Intro AI Lefsa

Curriculum 2018

- Lecture notes
- Text material about multiagent systems and game theory from outside the R&N book
- Textbook. Russell and Norvik. Artificial Intelligence - A modern approach. 3. ed. Chapters 1- 12; chap 22.
    - Chapter 4 is only section 4.1, 4.3 and 4.4 - the rest is not in curriculum
    - Chapter 22: only the parts of the chapter that are presented in the lecture/slides are included. So, it is mainly slides and use Chapter 22 in the book as reference.
    - Chapter 11: 11.4 is not included in the curriculum

Lectures from YouTube
https://www.youtube.com/playlist?list=PLjTSKEJpqIeDrUYF7DKspT2r9H38vg5dC

Chapter 1 - Introduction

- Many definitions to what AI is
- Rationality - a system is rational if it does the "right thing" given what it knows
- Four approaches/definitions of AI

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

    - Acting humanly
        - The Turing Test (by Alan Turing, 1950) was designed to provide a definition to intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.
        - To do this, a computer needs to have these capabilities

- 1. Natural language processing - communicate in English
- 2. Knowledge representation - store what it knows or hears
- 3. Automated reasoning - use stored information to answer questions and draw new conclusions
- 4. Machine Learning - adapt new circumstances, detect patterns

- Total Turing Test
  - In addition to Turing Test, includes a video signal so the interrogator can test the subject's perceptual (sanser) abilities, as well as the opportunity to pass physical objects.
  - To pass the test the computer need
    - 5. Computer vision
    - 6. Robotics - move objects
- These six disciplines are still relevant today, even though the test itself is not much used

- Thinking humanly
  - Must know how humans think
    - Can do this through introspection, psychological experiments and observing the brain
  - Cognitive science
    - Brings together computer models from AI and experimental techniques from psychology to construct theories of the human mind
  - More about this in TDT4137
- Thinking rationally
  - That a computer could solve any problem by being logical in every choice
  - "Laws of thought"
  - Problem - Just because it is logical, doesn't mean it is intelligent
    - Big difference between solving something in theory than in practice
- Acting rationally - rational agent approach
  - Agent - something that acts
    - operate autonomously, perceive their environment, persist over a time, adapt to change, create and pursue goals
  - Rational agent
    - Acts to achieve the best outcome or the best expected outcome
  - Two advantages over the other approaches
    - More general than thinking rationally because correct inference is just one of several possible mechanisms for achieving rationality
    - More amenable to scientific development than approaches based on human behavior or thought
- Tarski's theory of reference
  - The theory of reference is about how to relate the objects in a logic to objects in the real world.

- Historien til kunstig intelligens
    - 1943 - Mcculloch & Pitts - boolsk krets-modell av hjernen
    - 1950 - Alan Turing
    - 1966-74 - Skuffelse, AI oppdager computational complexity
    - 1969-79 - Tidlig fase med Knowlegde Base systemer
    - 1980-88 - AI industry is boomin
    - 1988-93 - "AI winter"
    - 1995 - Agents everywhere
    - 2001 - Store datasett ble tilgjengelige, Human-AI tilbake
    - 2005-10 - AI er ikke satt pris på
- Good old fashioned AI - GOFAI
    - Denne approachen var lite robust når den ble brukt på real-world problems.
- Situated embodied AI - SEAI
    - Fokuserte på å ha en kropp/motorskills i et fysisk environment

Chapter 2 - Intelligent Agents

2.1 Agents and Environments

- Agent - Alt som kan bli sett på som at det oppfatter sitt miljø gjennom sensorer og gjør handlinger på miljøet gjennom actuators/aktuatorer
    - Relatert til mennesket - øyne er sensorer, armer er aktuatorer
- Percept - agentens perseptuelle input ved et gitt tidspunkt
- Percept-sekvens - historien over denne inputen over tid
- Agent function - en agents behaviour er beskrevet av agentfunksjonen som mapper en gitt percept-sekvens til en aksjon.
    - Det er en abstrakt matematisk beskrivelse
    - Agentfunksjonen for en kunstig agent vil bli implementert av et agentprogram
        - Agentprogram er implementasjonen som kjører på et fysisk system
- Vacum cleaner eksempelet
    - To lokasjoner A og B, støvsugeren kan sjekke om det er skittent der den er, hvis ikke bevege seg. Kan også velge å ikke gjøre noe.

2.2 Good Behaviour: The concept of rationality

- Performance measure - notasjon for ønskelig oppnåelse, en evaluering av miljøtilstandene
    - Finnes ingen fasitsvar på hvilken performance measure som er best, det avhenger av omgivelsene.
- Rasjonell - avhenger av
    - Performance measure er kriteriet for suksess
    - Agenten kjenner til nåværende miljø
    - Agenten vet hvilke handlinger den kan utføre
    - Kjenner til den perseptuelle sekvensen så langt
- Rasjonell agent

    - For hver perseptuelle sekvens, skal den rasjonelle agentene bestemme en handling som er forventet å maksimere dens performance measure, gitt info den har fra den perseptuelle sekvensen og den kunnskapen den sitter på
        - Perfect rational - Evnen å ta gode valg gitt informasjonen, kan likevel være et miljø der man kun har noe informasjon, men hvis man velger det riktige gitt informasjonen man har så er man perfect rational likevel
    - Er støvsugeren rasjonell?
        - Vi må definere performance measure - feks poeng for hver rene plass per tidssteg. Ut i fra dette er agenten rasjonell, men ikke perfekt
- Omniscience
    - Ikke det samme som rasjonell. En omniscient agent kjenner til utfallet av dens handlinger, og handler deretter. Umulighet i praksis
- Information Gather
    - Handlinger som modifiserer fremtiden blir av og til kalt for informasjonsinnhenting. Dette er en viktig del av å være rasjonell.
    - Vår definisjon krever at agenten kan ta til seg og lære av det den henter av informasjon
- Autonomi
    - At agenten belager seg på kunnskapen til designeren og ikke sine egne oppfattelser kaller vi mangel av autonomi
    - En rasjonell agent burde være autonomisk, feks burde støvsugeren vår lære å forutse hvor og når det blir støvete

2.3 - The nature of environments

- PEAS
    - Performance measure
    - Environment
    - Acutators
    - Sensors
- Faktisk eksempel for en automatisert taxi
    - P - trygg, rask, lovlig, komfort, profitt
    - E - gater, motorvei, trafikk, fotgjenger, vær
    - A - Styring, bremse, gass, tute, signaler
    - S - Video, kamera, GPS, motorsensorer, speedometer
- Fully observable vs partially observable
    - Fully - Hvis en agents sensorer gir agenten tilgang på den fullstendige tilgangen til miljøet. Dette er ønskelig fordi agenten slipper å vedlikeholde en indre tilstand som holder styr på omgivelsene.
    - Partially - Kan være delvis pga av støy eller inaccurate sensorer.
        - Også hvis sensorene gir info kun om deler av miljøet.
    - Hvis en agent ikke har sensorer, er den unobservable
- Single agent vs multiagent
    - Er det flere agenter i miljøet? Feks kryssord vs sjakk.
    - Vi regner det som multiagent hvis vår agents handlinger har noe å si for andre agenter sine handlinger

    - Competitive multiagent environment - sjakk, vil slå hverandre
    - Cooperative multiagent environment - kjører bil, vil ikke kræsje
- Deterministic vs stochastic
    - Deterministisk - Hvis neste tilstand er bestemt ut fra hva den nåværende tilstanden er. Har vi et deterministisk, fullt observerbart miljø, kan man se bort fra usikkerhet
    - Strategic - Hvis miljøet er deterministisk bortsett fra actions fra andre agenter, sier vi at den er strategisk
    - Stokastisk - Det er noe usikkerhet rundt utfallet av en action
    - Ikke-deterministiske miljøer - bare mulige utfall, ikke sannsynligheter. Presenterer større problemer for agentdesign
- Episodic vs sequential
    - Episodic - agentens opplevelse blir delt opp i episoder. Den ene episoden følger av hvilken handling agenten gjorde på den forrige.
        - Hver episode mottar agenten en persepsjon og gjør en enkelt handling
        - Episodene er uavhengige og valget i episoden avhenger kun av episoden selv
    - Sekvensiell - Det nåværende valget kan gå ut over alle fremtidige valg
- Static vs dynamic
    - Dynamisk - hvis miljøet kan endre seg mens en agent gjør sine vurderinger, da er det dynamisk for den agenten
    - Statisk - statiske miljøer er enklere da agenten ikke trenger å se på miljøet mens den bestemmer seg for handlinger, og trenger ikke tenke på tid.
    - Semi-dynamisk - hvis miljøet er statisk, men agentens performance score endres underveis
- Discrete vs continuous
    - Diskret - et endelig antall distinkte tilstander og persepsjoner/handlinger (feks sjakk).
    - Kontinuerlig - kontinuerlig tilstand og tid (feks taxidriver), der fart og lokasjon til taxi og andre kjøretøy har flere kontinuerlige verdier
- Known vs unknown
    - Handler mer om agenten eller utviklerens kunnskap enn om selve miljøet
    - Kjent - outcomes for alle handlinger er kjent.
    - Ukjent - kan ikke forutse alle outcomes, selv om du har full oversikt over miljøet. Dette betyr at agenten må lære fra sine handlinger, og hvordan miljøet responderer til disse
- Worst case environment
    - Et delvis observerbart, multiagent, stokastisk, sekvensiell, dynamisk, kontinuerlig og ukjent miljø

2.4 The structure of agents

- For å designe en agent trenger vi to ting - agentprogrammet og en arkitektur
    - Disse må passe sammen (programmet kan ikke be agent om å "se" uten at den har øyne)
- Agentprogrammer
    - Alle agentprogrammer i dette faget har samme skjelett/oppbygning

