University of Oslo
Department of Informatics

Modellering og simulering av jernbanesystemer

En lagdelt tilnærming

Pål Enger

Hovedoppgave

21. mai 2007

Forord

Denne oppgaven er en del av en canditatus scientarium-grad. Arbeidet er utført ved forskningsgruppen Presis modellerering og analyse(PMA) ved Institutt for informatikk, Universitetet i Oslo.

Først vil jeg takke mine veiledere. Tusen takk til Anders Moen Hagalisletto, som har vært til stor inspirasjon gjennom litt for lang tid med denne oppgaven. Også stor takk til Ingrid Chieh Yu og Joakim Bjørk som tok over som veiledere da Anders tilslutt måtte prioritere sin egen doktorgradsoppgave. En særskilt takk til Joakim for faglige diskusjoner gjennom hele prosessen. Uten disse tre ville det ikke vært noen oppgave.

Takk til Njål Svingheim ved Jernbaneverket for norske jernbaneuttrykk og norske navn på jernbanekomponenter, og til Willy Ulseth ved Oslo Sporveier, Banedivisjonen, for innsikt i Oslo Sporveier.

Tusen takk til Studentorchesteret Biørneblæs som har gjort studietiden til en hyggelig, om enn noe lenger periode. En spesiell takk til Heidi, Marit og de andre Biørneblæserene som har presset meg til å bli ferdig.

Tilslutt vil jeg takke Siri Lajord for all støtte, hjelp, innspill og ikke minst tålmodighet.

Pål Enger 21. mai 2007

Innhold

Figurer

Kapittel 1

Introduksjon

Denne oppgaven er en del av et prosjekt ved PMA der man har undersøkt petrinet, og hvorvidt det er et praktisk verktøy til å modellere større sanntidssystemer. I den forbindelse er det publisert en serie artikler og oppgaver. (22) (5) (9) (8) (10)

Som eksempel på et slikt system har man valgt å bruke jernbanenett. Jernbanen er en spennende utfordring å modellere, ettersom det er et høyst reelt felt som angår mange. Jernbanen består av en fast struktur og mange tog som beveger seg samtidig, både avhengig og uavhengig av hverandre. En modell av et jernbanenett må kunne simulere mange parallelle hendelser.

Petrinet er et formelt modelleringsspråk foreslått av Carl Adam Petri i doktoravhandlingen hans fra 1962(19). Petrinet er en generalisering av automatteori der både tilstander og hendelser representeres eksplisitt, og hendelser kan forekomme samtidig, slik at man kan modellere parallellitet.

Problemstillingen

Arbeidet med denne oppgaven har hatt to hovedmål.

1. Bygge videre på arbeidet som er gjort tidligere, ved å vise hvordan man kan modellere spesifikke egenskaper. Det er brukt tre eksempler, og disse er:

   (a) informasjonsbærende komponenter, komponenter som kan inneholde informasjon. Det er ikke så viktig hva denne informasjonen er, så i dette tilfellet er lengden jernbane som komponenten representerer brukt.

(b) kollisjonsvare komponenter, komponenter som kan brukes til å undersøke mulighet for ulykker i form av kollisjoner mellom tog.

(c) signalførende komponenter, komponenter som sender signaler langs linjene, og som oppdateres automatisk ved hendelser i nettet.

2. Systematisere måten man modellerer nye egenskaper for å gjøre det enklere.

I tillegg er det i forbindelse med dette prosjektet er det utviklet et verktøy som kan simulere nettene som er presentert. En del av dette verktøyet presenteres i denne oppgaven. De andre delene er presentert i (22) og (5). Resultater av dette verktøyet er presentert i (10).

Oversikt

Oppgaven starter med å presentere petrinet med definisjoner og nyttige teknikker i Kapittel 2.

Kapittel 3 beskriver jernbanekomponenter og hvordan disse kan uttrykkes ved hjelp av petrinet. Her vil det også beskrives noen eksempler på egenskaper man kan modellere, og hvordan dette kan gjøres.

I kapittel 4 vil man se etter system i komponentene for å finne måter å redusere mengden med petrinet som må lages på nytt for hver ny egenskap som presenteres.

Kapittel 5 presenterer kort verktøyet som er laget.

Tilslutt, i kapittel 6, kommer konklusjon og åpne problemstillinger for fremtidig arbeid.

Språk

For å være konsistent med tidligere arbeid og med implementasjonene som er gjort, vil det forekomme engelske termer og forkortelser. Det er forsøkt så langt det lar seg gjøre å oversette uttrykk til norsk, eller bruke allerede eksisterende norske uttrykk.

Kapittel 2

Bakgrunn

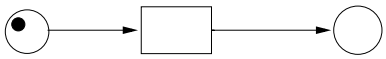
2.1 Petrinet

Petrinet (PN) er en grafisk teori for å modellere dataprosesser. Den ble foreslått av Carl Adam Petri i doktoravhandlingen hans fra 1962(19), for å modellere interaksjon mellom hardware. Nå brukes det hovedsaklig fordi det er et verktøy som både er teoretisk velforstått og utrykkskraftig nok til å modellere større systemer.

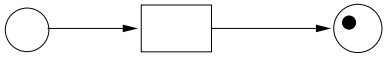
2.1.1 Oppbygning

PN har noen få forskjellige elementer, eller primitiver: steder (places), overganger (transitions), piler (arcs), og brikker (tokens). Pilene kommer i to utgaver. Den ene går fra steder til overganger og den andre fra overganger til steder. Disse kalles henholdsvis innpiler og utpiler. Stedene kan ha brikker. Overgangene representerer hendelser i nettet. Stedene og hvorvidt de har brikker uttrykker tilstanden til nettet.

Nettet endrer tilstand ved at overgangene **fyrer**. Dette foregår ved at alle innpilene knyttet til en overgang fjerner hver sin brikke fra stedet de er knyttet til. Deretter plasseres det for hver utpil en brikke i de respektive utstedene. Overgangen kan ikke **fyre** dersom det ikke finnes en brikke for hver av innpilene. Dersom overgangen kan **fyre** er den **aktiv**. Figurene 2.1 og 2.2 viser et eksempel på fyring.



Figur 2.1: Før fyring



Figur 2.2: Etter fyring

2.2 Uformell beskrivelse

Petrinetprimitivene ser slik ut:

- Steder (places) 
- Overganger (transitions) 
- Piler (arcs)
- Brikker (tokens) ●

2.2.1 Typer og utvidelser av petrinett

Det er foreslått flere typer petrinet. Dette er godt beskrevet i (16)(15) .Disse kan deles inn i tre hovedklasser:

- **Elementære nett**
  Elementære nett har som begrensning at det kun kan være en brikke på hvert sted. Dette medfører at overganger ikke er aktivert dersom det er brikker i minst et av overgangens ut-steder. Stedene vil således representere boolske verdier.
- **Sted-overgangsnett**
  Sted-overgangsnett har ikke den samme begrensningen som elementære nett. Et sted kan ha vilkårlig mange brikker. Stedene kan her uttrykke naturlige tall.
- **Fargede petrinet** (fargede brikker)
  Her har brikkene farger, dvs. de bærer ytterligere informasjon.

Informasjon kan f.eks være en tekststreng som inneholder navnet på personen hvis sak behandles, eller mer komplekse datatyper.

Det finnes mange nyttige utvidelser til dette igjen. Noen viktige er:

- **Pil-uttrykk**
  Her er det uttrykk tilknyttet pilene som kan spesifisere hvilke datatyper og eventuelt hvilke verdier brikkene de virker på skal ha. For eksempel kan det tenkes at det i et sted ligger både en brikke med en tekststreng og en brikke med en integerverdi. En innpil som spesifiserer en integerverdi vil ikke kunne røre brikken med tekststrengen, og en innpil som spesifiserer en boolsk verdi vil ikke kunne ta noen av brikkene, og den tilhørende overgangen vil ikke kunne aktiveres før en slik brikke dukker opp.

- **Voktere**
  Overgangene kan også ha en funksjon tilknyttet seg, som må være oppfylt før den er aktivert. Dette er således en boolsk funksjon over egenskapene til brikkene innpilene berører.

- **Tid**
  Mange systemer behøver et begrep om tid. Dette kan implementeres på flere måter. En av disse er å ha en global klokke som kun er aktiv, dvs går, når ingenting annet er aktivt, samt at man har en forsinkelse på brikker, dvs at en brikke ikke kan røres før klokka minst har nådd en gitt verdi. Overganger, eller strengt tatt utpiler, legger forsinkelser på brikkene.

- **Brytere**
  Med brytere kan man modellere ytterligere krav til hva som skal til før en aktivert overgang kan utføres. Man kan f.eks ha en tidsbegrensning, man kan vente på bruker etc. Dette er nyttig for å modellere ekstern kontroll.

2.3 Formell beskrivelse

Kurt Jensen har definert fargede petrinet slik:(13)
Et farget petrinet er et tuppel $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ slik at

1. Σ er et endelig sett ikke-tomme typer kalt fargesett.

2. P er et endelig sett steder.

3. T er et endelig sett overganger.

4. A er et endelig sett piler slik at $P \cap T = P \cap A = T \cap A = \emptyset$

5. N er en nodefunksjon. $N : A \rightarrow P \times T \cup T \times P$

6. C er en fargefunksjon. $C : P \rightarrow \Sigma$

7. G er en vokterfunksjon. $G : T \rightarrow$ *uttrykk* slik at $\forall t \in T : [Type(G(t)) = Boolean \wedge Type(Var(G(t))) \subseteq \Sigma]$

8. E er en piluttrykkfunksjon. $E : A \rightarrow$ *uttrykk* slik at $\forall a \in A : [Type(E(a)) = C(p(a)_{MS}) \wedge Type(Var(E(a))) \subseteq \Sigma]$ der $p(a)$ er stedet til $N(a)$.

9. I er en initialisatorfunksjon. $I : P \rightarrow$ lukkede uttrykk slik at $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$.

$Type(t)$ betyr typen(e) til t, $Var(E)$ betyr variablene i E og C_{MS} betyr at C er et multisett.

2.3.1 Fargesett

Fargesettene avgjør typene, operasjonene og funksjonene som benyttes i nettet. Det er ingen begrensning på kompleksiteten til fargene og de tilhørende uttrykkene. Det antas at fargesettene har minst ett element.

2.3.2 Steder, overganger og piler

Steder, overganger og piler er beskrevet med de tre settene P, T og A, og er endelige og parvis disjunkte. Steder og overganger kalles med en fellesbetegnelse **noder**.

2.3.3 Nodefunksjonen

Nodefunksjonen kobler hver pil med et par noder, hvor det første elementet er en kildenode og det andre er en destinasjonsnode. De to nodene må være av forskjellig slag, dvs ett sted og en overgang.

2.3.4 Fargefunksjonen

Fargefunksjonen kobler hvert sted p til et fargesett $C(p)$. Dette betyr at hver **brikke** i p må ha en farge som hører til $C(p)$.

2.3.5 Vokterfunksjonen

Vokterfunksjonen G kobler hver overgang t til en boolsk funksjon. Alle variablene i t har typer som hører til Σ. Typene til vokterfunksjonen til en overgang t er typene til variablene til vokterfunksjonen, samt de boolske verdiene *sann* og *usann*. Dersom vokteren er utelatt, evalueres den til *sann*.

2.3.6 Piluttrykkfunksjonen

Piluttrykkfunksjonen E kobler hver pil a til et uttrykk som må være av typen $C(p(a))_{MS}$. Dette innebærer at piluttrykket evaluerer til et multisett over fargesettet til det korresponderende stedet $p(a)$

2.3.7 Initialisatorfunksjonen

Initialisatorfunksjonen I kobler hvert sted p til et lukket uttrykk som er et multisett av $C(p)$. Det er disse elementene i multisettene $C(p)_{MS}$ som representeres ved brikkene, og initialisatorfunksjonen uttrykker den initielle merkingen.

2.4 Innsteder og utsteder

Innsteder og *utsteder* er en måte å referere til steder i forhold til overganger. Innstedene til en overgang t er alle stedene som er knyttet til en pil som går til t, og likeledes er utstedene til t alle stedene som har en pil tilknyttet seg, som kommer fra t. Utstedene til t skrives $t\bullet$, mens innstedene skrives $\bullet t$. Det er ingenting i veien for at $t\bullet$ og $\bullet t$ overlapper helt eller delvis.

2.5 Hierarkiske petrinet

Hierarkisk petrinet(13) (**HP**) er presentert som en metode for å holde store petrinet mer oversiktlige. Ideen er å kunne representere subnett ved enkle overganger. HP benytter seg av to teknikker: *overgangssubstitusjon* og *stedfusjon*. HP består av et sett ikke-hierarkiske petrinet, kalt *sider*. En side kan ha flere instanser. Et sted s som forekommer på en side som har flere instanser anses ikke som samme sted medmindre det eksplisitt er det som følge av stedfusjon.

2.5.1 Overgangssubstitusjon

Ideen med *overgangssubstitusjon* er å la en *substitusjonsovergang* med tilhørende piler representere et mer komplekst subnett. En substitusjonsovergang (t_{HS})[1] fungerer på samme måte som en vanlig overgang, men er i tillegg knyttet til en side. Det er overgangssubstitusjonen som knytter de ulike sidene sammen. En substitusjonsovergang representerer en hel side. Siden til en substitusjonsovergang t_{HS} inneholder også alle innstedene og utstedene til t_{HS}.

2.5.2 Stedfusjon

Ideen med *stedfusjon* er at man har et sett steder som er identiske, dvs at de representerer ett og samme konseptuelle sted selvom de er fremstilt som flere. Stedene i settet skal alltid ha identisk merking, og dersom en brikke legges til eller fjernes fra ett av stedene, skal det samme skje med de andre stedene. Stedene i et slikt *fusjonssett* kan være på samme side eller på forskjellige. Dersom stedene er på samme side, er det for å gjøre tegning av nettet mer praktisk. Dersom de derimot er på hver sine sider eller instanser av samme side er det for å kunne knytte sidene sammen.

2.6 Hierarkisk petrinet i denne oppgaven

I denne oppgaven forekommer flere elementer som minner om HP. Man har tegninger og ikoner som representerer petrinet, og som kan sies å være sider. Det ble her nødvendig å utvikle en ekstra teknikk, nemlig *overgangsfusjon*.

2.6.1 Overgangsfusjon

Overgangsfusjon fungerer på en litt annen måte enn stedfusjon. De representerer begge den samme noden, multisettet av piler til en fusjonert overgang t er summen av multisettene av piler til alle overgangene i fusjonssettet til t. Således er en fusjonert overgang ikke aktiv medmindre alle overgangene i fusjonssettet til t er aktive, og fyrer en av overgangene, fyrer alle.

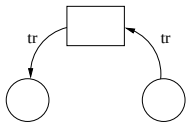
[1] for Hierarkisk Substitusjon

2.7 Jernbanekomponenter

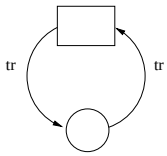
Det er også nødvendig å definere hva et jernbanenett og hva en jernbanekomponent er. Jernbanenett er enkelt og greit et farget petrinet hvor fargesettet inkluderer brikker av typen *tr*, som betyr *tog*-brikker, samt piler som opererer på disse.

En jernbanekomponent har minst en overgang som har tog både på en innpil og en utpil, og har stedene til disse.

Uformelt er en jernbanekomponent noe som utfører noe på en togbrikke. Således har den minst en overgang. For å gi mening, lar vi den ha alle stedene som har piler inn til overgangen, og alle stedene som har piler ut fra overgangen.(figur 2.3) Overgangen må ha minst et utsted og et innsted. Dette kan være det samme stedet. Således vil den minste mulige komponenten ha en overgang, ett sted, og to piler. Ikke helt tilfeldig er dette endestykkekomponenten. (figur 2.4)



Figur 2.3: Jernbanekomponent



Figur 2.4: Minste jernbanekomponent

Kapittel 3

Modellering og forfining

Når man modellerer et stort system, kan man ikke simulere med hensyn på alle egenskaper samtidig. Til det måtte man hatt en tro kopi som kjørte sanntid, og det er ofte upraktisk når man snakker om jernbanenettverk. Selv det å skulle simulere og resonnere over alle de viktigste egenskapene blir fort svært komplekst, ettersom mulige tilstander systemet kan ha fort blir veldig mange, og i noen tilfeller uendelig. Man velger seg derfor ut noen egenskaper man ønsker å simulere.

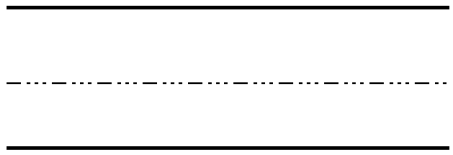
For å enklest mulig kunne endre egenskaper man ønsker å undersøke, kan man dele nettet i to abstraksjonsnivåer. Det ene nivået består av et sett udefinerte byggeklosser, samt regler for sammensetting av disse til større nett. Dette har fått navnet **spesifikasjonsnivået**. Det andre består av byggeklossene, spesifisert som petrinet. Dette har fått navnet **implementasjonsnivået**.

