Kehittää
ohjelmointitehtävien
ratkaisemisessa
tarvittavia
metakognitioita!

ESTELLE **ja** LOTOS

- ISO:n ja CCITT/ITU-T:n standardoimia formaaleja määrittelymenetelmiä
- ESTELLE (Extended Finite State Machine Language, ISO 1989)
- LOTOS (Language of Temporal Ordering Specification, ISO 1981)
- SDL (Specification and Description Language, CCITT alk. 1976)
- Alunperin OSI-standardien kuvaamiseen
  → käyttöalue laajentunut:
  - monimutkaisten
  - hajautettujen
  - rinnakkaisten
  - laatukriittisten
  - turvallisuuskriittisten
  - standardoitujen

  systeemien rakenteen ja toiminnan kuvaaminen.
- Kuvauskielen standardointi antaa “turvallisuutta”!
- Pyritään luomaan yleiskuva menetelmistä!
- HUOM: ollaan määrittelemässä ei implementoimassa! (vaikka käytettyjen “rakennuspalikoiden” esim. oliopohjainen toteutus onkin ilmeinen, vrt. abstraktit tietorakenteet)

ESTELLE

- Formaali kuvauskieli hajautetuille ja samanaikaisille systeemeille
  → OSI-palvelut ja -protokollat!
- ESTELLEn struktuuri:

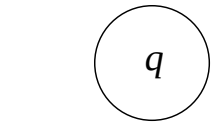
  Vaihtuvamuotoisten (non-deterministic) tila-automaattien rakenteen ja keskinäisen kommunikoinnin kuvaaminen

- Systeemin rakenne:

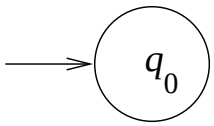
  hierarkkinen joukko tila-automaatteja, jotka kommunikointiporttiensa kautta vaihtavat viestejä kaksisuuntaisia viestinvälityskanavia pitkin.

- Viestit joko lähetys- tai vastaanottojonossa!
- ESTELLEn formaatti muistuttaa PASCALia
- Vrt. automaatit+viestit ↔ rinnakkaislaskenta
- Vrt. automaatit+keskeytyspisteet ↔ komponenttipohjainen ohjelmakehitys
- Vrt. automaatit+kanavat ↔ automaattiolio+sanomanvälitysolio
- Synkronoidut tai synkronoimattomat rinnakkaisjärjestelmät
- Systeemin rakenne voi muuttua dynaamisesti
- Abstraktissa määritysprosessissa pyritään monitasoisuuteen
- ESTELLEn peruskomponentteja moduulit, kanavat ja rakenne
- Systeemikuvaus voidaan antaa tilasiirtymäkaaviona

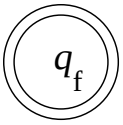
Tilasiirtymäkaavion merkintöjä:



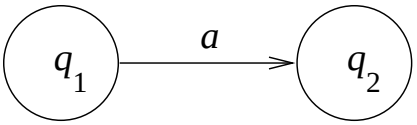
Automaatin tila nimeltä q



Alkutila



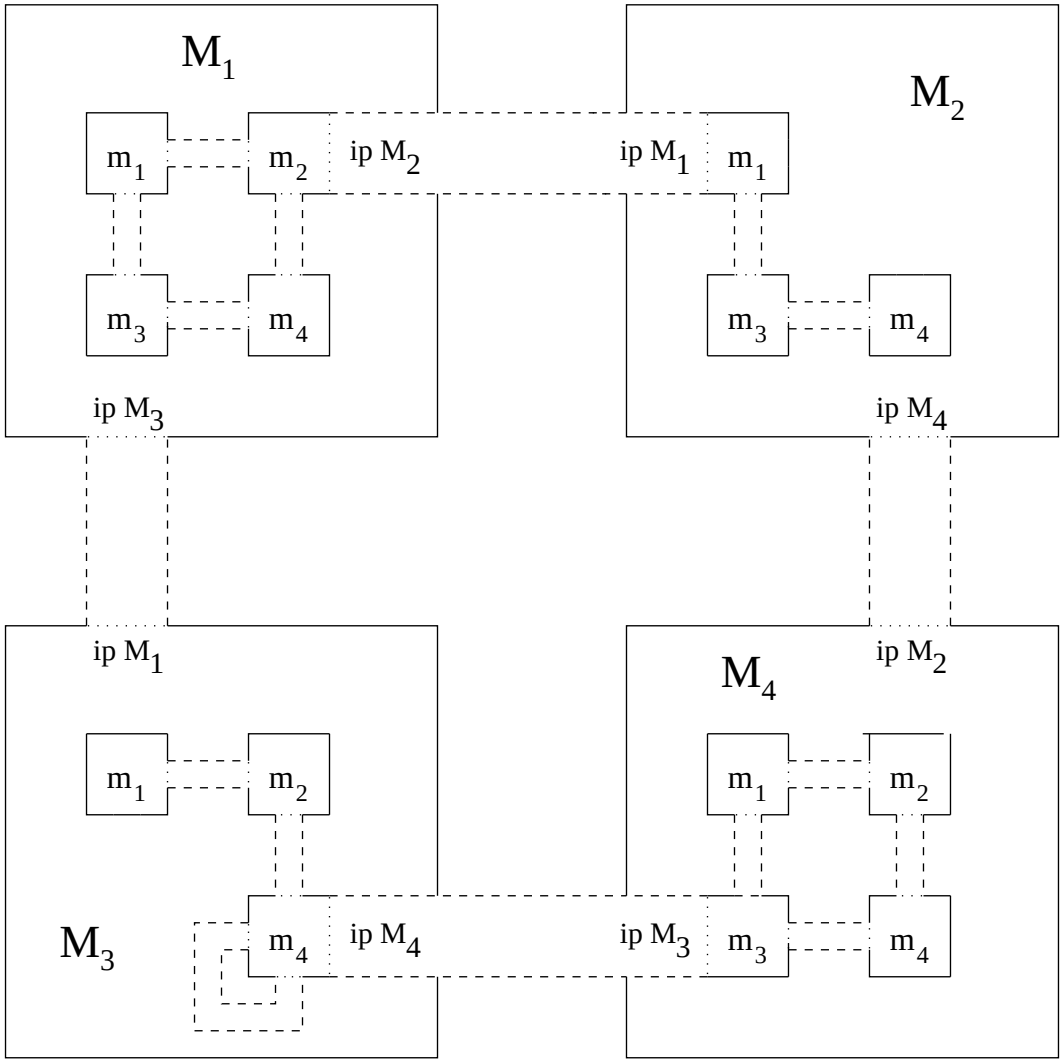
Lopputila: automaatti "hyväksyy" syötejonon, joss se on jonon loppuessa tässä tilassa



Syötemerkin a aikaansaama siirtymä tilasta q_1 tilaan q_2

HUOM: Annettu tila-automaatti voidaan korvata (toiminnaltaan) ekvivalentilla, jossa on minimäärä erilaisia tiloja!

ESTELLEn systeemin rakenne:



Moduulit:

- Äärellinen tila-automaatti
- Siirto tilasta toiseen keskeytyspisteessä (ip = interaction point)
  saadun hyväksyttävän syötteen (viestin) perusteella
  → tulosteena viestejä muille moduuleille
- Tyypillinen protokollamääritys: automaatti,
  joka huolehtii puhelun reitityksestä tilaajan ja keskuksen välillä
- Tila: yhteystilanne *closed, opening, open, closing, ...*
- Tilan muutos: *ConnectRequest*
  → yhteystilanteen muutos *closed*↔*opening*
- Syötteet:
  - käyttäjän (asiakkaan) pyynnöt *Connect, Send, ...*
    → tulosteena vaikkapa PDU (Protocol Data Unit)
  - yhteyden tarjoaja *DataIndication, Reset, ...*
    → tulosteena vaikkapa *ReceiveResponse, ErrorIndication, ...*
- Moduulin toiminta $\simeq$ siirrot tilasta toiseen
  → tilansiirron laukeaminen (firing): syötteen käsittely, tilan muuttaminen ja tulosteen muodostaminen
- Moduulin vaihtuvamuotoisuus (non-determinism)
  = nykytila ja saatu syöte määrittävät joukon sallittuja toimintoja

Kanavat:

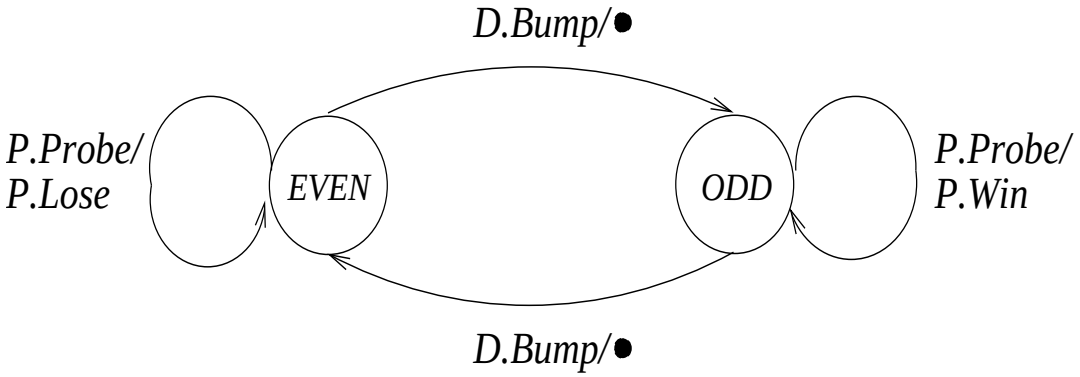
- Moduulit kommunikoivat kanavien kautta
- Kanava sitoo yhteen lähettävän moduulin ulostulon ja vastaanottavan moduuliin viestijonon
  $\rightarrow$ moduulit nimetty yksikäsitteisesti
- Huom: moduulin tilansiirto ilman ulkoista “ärsykettä“ voidaan toteuttaa viestin lähetyksenä moduulille itselleen
- Kanavat liittyvät kahden moduulin välille keskeytyspisteissä, kumpikin moduuleista voi alustaa yhteyden
- Vastaanotetuille keskeytyksille erikseen omat jononsa tai kaikki keskeytykset samaan vastaanotettujen viestien jonoon

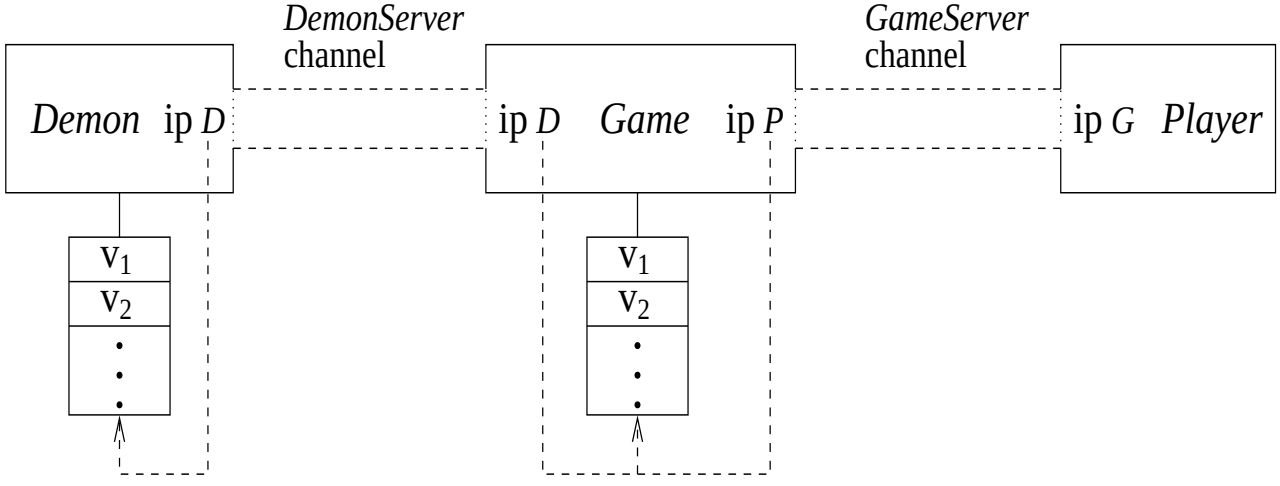
Rakenne:

- Moduuli “black box” $\simeq$ ulkopuolelle näkyvät vain lähetetyt viestit (kapselointi)
- Perustilassa moduulit toimivat rinnakkain, voidaan myös synkronoida
- Moduuli rakentuu alimoduulista (*parent-child*), joita esitellään (create), resetoidaan (release) ja tuhotaan (terminate) dynaamisesti suorituksen aikana.
- Kommunikointi keskeytyspisteissä, jotka voivat alimoduuleissa kytkeytyä (connect) joko vanhempimoduulin keskeytyksiin tai muiden moduulien keskeytyspisteisiin.
- Vanhempimoduuli voi liittää (attach) toiseen moduuliin viittavan keskeytyspisteen suoraan lapsensa viestijonoon.