        - Tar inn det agenten oppfatter gjennom sensorer og returer en aksjon.
        - Skiller mellom agentprogram (tar inn en nåværende oppfatning og gjør en handling) og agentfunksjonen (baserer seg på hele historien av oppfatninger)
    - Beskriver agentprogrammer som enkle pseudokoder
- Simple reflex agent
    - Enkleste formen for agenter, velger aksjon på grunnlag av den nåværende oppfatningen og ignorerer all historie. Agenten vil bare fungere hvis det riktige valget kan bli tatt på nåværende persepsjon, altså bare hvis miljøet er fullt observerbart.
    - Har ikke noe minne
- Model-based reflex agent
    - Den mest effektive måten å håndtere delvis observerbare omgivelser på er at agenten holder styr på den delen som ikke er synlig.
        - Har en internal state som holder styr på dette
    - For å gjøre dette trenger vi to ting
        - Informasjon om hvordan omgivelsene endrer seg, uavhengig av agenten
        - Informasjon om hvordan agentens handlinger vil påvirke omgivelsene
- Goal-based agents
    - Det holder ikke alltid å kjenne til omgivelsene for å vite hva som er den beste handlinger, trenger også informasjon om hva handlingen vil oppnå
- Utility-based agents
    - Nyttebasert - mål holder ikke alltid (feks taxi, mange veier til målet, men noen er raskere og tryggere enn andre)
    - Mål skiller mellom gode og dårlige valg, med utility kan vi finne ut hvor gode de faktisk er. Kan si at agenten prøver å maksimere sin “lykke”.
    - Beregnes med en utility function
- Learning agents
    - Lærer av sine aksjoner, kan deles inn i fire konseptuelle komponenter
        - Learning element - Ansvarlig for å gjøre forbedringer
        - Performance element - Ansvarlig for å velge eksterne aksjoner
        - Critic - Gir feedback til learning element om hvordan agenten gjør det og bestemmer hvordan performance element burde modifiseres for å gjøre det bedre i fremtiden. Forteller agenten hvor godt den gjør det mtp en bestemt performance standard
        - Problem generator - ansvarlig for å foreslå handlinger som vil lede til nye og informative opplevelser

Chapter 3 - Solving problems by searching

3.1 Problem-solving agents

- Intelligente agenter skal maksimere dens performance measure. For å forenkle gir vi agenten et mål å strekke seg mot.
- Definerer først mål, deretter søker løsninger. Når løsning er funnet, formulerer vi et nytt mål.
    - Agenten kjører mål-sekvensen og ignorerer det den oppfatter på veien, fordi den allerede vet hva den vil
- Et problem kan bli definert gjennom 5 komponenter
    - Init-tilstanden - tilstanden agenten starter i
    - Actions - En beskrivelse av mulige handlinger tilgjengelige for agenten. Gitt en tilstand s, actions(s) returnerer settet av handlinger som kan gjøres i s.
    - Transition model - en beskrivelse av hva hver handling gjør
    - Goal test- avgjør om en tilstand oppfyller kravene til å være en måltilstand (feks er sjakkmatt en måltilstand i sjakk, måltesten vil da sjekke om hvert steg er målet for spillet, hvis ikke fortsetter spillet)
    - Path cost - funksjon som gir en numerisk kostnad til alle stier
- Abstraction
    - Prosessen å fjerne detaljer fra representasjonen.
    - I tillegg til å fjerne detaljer må man fjerne handlingen.
        - En kjørende handling kan ha mange konsekvenser

3.3 Searching for solutions

- Søkealgoritmer jobber med å vurderer mulige aksjonssekvenser, der en aksjonsrekke er en løsning.
- Essens i søk
    - Sjekk om roten er en løsning
    - Vurder ulike aksjoner gjennom å ekspandere den nåværende tilstanden
    - Følg et valg og legg de andre til siden enn så lenge
- Frontier
    - Settet av alle løvnoder du kan nå (Open list)
    - Prosessen å ekspandere noder fortsetter til vi enten finner en løsning, eller det ikke finnes flere tilstander å ekspandere
    - Foretrukket datastruktur er en queue, med
        - Empty(q) - returnerer true hvis køen er tom
        - Pop(q) - fjerner og returnerer første element
        - Insert(element, q) - setter inn element og returnerer resulterende kø
- Redundante stier
    - Gjentagende stier
    - Måten å unngå redundante stier er å huske på hvor vi har vært.
        - For å gjøre dette trenger vi en “closed list”, som husker de ekspanderte nodene
- Grafsøk (Graph-Search) er det samme som tresøk (Tree-Search), bare med historie, altså husker du hvor du har vært
- Infrastruktur i søkealgoritmer

    - Datastruktur som holder orden på søketreet som blir dannet
    - For hver node n i treet har vi en struktur med følgende komponenter
        - n.state - tilstanden til node n
        - n.parent - noden n i søketreet som genererte denne noden
        - n.action - aksjonen som var utført på foreldrenoden for å generere n
        - n.pathcost - kostnaden fra roten til noden, generert vha pekere til foreldre
- Kø-struktur
    - FIFO, LIFO og prioritetskø
        - Prioritetskø - pop'er elementet med den høyeste scoren, basert på en eller annen rangeringsfunksjon
- Measuring problem-solving performance
    - Completeness/fullførbarhet - er det garantert at algoritmen finner en løsning, når løsning finnes?
    - Optimalitet - Finner algoritmen den optimale løsingen?
    - Tidskompleksitet - Hvor lang tid tar det å finne løsning?
    - Minnekompleksitet - Hvor mye lagringsplass krever algoritmen for å utføre søket?
        - Tid- og minnekompleksitet er målt etter
            - b - maksimum branching factor for søketreet
            - d - dybden for minste-kost løsning
            - m - maks dybde for tilstandsrommet (kan være inf)

3.4 - Uninformed Search Strategies

- Uninformed søkestrategier har ingen tilleggsinformasjon som gjør det mulig å vite noe mer om tilstandene utover det som er presisert i problemet. Det eneste de kan gjøre er å generere successors og skille en måltilstand fra en ikke-måltilstand.
    - Kalles også Blind Search
- Bredde-først-søk
    - Ekspanderer alltid grunneste, FIFO-kø
    - Måltest kjøres når en node blir generert
    - Komplett - ja
    - Tid - O(b^d), som er dårlig
    - Minne - O(b^d), lagrer alle noder i minnet (største problemet)
    - Optimal - Ja, hvis alle paths har lik kostnad er denne fin for korteste vei
- Uniform-cost search
    - Utledning av BFS
        - Også specialcase av A* der h=0 for alle
    - Optimal med enhver step-cost funksjon
        - I stedet for å ekspandere den nærmeste (minst dype) noden, så ekspanderer man noden med lavest kostnad g(n).
        - Gjøres ved en prioritetskø på g
    - Måltest gjøres når en node er valgt for ekspansjon, fordi første genererte kode kan være på en suboptimal sti. I tillegg er det en test i tilfelle en bedre sti er funnet til en node på frontieren.
    - Komplett - Ja, hvis step cost >= e

    - Tid - Antall noder med g <= cost til optimal løsning, O(b^(roof(C*/e))) hvor C* er kost til optimal løsning
    - Minne - Antall noder med g <= cost til optimal løsning
    - Optimal - Ja, noder ekspanderes i økende orden av g(n)
- Dybde-først-søk
    - Ekspanderer alltid dypeste, LIFO-kø (stack)
    - Brukes til topologisk sortering (DAG) for korteste vei
    - En variant er backtracking search
        - Kun en etterfølger blir generert om gangen og en delvis ekspandert node husker hvilke noder den har utelatt.
    - Komplett - Nei, feiler ved loops og uendelig-dybde rom
        - Modifiser for å unngå repeterende states
    - Tid - O(b^m), dårlig hvis m er mye større enn d, men kan være raskere enn BFS også
    - Minne - O(bm) - lineært
    - Optimal - Nei
- Depth-limited search
    - Løser problemet med infinite state space
    - Legger til en dybdegrense l, og behandler alle noder på nivå l som om de ikke har noen etterfølgere.
    - Med kunnskap om problemet kan dette gjøre problemet enklere å løse
- Iterativ deepening DFS
    - En strategi ofte brukt sammen med DFS for å finne den beste dybdegrensen
    - Gjør dette ved å stadig pushe grensen, helt til mål er funnet
    - Tilstander blir generert gjentagende
    - Komplett - Ja
    - Tid - O(b^d)
    - Plass - O(bd)
    - Optimal - ja, hvis step cost = 1, alle har lik stikostnad
- Bidirectional search
    - Gjør to søk samtidig
        - Et fra init state og et fra goal state, og håper at de møtes i midten
    - Fungerer slik at måltest er byttet ut med en sjekk for å se om frontiers til de to søkene intersects (krysses)
    - Første løsning trenger nødvendigvis ikke å være optimal
    - Komplett - ja, hvis b er endelig og begge retninger bruker bredde først
    - Tid - O(b^d/2)
    - Minne - O(b^d/2)
    - Optimal - Ja, hvis alle stier er like og begge retninger bruker BFS

3.5 - Informed (heuristic) search strategies

- Strategier som bruker kunnskap om problemer for å løse mer effektivt
    - Bruker informasjonen gjennom søket, ikke bare informasjonen gitt i problemet
- Generell tilnærming - Best-first-search
    - Grafsøk der nodene blir valgt basert på en evalueringsfunksjon, f(n)