Gevinsten med å organisere nettet slik kommer ved at man kan ha mange forskjellige implementasjoner. Om man ønsker å simulere med hensyn på en ny egenskap, behøver man nå ikke lage hele nettet på nytt, men kun settet med byggeklosser i implementasjonsnivået. Man kan så få en datamaskin til å koble sammen de nye komponentene slik man har gjort på spesifikasjonsnivået.

3.1 Atomære komponenter

Byggeklossene i spesifikasjonsnivået har fått navnet **atomære komponenter**. Anders Moen og Ingrid C. Yu(9) har foreslått et sett med atomære komponenter. De er presentert nedenfor.

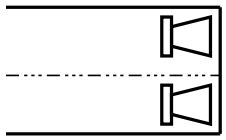
linjestykke



Figur 3.1: Linjestykke

Figur 3.1 viser hva komponenten linjestykke skal representere. Dette behøver ikke være et rett spor, det kan f.eks være en sving eller en bakke.

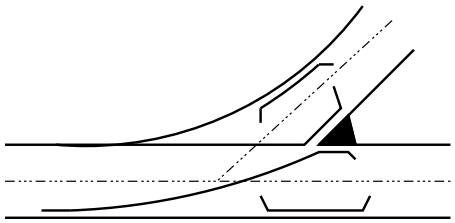
endestykke



Figur 3.2: Endestykke

Figur 3.2 viser et endestykke. Dette er slutten av en linje, med buffere for å stoppe et tog[1].

pens

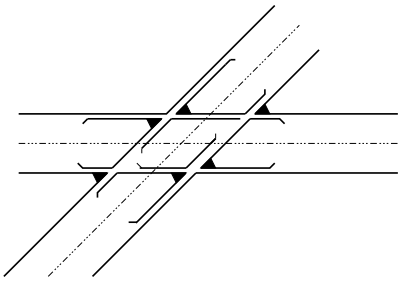


Figur 3.3: Pens

[1] som ikke kjører for fort

Figur 3.3 viser en pens, en komponent som splitter et spor i to. En pens har en mekanisk innretning kalt pensdrivmaskin som sørger for å fysisk flytte skinnene i pensen, og slik åpner for kjøring i en av retningene og lukker for den andre. For nå antar vi at det ikke går an å *kjøre opp pensen*, dvs komme fra den lukkede retningen og fysisk brøyte den åpen.

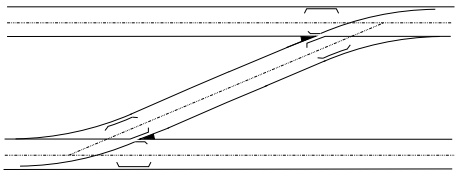
kryss



Figur 3.4: Kryss

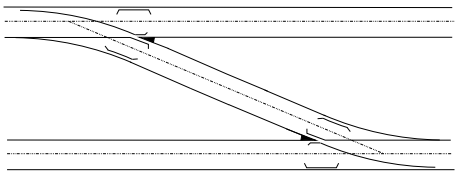
Figur 3.4 viser et kryss der to spor krysses, og tog ikke kan gå fra det ene sporet til det andre.

venstre- og høyresporskiftere

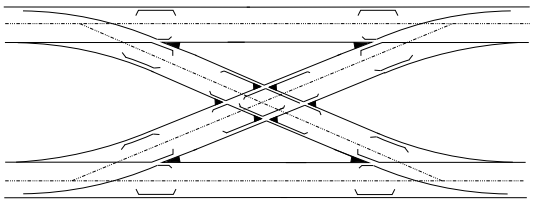


Figur 3.5: Venstresporskifter

Figurene 3.5 og 3.6 viser sporskiftere, komponenter mye brukt på dobbeltspor for å la tog flytte seg fra det ene til det andre av to parallelle spor.



Figur 3.6: Høyresporskifter

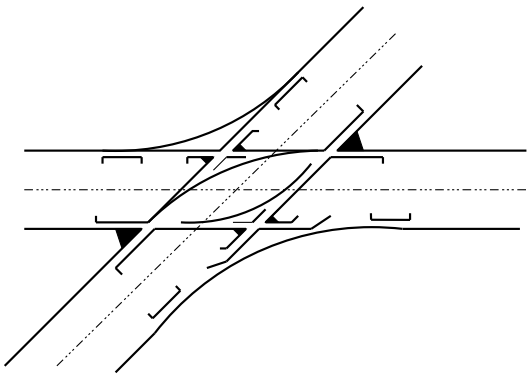


Figur 3.7: Saks

saks

Figur 3.7 viser en komponent som forbinder to parallelle spor slik at tog kan bytte spor uavhengig av hvilket spor det er på og hvilken retning det kjører.

engelskmann



Figur 3.8: Engelskmann

Figur 3.8 er en komponent som brukes inne på stasjonsområder hvor det

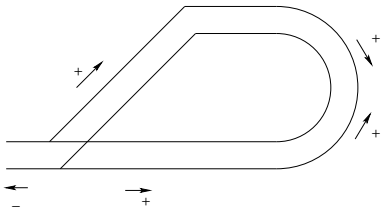
er lav fart. Her krysser to spor hverandre, og det er mulig for et tog å bytte spor.

3.1.1 Retning

Ettersom man skal lage en datamodell av dette, må man ta et hensyn til: Man må innføre et begrep om retning. Når et tog kjører langs et spor, vil det ikke være noen tvil om hvilken retning toget kjører, og således i hvilken retning det skal fortsette å kjøre. Man må overføre dette til modellen, og man behøver derfor at modellen inneholder et begrep om retning. Dette vil være rent syntaktisk.

Et virkelig jerbanespor strekker seg gjennom et tredimensjonalt rom. Det er flere tenkelige måter man kunne representert retningen på. Hadde man ønsket å være pinlig nøyaktig, kunne man knyttet dette til et koordinatsystem, som f.eks EUREF89[2] og UTM[3], og oppgitt retningen som en vektor. I praksis er dette unødvendig. Fra togets synspunkt er det kun to retninger å velge mellom: frem eller tilbake langs skinnegangen. Dette er kun én dimensjon. De to andre dimensjonene kommer til uttrykk på andre måter, som f.eks ved at det blir tyngre eller lettere å kjøre.

Man kan kalle retningene positiv og negativ retning, eventuelt pluss- og minusretning $(+/-)$. Det er nødvendig at det i hvert punkt i nettet er entydig hvilken retning som er hvilken, ettersom dette er en egenskap ved virkeligheten det er naturlig å benytte. Mangler man denne egenskapen vil ikke et tog kunne bevege seg jevnt langs et linjestykke. Det viser seg imidlertid at det er behov for at retningen som oppfattes som fremover kan endre seg fra en komponent til neste, og naturlig nok samtidig i et tog som traverserer komponentene. Dette kan lett illustreres ved følgende eksempel:



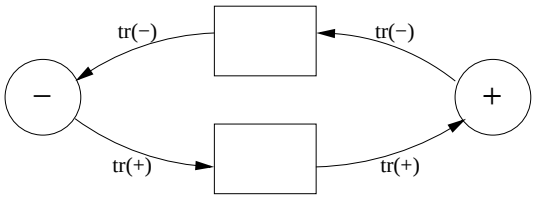
Tegningen illustrerer en pens hvor sporene går sammen i en løkke. Man kan anta at ut av pensen mot venstre er negativ retning, og de to veiene ut av

[2]Matematisk modell av jordkloden. EUREF89 er knyttet til den eurasiske kontinentalplaten.

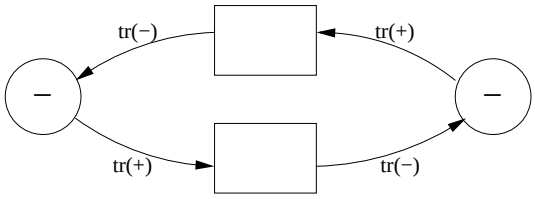
[3]Universal Transversal Mercatorprojeksjon, et globalt koordinatsystem som forankres i en matematisk modell av jordkloden, som f.eks EUREF89

pensen til høyre er positiv retning. Når de to veiene fra pensen møtes vil det finnes et punkt som er i positiv retning fra begge nabopunktene sine, og således må ha negativ retning til begge nabopunktene. Fra dette punktet er det altså ikke entydig hvilken retning som er hvilken. Andre antakelser om retning fra pensen vil lede til samme problem.

Løsningen man har valgt går ut på at man skifter retning i overgangen mellom to punkter. Dersom et vanlig linjestykke implementeres som i figur 3.9 kan en retningsendrer implementeres som i figur 3.10



Figur 3.9: Implementasjon av linjestykke



Figur 3.10: Implementasjon av retningsendrer

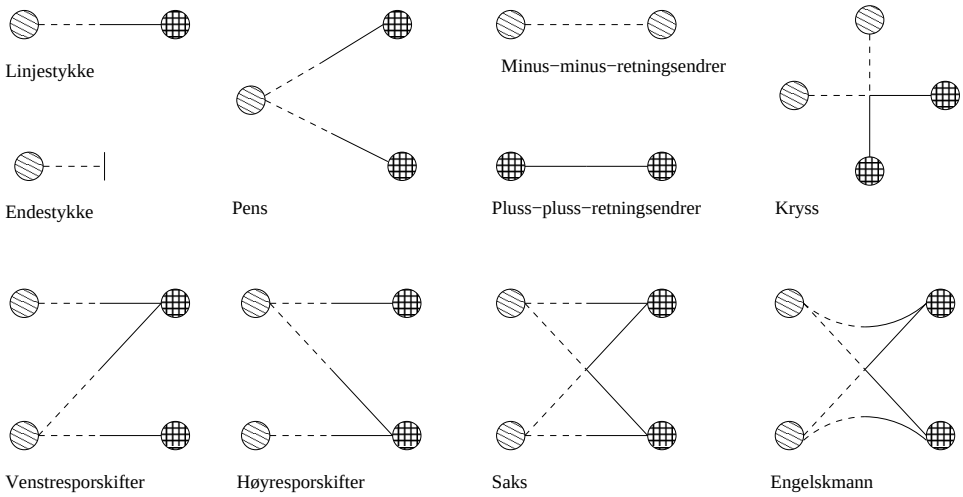
Et tog som forlater plassen til venstre i figur 3.9 for å forflytte seg til den høyre plassen starter med å bevege seg i positiv retning, og kommer frem til den venstre plassen fremdeles i positiv retning. Derimot vil et tog som forlater plassen til venstre i figur 3.10 for å forflytte seg mot høyre i positiv retning få retningen sin endret til negativ retning idet det plasseres på den høyre plassen. I tilfellet med pens-løkka over, vil toget da kunne fortsette rundt løkka, og problemet er løst. Man behøver da to slike komponenter, en hvor begge veier er positiv retning og en der begge veier er negativ retning.

For å ta hensyn til retning, behøves to komponenter til, som er:

- minus-minus-retningsendrer

- pluss-pluss-retningsendrer

En illustrasjon av de atomære komponentene som ikke viser de bakenforliggende petrinettene kan sees i figur 3.11. Dette er de atomære komponentene, altså spesifikasjonsnivået. De skråstrekfylte sirklene representerer komponentens sted i minusretning og de stiplede linjene pilene som flytter tog i minusretning til stedet som er i minusretning. På samme måte representerer de rutefylte sirklene steder i plussretning, og de heltrukne linjene piler som flytter tog i plussretning mot pluss-stedet.



Figur 3.11: Atomære komponenter.

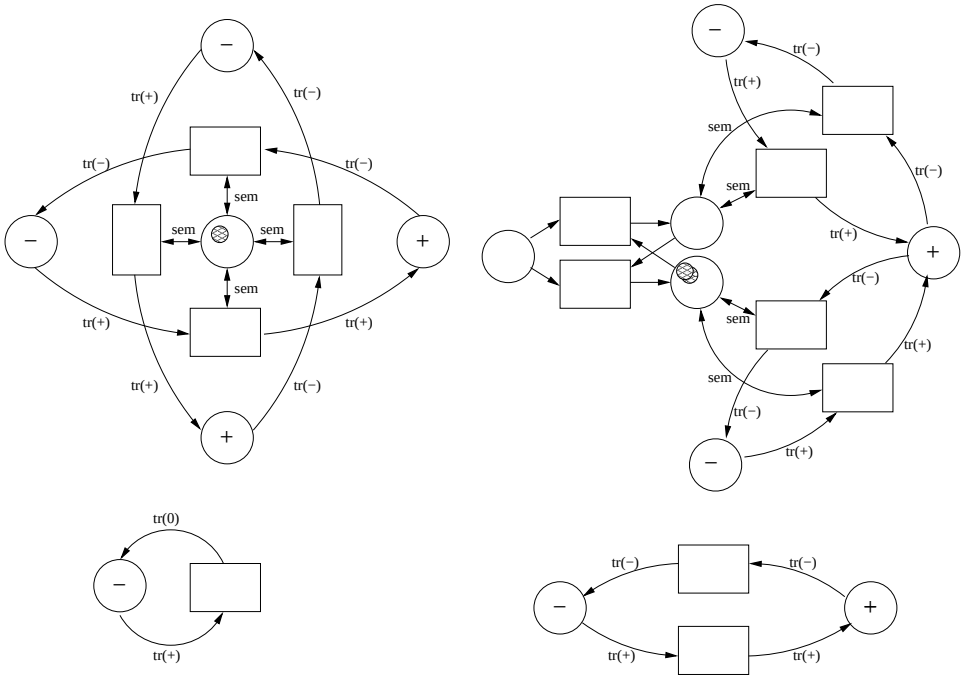
3.1.2 Sammensetting

Komponentene settes sammen på samme måte som hierarkiske petrinett. Det er imidlertid nødvendig med noen regler for hvordan man kan sette sammen komponentene. Man ønsker på dette nivået å holde reglene så lite restriktive som mulig, og kun ta hensyn til at nettet skal virke som vi ønsker, altså at det overalt er entydig hvilken retning som er hvilken. Alle komponentene har ender, og disse er enten en pluss-ende eller en minus-ende, avhengig av hvilken retning et tog som har kjørt fra den ene enden av komponenten til den andre har når det er ferdig med bevegelsen. Regelen for sammensetting er enkel og grei: Pluss kan kun settes sammen med minus, og minus kan kun settes sammen med pluss.

Slik settes nettet sammen ved stedfusjon, der man fusjonerer par med et pluss-sted og et minus-sted.

3.2 Grunnleggende komponenter

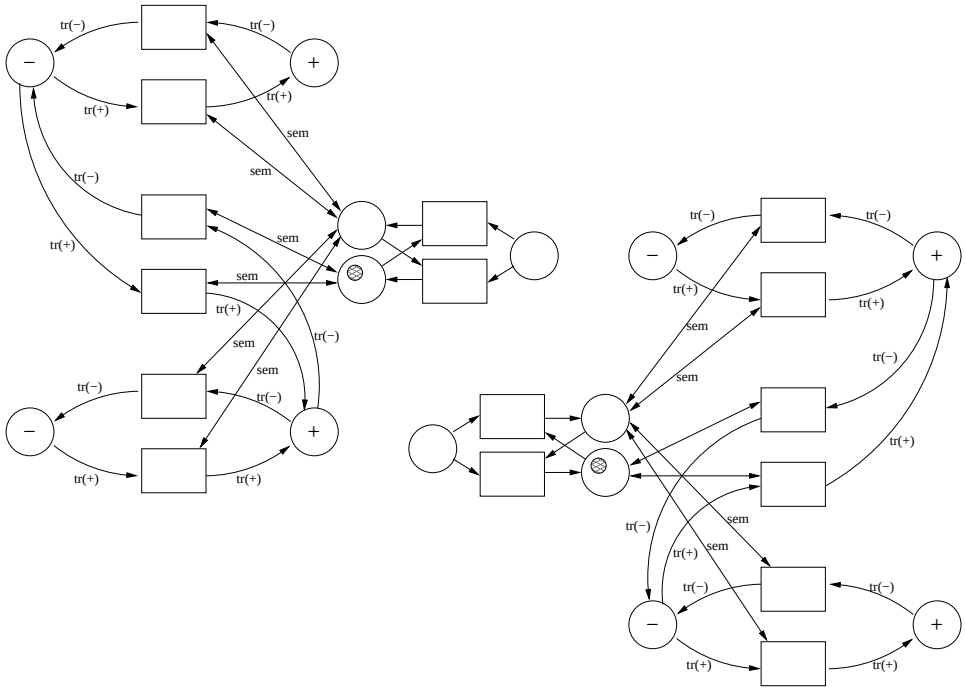
A.M.Hagalisletto og I.C.Yu(22) har laget noen grunnleggende sett med jernbanekomponenter de har kalt *basic* og *safety*. *Basic*-komponentene er presentert i figurene 3.12, 3.13, 3.14 og 3.15.



