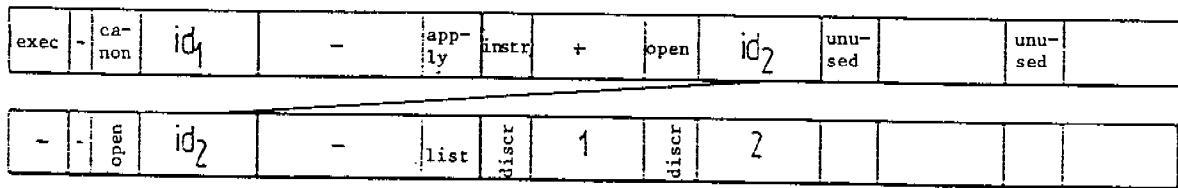


FIG. 12B

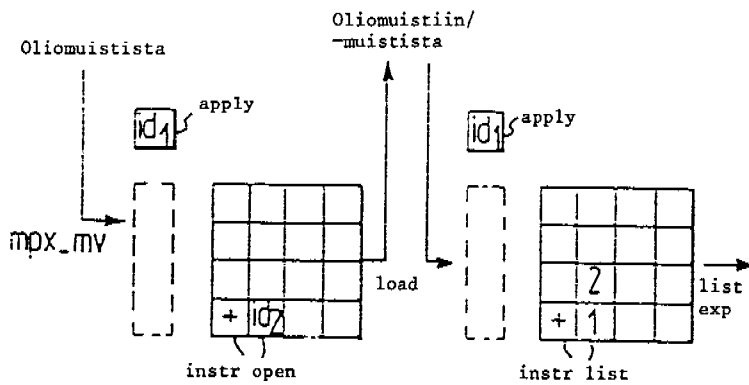


FIG. 12C

FIG. 12D

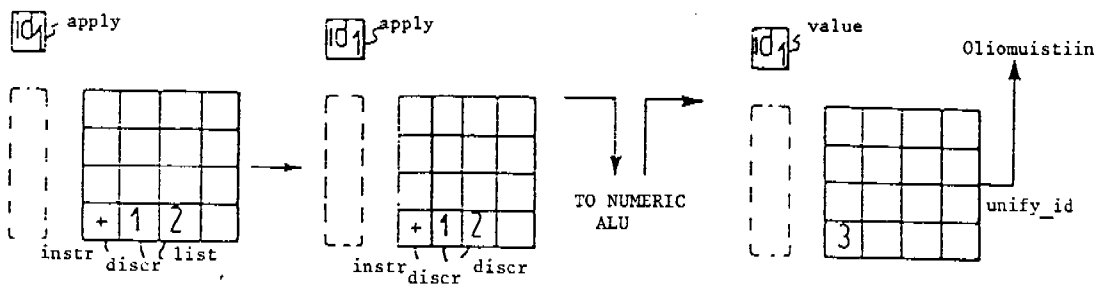


FIG. 12E

FIG. 12F

FIG. 12G