        - f(n) er kostnadsfunksjon, slik at noden med lavest evaluering blir ekspandert først
    - Implementert med prioritetskø
    - Bruker en heuristikk h(n) som estimerer billigste vei fra tilstanden til målet
        - h(n) tar en node som input, men skiller seg fra g(n) ved at den kun avhenger av tilstanden i noden.
    - Dette er den mest vanlige metoden for tilleggskunnskap
- I rutenett kan disse heuristikk-funksjonene brukes
    - Manhattan distance - Kan bare gå opp/ned/høyre/venstre
    - Euclidean distance - Lengden målt med linjal (skrått vil være feks sqrt(2))
- Greedy best-first-search
    - Forsøker å ekspandere noen som er nærmest målet, basert på at det trolig gir en løsning raskest.
    - Evaluerer kun ved bruk av heuristikk, f(n) = h(n)
    - Komplett - Nei, kan bli stuck i sykler, men er komplett for et endelig område med repeterende tilstandssjekk
    - Tid - O(b^d), man kan med en god heuristikk forbedres dramatisk
    - Plass - O(b^d), alle noder i minnet
    - Optimal - nei
- A* search
    - Ideen - unngå ekspanderende stier som allerede er dyre
    - Kombinerer g(n) og h(n), slik at f = g + h, der f(n) er totale kostnad
    - Betingelser for optimalitet
        - h(n) må være en admissible heuristikk, altså aldri overestimerer kostnaden til målet
            - Admissible heuristikk-funksjoner er optimister, fokuserer på beste scenario. Dette er typisk korteste vei i luftlinje (ignorer hindringer på veien)
        - Consistancy- h er konsistens hvis den for hver node n og hver etterførlger n', er den estimerte kostnaden for n større enn n'
        - h(n) <= c(n, n') + h(n')
    - Fremgangsmåte
        - A* starter med rotnoden og legger den til i closed liste (ekspanderte noder)
        - Alle barnnoder blir lagt til i open-liste
        - Hver node har en liste med foreldrenoder, så rot blir lagt til som forelder for alle barna. Siden dette er eneste forelder hittil er den best-parent
        - Velger ut noden med lavest f(n) fra open-settet.
        - Sjekker om denne noden er en løsning, hvis ikke fortsetter vi prosessen med å flytte noden fra open til closed, sjekker og legger til naboene i open-list (med mindre de er der fra før).
        - Nåværende node blir satt som parent
        - Hvis noden var i openlist fra før av, sammenligner vi g(n)-verdiene (basically f(n) fordi h(n) vil jo være lik for samme node) og eventuelt oppdaterer best-parent

        - Fortsetter slik til vi har en løsning, og finner optimal aksjonssekvens ved å følge best-parent tilbake til root
    - Komplett - Ja, med mindre det er uendelig mange noder
    - Tid - Eksponensiell, hvis heuristikk ikke informativ og alle stepcosts er like, er det samme som BFS
    - Minne - alle noder i minne
    - Optimal - Ja, siden h aldri overestimerer og vi alltid velger den med minst kostnad innebærer det at om vi finner en løsning, er den garantert optimal
    - A* tar stor plass
        - De ekspanderte nodene holdes i minnet, så man vil gå tom for minne før man går tom for tid. Skalerer derfor ikke særlig bra
- Memory-bounded heuristic search
    - Den enkleste måten å reduserer minnekravene til A* er å adaptere ideen av iterativ dybde. Kalles IDA*.
    - Forskjellen er at cutoff er f(n) i stedet for dybden
- Recursive best-first search
    - Enkel rekursiv algoritme som forsøker å herme etter operasjonene i en standard BFS, men på lineær plass.
    - Holder styr på den til nå beste alternative veien fra hvilken som helst node til den nåværende noden underveis

3.6 Heuristic functions

- Vi sier at en heuristikk-funksjon h1 dominerer en annen h2, hvis h1(n) >= h2(n) for alle n
    - Så lenge begge er tillatt, antar vi at h1 genererer færre noder enn h2, som gir et mer effektivt søk til målet
- Egenskaper til en god heuristikk
    - Admissible/overtakbar - aldri overestimerer kosten for å nå målet. Garanterer minimum cost path
    - Consistancy - kostnaden å bevege seg er høyere enn reduksjonen i heuristikken. Endelig estimerte kost synker ikke
        - Consistent hvis (trekantregel) h(n) <= c(n, a, n') + h(n')
    - Hvis en heuristikk er consistant, så er den også admissble, men ikke omvendt
    - Bevis:
        - **Solution:** An admissible heuristic is one that never overestimates the cost to reach the goal. A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the setp cost of getting to n' plus th estimated cost of reaching the goal from n'.

          $$h(n) \leq c(n, a, n') + h(n'), \forall n, n', a$$

          Let $k(n)$ be the cost of the cheapest path from n to the goal node. We will prove by *induction on the number of steps to the goal* that $h(n) \leq k(n)$.
          Base case: If there are 0 steps to the goal from node n, then n is a goal and therefore $h(n) = 0 \leq k(n)$.
          Induction step: If n is i steps away from the goal, there must exist some successor n' of n generated by some action a s.t. n is on the optimal path from n to the goal (via action a) and n is $i-1$ steps away from the goal. Therefore,

          $$h(n) \leq c(n, a, n) + h(n)$$

          But by the induction hypothesis, $h(n) \leq k(n)$. Therefore,

          $$h(n) \leq c(n, a, n) + k(n) = k(n)$$

          since n is on the optimal path from n to the goal via action a.

- Relaxed problem
    - Færre restriksjoner på aksjonene
    - Gjøre prosessen å velge heuristikk enklere

Chapter 4 - Beyond classical search

4.1 Local Search algorithms and optimization problems

- I optimaliseringsproblemer der path ikke er relevant, men goal state er det som er løsningen
- Local search
    - Local Search algoritmer opererer med en enkelt current node i stedet for flere paths og generelt sett beveger de seg kun til naboer for den gitte noden.
    - Bruker ikke heuristikker, men et konsept kalt objective functions - denne svarer på spørsmål om hvor optimal løsningen er
    - Local search algoritmer er ikke systematiske, men har to fordeler
        - 1. Bruker veldig lite minne - vanligvis en konstant mengde
        - 2. Kan ofte finne greie løsninger i store eller uendelige state spaces der systematiske algoritmer ikke kan brukes
    - Snakker mye om state-space landscape, som både har location og elevation (lokasjon og høyde). Høyden er enten heuristikk-cost eller objective function.
        - Målet er dermed å finne enten et globalt minimum (ved heuristikker - lavest cost) eller et globalt maximum (ved objective functions)
    - En komplett local search algoritme finner alltid et mål hvis det eksisterer, og en optimal algoritme finner alltid et globalt max/min
- Hill climbing
    - En algoritme som ofte kalles greedy local search, fordi den alltid velger neste state som den beste uten å tenke fremover.
    - Rask for smoothe landskap, men kan fort bli stuck i variert landskap
        - Blir stuck i local maxima, fordi da er ingen av nabostates bedre og den forblir der den står
        - Stuck i rigdes - se for deg 3D, der du går oppover et fjell langs en rygg, men punktene man sjekker er på hver sin side av ryggen, og ikke opp mot fjellet. Dermed tror den at den er på et maxpunkt og sjekker ikke mer
        - Stuck i plateux - kan være en flat topp eller en skulder, der landskapet er likt rundt deg. Kan komme seg ut av en skulder, men ikke et flatt local max
        - Kan tillatte "sideways" der man tar høyde for platå-skuldere, men kan også implementere et maksgrense på hvor mange sideways det er lov og ta, slik at man ikke blir stuck i en evig loop
    - Stochastic hill climbing
        - Velger random av uphill moves - sannsynligheten av seleksjon kan variere med brattheten til uphill-movet. Konvergerer som regel tregere enn de bratteste, men finner i noen tilfeller en bedre løsning
    - First-choice hill climbing

        - Implementerer stokastisk hill climbing ved å generere etterfølgere random helt til en av de genererte er bedre enn current state. Dette er en god strategi når en state har flere tusen etterfølgere
    - Random-restart hill climbing
        - Eneste som er komplett i forhold til de tidligere nevnt, fordi den uansett til slutt (worst case) vil generere en init state som er goal state.
        - Utfører en serie av hill-climbing søk fra random genererte init states, helt til et mål er funnet
- Simulated Annealing
    - Oppfører seg som hill climbing, men er ikke greedy, heller mer random
    - Er en stokastisk optimaliseringsmetode
    - Bruker en sannsynlighet for å ikke velge nabo med umiddelbar benefit.
    - Se for deg at du har et landskap med mange lokale minimum, og skal på en ball til å lange i det globale minimum. Når du lander i et minimum, rister du på landskapet slik at ballen flyttes til et annet
    - Bruker en temperatur T mellom 0 og 1 som starter ved å være høy, og som senkes gradvis
    - Innerste del av loopen i simulated annealing ligner på hillclimbing, men i stedet for å velge det beste movet, velger den et random move.
        - Hvis det random movet er bedre, godtas det. Hvis ikke godtar algoritmen det med en sannsynlighet mindre enn 1
        - Sannsynligheten synker eksponensielt med "dårligheten" til movet
        - Sannsynligheten går også ned når temperaturen synker
        - Dårlige moves er mer tillatt ved en høy temp/i starten
    - Hvis man senker T gradvis nok, vil algoritmen finne et globalt optimum med en sannsynlighet som nærmer seg 1
- Local beam search
    - Holder orden på k states i stedet for kun én
    - Begynner med k random generert states
        - Ved hvert steg genereres alle etterkommere til alle k states
        - Hvis noen av de er goal state, stopper algoritmen
        - Hvis ikke velger den de k beste etterkommerne fra den komplette lista og repeterer
    - Skiller seg fra random-restart hill climbing fordi disse kjører i parallell og gir informasjon til hverandre underveis, de kjører ikke uavhengig av andre
    - I en enkel form av algoritmen kan det virke som om det er lite mangfold blant de k states, fordi de fort kan bli konsentrert i små regioner. Da er det ikke noe bedre enn en litt "dyr" hill climbing
    - Stochastic beam search fikser dette
        - I stedet for å velge de k beste hver gang, velger den k random, med sannsynligheten for å velge en gitt etterfølger som en økende funksjon av sin verdi
- Genetic Algorithms
    - En genetisk algoritme er en variant av stokastisk beam seach hvor etterfølger til tilstandene er generert ved å kombinere to tilstander i steder for å modifisere én - en analogi til naturlig seleksjon