Figur 3.12: Basic kryss, vei, pens og endestykke

Kort fortalt fungerer komponentene slik: *Basic* et sett komponenter som har som eneste egenskap at tog kan flytte seg. De er således et godt utgangspunkt å bygge videre på, da det er naturlig å anta at de aller fleste ønskelige simuleringer forutsetter at tog flytter seg. Fargesettet til *basic* er $tr(r)$[4] med de mulige verdiene $tr(+), tr(-), tr(0)$ og *sem* som er en semaforbrikke. Semaforer brukes i flere komponenter, f.eks i **kryss** for

[4]train



Figur 3.13: Basic venstre- og høyresporskifter

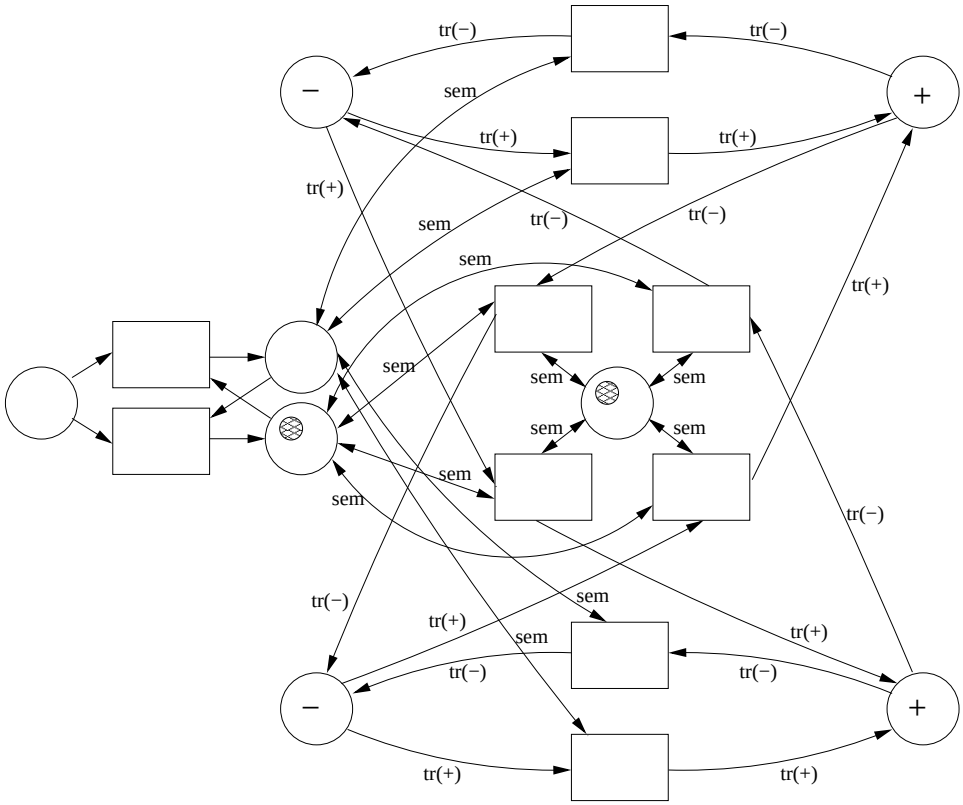
å sørge for at ikke tog passerer krysset samtidig, og i **pens** for å indikere hvilken retning som er åpen.

Safety er en utvidelse av *basic* som har som tilleggsegenskap at petrinettet alltid har nøyaktig en brikke på hver plass; dette er enten en togbrikke eller en ikke-togbrikke. Dette forhindrer en rekke situasjoner i modellen som åpenbart ville passet dårlig med virkeligheten, slik som at to tog passerer hverandre i samme spor. At ethvert sted kan ha høyst én brikke, kalles *1-safe*. Dette er altså også et sett komponenter det kan være naturlig å bygge videre på. Fargesettet til *safety* er det samme som for *basic* med tillegget *ikke-tog*-brikken nt[5]

Et eksempel på en *safety*-komponent kan sees i figur 3.16.

For å kunne bygge vilkårlige nett, behøver man også de syntaktiske komponentene som endrer retning på tog og skinner. Da disse behøves i alle utvidelsene, legges de til i settet av atomære komponenter.

[5]NoTrain



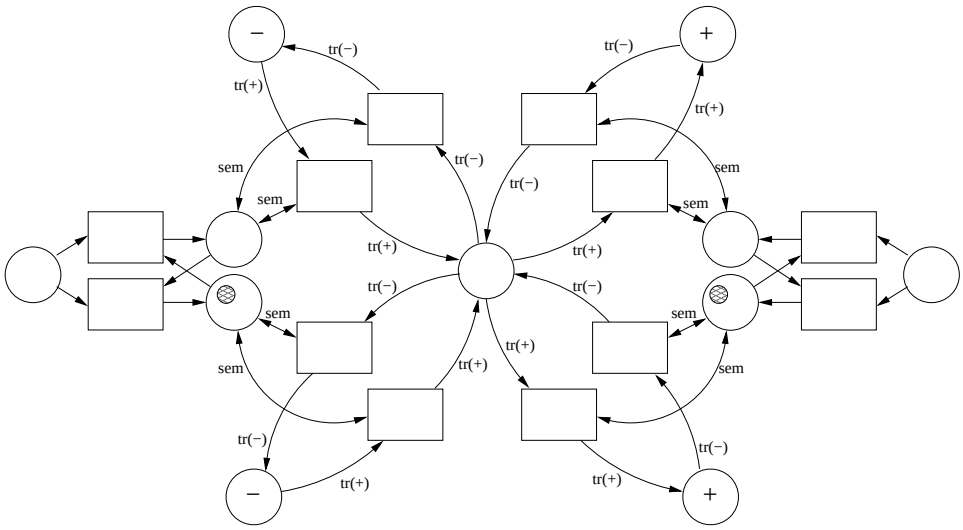
Figur 3.14: Basic saks

3.3 Informasjonsbærende komponenter

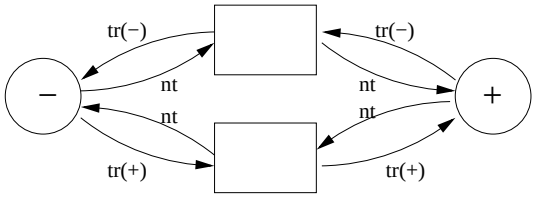
Noe det også lett kan finnes nytte i, er at komponentene har ytterligere informasjon knyttet til seg. Dette kan være hva som helst, men for å velge en grunnleggende egenskap lar vi her komponentene vite hvor mange meter virkelig skinnegang de representerer. Andre typer informasjon kunne vært hellningsgradient på bakken, lokale fartsgrenser etc. Hva denne informasjonen brukes til vil være opp til ytterligere utvidelser av komponentene. Fargesettet utvides derfor med fargen$info = nat$.

De to syntaktiske komponentene $+ +$ og - - skiller seg fra de strukturelle komponentene ved at de ikke representerer skinnegang, men er med for å få nettet til å virke. På grunn av dette får de ikke noen informasjonsplass.

De informasjonsbærende komponentene vises i figurene 3.17, 3.18 og 3.19.



Figur 3.15: Basic engelskmann

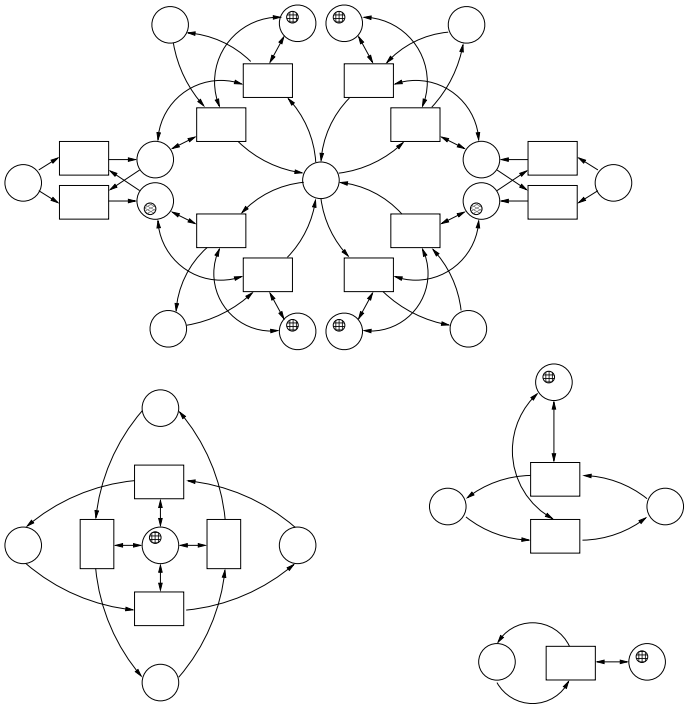


Figur 3.16: Veistykke med safety

Brikken som synes på tegningene er en informasjonsbrikke, som inneholder data. I noen av tilfellene, som f.eks i komponenten kryss, har man latt semaforbrikken som allerede finnes også fungere som informasjonsbrikke.

3.4 Kollisjonsdeteksjon

En naturlig ting å bruke en modell av et jernbanesystem til, er å forebygge kollisjoner. For å få til dette, må vi først og fremst ha en modell der togene *kan* kollidere. Dernest må modellen sørge for at hvis tog oppfører seg slik at de i en virkelig situasjon ville kollidert, slik som å kjøre mot hverandre i samme spor, vil de faktisk kollidere. Når man har en slik modell, kan man



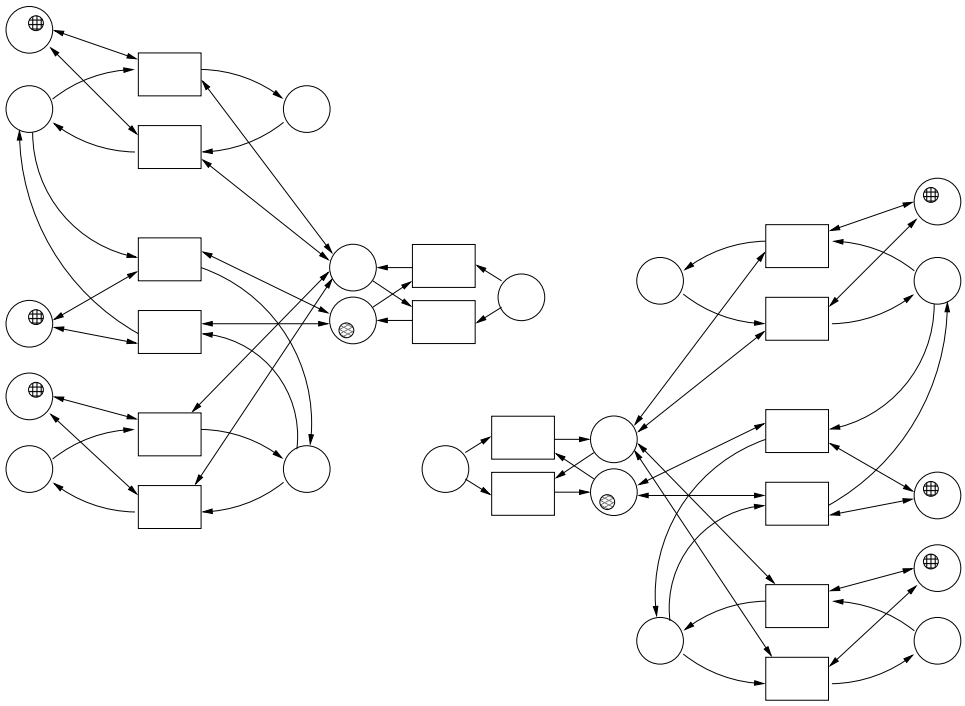
Figur 3.17: Informasjonsbærende komponenter

analysere denne med tanke på å redusere fare for ulykker. Dette kan f.eks gjøres i modellsjekkere, ved at man søker etter situasjoner der det finnes et kollidert tog. Om man er i stand til å finne en slik tilstand, kan man undersøke hva slags betingelser som må til for at dette skal skje, og deretter iverksette tiltak.

Det man krever av en slik modell er:

1. Tog kan kollidere.
2. To tog kan aldri bytte plass i ett steg.
3. To tog vil alltid kollidere dersom de prøver å oppta samme plass.
4. Dersom man finner ytterligere situasjoner i virkeligheten hvor tog kolliderer, vil tog i modellen som oppfører seg slik også kollidere.

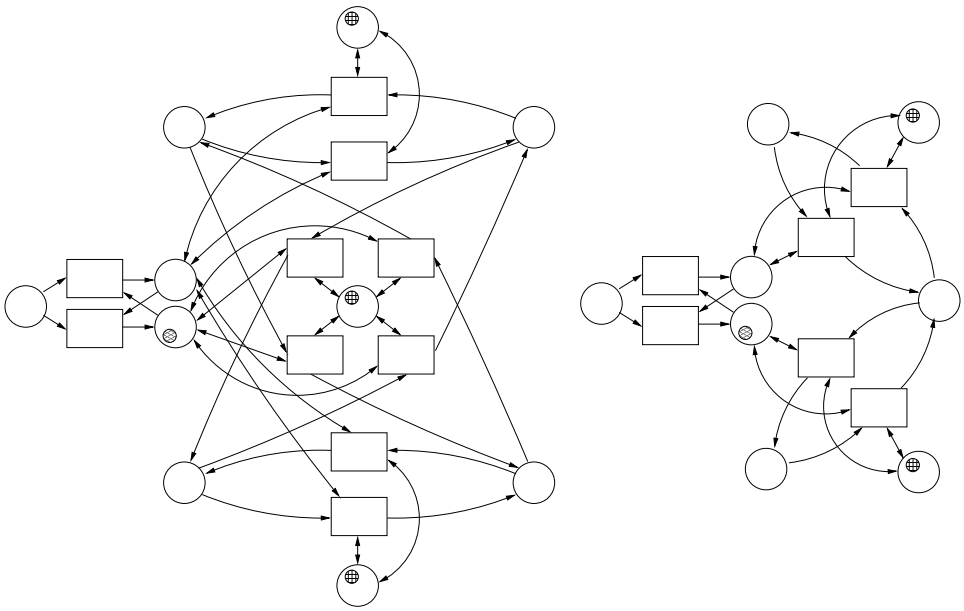
Et eksempel på hvorfor det siste punktet er nødvendig, er komponenten **kryss**. To tog som kommer på hvert sitt spor, og passerer krysset samtidig,



Figur 3.18: Informasjonsbærende komponenter

vil i modellen aldri oppta samme sted, men bør likevel kollidere. Komponenten **kryss** må dermed ta høyde for dette ved å la to tog som prøver å passere et kryss samtidig, kollidere istedet. Man kan selvsagt gjøre betraktninger om fart og lignende, men for nå antar vi at tog enten har stoppet før de har møtt hverandre; eller ikke har det, og dermed kollidert.
Et sett komponenter det er nærliggende å prøve å basere kollisjonsdetekterende komponenter på, er *safety*. De har allerede egenskapen at to tog ikke kan bytte plass i ett steg, og at de ikke kan oppta samme plass. Man behøver altså å legge til at dersom togene "forsøker" sistnevnte, vil de kollidere. *Safety*-komponentene er 1-safe. Dette er en egenskap vi åpenbart ønsker å beholde, ettersom brudd på dette kan medfører brudd på punkt tre ovenfor.

Denne utvidelsen behøver noen tillegg i fargesettet. Togene får en ny mulig verdi: $tr(Crash)$. Denne representerer at toget har havarert og ikke kommer noen vei.



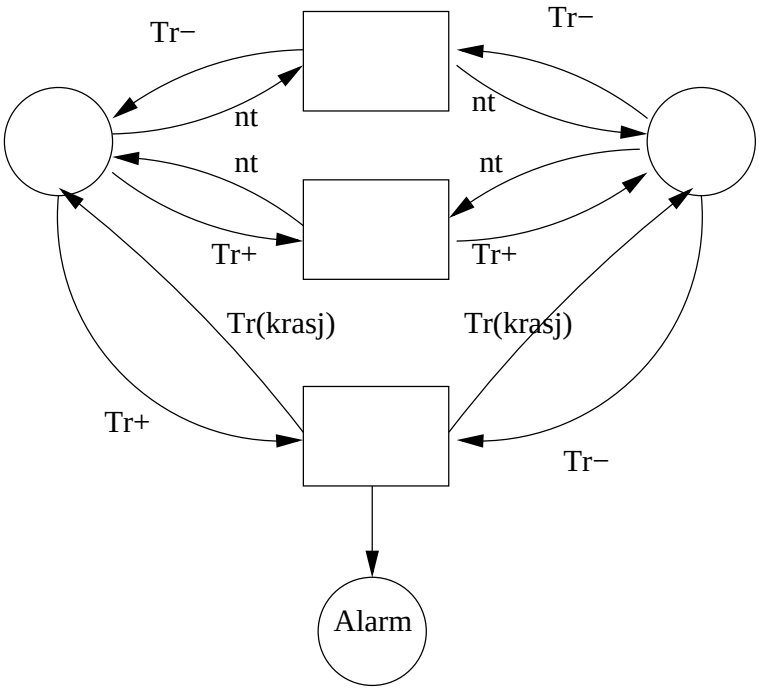
Figur 3.19: Informasjonsbærende komponenter

3.4.1 Forklaring og eksempler

Figur 3.20 viser hvordan et linjestykke med kollisjonsdeteksjon kan se ut. Hvis det i utgangspunktet står en togbrikke med retning + i den venstre plassen og en ikke-togbrikke i den høyre, eller omvendt, vil toget flytte seg som for et vanlig linjestykke. Dersom det imidlertid står en togbrikke med retning + i den venstre plassen, og en annen togbrikke med retning − i den høyre, kan de kollidere. Vi ser at den eneste overgangen som er aktiv her, er den merket *Crash*. Da vil togene fjernes fra plassene sine, og returneres i en krasjet tilstand. Dette kan man modellere på flere måter. I tillegg vil en semaforbrikke legges i plassen merket *Alarm*. I en implementasjon vil dette antakeligvis kobles til store varsellamper etc. i et kontrollrom. Figur 3.21 viser et kryss med kollisjonsdeteksjon. For å bedre oversikten er piluttrykkene utelatt. Her kan man se at tog også kan kollidere med andre tog på det kryssende sporet.