ESIMERKKIPELI:

Peliä (*Game*) ylläpitää pelaajalle näkymätön ’demoni’ (*Demon*; suom. hiisi, paholainen, peikko, riivaaja), joka satunnaisin väliajoin innostuu ’lyömään’ (*Bump*) pelin tilasta toiseen. Pelin tila muodostuu ’lyöntien’ lukumäärän perusteella. Jos ’demoni’ on ’lyönyt’ parittoman (*ODD*) määrän kertoja, voittaa (*Win*) pelaaja (*Player*) peliä pelatessaan (*Probe*). Vastaavasti, kun ’lyöntejä’ on kertynyt parillinen (*EVEN*) määrä, pelaaja häviää (*Lose*) pelatessaan.





Kanavien kuvaukset:

- Kuvataan kanavien *DeamonServer* ja *GameServer* toiminta.
- Kuvauksessa määrätään

  i) minkä moduulien välillä kanava sijaitsee sekä

  ii) minkätyyppisiä viestejä moduulit kanavaa pitkin lähettävät.

```
channel DeamonServer (User, Provider);
   by Provider:
      Bump;

channel GameServer (Player, Machine);
   by Player:
      Probe;
      Result;
      Newgame;
      Endgame;
   by Machine:
      Win;
      Lose;
      Score (nwon: integer);
```

Moduulien kuvaukset:

- Kuvataan moduulien sisäinen rakenne ja tilansiirrot.
- Kuvauksessa määrätään

  i) moduulin rooli siihen keskeytyspisteissä (ip) liittyvissä kanavissa,

  ii) viestijonojen tyypitykset,

  iii) moduulin tilan ja toimintojen määrittäminen tilansiirron lauetessa.
- Kohdat *i*) ja *ii*): moduulien *Game* ja *Deamon* sisäinen rakenne:

```
module Game activity;
   ip
      P: GameServer (Machine) common queue;
      D: DeamonServer (User) common queue;
end; { Game }

module Deamon systemprocess;
   ip
      D: DeamonServer (Provider) individual queue;
end; { Deamon }
```

Moduulien kuvaukset II:

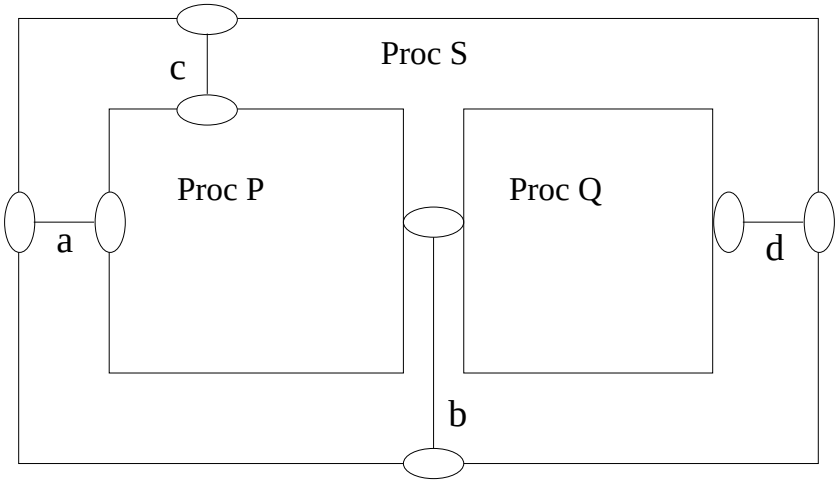
- Kohta *iii*): moduulin *Game* “runko” (**body**):

```
body GameBody for Game;
   state EVEN, ODD;

   initialize
      to EVEN;
         begin
         end;

   trans
      { *** Pelaaja pelaa *** }
      when P.Probe
         from EVEN to EVEN
            begin
               output P.Lose
            end;
      when P.Probe
         from ODD to ODD
            begin
               output P.Win
            end;
      { *** Demoni jysäyttää *** }
      when D.Bump
         from EVEN to ODD
            begin
            end;
      when D.Bump
         from ODD to EVEN
            begin
            end;
end; { GameBody }
```