- Begynner med et sett av k random genererte tilstander, representert som bitstrenger - dette kalles populasjonen
- Hver state eller individual blir ratet etter fitnessfunksjonen, som returnerer høyere verdier for bedre tilstander.
- Etterfølgertilstanden er dermed generert ut i fra to random tilstander for å produsere et "barn".
  - Dette gjøres ved å velge et random crossoverpunkt x, der de x første bits er fra den ene forelderen, og resten fra den andre
  - Den nye staten lider av mutasjon

Chapter 5 - Adversarial search

Adversarial search - motstridende søk

5.1 Games

- Agenter kan være collaborative eller competitive
- Klasser av spill
  - Perfekt info: tilstanden til spillarena og andre spilleres beholdning er kjent til enhver tid
  - Imperfekt info: noe informasjon om arena eller spillere er uvisst
  - Deterministisk: Utfallet av enhver aksjon er sikker
  - Stokastisk/sjanse: Noen utfall har sannsynlige utspill, feks kaste terning

	Deterministic	Chance
Perfect information	Chss, checkers, go, othello	Backgammon monopoly
Imperfect information	Battleship Blind tic-tac-toe	Bridge, poker, scrabble

- Søketreet er forskjellig fra standard søk - her er et tre der hver spiller har kontroll/sin tur ved annenhver nivå
  - Spilleren vi vil at skal vinne starter spillet
  - Søket handler om å maksimere utility/nytten
  - Antar at motstander er unpredictable - løsningen er en strategi som spesifiserer et trekk for hvert mulige mottrekk

5.2 Optimal decisions in games

- Minimax adversarial search algorithm
  - Perfekt for deterministiske, perfekt-info games som kalles kombinatoriske spill
    - Dette er cero sum games, 2ply
  - Ideen - Velg trekk av posisjonen med høyeste minimax verdi = beste mulige payoff mot beste play
  - Minimax value er utility til MAX spilleren

- Algoritmen
  - Starter på toppen, som er spilleren vi vil skal vinne
  - Hvert trekk er noder i treet helt ned til terminal-noden, som er vinn eller tap posisjon
  - For hver løvnode blir det bestemt en evaluation function value
    - Setter vinn-situasjoner til høye verdier og tap-situasjoner til lave verdier. Uavgjort er ofte 0
  - Verdiene sendes opp igjen i treet slik at MAX får vite hvor den høyeste av sine løvnoder, og MIN får laveste
- Kun komplett hvis treet er endelig, og den er kun optimal mot en optimal motstander
- Treet er undersøkt som et dybde først søk, heuristikker er bare brukt på bunnen av treet
- Med m som dybde på tre og b er antall lovlige moves er kjøretid O(b^m)

5.3 Alpha-beta pruning

- Problemet med minimax er at antall spilltilstander den må sjekke i treet er eksponentielt voksende
- Kan innføre pruning - som gjør at vi slipper å sjekke alle mulige kombinasjoner
- Alpha beta pruning vil gi samme resultat som minimax, men pruner bort grener i treet som umulig kan endre utfallet og verdiene
  - Kan implementeres på trær av alle lengder, også mulig å prune vekk hele subtre
- Gjøres ved å gi verdier til nodene på veien, der alpha-verdi er beste for MAX og beta-verdi er beste for MIN
  - Alpha oppdateres når en større evaluering er returnert fra et MIN-barn-node. Brukt for å prune MIN noder
  - Beta oppdaterer når en mindre evaluering er returnert fra en MAX-barn-node. Brukt for å prune MAX noder
- Hvis man kunne fått til å alltid sjekke den beste/verste (alt ettersom du er MIN eller MAX) av terminaltilstandene nederst i treet, vil kjøretiden kunne bli O(b^m/2), som er en drastisk endring
  - Hvis etterfølgere blir sjekket ut i random rekkefølge, vil totalt antall noder som er eksaminert være O(b^3m/4) for moderate b.
- Kan også prøve å velge beste trekk ved å se beste moves fra tidligere.
  - Kan bruke iterativ deepening, som legger på litt på den eksponentielle tiden, men med en move-ordering på denne vil det gjøre opp for seg.
  - De beste trekkene kalles killer moves, og det å prøve disse først kalles the killer move heuristic
- I mange spill kan det oppstå transpositions - forskjellige permutasjoner av trekksekvenser som ender opp i samme posisjon
  - Kan dermed lagre en ny posisjon i en hash tabell første gang den oppdages, slik at man kan slå opp i den neste gang og ikke trenger å gjøre utregninger på nytt
    - Dette kalles en transposition table
    - Ligner på explore-list i graf-søk

        - Kan ha en god effekt, nesten doble dybden i søket for sjakk
        - Ikke hensiktsmessig å spare på alle, men er strategier for hvilke man skal velge her også

5.4 Imperfect real-time decisions

- Selv om alpha beta pruning gjør minimax mer effektivt, så må den likevel gå helt ned til terminalnodene
- Innfører derfor cutoff, der man slutter søket tidligere, men innfører en heuristisk evalutation function til tilstander i søket, som gjør ikke-terminaler om til terminaler
    - Bytter ut utility funksjonen med en heuristisk EVAL func, som estimerer posisjonens nytte, og bytter ut terminaltesten (om man er i mål eller ikke) med en cutoff-test som bestemmer når man skal implementere EVAL.
- Evaluation functions
    - Returnerer et estimat av forventet nytte til spille fra en gitt posisjon
    - EVAL burde velge terminaltilstander slik som utility-funksjonen gjør
        - Tilstander som er vinnende blir evaluert bedre enn uavgjort og tapende
    - I tillegg må ikke utregningen ta for lang tid
    - For ikke-terminaltilstander burde eval func ha sammenheng med faktiske sjanser for å vinne
    - For sjakk er det typisk en lineær vektet sum av features

5.5 Stochastic games

- I spill der tilfeldighet eller sjanse er en del av utfallet (terning, backgammon) må man bruke chance nodes i minimax for å regne ut beste trekk
    - For disse nodene regner vi ut forventet verdi, som er summen av verdien til alle utspill, vektet med sannsynligheten for hver sjanse-aksjon
    - Chance-noder er representert som sirkler i minimax treet
- Expectiminimax gir perfect play
    - Minimax med chance noder

Chapter 6 - Constraint satisfaction problems

6.1 Defining constraint satisfaction problems

- Et problem av typen tilstand-søk som er definert av
    - X - et sett med variabler
    - D - et sett med domener, en per variable
        - {D1, ..., Di} der hver D er et sett av lovlige verdier for en variabel X
    - C - et sett med begrensninger som forteller hvilke kombinasjoner av verdier som er lovlige
        - Hver C består av et par (scope, rel), der scope er en tuppel av variabler som deltar i begrensningen og rel er en relasjon som definerer verdiene som disse variablene kan være
- For å løse en CSP må vi definere et tilstandsrom (state space) og notasjonen for løsningen
    - Hver state i en CSP er definert av en assignment av verdier til noen eller alle variablene

    - En assignment som bryter ingen constraints kalles consistent eller legal assignment
    - En complete assignment er en hvor hver variabel er assigned, og en løsning til CSP er konsistent
    - En partial assignment er en som gir verdier til kun noen av variablene
- Et problem er løst når hver variable har en verdi som tilfredsstiller alle begrensninger på variabelen
- Felles for alle CSP løsninger
    - Initial state: en tom assignment {}
    - Successor function: assign en verdi til en unassigned variabel som ikke er i konflikt med den nåværende assignment → fail hvis det ikke er noen lovlige assignment
    - Goal test: den gjeldende assignment er komplett
- Kan lage en constraint graf der noder er variabler og kanter er constraints
- De enkleste formene for CSP inneholder variabler som har diskret og endelige domener
    - Kan også ha uendelige diskrete domener, da er det ikke lenger mulig å beskrive contraints ved å nummerere alle lovlige kombinasjoner av verdiene
    - I stedet må en constraint language bli brukt som forstår constraints direkte, uten å nummerere
    - Kan bevises at det ikke finnes algoritmer som løser generelle ikke-lineære constraints på integer variabler
- CSPer med kontinuerlige domener er vanlige i den virkelige verden og blir forsket på
    - Den mest kjente kategorien av kontinuerlige domene CSPer er lineær programmeringsproblemer, hvor constraints må være lineære likheter eller ulikheter. Disse kan løses i polynomisk tid
- Ulike type constraints
    - Unary
        - Involverer en singel variabel, feks variabel A må være grønn
    - Binary
        - Involverer et par med variabler, feks variabel A må være ulik variabel B
    - Higher order
        - 3 eller flere variabler, feks at alle variabler må ha forskjellige verdier
    - Preferences (soft constraints)
        - Disse kan bli brutt i en lovlig løsning, kan feks være “bedre enn” constraints
        - CSPer med preferences kan bli løst med optimaliserings-søke-metoder, da kalles det COP, constraint optimization problem
- En hver endelig-domene constraint kan bli redusert til et sett av binary constraints hvis nok hjelpe-variabler blir introdusert. Så vi kan transformere ethvert CSP til en med kun binære constraints
    - En annen måte å gå fra n-ary til binary er en dual graph transformasjon, der man lager en ny graf hvor det er én variabel for hver constraint i den originale

grafen, og en binær constraint for hvert par av constraint i den originale grafen som deler de samme variablene
    - Er likevel to grunner til at vi kanskje ønsker en global constraint i stedet for et sett med binære constraints
        - Det er enklere og unngår flere feil ved å skrive problemet som feks Alldiff
        - Det er mulig å designe special-purpose inference algorithms for globale constraints som ikke er tilgjengelige for et sett av primitive constraints