3.4.2 Betraktninger

Det er nødvendig å vise at disse komponentene oppfyller kravene til en modell for kollisjonsdeteksjon slik de ble presentert i begynnelsen av



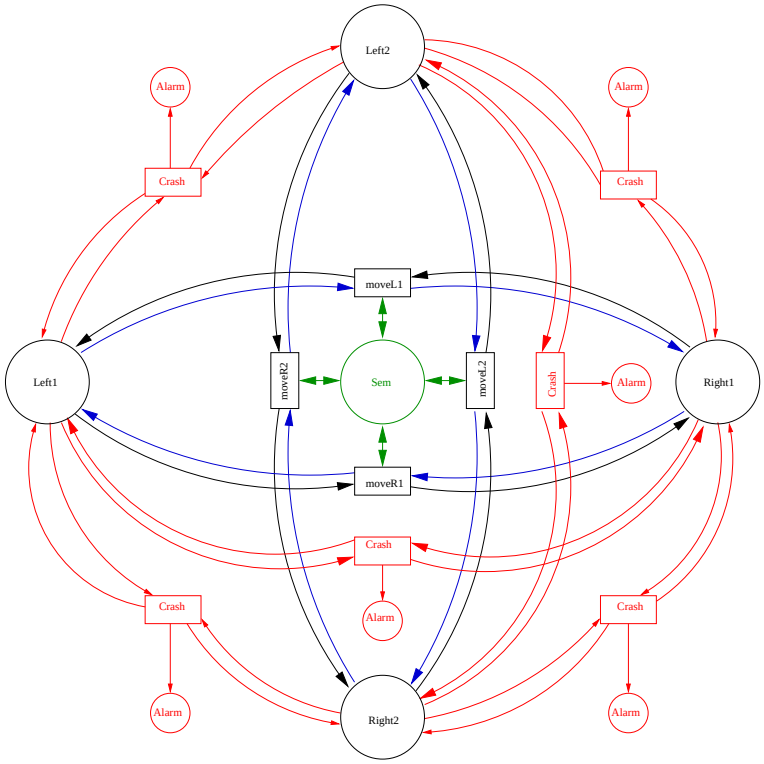
Figur 3.20: Linjestykke med kollisjonsdeteksjon

seksjonen. For å gjøre dette, må man først vise at komponentene enkeltvis oppfyller kravene, og deretter at ikke sammensettingen endrer dette på noe vis.

Punkt 1. Dersom det finnes en komponent der tog kan kollidere, er det oppfylt. At tog kan kollidere defineres som at det finnes en merking slik at en togbrikke kan få verdien **Crash** uten å ha denne verdien fra før.

Her kan man ta for seg komponenten *linjestykke*. Komponenten *Linjestykke* har utpiler som setter togbrikker til **Crash**. Dernest må man se om det kan tenkes en merking som gjør overgangen som har disse utpilene aktiv. En merking som har togbrikker i begge stedene til linjestykket, og nevnte togbrikker har retning mot hverandre, oppfyller dette.

Punkt 2. Alle komponentene er her bygget over *safety*, som har som egenskap at det til en hver tid ikke befinner seg mer enn én brikke på hvert sted. Dette er bevart i alle de kollisjonsvare komponentene. Alle overganger som legger ut en tog- eller en ikketogbrikke har først fjernet en tog- eller ikketogbrikke fra det samme stedet, slik at dersom den initielle merkingen



Figur 3.21: Kryss med kollisjonsdeteksjon

ikke har flere enn én tog- eller ikketogbrikke noe sted, vil det ikke kunne oppstå situasjoner der det er flere enn én slik brikke på samme sted.

Alle overganger som fjerner to togbrikker returnerer to togbrikker med verdien **Crash**. Alle overganger som fjerner én togbrikke og setter ut en togbrikke et annet sted, fjerner også en ikketogbrikke fra dette andre stedet. Dersom dette stedet okkuperes av en togbrikke, vil sistnevnte overgang ikke være aktiv, ettersom det er vist at nevnte sted ikke også kan inneholde en ikketogbrikke. flyttes vekk fra hverandre. To togbrikker kan således ikke bytte plass i ett steg.

Punkt 3. Det forstås slik at dersom en togbrikke oppfyller noen av pilene til en overgang som kan flytte togbrikken til et sted som har en annen togbrikke, og denne andre togbrikken likeledes oppfyller noen av pilene til en overgang som kan flytte togbrikken til det første stedet, prøver de å oppta

samme plass, og skal således få verdien **Crash**.

Dersom togbrikkene har retning mot hverandre, vil den eneste overgangen som er aktiv være den som setter begge togene til **Crash**. Dersom togbrikkene ikke har retning mot hverandre, vil minst det ene toget prøve å oppta en plass der det andre toget ikke er, og togbrikkene vil flytte seg vekk fra hverandre.

Punkt 4. Hittil er det kun presentert én slik situasjon, som er at dersom to tog prøver å passere samme krysset samtidig, vil de kollidere. Dersom man finner flere situasjoner, må man utvide beviset for disse.

I komponenten kryss er det slik at det finnes nøyaktig én semaforbrikke i semaforplassen i midten, og at alle overganger som flytter en togbrikke forbi krysset benytter denne semaforbrikken. To overganger kan ikke benytte den samme brikken i samme steg. Den eneste overgangen som fjerner to togbrikker setter begge brikkene tilbake med verdien **Crash**.

Når to togbrikker er på vei inn i krysset på hvert sitt spor er de mulige hendelsene disse:

1. Begge togbrikkene står stille.
2. En av togbrikkene flytter seg forbi krysset mens den andre togbrikken står stille.
3. Begge togbrikkene får verdien **Crash**

Det er ikkedeterministisk hvilken av disse hendelsene som inntreffer. Den første hendelsen skaper samme situasjon om igjen, og vil fortsette med det til en av de andre hendelsene inntreffer, så den kan man se bort ifra. Den andre hendelsen vil tilsvare at det ene toget passerer krysset rett før det andre. Vi har da ikke lenger situasjonen at togene passerer samtidig, så dette er også greit. Da har vi kun igjen at togene har prøvd å passere krysset samtidig, og har kollidert.

3.5 Følsomme nett

Noe man gjerne har nytte av når man modellerer jernbane, er et signalsystem. I første omgang lar man signalsystemet kun behandle lengden fri bane i hver retning, slik at tog kan kunne vite hvor fort det kan kjøre. Dette er ment som en grunnstamme man kan bygge videre på. Man kan lett tenke seg en implementasjon av blokkstrekninger, lyssignaler et cetera, som benytter seg av denne grunnstammen.

En av de største utfordringene med dette, er å få det modulært, slik at man fortsatt kan bygge nettene av en liten mengde små komponenter. Utfordringen kommer av at hver komponent plutselig må ta hensyn til informasjon mange andre plasser i nettet, og ikke bare i nabokomponentene.

Det man ønsker av et signalsystem blir da:

1. Det kan bygges inn i eksisterende komponenter, som en utvidelse (modulært)

2. Det har korrekt informasjon om avstanden til hindringer i hver retning fra ethvert punkt

I tillegg må vi definere hva som er en hindring:

- Et tog er en hindring
- Et endestykke er en hindring
- Komponenter kan ha situasjoner som er hindringer

Eksempler på det siste punktet kan være komponenten *pens*, der pensen til enhver tid står i en av to retninger. For et tog som kjører mot pensen den veien som ikke er åpen, vil det være en hindring i pensen. For et kryss kan det også være hensiktsmessig å anta at for et tog er det nærmeste toget en hindring, uavhengig om det står på det samme sporet, eller om det står på det kryssende sporet. Om man lager utvidelser vil man kunne finne flere situasjoner som kan være en hindring, f.eks rødt lys, om man implementerer lyssignaler.

Løsningen på utfordringen ble å lage signalsystemet rekursivt, slik at komponenter som hadde en hindring, sendte beskjed om dette i en eller begge retnigner. Signalsystemkomponentene er en utvidelse av *safety. nt*-brikken får et sett ekstra verdier. Disse er:

- Lengden fri bane i plussretningen
- Lengden fri bane i minusretningen
- Semafor for å vite om signal skal oppdateres i plussretningen
- Semafor for å vite om signal skal oppdateres i minusretningen

Tog-brikken får også et nytt felt. Dette er en semafor som indikerer hvorvidt toget har flyttet seg, og således om det skal oppdatere nabofeltene sine.

Komponenter som får en beskjed om en hindring, øker avstandsverdien, og sender beskjeden videre. For å gjøre det mer effektivt, sender kun komponenter som har fått en ny hindring, som f.eks at det har kommet et tog eller en pens har skiftet posisjon, beskjed. Endestykker sender kun informasjon videre når de mottar informasjon. Dette er fordi det ikke er noen grunn til å sende nye signaler langs tomme linjer. Et linjestykke med signaler er illustrert i 3.22.

3.5.1 Forklaring og eksempler

Signalsystemet for et rett linjestykke som vist i figur 3.22 fungerer slik: *nt*-brikkene inneholder følgende fire ekstra felter. $+d$, $-d$[6] er naturlige tall som inneholder informasjon om lengden fri bane i henholdsvis pluss-retningen og minus-retningen. $+ss$ og $-ss$[7] er boolske verdier som forteller om avstandsverdiene er blitt oppdatert og skal sendes videre. *Tog*-brikkene har det ekstra feltet ss[8], som er en boolsk verdi som forteller om toget har flyttet seg, og således skal sende informasjon fremover. Togets retning blir nå eksplisitt kalt d. Fargesettet blir således oppdatert ved at nt erstattes med $nt(+d = nat, -d = nat, +ss = boolean, -ss = boolean)$ og $tr([+-0])$ erstattes med $tr(d = [+-0], ss = boolean)$.

Signalsystemet demonstreres i figurserien 3.23, 3.24, 3.25, 3.26. Til å starte med er situasjonen som i figur 3.23. Merkingen består av ikketogbrikker(ikkeTog) i stedet $p1$, $p2$ og $p3$, og en togbrikke(Tog) i stedet $p4$. Brikkenes verdier er beskrevet i rammen i illustrasjonen; at en verdi er lik n betyr at den kan være et hvilket som helst naturlig tall.

Toget har $d = -$ og $ss = false$. Overgangen lengst til høyre merket *move-* er aktiv, og toget kan flytte seg. Toget fjernes fra plassen $p4$, og ikkeToget fra plassen $p3$. Toget plasseres i $p3$, og får satt $ss = true$, ettersom det nå er flyttet, og skal gi beskjed om det videre i systemet. IkkeToget plasseres i $p4$, der det får satt $-d = 0$ og $-ss = true$. Dette betyr at avstanden i $-$retningen fra ikkeToget er 0, og denne beskjeden skal sendes videre.

Nå flyttes fokus til det midterste partiet, slik det er illustrert i figur 3.24. Her har toget nylig ankommet fra høyre, og har derfor $ss = true$ og $d = -$. Plassen merket $p2$ har igjen et ikkeTog. Den midterste overgangen merket $nt-tr+$ er da aktiv. Merk at den midterste overgangen merket $move-$ **ikke**

[6] +distance og −distance
[7] +SendSignal og −SendSignal
[8] SendSignal

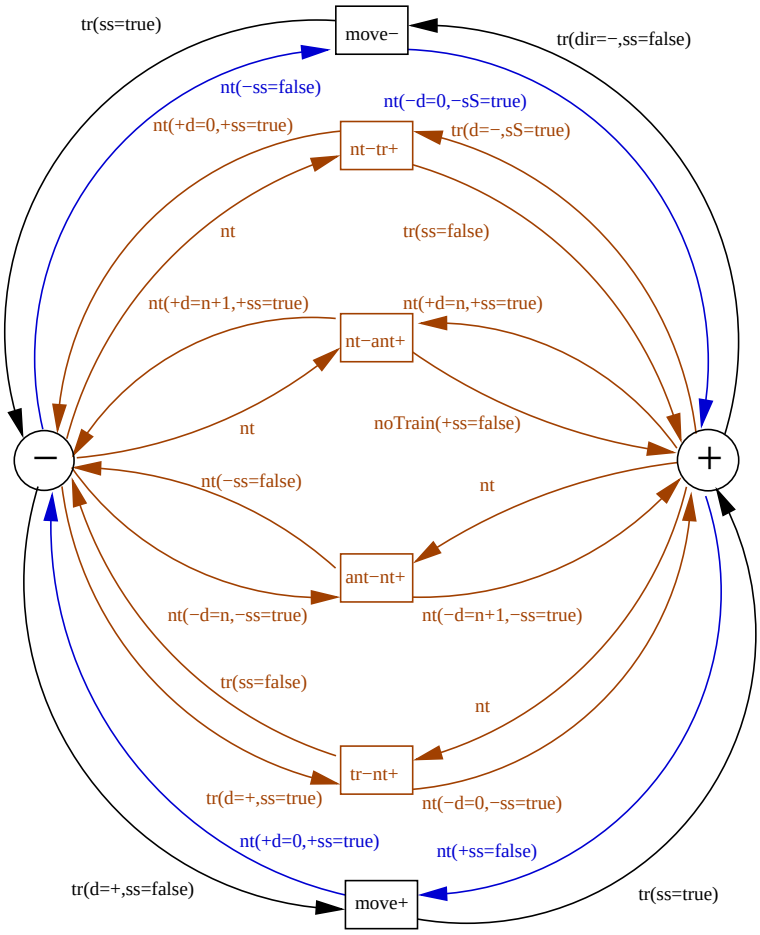
er aktiv. Dette er viktig for at ikke toget skal fjerne ikketogbrikken foran seg før toget har sendt signalet sitt.

Toget fjernes fra plassen $p3$, og ikkeToget fra plassen $p2$. Til forskjell fra forrige tilfelle, settes brikkene tilbake der de kom fra, men med oppdaterte verdier. Toget får satt verdien $ss = false$. IkkeToget får satt $d+ = 0$ og $+ss = true$.