LOTOS

- Teoreettiset perusteet:
  - CCS (Calculus of Communicating Systems, Milner)
  - CSP (Communicating Sequential Processes, Hoare)
  - ACT ONE (Ehrig ja Mahr)
- Systeemin rakenne: peräkkäisprosessit (sequential processes) ja/tai samaan aikaan toimivat yhteisprosessit (concurrent processes)
- Prosessien välinen kommunikointi synkronoitua tai synkronoimatonta
- Spesifikaation peruselementtejä ovat
  - prosessit (*process*) ("kuuntelevat")
  - tapahtumat (*event*) ("herättävät")
  - käyttäytymislausekkeet (*behaviour expression*) ("toimittavat")
- Prosessit kommunikoivat toistensa kanssa porttien (*gate*) kautta
- LOTOS-pohjainen systeemikuvaus määrittää ohjelmalle asetetun toimintamallin eli joukon sallittuja käyttäytymisiä
  → soveltuu hyvin esim. avoimien standardien kuvaamiseen

PerusLOTOS




```
process S [a, b, c, d] : noexit :=

   P [a, b, c] |[b]| Q [b, d]          (* behaviour expression *)

   where

   process P [t, u, v] : noexit :=
      t; (u; stop [] v; stop)          (* behaviour expression *)
   endproc (* P *)

   process Q [x, y] : noexit :=
      x; y; stop                       (* behaviour expression *)
   endproc (* Q *)

endproc (* S *)
```

TäysiLOTOS **= Perus**LOTOS ∪ **ADT:**

- TäydenLOTOSken kaikki tietotyypit määritellään abstraktisti!
- Tyyppikuvaus: muuttujan perustyyppi ja tyypin operaatiot:

```
type Boolean is
   sorts Bool
   opns
      true, false :                  –> Bool
      not :              Bool        –> Bool
   eqns
      ofsort Bool
           not (true) = false;
           not (false) = true;
endtype (* Boolean *)

type NaturalNumber is
   sorts Nat
   opns
      0 :                            –> Nat
      Succ :             Nat         –> Nat
endtype (* NaturalNumber *)

type NaturalExtended is NaturalNumber
   opns
      _ + _ :            Nat, Nat    –> Nat
   eqns
      forall x, y: Nat
         ofsort Nat
            x + 0 = x;
            x + Succ (y) = Succ (x + y);
endtype (* NaturalExtended *)
```

- Olemassaolevia tyyppimäärityksiä käyttöön standardikirjastosta:

```
library
   Boolean, Set, NaturalNumber
endlib
```

TäysiLOTOS **= Perus**LOTOS ∪ **ADT (cont.):**

- ADT:t ovat niitä tapahtumia l. parametreja, joita prosessien portteihin liitetään kommunikointia varten
- esimerkki:

```
process Telephone [phone] : noexit :=
   phone ? dialled : TelephoneNumber ? money : Currency
      [InternationalCall (dialled) implies (money gt Cents50)];
   (Conversation [phone] (money) >> Telephone [phone])
endproc (* Telephone *)
```

- interaktion muoto määritellään prosessien välisten ns. *kokeilutarjousten* (*experiment offer*) avulla:

käyt. laus. A	käyt. laus. B	Ehto	Interaktion tyyppi
$g!E_1$	$g!E_2$	$value(E_1) = value(E_2)$	arvon yhteensopivuus
$g!E_1$	$g?x : t$	$sort(E_1) = t$	arvon välitys
$g?x : t$	$g?y : u$	$t = u$	arvon generointi

```
process MaxCalc [input, max] : exit :=
   input ? x : Nat ? y : Nat;
   (
      [x ge y] ->
         max ! x; exit
   []
      [y ge x] ->
         max ! y; exit
   )
endproc (* MaxCalc *)
```