6.2 Constraint propagation: inference in CSPs

- I CSP kan en algoritme enten søke eller gjøre en spesifikk type inference som kalles constraint propagation
    - Bruke constrainten til å redusere antall lovlige verdier for en variabel, som igjen kan redusere lovlige verdier for en annen variabel, osv
    - Kan gjøres sammen med søk, eller før et søk starter, som en preprocessings-steg
        - Noen ganger kan preprocessingen løse hele problemet, uten å måtte utføre søket i det hele tatt
    - Nøkkelideen er local consistency - hvis vi behandler hver variabel som en node i en graf og hver binary constrant som en arc (kant), så vil prosessen av å tvinge lokal consistensy i hver del av grafen føre til inkonsistente verdier som elimineres gjennom grafen
- Ulike typer local consistency
    - Node consistency
        - En enkelt variabel er node-konsistent hvis alle variablene i denne variabelen sitt domene tilfredsstiller variabelen sin unary constraint.
            - Hvis A har D = {1,2,3} og det er en unary constraint som sier at A ikke kan ha even-numbered verdier, så kan vi gjøre den konsistent ved å fjerne 2
        - Sier at nettverket er node-konsistent hvis hver variabel i nettverket er node-konsistent
    - Arc consistency
        - En variabel i CSP er arc-konsistent hvis hver verdi i dens domene tilfredsstiller variabelens binary constraints.
            - X er arc consistent med Y hvis for hver verdi i Dx så er det en verdi i Dy som tilfredsstiller den binære constrainten på arc(x,y)
        - Et nettverk er arc-consistent hvis hver variabel er arc-consistent med alle andre variabler
        - Filtrerer på constraints med binære constraints
            - Filtrerer på en variabel av gangen, bruker en stack av constraints hvor vi begynner med den øverste og jobber oss nedover
    - Path consistency

        - Et to-variabel sett {X,Y} er path-consistent med respekt til en tredje variabel Z hvis for hver assignment {X=a, Y=b} konsistent med constraint {X,Y}, det er en assignment til Z som tilfredsstiller constraints på {X,Z} og {Y,Z}
        - Fokus på to eller flere constraints samtidig, så vi filtrere med mer enn én constraint av gangen (X>Z and Y<Z)
    - K-consistancy
        - En CSP er k-consistant hvis for hvert sett av k-1 variabler og for alle consistant assignment til disse variablene, en consistent verdi kan alltid bli assigned to enhver kth variabel
        - 1-consistancy sier at gitt et tomt sett, så kan vi gjøre hvert sett av en variabel consistent → Node consistancy
        - 2-consistancy er det samme som Arc consistancy
        - For binære constraint nettverk er 3-consistancy samme som path consistancy
- Mest populære algoritme for arc-consistency er AC-3
    - For å gjøre hver variabel arc-consistent, opprettholder AC-3 en kø av arcs som den skal vurdere
        - Ved start inneholder køen alle arcs i CSP
        - Pop-er en vilkårlig arc {X,Y} fra køen og gjør X arc-consistent med Y
            - Hvis dette gjør Dx uendret, kan algoritmen gå til neste arc
            - Hvis det endrer Dx (forminsker domenet), så legger vi til i køen alle arcs {Z,X} hvor Z er nabo til X
            - Må gjøre dette fordi endringen i Dx kan muliggjøre flere reduksjoner i Dz, selv om vi tidligere har sjekket Z
            - Hvis X er revidert ned til ingenting, vet vi at hele CSPen har ingen konsistent løsning, og AC-3 returnerer failure
            - Hvis ikke fortsetter vi å sjekke, ved å fjerne verdier fra domenene til variabler helt til det ikke er flere arcs igjen i køen
            - På det punktet står vi igjen med en CSP som er ekvivalent med den originale CSPen - begge med samme løsning - men den arc-consistent CSP vil i mange tilfeller være raskere å søke fordi variablene har mindre domener
    - Worst case kjøretid er O(cd^3) der c er binary constraints og d er største størrelse på domene for hver variabel

6.3 Backtracking search for CSPs

- En viktig egenskap for alle CSPer: commutativity - at resultat blir det samme uavhengig av rekkefølge
- Backtracking search er brukt for et dybde-først søk som velger verdier for en variabel av gangen og backtracker når en variabel har ingen lovlige verdier igjen å assigne
    - Holde orden på variablenes domene for å gjøre det enklere å detektere feil
    - Så når man plasserer en variabel på et brett, så må alle andre variablene sine domener bli sjekket
- Den velger en unassigned variabel, og prøver deretter alle verdiene i domenet til den variabelen etter tur, for å finne en løsning

    - Hvis en inkonsistens er funnet så returnerer backtracking failure, som får det forrige kallet til å prøve en annen verdi
- Siden representasjonen av CSPer er standardisert, trenger man ikke å supplere backtracking search med en domenespesifikk initial state, action function, transition model eller goal test
- Variabel og verdi ordering
    - Minimum Remaining Value (MRV) heuristikk
        - Velg variabelen med færrest gjenværende lovlige verdier, variabler med minst domene
    - Degree heuristics
        - Velger variabelen som er involvert i flest antall constraints, et bra valg når MRV gir uavgjort
    - Least constraining value
        - Foretrekker verdien som utelukker færrest valg for gjenværende variabler i constraint grafen
- Innflettede søk og inference
    - Forward checking
        - Holder orden på gjenværende lovlige verdier for unassigned variabler. Terminerer når en variabel ikke har noen lovlige variabler
        - Når en variabel er assigned, etablerer forward checking arc consistency for den
            - Siden forward checking bare gjør arc consistency, trenger vi ikke gjøre det hvis vi gjorde arc-cons før start
    - Selv om forward checking oppdager mange inkonsistenser, oppdager den ikke alle, den ser ikke langt nok frem
    - Maintaining Arc Consistency (MAC) gjør dette
        - Når en variabel Xi er assigned, kalles AC-3
            - I stedet for å bruke en kø for alle arcs i CSP, starter vi kun med arcs (Xj, Xi) for alle Xj som er unassigned variabler som er naboer til Xi
            - Derfra gjør AC-3 contraint propagation som vanlig
    - Intelligent backtracking
        - I stedet for å backtracke til foregående variabel og prøve en annen verdi for den ved failure, kan man backtracke til en variabel som prøver å fikse problemet - en variabel som er ansvarlig for å gjøre en av de mulige verdiene til X umulige.
        - Holder orden på et sett av assignments som er i conflict med en verdi av X, kalles conflict set
        - Backjumping-metoden backtracker til en mest recent assignment i conflict settet

6.4 Local search for CSP

- Når algoritmen starter er initial state fylt ut og søket endrer en variabel av gangen
- Poenget med algoritmen er å eliminere constraints som er foreløpig violated
- Hvilken variabel som endres er valgt random, men hvilken endring som gjøres er bestemt av heuristikken

- Heuristikken velger trekket som vil violate færrest constraints (min-conflicts) fra nåværende ståsted

6.5 The structure of problems

- Kompleksiteten av å løse en CSP er sterkt relatert til strukturen til sine constraints grafer
- Tre-strukturerte problemer kan løses i lineær tid
- Cutset conditioning kan redusere en generell CSP til en tre-strukturert og er ganske effektiv hvis en liten cutset kan finnes
- Tre-dekomposisjon-teknikker transformerer CSP til et tre av subproblmer og er effektivt hvis tre-bredden til constraint grafen er liten

Chapter 7 - Logical Agents

7.1 Knowledge-based agents

- Agenten baserer seg på kunnskap i Knowledge base (KB), som inneholder et sett med sentences på et knowledge representation language.
- Hver gang agentprogrammet kalles, gjør den tre ting
    - Tell - forteller KB hva den oppfatter
    - Ask - spør KB hvilken handling den bør gjøre
    - Tell - Forteller KB hvilken handling den velger, og utfører denne
- Declarative vs procedural system building
    - Om man skal gi en agent en tom KB og lære den ting underveis, eller gi den en ferdig programmert KB fra starten
- Inference engine
    - Sett med prosedyrer som bruker representational language for å interferere nye fakta fra allerede kjente fakta, så vel som å svare på ulike KB queries

7.2 The wumpus world

- Et slags spill:
    - Agenten befinner seg i en hule der det er pits og et monster - wumpus. Målet er å finne gullet i hulen og komme seg ut, uten å falle i en pit eller å bli drept av wumpus. Agenten kan drepe wumpus, men har bare én pil å gjøre det på
    - Diverse regler, actions og performance measure, bare les boka
- Environment
    - Deterministisk - utfallene er spesifisert
    - Statisk - ja, pits og wumpus flytter seg ikke
    - Diskret - ja
    - Single agent - ja, hvis wumpus ikke kan utføre handlinger
    - Fully observable - nei, bare lokal persepsjon
    - Episodic - nei, forrige handling går ut over nåværende og fremtidige handlinger. Å huske hva som er observert er veldig useful

7.3 Logic og 7.4 Propositional logic: a very simple logic og 7.5 Propositional theorem proving

- Logics - formelle språk som representerer informasjon
- Syntax - definerer hvordan setninger gir mening