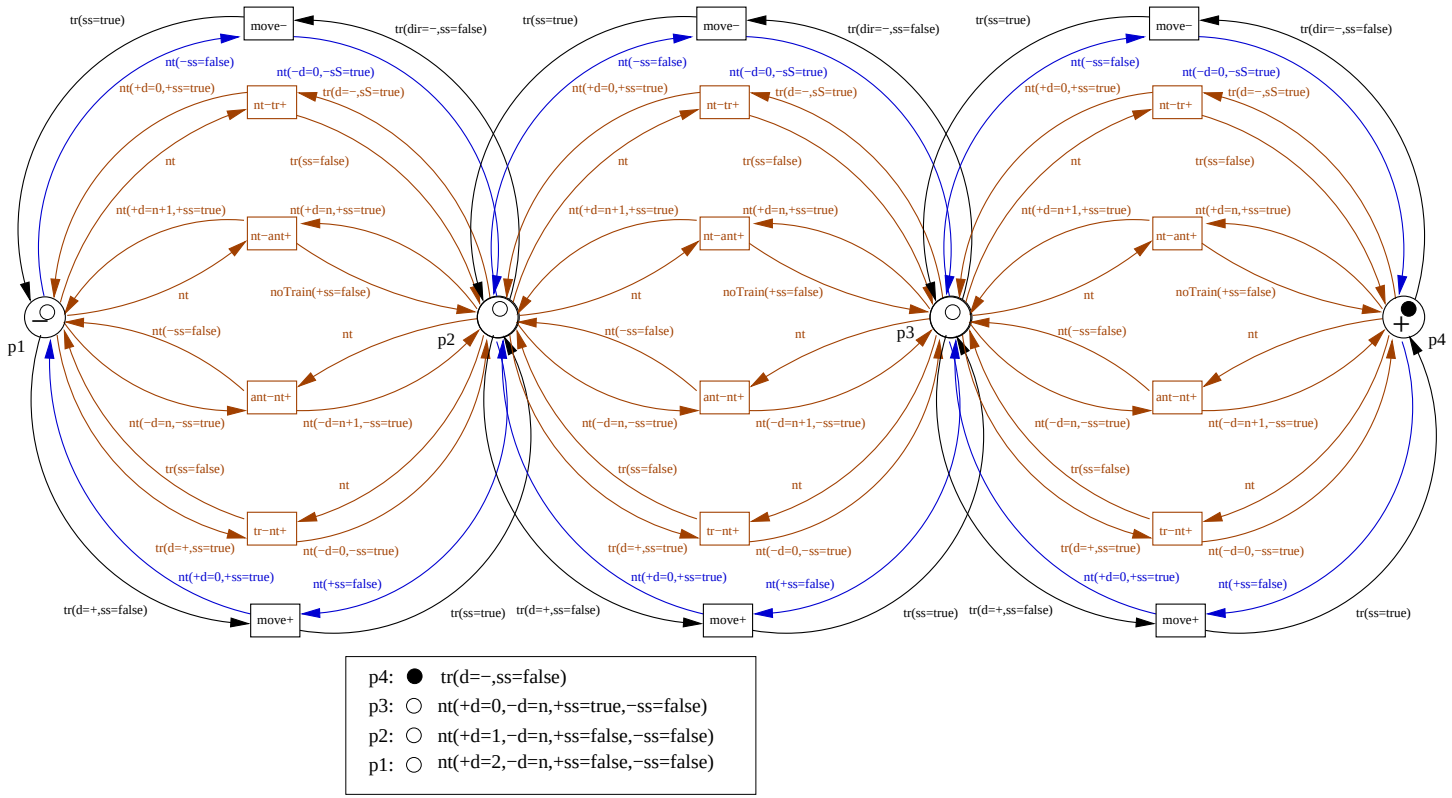
Fokus flytte nok en gang til venstre. Situasjonen ser nå ut som i 3.25. Her har vi ikkeTog i både $p1$ og $p2$. IkkeToget i $p2$ har verdien $d+ = 0$, og $+ss = true$. Den venstre overgangen merket $nt - ant+$ er da aktiv. Merk at heller ikke nå er den midterste overgangen merket $move-$ aktiv. Dette er igjen fordi det oppdaterte signalet skal sendes videre før det risikerer å bli fjernet.

Brikkene fjernes, får oppdatert verdiene, og settes tilbake. Situasjonen ser nå ut som i 3.26. IkkeToget i $p2$ har fått $-ss = false$. IkkeToget i $p1$ har fått $+d$-verdien sin satt til $+d = n + 1$ og $+ss$ til $+ss = true$.

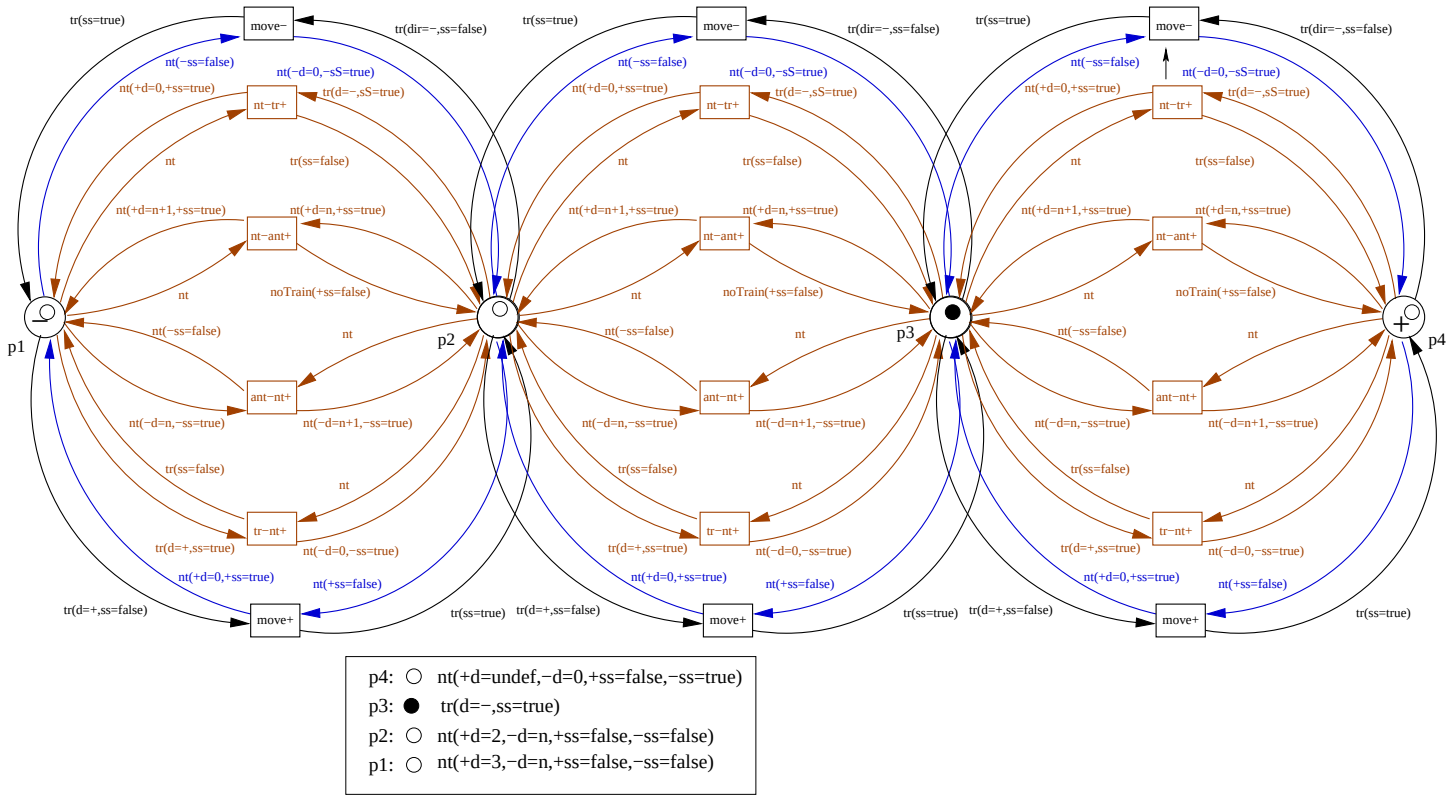
Neste situasjon enda et hakk til venstre vil være lik den tredje situasjonen skissert rett over, og slik vil signalet gå foran toget langs linja, og bli oppdatert hver gang toget flytter seg. Ikketogbrikken som bytter plass med toget i den første situasjonen lager en signalstrøm i motsatt retning på samme måte som i den tredje situasjonen, der overgangen $ant - nt+$ er aktiv, og $-d$-variablene blir oppdatert mot høyre.



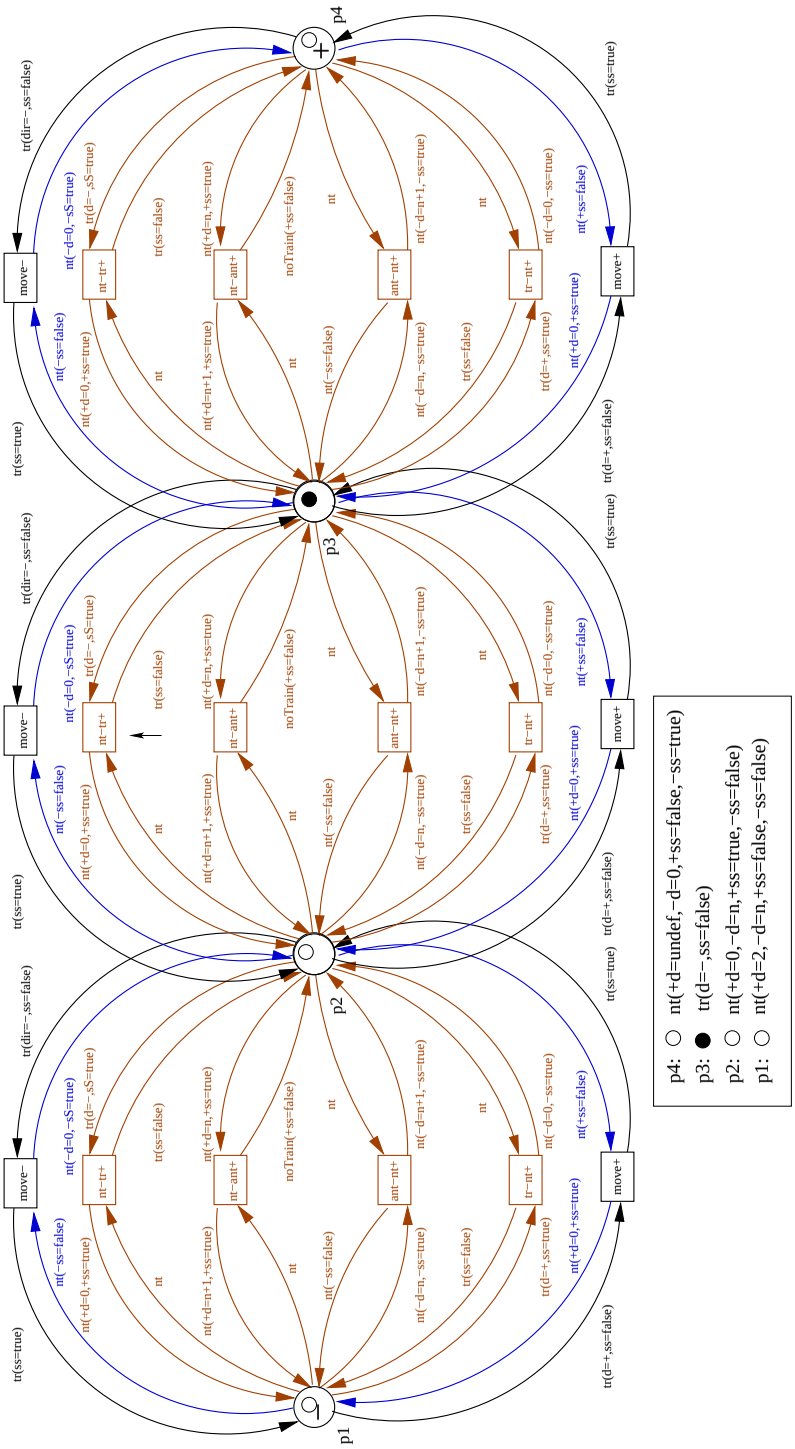
Figur 3.22: Linjestykke med signaler



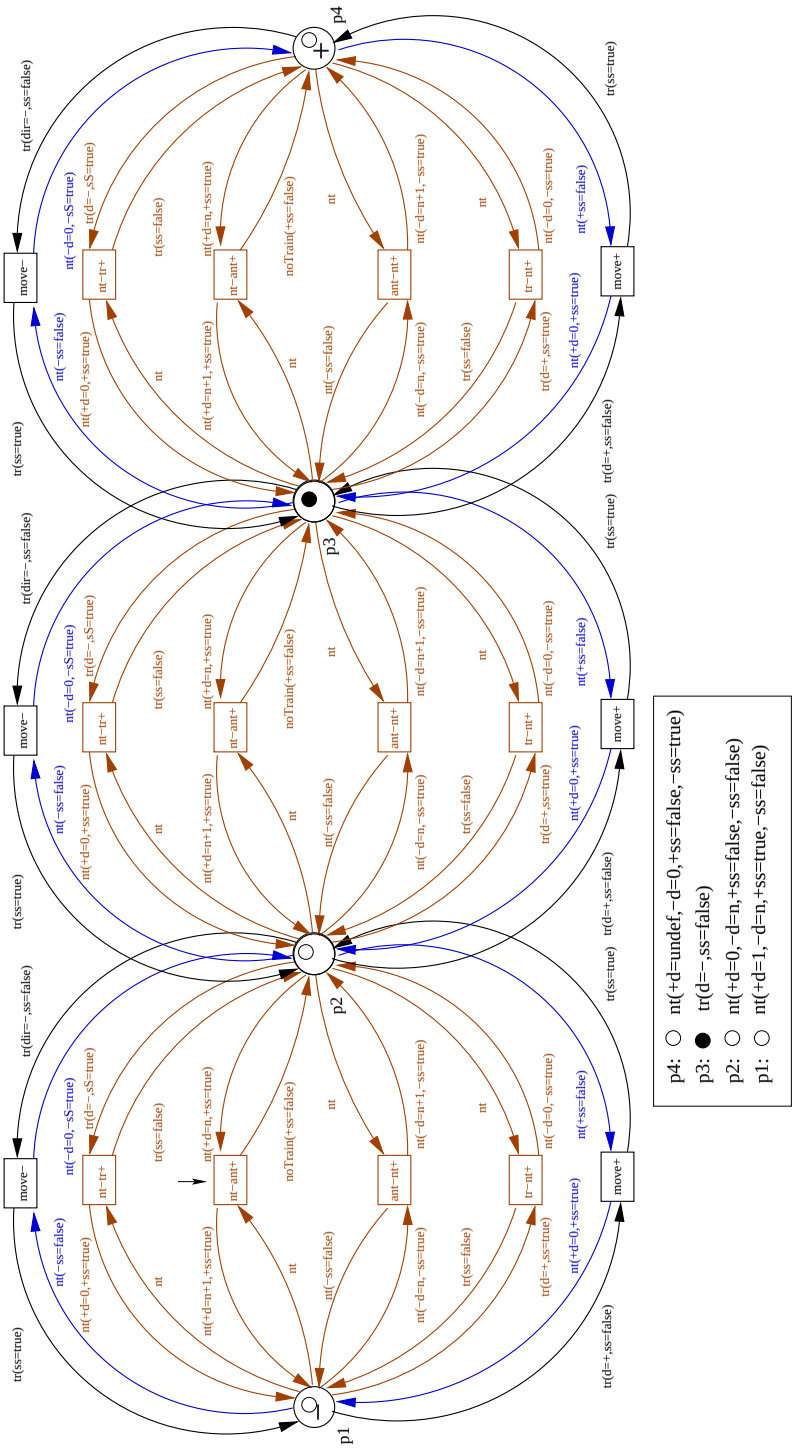
Figur 3.23: Illustrasjon av signaler, side 1



Figur 3.24: Illustrasjon av signaler, side 2



Figur 3.25: Illustrasjon av signaler, side 3



Figur 3.26: Illustrasjon av signaler, side 4

Kapittel 4

Dekomponering

4.1 Nukleoner

I komponentene som allerede er foreslått, ser man likheter og gjentakelser. Det kan være interessant å undersøke hvor få, og hvor små komponenter man kan klare seg med. Elementene i dette minimale settet har fått navnet *nukleoner*.

4.1.1 Hensikt

Det er tre grunner til å lage så få og så små komponenter som mulig:

1. Når man ønsker å utvide systemet, vil det være mindre jobb jo færre komponenter man behøver å lage.
2. Det blir færre kilder til feil med færre komponenter.
3. Få og små komponenter antas å være mer oversiktlige enn mange store.

4.1.2 Minimalt sett

Egenskaper

For å finne ut hvilke komponenter man kan klare seg med, er det viktig å kartlegge hvilke strukturelle egenskaper som eksisterer i de forskjellige komponentene. Disse er:

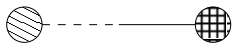
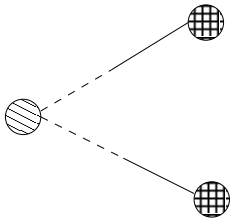
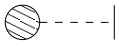
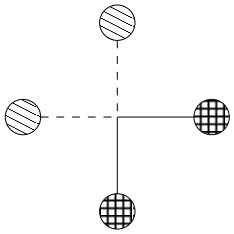
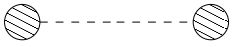
- Rett linjestykke
- Sporet deler seg

- Sporet slutter
- To spor krysser hverandre

Dette er alle strukturelle egenskaper som finnes i de tidligere skisserte komponentene. I tillegg har komponentene retning, slik at tog som kjører fremover vet hva fremover er. Dette er syntaktiske egenskaper.

Skal man ha det minste mulige settet, lager man en komponent for hver av de struktruelle egenskapene. Man behøver også de to retningsendrende syntaktiske komponentene.

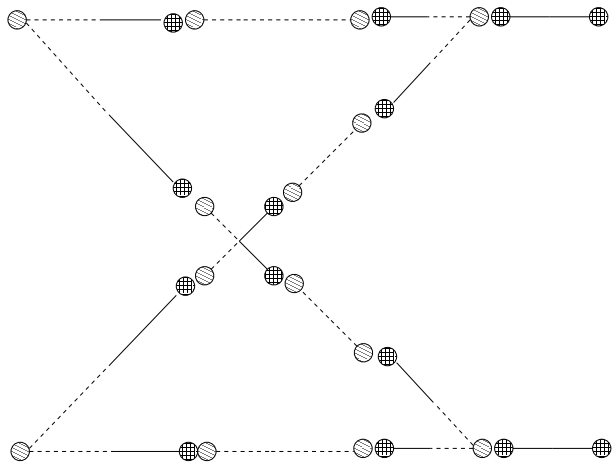
Komponentene blir da:

1. Rett linje 
2. Pens 
3. Endestykke 
4. Kryss 
5. Minus-minus-retningsendrer 
6. Pluss-pluss-retningsendrer 

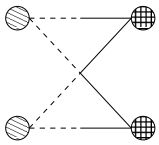
4.1.3 Sammenstilling

Man kan sette sammen disse nukleonkomponentene slik at de utfører oppgavene til de av de opprinnelige atomære komponentene som ikke finnes i det minste settet. Figur 4.2 viser den atomære komponenten saks. Figur 4.1 viser et nett av nukleonene som ligner på figur 4.2. Nukleon-nettet består av fire penser, fire minus-minus-retningsendrere, to pluss-pluss-retningsendrere

og et kryss. De to pluss-pluss-retningsendrerne er nødvendige for å få retningen i de ytterste stedene til å være like i de to nettene.



Figur 4.1: Sammensatt saks



Figur 4.2: Saks

Man må her passe på når man setter sammen komponentene at de fire pensene har den samme pensdrivmaskinen, dvs at alle skifter retning samtidig.

Disse to jernbanenettene er åpenbart ikke like, men de utfører den samme oppgaven, dog bruker det ene nettet noen flere eksekveringssteg enn det andre. Dersom det er slik at uansett hvordan man sender de samme togene inn i de to nettene vil de komme ut likt i de to nettene, kan man si at det er to modeller med forskjellig oppløsning av det samme stykket jernbane.

Hensynet til hva slags oppløsning man skal ha er det samme hensynet man må ta når man skal lage et rett linjestykke: hvor nøyaktig man skal kjenne et togs plassering, og hvor mange tog det skal være plass til langs

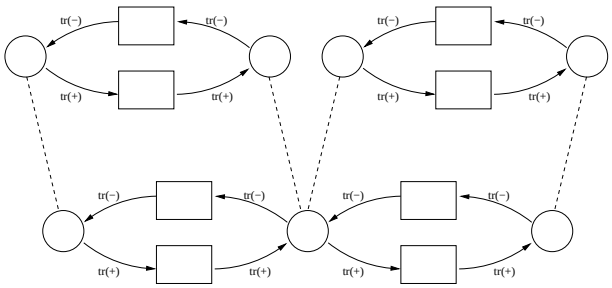
en gitt strekning. I teorien kunne man modellert enhver strekning som ett sted, men i praksis behøver man å kunne ha flere tog etterhverandre f.eks mellom to stasjoner. Samtidig er det ikke spesielt interressant å ha modellen for detaljert; det gir ikke nevneverdig mer informasjon å ha et sted for hver meter skinnegang, og implementasjonen av en slik modell ville vært enorm.

4.1.4 Sammenslåing

Man er kanskje ikke fornøyde med at nettene utfører samme oppgave, men med forskjellig antall eksekveringssteg. Det neste spørsmålet er således om man kan slå komponenter sammen slik at de blir identiske.

To linjestykker

I (22) presenteres det en måte å sette sammen komponentene til større nett, illustrert i figur 4.3.

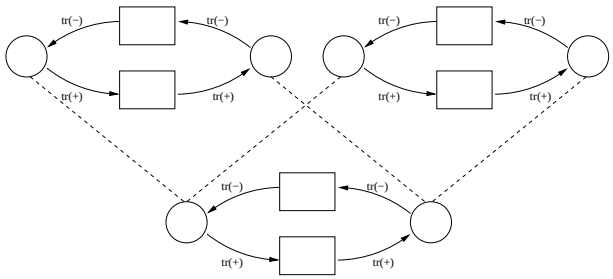


Figur 4.3: Sammenstilling av to linjestykker

I teorien kunne man slått denne veien sammen til ett veisegment slik som i figur 4.4

To linjestykker med informasjon

Hva skjer når man ønsker å slå sammen to linjestykker som har tilhørende informasjon? Man behøver en spesifikasjon som omhandler hva man gjør med informasjonen. Hvis man fremdeles antar at den tilhørende informasjonen til et linjestykke er linjestykkets lengde i meter, vil operasjonen som utføres på informasjonsbrikkene være å legge sammen verdiene. Sammenslåingen vil således bli slik som i figur 4.5.



Figur 4.4: Sammensetting av to linjestykker

Linjestykke og pens

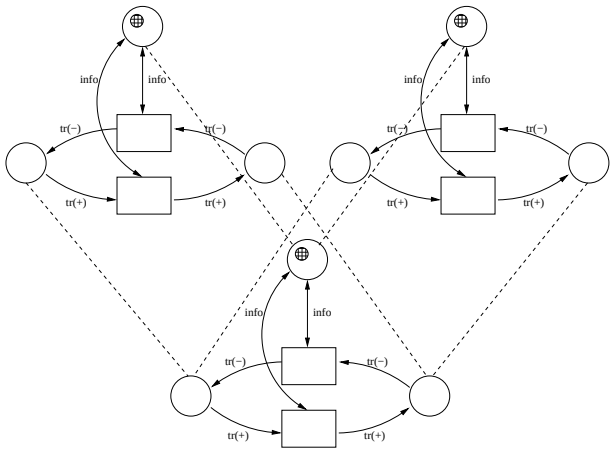
Videre kan man undersøke hvorvidt man kan slå sammen deler av to ikke-identiske komponenter. Dette kan gjøres som i figur 4.6.

Her ser man at alle egenskaper beholdes en gang. Alle egenskapene til linjestykket opptrer også i det aktuelle linjestykket i pensen, og behandles som om de er de samme. I tillegg beholder man de pens-spesifikke egenskapene. En anelse mer formelt kan man si at man kan sette merkelapper på alle steder, overganger og piler i de to utgangskomponentene slik som i 4.7 Merk at alle bestanddeler av A også finnes i B. Man tar deretter settet av alle bestanddelene, dvs at alle bestanddelene forekommer én gang i den nye komponenten.

Linjestykke og retningsendrer

Det hele blir en tanke mer komplisert når man skal slå sammen et linjestykke og en retningsendrer. Man ønsker åpenbart å beholde de syntaktiske egenskapene til retningsendreren. Uformelt tenker man seg at man lar retningsendrerens egenskaper gå foran der de to komponentenes tilsvarende elementer ikke er identiske. Det kan da se ut som i figur 4.8.

En antakeligvis mer intuituv måte å se på det på er at begge sider av det opprinnelige nettet har en retning som gir mening i forhold til det som ligger videre i begge retninger. Man kan forestille seg at man tar bort informasjon fra midten slik som i figur 4.9.



Figur 4.5: Sammensetting av to linjestykker med informasjon

Retningsendrere og pens

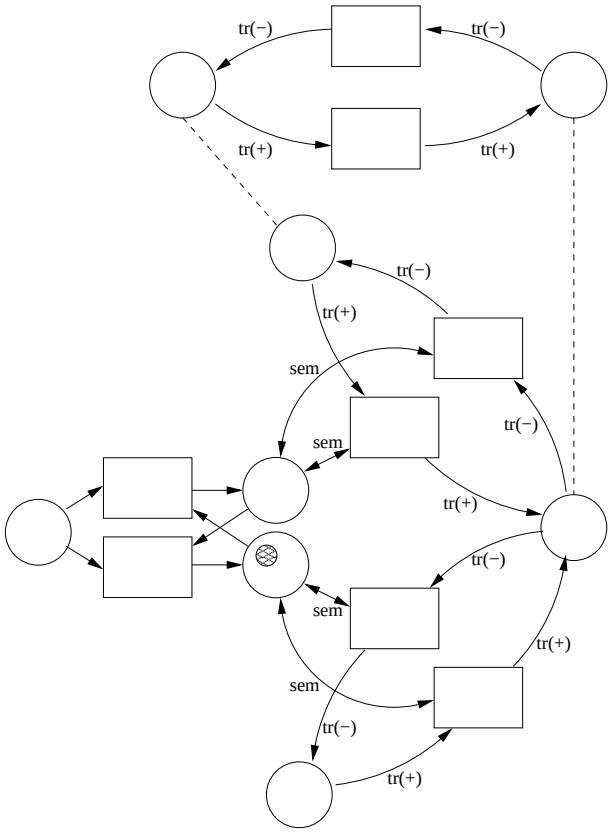
Det neste som er interressant er om man kan snu retningen til en pens. En pens har, slik den er presentert, én minusende og to plussender. Man kan se behovet for en pens med én plussende og to minusender. Uten en sammenslåing vil dette se ut som i figur 4.10, eller på spesifikasjonsnivå som i figur 4.11.

Man ser først på pensens minusside og *++retningsendreren*. For at man skal unngå at et tog tar en "u-sving" i pensen, må begge sporene fra pensen endres samtidig. Når man har gjort dette, kan man legge til retningsendrerene på pluss-sidene av pensen. Man ønsker naturlig nok å beholde endringene som nettopp er gjort. Dette kan man gjøre ved å behandle sammenslåingen som i avsnittet over. Dette er kanskje lettest å se på spesifikasjonsnivået (illustrert med figurene 4.11 til 4.15).

Komponenten man nå har kommet frem til, kan kalles *antipens*. Den eneste forskjellen mellom pens og antipens er retningen en togbrikke med en gitt retningsverdi flytter seg.

Det store tilfellet

På samme måte som vist i foregående avsnitt, kan man slå kryss sammen med linjestykker, retningsendrere eller en av sidene i en pens. Det burde nå være overkommelig å slå sammen figur 4.1 til figur 4.2.



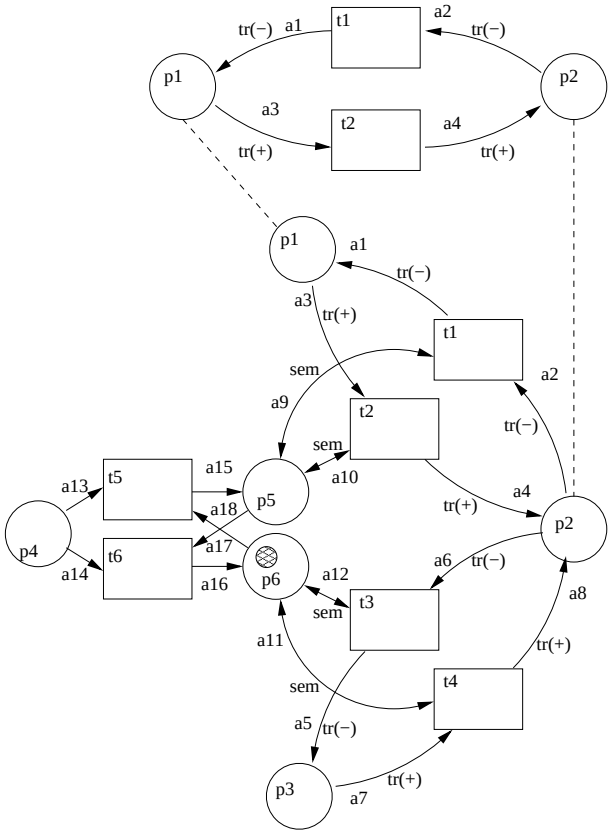
Figur 4.6: Sammensetting av linjestykke og pens

4.2 Kvarker

4.2.1 Presentasjon

I arbeidet med å slå sammen nukleonene ble det etterhvert åpenbart at man med hell kunne gå enda et abstraksjonsnivå ned. Man kunne beholde de strukturelle egenskapene som tomme skall, og putte mest mulig av syntaksen i nivået under. Komponentene i dette nivået har fått navnet **kvarker**[1] Det viser seg at man kun behøver to kvarker for hver forfiningsgrad: en plusskvark og en minuskvark. Så lenge man er litt taktisk og forutseende når

[1]Tro mot fysikkens tradisjon med å sette nye navn på de stadig mindre byggestenene til det man en gang trodde var udelelig og minst.

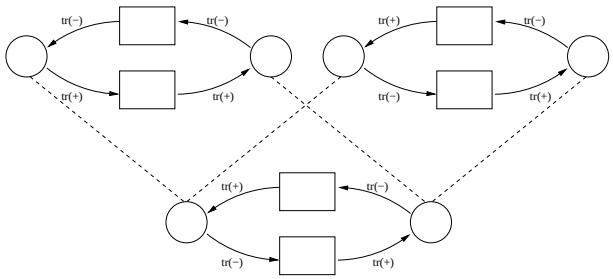


Figur 4.7: Sammensetting av linjestykke og pens med merkelapper

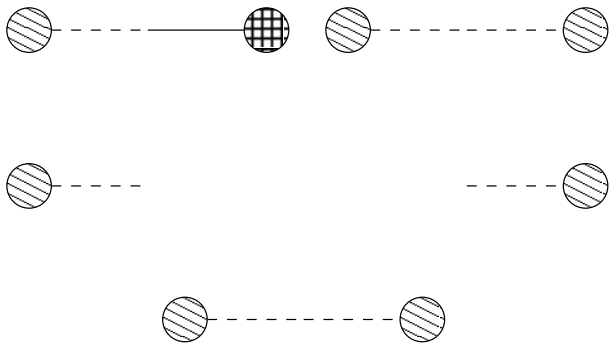
man lager de strukturelle skallene, kan man legge nær sagt alle forskjellene i de forskjellige forfiningsgradene i kvarkene.

De strukturelle skallene representerer de strukturelle egenskapene som er presentert i forrige kapittel. De semantiske egenskapene ved retning er representert i de to kvarkene. Slik unngår man de komponentene som kun er semantiske. For den strukturelle egenskapen **endestykke** var det imidlertid ingen praktisk nytte av å lage noe skall. Dette var fordi **endestykket**, i motsetning til de andre komponentene, ikke flyttet noen brikker, men istedenfor kun utførte operasjoner på dem[2]. Heldigvis taper man lite på dette. For konstruksjon av jernbanenettenes del er det ingen forskjell om en

[2]Endret bevegelsen til null



Figur 4.8: Sammensetting av linjestykke og retningsendrer

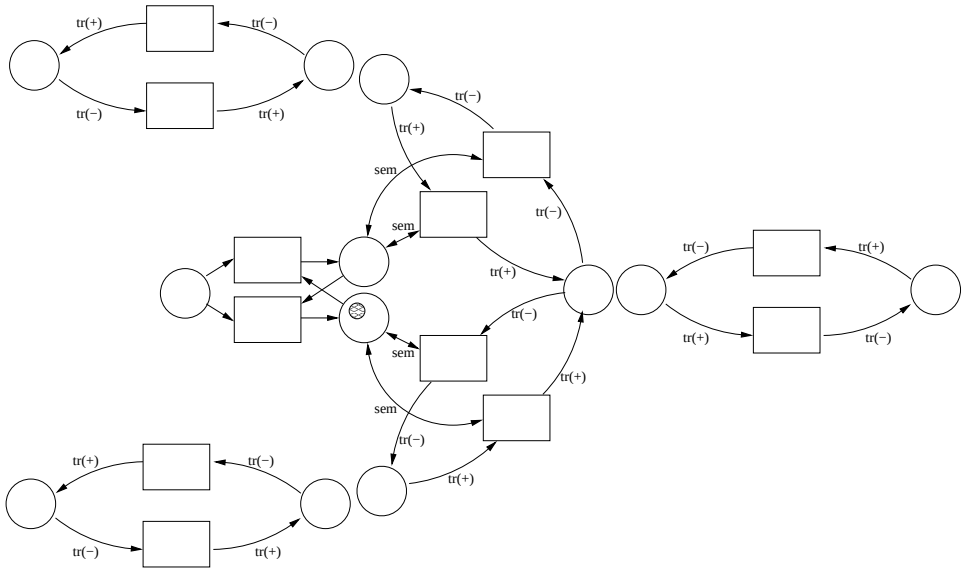


Figur 4.9: Sammensetting av linjestykke og retningsendrer II

komponent har vært skall som har blitt mettet først , altså at kvarkene er satt inn, eller om komponenten har vært et eget jernbanenett hele tiden. Endestykket er forøvrig det samme som i nukleonkomponentene og i de atomære komponentene.

Det man derfor må lage for hver forfiningsgrad er:

- plusskvark
- minuskvark
- endestykke



Figur 4.10: Pens med retningsendrere på alle kanter

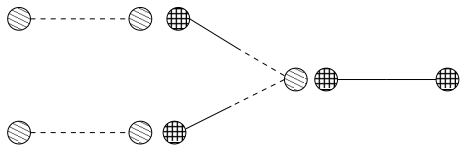
Fordeler

Det er tre klare fordeler med å gå ned til **kvark**-nivået. For det første er det klart arbeidsbesparende, da måten å flytte rundt på brikker i stor grad er lik for alle komponentene. Man får således unngått å gjøre det som er likt i alle komponentene mange ganger. For det andre får man redusert feilkildene ytterligere ved å ha færre syntakstunge komponenter å holde styr på. Sist, men ikke minst viser det seg at man slipper retningsendrerne, da muligheten for de retningsendrende egenskapene blir innebygd i alle komponenter, og ved at retningene blir implisitt utifra valget av kvarker.