- Semantics - definerer meningen til en setning
- World - en mulig verden/modell er et komplett sett av sannhetsverdier til alle logiske variabler
- Entailment - At en setning følger logisk av en annen. Vi sier a |= b, a I hver modell der a er sant, er også b sant
    - En inference/slutning algoritme som uteleder bare entailements kalles sound eller truth-preserving
- Completeness - en slutningsalgoritme er complete hvis den kan utlede enhver setning som er entailed
- Grounding - connection mellom logisk reasoning prosesser og the real environment som agenten eksisterer i
- Satisfiable - setning er tilfredsstillende hvis den er sann i (minst) én modell
- Unsatisfiable - setning ikke tilfredsstillende
- Valid - en setning er valid hvis den er sann for alle modeller
- Number of model - antall modeller som er sanne
- Horn clause - maks en variabel i en logisk setning er unnegated (positiv)
- Definite clauses - akkurat en variabel er unnegated
- Goal clasues - ingen variabler er unnegated

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \implies \beta) &\equiv (\neg\beta \implies \neg\alpha) && \text{contraposition} \\
(\alpha \implies \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \implies \beta) \wedge (\beta \implies \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

    - De Morgan og implication elimination er mye brukt
- Modus Ponens
    - I AI: forward chaining
    - P → Q, P |- Q
        - Hvis idag er tirsdag, John drar på jobb
        - I dag er tirsdag
        - Derfor drar John på jobb
- CNF - conjunction normal form
    - Skriv om alle logiske uttrykk til å være et produkt av summer, altså er det en AND av ORer: (A \/ B) /\ (-B \/ C \/ D) /\ (D \/ -E)
- Resolution
    - Først må gjøre alle fakta om til CNF og så ta negasjonen til det du vil bevise og kansellere ut fakta basert på hva du vet
    - Når du kan kansellere ut hva du ønsker å bevise med hva du kan utlede fra hva du vet så har du løst problemet

    - Kan være at man utleder fakta som kan være irrelevant for hva du prøver å bevise
- Completeness of resolution
    - Kalles ground resolution theorem
        - Hvis et sett med clauses er ikke tilfredsstilt, så inneholder resolution closure til disse clausene empty claus
- Forward chaining
    - Når egenskapene til en regel er tilfredsstilt, konkluder. Fortsett med regler som ble tilfredsstilt
    - Kjører i lineær tid
    - Er sound - hver slutning er en applikasjon på Modus Ponens
    - Complete - hver entailed atomisk setning vil bli avledet
    - Eksempel på data-driven reasoning - reasoning hvor fokuset starter med kjent data, som kan brukes av agenter til å trekke konklusjoner med innkommende persepsjoner, ofte uten en spesifikk query i tankene.
- Backward chaining (goal reduction)
    - Bevis det faktum som forekommer i konklusjonen av en regel, bevis egenskapene til regelen. Fortsett rekursivt.
    - Jobber seg fra målet og bakover til den finner kjente sannheter
    - Også en løsning i lineær tid, men ofte veldig mye mindre fordi den bare ser på de relevante faktaene i KB
    - En type goal-directed reasoning
- Begrensninger til propositional logic
    - Representerer utsagn om verden uten å reflektere strukturen og uten å modellere disse entitetene eksplisitt
        - Konsekvensen er at noe kunnskap er vanskelig eller umulig å encode i propositional logikk
    - To caser som er vanskelig å representere
        - Utsagn om like objekter og relasjoner
            - Må bli nummerert, og KB blir stor
        - Utsagn som refererer til grupper av objekter
            - Må bli nummerert, KB blir stor

Chapter 8 - First-order logic

- FOL er mer expressive enn propositional logic
    - Representerer objekter, deres egenskaper, relasjoner og utsagn om dem
    - Introduserer variabler som refererer til et vilkårlig objekt og kan bli substituert av et spesifikt objekt
    - Introduserer kvantifiserere som lar oss gjøre utsagn over grupper av objekter uten å måtte representere hver av de separat

8.1 Representation revisited

- Objects - ting, mennesker, hus, bil, teori, farge, gjenstand
- Relations - egenskaper feks, rød, rund.
    - N-ary relations, sønn av, inni, har størrelsen, kommer etter, mellom
- Functions - en mer enn, bestevenn, tredje element av

- Ontological commitment - store forskjellen mellom propositional logic og FOL ligger i ontologiske commitment gjort av hvert language, hva den antar om naturens virkelighet

8.2 Syntax and semantics of FOL

- Domain
    - Domenet til en modell er settet med objekter eller domain elements den inneholder.
        - Krav om at den må være nonempty, hvert ord må inneholde minst ett objekt
- Quantifiers
    - ∀ - For alle
        - Bruker → heller enn /\
    - ∃ - Det finnes (minst én)
        - Bruker /\ heller enn →

Chapter 9 - Inference in FOL

9.1 Propositional vs FO inference

- Universal Instantiation (UI) sier at vi kan inferere enhver setning hentet fra å substituere en ground term (en term uten variabler) for variabelen
- Existential Instantiation (EI) sier at variabler er byttet ut av en enkelt ny konstant symbol.
- Propositionalization
    - Hver KB i FOL kan bli propositionalized
    - Hver ground sentence bevarer entailment
- Propositinalization process
    - Instansiér universal quantification
    - Instansiér existential quantification
    - Bruk propositional logic inference metoder som nummerering, resolution etc

9.2 Unification and lifting

- Unification - matche en fri variabel med en konstant. Thetha = {x/John}
    - Kan også kalles substitusjon
- Standardizing apart
    - Eksempel: Knows (John, x) and Knows (x, OJ) trenger standardizing apart fordi x kan ikke være både John og OJ samtidig.
    - Skriver dermed feks Knows (John, x17) and Knows (x27, OJ)

9.3 Forward chaining

- Start med base facts og prøv å finne ut hvilke fakta som fører til hvilke regler
- Stop når du har nådd målet
- Optimalisering
    - Hvordan gjøre dette raskere?
    - Hvis det er mange objekter eid av Nono, og få missiler, er det bedre å lete gjennom missilene og sjekke hvilke som er Nono siden, enn å sjekke alle tingene til Nono og se hva som er missiler (sjekker gjennom en mindre liste, alltid lurt det ja)
    - Kalles conjunct ordering problem - NP hardt
        - Gode heuristikker er tilgjengelige
        - Feks MRV fra CSP
- Egenskaper
    - Sound og complete for FO bestemte clasues
    - Datalog = first-order bestemte clauses + no functions
    - FC terminerer for Datalog i endelig antall iterasjoner - på det meste p*n^k literaler
    - Kan hende ikke terminerer hvis a ikke er entailed
    - Uunngåelig - entailment med endelige clauses er semidecidable
- Ofte brukt i deductive databaser

9.4 Backward chaining

- Start med målet og jobb deg nedover

- Legg til regler på målet og bruk unification for å binde variabler til å bevise målet
- Når vi legger til regler får vi subproblemer om må prøve å løse disse
- Hvis vi kan løse subproblemene så kan vi løse hele problemet (basically DFS)
- Incomplete pga uendelig løkker
    - Fikse ved å sjekke current goal mot hvert mål på stack
- Ineffektivt pga repeterende submål
    - Fiks ved å cache forrige resultat (bruker ekstra minne)
- Ofte brukt (uten forbedringene) for logisk programmering

9.5 Resolution

- Enhver setning i FOL kan bli konvertert til en indirekte lik CNF setning
- Konvertere til CNF
    - Eliminer → og <-->
    - Flytt negasjon innover
    - Standardiser variablene - hver quantifier burde ha ulike variabler
    - Hver eksistensielle variabel er byttet ut av en Skolem funksjon av de omsluttede universelle kvantifiserte variablene (F(x), G(x)..)
    - Dropp universale quantifiers
    - Bruk lover for å få det på CNF form
- Skolemization
    - Prosessen ved å fjerne eksistensielle quantifiers ved eliminasjon
    - Oversett Ex P(x) til P(A), hvor A er en ny konstant
    - Argumentet i skolem function (feks F(x)) er alle universelle kvantifiserte variabler som forekommer i scopet til den eksistensielle kvantifisereren
    - Bytter basically ut kvantifisererne med en funksjon for verdien kvantifisereren gjaldt for

Chapter 10 - Classical Planning

10.1 Definition of classical planning

- Planning Domain Definition Language (PDDL)
    - Et språk som lar oss beskrive alle 4Tn^2 handlinger med ett action schema
    - Beskriver de fire tingene vi trenger for å definere et søkeproblem
        - Init state
        - Tilgjengelige handlinger i staten
        - Resultatet til den brukte handlingen
        - Goal test
    - Tillater negative literaler i preconditions og goals, heller ikke quantifiers
    - Baserer seg på STRIPS planning language - Stanford Research Institute Planning system
        - Tillater da ikke negative literaler i preconditions og goals
        - Domain - et sett av skrevne objekter, vanligvis representert som propositions
        - States - representert som første ordens predikater over objekter, sammensetning av ground atoms (variabelfrie)

        - Closed-word assumption - alt som ikke er oppgitt er false, de eneste objektene i verden er de som er definert
        - Operators/actions - Definert av preconditions - når kan handlingen bli gjort?
            - Effects - hva skjer etter handlingen
            - Representert in terms of
                - Add-list - liste av proposisjoner som blir true etter aksjon
                - Delete-list - som blir false etter aksjon
            - Goals - Kombinasjon av literaler
        - Fordeler
            - Siden den er restricted så kan inference bli gjort effektivt
            - Alle handlinger er additions eller deletions av propositions er i KB
        - Ulemper
            - Antar at bare et lite antall propositions vil endre for hver handling
            - Begrenset språk, så ikke brukbart til alle domains of interest
- PlanSAT - om det finnes en plan som løser planproblemet
- Bounded PlanSAT - om det finnes en løsning av lengden k eller mindre, dette kan brukes for å finne en optimal plan