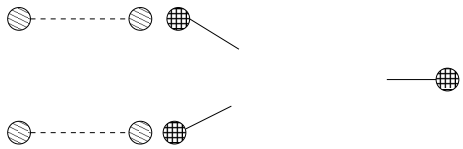
Figur 4.16 viser kvarker for noen av de forskjellige forfiningsgradene: De strukturelle skallene vises i figur 4.17.

4.2.2 Sammensetting

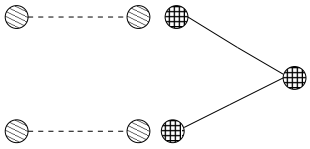
For å få sammensettingen til å gå enklest mulig, har overgangene i kvarkene fått et nummer og en bokstav i navnet. Overgangene er av to typer: L for linje og S for spesiell. For L-overgangene er numrene positive og negative slik at man kan organisere overgangene parvis slik at indeksene blir null summert, mens for S-overgangene er de kun positive. Steder kan også få



Figur 4.11: Pens med retningsendrere på alle kanter

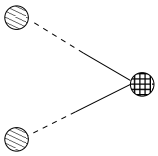


Figur 4.12: I ferd med å slå sammen pensen og pluss-pluss-retningsendreren

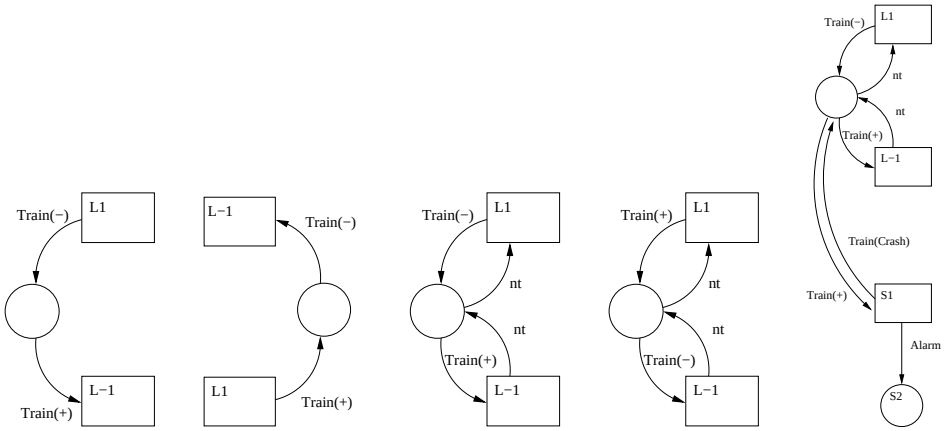


Figur 4.13: Ene siden sammenslått

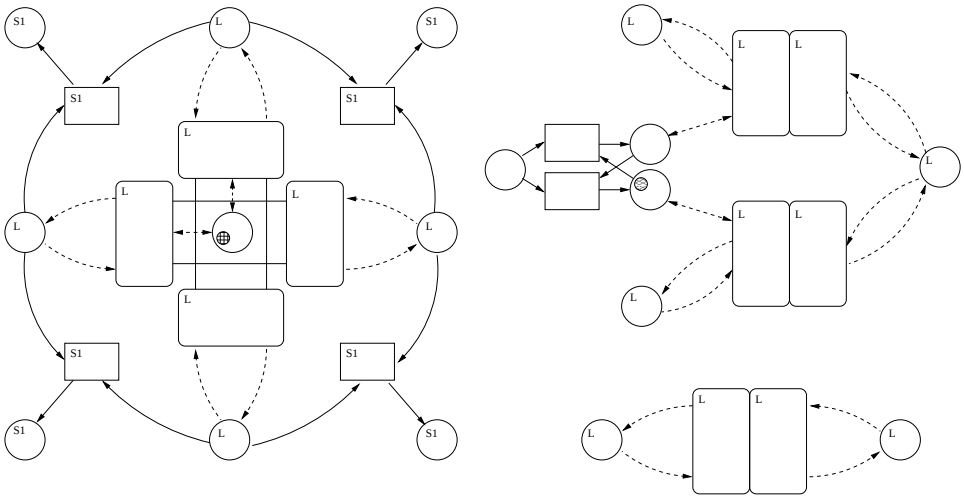
Figur 4.14: I ferd med å slå sammen pensen og minus-minus-retningsendreren



Figur 4.15: Antipens



Figur 4.16: Minus- og plusskvarker for *basic* og *safety*, samt minuskvark for *kollisjonsvar*

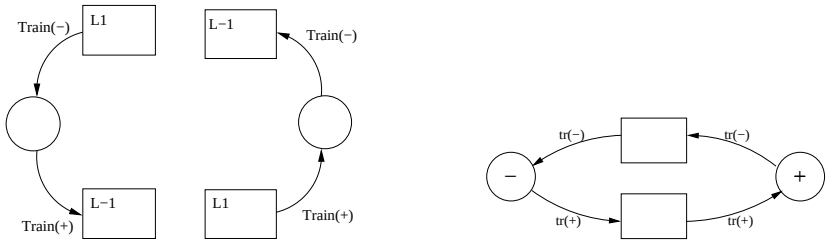


Figur 4.17: De strukturelle skallene: Kryss, pens og linje

S-benevning, og skal slås sammen på samme måte. Dette er for å kunne ha en algoritme for å sette sammen kvarkene. Eksempler på sammensetting er beskrevet nedenfor. Overgangene med positive numre leder *tog*-brikker til stedene, mens overgangene med negative numre leder togene vekk fra stedene. For at oppførselen skal bli slik som i de atomære komponentene de skal kunne bygges opp til, behøver man at overgangene fjerner en *tog*-brikke fra fra ett sted og setter dem ut et annet sted. Nummerering av overgangene sørger for dette ved at man vet hvilke overganger som skal fusjoneres. Det finnes utvidelser der overgangene ikke flytter tog, men prinsippet er det samme; man sørger for at hver overgang både kobles til der informasjonen kommer fra, og dit den skal.

Eksempel 1

Man ønsker å lage et linjestykke. Man ser først på skallet *linje* i figur 4.17, og ser at det består av to kvarker. I dette tilfellet er det ikke behov for noe annet enn et vanlig linjestykke med pluss i den ene enden og minus i den andre, så man bruker således en plusskvark og en minuskvark. Dernest må man avgjøre hvilken forfiningsgrad, altså kvarkgruppe, man behøver. Man kan starte med *basic*, illustrert i 4.18.

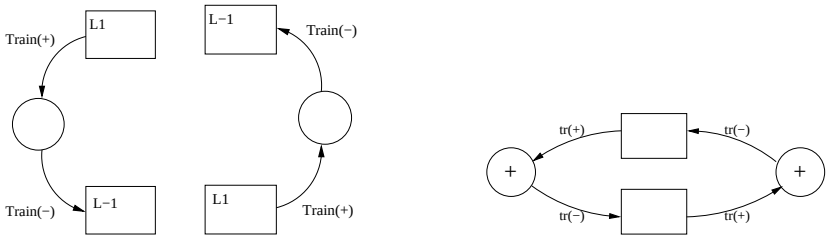


Figur 4.18: Sammensetting av kvarker til linjestykke

Man setter altså L-overgangene sammen slik at summen av indeksene blir null. Dersom man behøver pluss i begge ender, går det for seg som i 4.19:

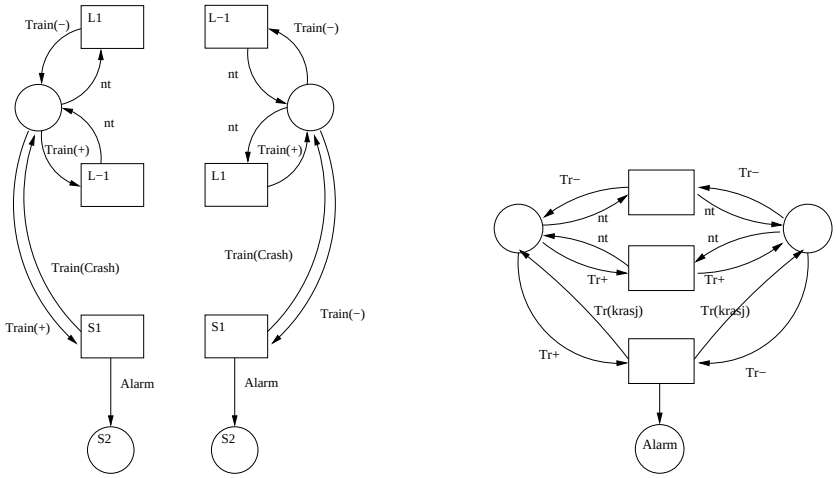
Eksempel 2

Man ønsker å lage et linjestykke med en forfining som innebærer spesialovergangene. Som over behøver man to kvarker, og man har sett at fremgangsmåten er lik uavhenging av hva som brukes av pluss- og minuskvarker.



Figur 4.19: Sammensetting av plusskvarker til retningsendrende linjestykke

Som eksempel på forfiningsgrad der S-overgangene brukes, benyttes de kollisjonsvare komponentene, og man bruker i eksempelet en plusskvark og en minuskvark. L-overgangene settes sammen som i forrige eksempel. S-overgangene skal samme nummer mot samme nummer. Fremgangsmåten er illustrert i 4.20



Figur 4.20: Sammensetting av kvarker til linjestykke med kollisjonsdeteksjon

4.2.3 Konstruksjon

Når man nå har vist hvordan man kan mette de strukturelle skallene med kvarker, og således med petrinet, kan man bruke disse skallene når man setter sammen større nett. Om man nå bruker alt som det her er kommet frem til, kan man relativt enkelt bygge vilkårlige jernbanenett.

Kapittel 5

Verktøyet

Det er laget et verktøy for å simulere jernbanenett modellert i petrinet. Det er fremstilt i forbindelse med arbeidet med de tre hovedoppgavene på dette temaet(22) (5). En beskrivelse av hvordan verktøyet virker finnes i (10). I arbeidet med denne oppgaven ble det laget et kontrollscript som gjorde det mulig for de forskjellige progammene å fungere sammen som ett system.

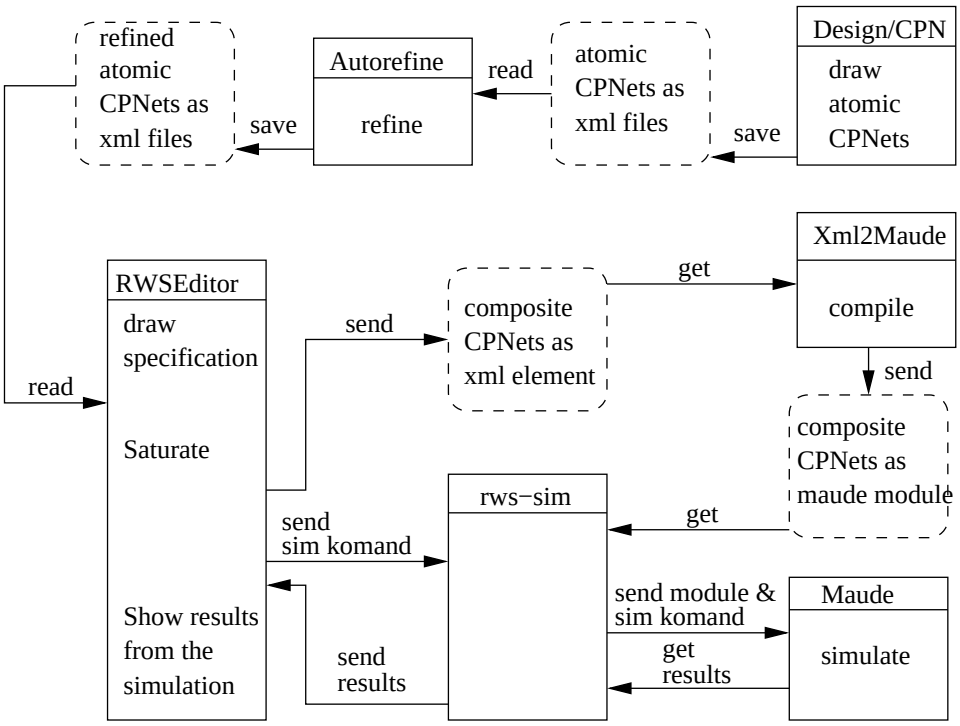
Scriptet sørget for kommunikasjon mellom prosessene. Det var programmet som startet det hele, og satte i gang de andre programmene, og "styrte" dem. I tillegg var det en parser, som sørget for å plukke ut det som var interressant fra de store mengdene med data som ble generert av maude-interpretet. Dette var nødvendig da man ofte måtte sende simuleringsforespørsler til maude-interpretet gjentatte ganger, og for hver gang undersøke om man var ferdig.

Valget av programmeringsspråk for denne oppgaven falt på perl(1). Dette fordi perl er godt egnet til å parse informasjon og til å håndtere informasjonsstrømmer, som er akkurat det man her behøver.

5.1 Arkitektur

Systemet består av mange forskjellige deler. Den delen av systemet som kjører kontinuerlig, og ikke bare startes for å løse enkeltoppgaver, og slutter etter det, er tredelt:

1. Javaprogram med brukergrensesnitt og kontroll over datastrukturen
2. maude-interpret som kjører selve beregningene
3. perl-script som sørger for smertefri kommunikasjon mellom de andre prosessene, inkludert parsing av output.



Figur 5.1: Arkitekturen til simuleringsverktøyet, hentet fra(10).

Et prosessflytskjema av hele systemet er vist i figur 5.1.

5.1.1 Interaksjon

Toveis kommunikasjon mellom prosesser der minst den ene prosessen er automatisert, er en nitidig affære. Grunnen til dette, er at det er viktig å være helt presis med hensyn på hvordan kommunikasjonen skal foregå, slik at man ikke risikerer 'deadlock', typisk i form av at begge prosessene sitter og lytter, og venter på at den andre prosessen skal skrive noe.

I dette systemet er det to tilfeller av toveis kommunikasjon mellom automatiserte prosesser. Det er kommunikasjonen mellom brukergrensesnittet (RWSEditor) og kontrollscriptet (rws-sim), og mellom rws-sim og maude-interpreteren. For å få kommunikasjonen korrekt, ble det laget protokoller for kommunikasjonen. Ettersom ingen av prosessene sender og lytter etter den samme informasjonen, ble det behov for fire protokoller.

1. RWSEditor til rws-sim

Dette er kommunikasjonen mellom brukergrensesnittet og kontrollscriptet. Her sendes datastrukturen, samt kommandoer som forteller hvordan simuleringen skal utføres, f.eks hvor lenge det skal simuleres, og hvor ofte mellomresultater skal returneres.

2. rws-sim til maude-interpreteren
   Her kommer maudemodulen og kommandoene. maude lytter og skriver kontinuerlig, så det siste som sendes her er en kommando som ber maude om å skrive at nå er den ferdig.

3. maude til rws-sim
   Her kommer alle resultatene og mellomresultatene.

4. rws-sim til RWSEditor
   Resultatene kommer i denne kommunikasjonsstrømmen, og blir benyttet i brukergrensesnittet.

5.2 Bruk

Verktøyet fungerer, og er benyttet til å simulere trafikk i Oslo Sporveiers nett. Data fra dette er publisert i (10).

Kapittel 6

Konklusjon

I denne oppgaven har man sett på hvordan petrinet kan brukes til å modellelere forskjellige egenskaper ved jernbanenett. Det har vist seg å være overkommelig å lage forskjellige modeller med grunnleggende jernbaneegenskaper.

Den første utfordringen var å bygge videre på arbeidet som er gjort ved å presentere måter å modellere spesifikke egenskaper. I oppgaven er det presentert tre forskjellige egenskaper og måter å representere disse ved jernbanekomponenter beskrevet i petrinet. Disse egenskapene må kunne sies å være representative utgangspunkt for mer komplekse egenskaper man måtte behøve dersom man skal lage en realistisk modell. Underveis er også noen problemer med de tidligere nettene adressert og løst.

Den andre utfordringen var å systematisere modellene. Her ble det først vist at noen av de presenterte komponentene kunne representeres ved hjelp av kombinasjoner av andre komponenter. Det ble foreslått et grunnleggende sett komponenter som kunne brukes til å presentere de andre komponentene. Teknikker for å gjøre dette ble beskrevet.

Videre ble det klart at det var mulig å finne ytterligere system. Det ble presentert en måte å minimalisere mengden petrinet som må lages for hver ny egenskap man ønsker å simulere med hensyn på. Hvorvidt dette er enklere eller ikke er vanskelig å si; det kreves muligens en større forståelse av systemet, men om man først har denne er systemet klarere og arbeidsmengden tilsynelatende mindre.

Tre lag abstraksjon (jernbanenettverk satt sammen av et lite sett modulære komponenter, som igjen er satt sammen av en struktur og en innholdsbit, hvor man i de fleste tilfeller kun behøver å lage de to innholdsbitene på nytt hver gang man lager en ny forfiningsgrad) ser ut

til å være en enkel måte å legge til nye egenskaper å simulere.

6.1 Beslektet arbeid

Arbeidet i dette prosjektet er inspirert av Wil van der Aalsts synspunkter på arbeidsflyt (2) (3), hvor petrinet er brukt til å presist definere, simulere, kjøre og analysere arbeidsflyt. Et fremtidig mål er å gjøre det samme for jernbanesystemer, hvor man kan undersøke hvordan petrinet kan brukes til å berike programvareutviklingen. Det er publisert noen artikler som tar for seg modellering og analyse av isolerte fenomenter (21) (11) (12) (6) (17) (4) (20) (7). I dette prosjektet er det forsøkt å modellere helheten, og å gjøre dette nært opp til virkeligheten.

6.2 Videre arbeid/åpne problemstillinger

Det kan være spennende å fortsette modelleringen ved å prøve å løse faktiske problemer ved jernbanenett. For å få til dette må man identifisere problemer og utfordringer og prøve å finne ut hvilke forutsetninger som ligger til bunn for å beskrive disse. Deretter må man lage en modell og utvide det eksisterende verktøyet, eller lage et nytt, for å undersøke disse problemene. Flere av disse dette ligger innefor andre grener av databehandling.

Det vil også være nyttig å undersøke om de nevnte kvarkene tåler å representere slike mer komplekse komponenter, eller om de blir upraktiske å bruke for tilstrekkelig infløkte systemer.

Bibliografi

[1] http://www.perl.org/.

[2] Will M.P. van der Aalst. Modelling and Analyzing Workflow using a Petrinet based Approach. I G. De Michelis, C. Ellis og G. Memmi, redaktører, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, side 31–50, 1994.

[3] Will M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[4] W.M.P. van der Aalst og M.A. Odijk. Analysis of railway stations by means of interval timed coloured petri nets. *Real Time Systems*, 9(3):1–23, November 1995.

[5] Joakim Bjørk. Executing large scale colored petri nets by using maude. Hovedfagsoppgave, Department of Informatics, Universitetet i Oslo, 2006.

[6] D. Decknatel. Modelling Train Movement with Hybrid Petri Nets. I *FMRail Workshop 4*, 1999.

[7] Maria Pia Fanti, Alessandro Giua og Carla Seatzu. A Deadlock Prevention Method for Railway Networks using Monitors for Colored Petri Nets. I *Proc. 2003 IEEE Int. Conf. Systems, Man, and Cybernetics (Washington, D.C., USA)*, side 1866–1873, October 2003.

[8] A. M. Hagalisletto, J. Bjørk, I. C. Yu og P. Enger. Constructing and Refining Large Scale Railroad Models Represented by Petri Nets, 2006. Accepted for publication IEEE Transactions on Systems, Man and Cybernetics, Part C.

[9] A. M. Hagalisletto og I. C. Yu. Large scale construction of railroad models from specifications. I Wil Thissen, Peter Wieringa, Maja Pantic og Marcel Ludema, redaktører, *IEEE SMC'2004. Conference Proceedings 2004 Systems Man and Cybernetics*, side 6212 – 6219. IEEE, October 2004.

[10] Anders Moen Hagalisletto, Joakim Bjørk og Pål Enger. Large scale simulations of railroad nets. *Proceedings of the Fourth International Workshop on Modelling of Objects, Components and Agents*, side 45–64, june 2006.

[11] W. Hielscher, L. Urbszat, C. Reinke og W. Kluge. On Modelling Train Traffic in a Model Train System. I *Workshop and Tutorial on Practical Use of Coloured Petri Nets and Design/CPN, June 8-12, 1998, Aarhus Denmark*, 1998.

[12] M. Meyer zu Hörste. Modelling and Simulation of Train Control Systems using Petri Nets. I *FMRail Workshop 3*, 1999.

[13] Kurt Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, bind 1 av *EATCS, Monographs on Theoretical Computer Science*. Springer-Verlag", 1997. Basic Concepts.

[14] Kurt Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, bind 2 av *EATCS, Monographs on Theoretical Computer Science*. Springer-Verlag", 1997. Analysis Methods.

[15] Kurt Jensen. An Introduction to the Practical Use of Coloured Petri Nets. Obtained from `http://www.daimi.aau.dk/PetriNets/`, 2002.

[16] Kurt Jensen. A Short Introduction to Coloured Petri Nets. Obtained from `http://www.daimi.aau.dk/PetriNets/`, 2002.

[17] Gabrielle Malavasi og Stefano Ricci. Petri nets theory in the railway signalling models. I *FMRail Workshop 5*, 1999.

[18] Joern Pachl. *Railway Operation and Control*. "VTD Rail Publishing", 2002.

[19] Carl Adam Petri. Kommunikation mit automaten. Rapport Schriften des IIM Nr. 2, Bonn: Institut für Instrumentelle Mathematik, 1962.

[20] X. Ren og M. C. Zhou. Scheduling of Rail Operation: A Petri Net Approach. I *Proc. of 1995 IEEE Int. Conf. on Systems, Man, and Cybernetics, Vancouver, Canada, Vol. 4*, side 3087–3092, October 1995.

[21] L. Tang, T. Chen og J. Xiao. Analysis of the concurrent model of train station based on Petri net. I *Proc. 2000 Int. Workshop on Autonomous Decentralized Systems, 21-23 September 2000, Chengdu, China*, side 92–96, 2000.

[22] Ingrid Chieh Yu. A Layered Approach to Automatic Construction of Large Scale Petri Nets. Hovedfagsoppgave, University of Oslo, 2004.

Tillegg

Protokoller for rws-sim, med forklaringer.

Protokollene er beskrevet som standard BNF.

fra RWSEditor til rws-sim:

```
<term>
 <kommando>|<kode>
<kommando>
 "kommando" <kom>"\n"|"kommando" <tall> <tall>"\n"|"kommando" <tall> "nointer\n"|
 "kommando <tall> <tall> time\n"
<kom>
 "debug"|"nodebug"|"convert"|"maxstep"|"EOS"
<tall>
 [0-9]+
<kode>
 <String>

<term>*
"kommando EOS\n"
```

Kort forklart fungerer det slik at alt som ikke gjenkjennes som en kommando, antas å være kode. Kode skal sendes rett videre til maude-interpreteren. Alle kommandoer er en egen linje, som starter med tekststrengen 'kommando ' etterfulgt av nøkkelord, som forklart over. rws-sim lytter på denne kanalen til den får tekststrengen kommando EOS\n".

rws-sim til maude:

```
<String>*
'red "EOS" .\n'
```

Alt som kommer her er gyldig maude-syntaks. Det er enten moduler eller kommandoer som f.eks 'frew InitState .' eller 'continue 1 .'. Det siste som

sendes i en omgang er 'red EOS".\n', som får maude til å returnere 'result String: EOS"' når den er ferdig med å behandle alle beregningene den er bedt om å gjøre.

maude til rws-sim:

```
<String>*
'result String: "EOS"'
```

Det som kommer på denne kanalen er resultatmerkingene fra hver av kommandoene maude er bedt om å utføre.

rws-sim til RWSEditor:

```
<term>
 <kommando>|<kode>
<kommando>
 "kommando endstep\n"|"kommando endOfSimulation\n"
<kode>
 <String>


<term>*
"kommando endOfSimulation\n"
```

Her har rws-sim parset ut den informasjonen man har bedt om i brukergrensesnittet, og returnert den i en mer praktisk form. Hvis simuleringen inneholder flere steg som skal vises, adskilles de med kkommando endstep\n". Når simuleringen er ferdig, sendes kommando endOfSimulation\n".

Kontrollscriptet rws-sim

```
#!/store/bin/perl

# Import av moduler
use locale;
use POSIX qw(locale_h);
setlocale(LC_CTYPE, "no_NO.iso88591");
use FileHandle;
use IPC::Open2;
use Text::Balanced qw ( extract_delimited
                        extract_bracketed
```

```
                        extract_quotelike
                        extract_codeblock
                        extract_variable
                        extract_tagged
                        extract_multiple

                        gen_delimited_pat
                        gen_extract_tagged
);


# Deklarasjon av variable, samt åpning av kanaler
$sem = 1;
$found=0;
$runder = 0;
$fortsett = 0;
$filnavn;
$temp ="";
$debug = 0;
$allStat = 0;
$frekvens = 1;
$time = 0;
$sekunder = 0;
$maksteg = 0;
$stegteller = 0;

$ENV{TERM} = 'dumb';

$pid = open2(\*Reader, \*Writer, "maude -no-banner -no-wrap" ) or die;
$pid2 = open2(\*JReader, \*JWriter, "java -Xmx1000m RWSEditor" ) or die;


sub xml2maude{ # Funksjon som konverterer xml til maudekode
    my $komm = "cd xml2maude; java -Xmx1000m Xml2Maude ../".$filnavn;
    system($komm);
#    system("del ".$filnavn);

}

sub noksteg{  # Funksjon for å teste om vi har simulert lenge nok.
```

```
    if($makssteg==0) {return 0;}
    if($makssteg < $stegteller) {return 1;}
    else {$stegteller++; return 0;}
}

sub readall{  # Funksjon som leser inn når vi ikke bekymrer
              #oss om begrepet tid.

    while($sem==1 && noksteg()==0){
my $tekst = $temp;
$temp = <Reader>;
if($debug>=3) {
    print STDOUT "Fra maude 1 -> $temp\n"}

# Test på om det er siste linje fra maude.
if($temp =~ /^result String: "EOS"/) {$sem=0}

elsif($temp =~ /^result /) {
    print JWriter "kommando EndStep\n";
    if($debug>=2)
    {print STDOUT "Til RWSEditor -> kommando EndStep\n";}
}

# Funksjon som parser $tekst etter tog-tokenene.
@fields = extract_multiple
($tekst,
  [
  sub { extract_bracketed( $_[0] , '(")', qr/.*?(?=\( ?tr)/ ) }, ]);

# Løkke som plukker ut posisjonen til tog-tokenene
foreach(@fields) {
    if($debug>=4) {
print STDOUT "Feltene er -> $_\n"}
if (/^(?!\( ?tr)/) { s/.* ?[\{]?[\w]+-(\d+-\d+)\s*$/$1/; if($1!=""){
print JWriter "$1\n";
if($debug>=2) {
    print STDOUT "Til RWSEditor -> $1\n"}}}}
    }
$sem = 1;
print JWriter "kommando endOfSimulation\n";
```

```
if($debug>=2)
{print STDOUT "Til RWSEditor -> kommando endOfSimulation\n";}

}


sub readingtime{  # Funksjon som leser fra maude med "time"
    if($debug>=2) {print STDOUT "I tidsklemma\n";}


    my $sem=1;
    while($sekunder<=$runder && noksteg()==0){
#while($sem==1){
my $tekst = <Reader>;
#     $_ = $temp;
if($debug>=3) {
    print STDOUT "Fra maude 1 -> $tekst\n"}

# En løkke som setter sammen mange maudelinjer til en tekststreng
$found=0;
    while(!($tekst =~ /\} time\((\d+)\)/)&& $found==0){
$temp = <Reader>;
$temp =~ s/^\s*//;
$tekst = $tekst . $temp;
$tekst =~ s/\s*\n\s*/ /mg;


if($debug>=3) {
    print STDOUT "Fra maude 2 -> $temp\n"}

if($tekst =~ /\} ?time ?\( ?\d+ ?\d* ?\)/){$found=1;}

if($debug>=3) {
    $tekst =~ /(.{1,140})$/s;
    print STDOUT "Fra maude, hale -> $1\n"}
    }


# Her er to uttrykk som fjerner linjeskift, og eventuell påfølgende
# blanke, og etterpå fjerner "Maude>"
```

```
    $tekst =~ s/\s*\n\s*/ /g;
    $tekst =~ s/Maude> //g;
    $_ = $tekst;

    if(/\} time\((\d+)\)/){$sekunder=$1;}
    if($debug>=2) {print STDOUT "time er -> $sekunder\n";}


    print JWriter "kommando EndStep\n";
    if($debug>=2)
    {print STDOUT "Til RWSEditor -> kommando EndStep\n";}
    #}

    if($debug>=3) {
print STDOUT "Fra maude -> $tekst\n"}

    # Funksjon som parser $tekst etter tog-tokenene.
    @fields = extract_multiple($tekst,
[
sub { extract_bracketed( $_[0] , '(")', qr/.*?(?=\( ?tr)/ ) },
]);

    # Løkke som plukker ut posisjonen til tog-tokenene
    foreach(@fields) {
if($debug>=4) {
    print STDOUT "Feltene er -> $_\n"}
if (/^(?!\( ?tr)/) { s/.* ?[\{]?[\w]+-(\d+-\d+)\s*$/$1/; if($1!=""){
    print JWriter "$1\n";
    if($debug>=2) {
print STDOUT "Til RWSEditor -> $1\n"}}}}
#}
#$sem = 1;

    if($sekunder<=$runder){
print Writer " continue $frekvens .\n";
if($debug>=2)
{print STDOUT "Til maude ->  continue $frekvens .\n";}
    }
    }
    print JWriter "kommando endOfSimulation\n";
```

```
    if($debug>=2)
{print STDOUT "Til RWSEditor (time) -> kommando endOfSimulation\n";}
    $sekunder=0;
}


sub jreading{   # Funksjon som leser fra RWSEditor
    my $sem=1;
    $time=0;
    $makssteg = 0;
    while($sem==1){
my $a = <JReader>;
$_ = $a;
if($debug>=3) { print STDOUT "leser fra RWSEditor -> $a\n";}
if(/^kommando EOS/) {$sem=0; if($debug>=3) {print STDOUT "-- . $a";}}
elsif(/^kommando ([0-9]+) ([0-9]+) time/){
    $runder = $1; $frekvens= $2; $time = 1;}
elsif(/^kommando ([0-9]+) ([0-9]+)/){ $runder = $1; $frekvens= $2;}
elsif(/^kommando ([0-9]+) noInter/) { $runder = $1; $allStat = 0;}
# elsif(/^kommando ([0-9]+)\+/) { $runder = $1; $fortsett=1;}
elsif(/^kommando ([0-9]+)/) { $runder = $1; $allStat = 0;}
elsif(/^kommando debug (\d)/) {$debug=$1; print STDOUT "--Debugmode $1--\n"}
elsif(/^kommando nodebug/) {$debug=0; print STDOUT "--Debugmode off--\n"}
elsif(/^kommando convert (\w+\.xml)/) { $filnavn = $1;
       xml2maude;
     }
elsif(/^kommando maxstep (\d+)/) {$makssteg = $1;}
else {
    print Writer "$a\n";
    if($debug>=3) {print STDOUT "Til maude -> ".$a;}
}
    }
    # Start for timed
    # $time=1;
    if($time==1){
if($runder>=$frekvens) {
    print Writer "frew [$frekvens] initState .\n";
    if($debug>=2) {print STDOUT "frew [$frekvens] initState .\n";}
    $runder=$runder-$frekvens;
}
```

```
# Denne er muligens feil, avhengig av intensjon. Må se på.
else{
    print Writer "frew [$runder] initState .\n";
    if($debug>=2) {print STDOUT "frew [$runder] initState .\n";}
    $runder=0;
}
    }
    # Start for øvrige
    elsif($frekvens >= 1) {
if($runder>=$frekvens) {
    print Writer "frew [$frekvens] initState .\n";
    if($debug>=2) {print STDOUT "frew [$frekvens] initState .\n";}
    $runder=$runder-$frekvens;
}
else{
    print Writer "frew [$runder] initState .\n";
    if($debug>=2) {print STDOUT "frew [$runder] initState .\n";}
    $runder=0;
}

for(;$runder>$frekvens;$runder=$runder-$frekvens){
    print Writer "continue $frekvens .\n";
    if($debug>=2) {print STDOUT "continue $frekvens .\n";}
}
if($runder>0){
    print Writer "continue $runder .\n";
    if($debug>=2) {print STDOUT "continue $runder .\n";}
}
    }

    elsif($allStat == 0) {
print Writer "frew [$runder] initState .\n";
if($debug>=2) {print STDOUT "frew [$runder] initState .\n";}
$runder=0;
if($debug>=2) {print STDOUT "Dropper mellomregning\n"}
    }
    elsif($runder>0){  # Funksjon som utfører kommandoen continue n
if($fortsett==0){
    print Writer "frew [1] initState .\n";
    if($debug>=2) {
```

```
print STDOUT "Til maude -> frew [1] initState .\n";}
#     print Writer " red \"EndStep\" . \n";
    $runder--;
}
else{$fortsett = 0;}
for(;$runder>0;$runder--){
    print Writer " continue 1 .\n";
    if($debug>=2) {print STDOUT "Til maude ->  continue 1 .\n";}
#     print Writer " red \"EndStep\" . \n";
}
    }
    if($time!=1){
print Writer "red \"EOS\" . \n";
if($debug>=2) {print STDOUT "Til maude -> red \"EOS\" . \n";}
$sem = 1;
    }
}

# "main()" finnes ikke i perl, men du starter her.
while(1){
    $stegteller=0;
    jreading;
    if($time==1){readingtime;} else {readall;}
    if($debug>=3) {print STDOUT "--Trinn--\n";}
}
```