10.2 Algorithms for planning as state-space search

- Planning problems kan løses som search problems
- Forward progression state-space search
    - Begynner med inital state og bruker problemets aksjoner til å søke fremover for å finne medlem av goal state settet
    - Må da jobbe med irrelevante aksjoner også, fordi vi vet ikke utfallet
- Backward regression relevant-states search
    - Begynner på goal states og jobber seg bakover helt til man finner initial state
    - Ser kun på de tilstandene som er relevante for goal/current state
    - Fungerer bare når vi vet hvordan vi finner forgjengere til tilstandene
- Ingen av disse er effektive uten en god heuristikk
    - Kan få en admissible heuristikk - en som ikke overestimerer
    - Feks ignore preconditions heuristikk - fjerner alle preconditions fra actions
    - Ignore delete lists heuristikk - fjerner delete-list fra alle aksjoner (altså fjerne alle negative literaler fra effects)
        - Fortsatt NP-Hard men kan finne en tilnærmet løsning i polynomisk tid med hill-climbing
    - Kan også bruke state abstraction, fjerner states for å gjøre det enklere
- Partial order planning
    - En plan som er et sett av handlinger og constraints på formen before(ai, aj) som sier at en handling forekommer før en annen
    - Søker i space of plans heller enn i state-space
    - Starter med en tom plan som kun inneholder goal og init state, uten handlinger mellom disse

        - Søket ser etter flaws i planen, og legger til noe i planen for å rette opp i flawen (hvis ingen endringer kan gjøres for å rette opp, backtrack og prøv noe annet)
        - En flaw er alt som gjør at partial plan ikke er en løsning.
    - Søk i plan-space og bruk least commitment når det er mulig.
        - Least commitment: bare ta valg som er relevante for å løse den nåværende delen av problemet
    - Search-space: sett av states i verden
    - Handlinger fører til overganger mellom states
    - Plan er en sti gjennom state-space
    - En plan er komplett iff hver precondition er oppnådd
        - En precondition er oppnådd iff det er effect av et tidligere steg og ingen mulig inngripende steg undoes/ugjør det.
    - Fordeler
        - Plan steg kan gjøres unordered
        - Håndterer samtidige planer
        - Least commitment kan lede til kortere søketider
        - Sound og complete
        - Produserer typisk en optimal plan
        - Typisk brukt i planer der det er viktig at mennesker forstår planen
    - Ulemper
        - Komplekse plan-operatorer leder til høy kostnad for å generere aksjoner
        - Stort search space pga samtidige handlinger
        - Har ingen eksplisitt representasjon av states i state-transition model

10.3 Planning graphs

- Rettet graf organisert i nivåer: S_0 er første nivået og er initial state
    - Initial state består av node som representerer hver fluent som holdes i s_0
    - Neste nivå A_0 består av noder som representerer hver ground aksjon som kan være aktuell for s_0
    - Alternerende nivåer S_i og A_i helt til vi kommer til terminal
    - En vedvarende handling skjer hvis ingen aksjoner negerer den
    - Gjensidige utlukkelseslinker (mutex - mutually exclusive) blir tegnet mellom tilstander når de er overfor hverandre eller når tilstander ikke dukker opp sammen uavhengig av valg av aksjon
        - Altså at bare en av de kan være resultatet
    - Hvis vi kommer til et punkt i grafer der to påfølgende nivåer er like, så har grafen level off
    - No-op: Vi kan si at en literal oppstår fordi en aksjon førte til det, men kan også si at en literal vedvarer hvis ingen aksjoner negerer den. Dette er vist ved en persistance action/no-op, som er tegnet som tomme bokser ved siden av handlingen
    - Partial order planning
        - Search space er sett av partially ordered plans
        - Plan operatorer fører til transitions

        - Mål er en legal plan
- To handlinger er mutually exclusive (mutex) på et visst punkt hvis det er ingen valid plan som kan inneholde begge på det tidspunktet
    - To handlinger på samme nivå kan være mutex pga
        - Inconsistent effects - en effekt av en handling negerer den andre
        - Inference - en negerer en precondition til den andre
        - Competing needs - handlingene har mutex preconditions
    - To proposisjoner på samme nivå er mutex hvis
        - Ene negerer den andre
        - Inconsistent support - alle måter å oppnå dem på er ved parvis mutex
- Observasjon
    - Antall prepositions øker alltid - fordi alle fra forrige nivå blir med videre
    - Antall handlinger øker alltid - fordi antall satisfied preconditions øker
    - Antall proposition som er mutex synker - fordi det er flere måter å oppnå samme propositions
    - Antall handlinger som er mutex synker - fordi det er synkende antall mutexes mellom propositions
- No-good: I tilfeller der extract-solutions failer med å finne en løsning for et sett av goals på et gitt nivå, så lagrer vi (level, goals) paret som no-good. Kan slå opp på dette og dermed direkte returnere failure uten å måtte søke gjennom de igjen.
- Valid plan
    - En valid plan er en subgraf av planning graf slik at
        - Alle goal propositions er tilfredsstilt i siste nivå
        - Ingen goal propositions er mutex
        - Aksjoner på samme nivå er ikke mutex
        - Hver aksjon sin precondition er gjort sann av planen
    - Basic algoritme (Graphplan algorithm)
        - Øk planning graf helt til alle goal propositions kan nåes og ikke mutex
        - Hvis grafen levels off først, return failure (da finnes ingen valid plan)
            - Levels off er når du påfølgende nivåer er identiske
        - Søk grafen for en løsning, CSP eller backward search
        - Hvis ingen valid plan er funnet, legg til et nivå og prøv igjen

Chapter 11 - Planning and acting in the real world

11.1 Time, schedules and resources

- Planning i den virkelige verden er mer kompleks
- Durations og resource constraints
    - Plan først, schedule senere, deles inn i
        - Planning phase - der aksjoner blir valgt, med noen constraints for rekkefølge for å møte målene for problemet
        - Schedule phase - temporal informasjon er lagt til planen for å forsikre seg om at det møter ressurs og deadline constraints
- Veldig store state spaces
    - Handlinger er ofte lavnivå
    - Dekomposér problem - hierarkisk planning

- Usikkerhet
    - Ikke-korrekte og ikke-komplett informasjon
    - Beredskaps-planning, re-planning
- Makespan - totale varighet for planen
- Aggregation - grupperer individuelle objekter inn i kvantiteter når objektene ikke kan skilles fra hverandre med hensyn til hensikten deres.
    - Essensielt for å redusere kompleksitet
- Critical path method
    - Den stien med lengst varighet
    - Å korte ned andre planer vil ikke endre noe for hele planen, da det står og faller på critical path
    - Aksjoner som ikke er på critical path får da et vindu på å utføre handlingene, beregnet med earliest possible start time ES og latest possible start time LS
        - LS - ES kalles slacket til en aksjon
        - Sammen utgjør LS og ES for alle aksjoner en schedule
    - En populær heuristikk er minimum slack algorithm
        - For hver iterasjon, schedule for den tidligste mulige starten hvilken som helst unscheduled aksjon som har alle sine forgjengere scheduled og har minst slack
        - Deretter oppdater ES og LS for hver aksjon som ble påvirket, og repeat dette
        - Ligner litt på MRV fra CSP

11.2 Hierarchical planning

- Lavnivå forklaringer fører til lange plans, men er alltid kjørbare
- Høynivå forklaringer er korte og forståelige plans, men kan være urealistiske
- Hierarkisk dekomposisjon - dele store oppgaver inn i mindre vha et hierarki
- Hierarchical Task Network (HTN)
    - To typer aksjoner
        - Primitive aksjoner - kan bli utført direkte, feks go-forward
        - Høynivå aksjoner HLA - kan raffineres til en sekvens av aksjoner, feks hent-posten
    - Refinements (raffineringer) gitt av en plan library
        - Som er definert av domene-eksperter
        - Lært gjennom erfaring
    - En HLA raffinering som bare inneholder primitive aksjoner kalles en implementasjon av HLAen
- Searching for løsninger
    - Starter med en top-level initial state - Act
        - Målet er å finne en implementasjon av Act som oppnår målet
    - Hierarkisk søk
        - Gjenta: Velg en HLA i den nåværende planen og bytt den ut med en av dens refinements, helt til planen når målet
- Downward refinement property
    - Hvis høynivå-planen påstår at den kommer til å nå goal, da finnes det en implementasjon som når goal

- Conditional planning
    - Plans inkluderer observation actions som inneholder informasjon
    - Subplaner er laget for hver beredskap (hvert mulige utfall av observation actions)
    - Dyr fordi den planlegger for mange usannsynlige utfall
- Demonic nondeterminism - hvor en motpart gjør valgene
- Angelic nondeterminism - Hvor agenten selv gjør valgene
- Angelic semantic
    - Reachable sets
        - Gitt en HLA h og en state s, vurder sett av states reachable fra s fra en hvilken som helst implementasjon av h: Reach(s,h)
        - Ideen er at agenten kan selv velge hvilket element av de reachable settene han skal ende opp i når den kjører HLAen
            - Dermed er en HLA med flere refinements mer kraftfull enn den samme HLAen med mindre refinements
    - Vanskelig å forutse hva en HLA vil gjøre med en bestemt state property
        - Bruker approksimasjoner for Reach
    - Optimistic description av en HLA Reach+: Overdriver settet av reachable states
        - Hvis Reach+ til planen ikke krysser målet, da funker ikke planen
    - Pessimistic description av en HLA Reach-: Underdriver settet av reachable states
        - Hvis Reach- krysser målet, så funker det definitivt
- Angelic search
    - Kalles med Act som initial plan
    - Kan oppdage plans som funker og ikke funker ved å sjekke kryssing av optimistisk og pessimistisk reachable sets mot målet
    - Når en abstrakt plan er funnet, dekomponerer algoritmen det originale problemet inn i subproblemer, en for hver del av planen
    - Initial state og goal state for hvert subproblem er ivaretatt ved å gå tilbake til en garantert reachable goal state gjennom action schema for hvert steg i planen

11.3 Planning and acting in nondeterministic domains

- Utvider planning til å dekke ikke-deterministiske miljøer - der utfall av aksjoner er uvisst
- Metoder for å håndtere delvis observerbare, ikke-deterministiske og ukjente miljøer inkluderer
    - Sensorless planning - miljøer uten observasjon
    - Contingency planning - Delvis observerbare og ikke-deterministiske miljøer
    - Online planning og replanning - Ukjente miljøer
        - Kan også måtte replanne hvis agenten sin modell av verden er feil
            - Kan ha en missing precondition - feks skal den slå inn en spiker, men vet ikke at den trenger en hammer for dette
        - Online agent har tre nivåer av valg for hvor forsiktig den skal monitore omgivelsene

- Action monitoring - før utførelse av handling, agenten verifiserer at alle preconditions holder
- Plan monitoring - før utførelse av handling, agenten verifiserer at den gjenstående planen fortsatt holder
    - Tillater serendipity - utilsiktet suksess
- Goal monitoring - sjekker om det er et bedre sett av mål som den kan prøve å oppnå

- Må bytte fra closed-world assumption (at bare de som er oppgitt å være sanne eksisterer, resten er false) til open-world assumption (der states om ikke er nevnt har en ukjent verdi, ikke false) for sensorless og partially observable planning.

Chapter 12 - Knowledge Representation

12.1 Ontological engineering

- Hvordan man representerer abstrakte konsepter som tid, hendelser, objekter og oppfatninger kalles ontologisk engineering
    - Alt i den virkelige verden, utenfor agenten
- Upper ontology - generelle rammeverk for konsepter
- Knowledge engineer - designer, bygger og tester ekspertsystemet
    - Bestemmer innholdet og organiserer kunnskapet som er nødvendig for domenespesifikke KB
    - Knowledge acquisition teknikker
        - Tradisjonell - jobbe sammen med menneskelige eksperter, manuelt arbeid
        - Nylig - Fra databaser, tekst og data - informasjon blir hentet ut og kunnskap oppdages derfra
- Domain expert - innehar ferdighetene og kunnskapen til å finne en løsning
- End user - det siste systemet burde møte kravene til brukeren
- General-purpose ontology burde være gjeldende i enhver special-purpose domain
    - Kan legge til domene-spesifikke aksiomer
- Declarative knowledge - Uttrykket i deklarative setninger
- Procedural knowledge - kunnskap som finnes i utføringen av en oppgave
- Domain knowledge - hva vi resonnerer om
- Strategic knowledge - hvordan vi resonnerer
- KR languages
    - Bruker språket for å representere kunnskapen i KB
    - The inference engine har evnen til å finne implisitt kunnskap ved argumentasjon på den eksplisitte kunnskapen. Bestemmer hva slags konklusjon som kan bli trukket
    - Requirements
        - Kraft, inferensiell tilstrekkelig, modifiserbar, forklarbar, lesbar
    - Noen typer KR språk
        - Logisk språk
        - Semantiske nettverk

    - Beskrivende logikk
    - Produksjonsregel

12.2 Categories and objects

- KR krever at objekter blir organisert i kategorier
    - Interaksjon på objektnivå
    - Argumentasjon/fornuft på kategorinivå
- Representere kategorier i FOL
    - Predikater og objekter - kan bruke predikat Example(x) eller "reify" (tingliggjøre, gjøre mer abstrakt) til et objekt Examples.
- Muliggjør reasoning gjennom inheritance
    - Arv basically, hvis vi sier at all Mat er spiselig, og dermed sier at Frukt er en underklasse av Mat, og Eple er en underklasse av Frukt igjen, så betyr det at alle epler er spiselige
- Kategorier spiller en rolle i spådommer rundt objekter
    - Oppfattet egenskap - Object som er oransje, ca 10cm diameter, lukter. Kan det spises?
    - I KB - Appelsin er en frukt, frukt er spiselig. Appelsin-kategori har samme egenskaper som det oppfattede objektet
    - Predict - Det kan spises
- Begrensninger til logiske representasjoner
    - Hva om et eple har en annen farge, eller lignende
        - Kan oppstå usikkerhet
- Disjoint - hvis to eller flere kategorier har ingen like medlemmer i seg
- Exhaustive decomposition - hvis vi har {Xi,..,Xn, Y} så må Y være av minst én Xi, feks ({Americans, Canadians, Mexicans}, NorthAmerican). Hvis du er nord-amerikaner, så er du også en av nasjonalitetene. Men du kan altså være både amerikaner og kanadier, pga delt statsborgerskap
- Partition - En exhaustive decomposition (som over), men der man kun kan være den ene. Altså hvis man er amerikaner, vil man i et partition-tilfelle ikke kunne være noe annet
    - Sagt på en annen måte: hvis man vet at personen hverken er amerikaner eller kanadier, kan man konkludere med at personen er meksikaner
- Kategorier som ikke har en klar definisjon i den virkelige verden kalles natural kind kategorier
    - Kan bruke Typical for dette: Typical(Tomatoes), x in Typical(Tomatoes) → Red(x) ^ Spherical(x)
- Physical composition
    - Feks PartOf, kan si at siden Norge er en del av Europa, som er en del av verden, så kan vi si at Norge er en del av verden
- Measurements
    - Objekter har høyde, masse, kostnad osv, dette kalles measures
    - Viktigste aspektet er at de kan settes i rekkefølge
- Objekter
    - Noen egenskaper er intrinsic - de tilhører selve substansen av objektet, heller enn selve objektet i sin helhet. Ting som tetthet, kokepunkt, smak, farge

- Extrinsic properties - vekt, lengde, shape

12.3 Events

- Event calculus
  - Adresserer hva som skjer under handlingen
    - At(Knut, NTNU) sier bare at han er på NTNU, men sier ikke noe om det er sant
    - Trenger et predikat T: T(At(Knut,NTNU), t)
  - Tidsintervaller
    - T(f, t) - f er true på tidspunkt t
    - Happens(e, i) e skjer over tidsintervallet i
    - Terminates(e, f, t) e får f til å opphøre ved tidspunkt t

12.4 Mental events and metal objects

- Hva med kunnskap om oppfatninger?
  - Krever en modell de mentale objektene i noens hode og prosessen å manupulere disse
  - Forhold mellom agenter og mentale objekter, propositional attitudes
    - Oppfatninger
    - "vet"
    - "vil"
  - En agent kan nå reason om oppfatninger til agenter

12.5 Reasoning systems for categories

- Semantic networks - Ross Quillian
  - Strukturen til nettverkene var bygget på labtesting av menneskelige svar til spørsmål som "Er en kanari en fugl", "kan en kanari fly?".
  - Quillian - mennesker organiserer kunnskap hierarkisk og lagrer informasjon på sitt mest abstrakte nivå
  - Reduserer størrelsen til KB, unngår inkonsistenser i oppdateringen
- Semantic network er en struktur som viser relasjonen mellom konsepter, instanser og verdier
  - Kunnskap er ikke en stor kolleksjon av mindre deler med kunnskap, men heller større biter som henger sammen
  - Konseptet kan være kategorier i sett av entiteter
    - Kan være medlemskap, subklasser og attributter
  - Lagrer vår kunnskap i en graf, med noder som representerer objekter i verden, og kanter representerer forhold mellom disse objektene
  - Bruker termene sin arv, som brukes når et objekt tilhører en klasse og dermed arver alle egenskaper til den klassen
  - Semantiske nettverk kan bli oversatt til frames.
    - Noder blir da frame names, links blir slot og noder i den andre enden blir slot value
    - Frames er en samling av attributter/slots og assosierte verdier og begrensninger til disse verdiene (F er false, T er True)
    - En slot består av en variabel med en verdi

        - Kan være declarative, men også procedural kunnskap gjennom demons (feks "age: (calculate-age-from-PID)")
    - I tilfeller der den expressive power er begrensende, så står mange semantiske nettverk for procedural attachment, som skal tette hullene
        - En teknikk hvor en query om en spesiell relasjon resulterer i et kall til en spesiell prosedyre som er designet for den relasjonen i stedet for en generell inference algoritme
    - En annen viktig ting ved semantiske nettverk er deres evne til å representere default values for kategoriene. Feks har et menneske to ben med mindre noe annet er presisert, dette blir ikke en kontradiksjon i semantiske nettverk. Da vil default bli overskrevet av den spesifikke verdien
- Multiple inheritance
    - Når et objekt kan tilhøre flere kategorier, eller når en kategori er et subsett av flere andre kategorier
- Description logics
    - Designet for å beskrive definisjoner og egenskaper til kategorier - en formalisering av semantiske nettverk
    - Subsumption - sjekke hvis en kategori er subset av en annen ved å sammenligne deres definisjoner
    - Classification - sjekke om et objekt tilhører en kategori
    - Consistency - et system sjekker også hvorvidt kategoriens medlemskap-kriterie er logisk tilfredsstillende

12.6 Reasoning with default information

- FOL er monotont, som er begrensende - settet av entailed setninger kan bare øke
- Nonmonotonic logic
    - Circumscription
        - Introdusert av McCarthy
        - Ideen er å spesifisere spesielle predikater som er antatt å være "så feil som mulig" - altså falsk for alle objekter bortsett fra de som vi vet er sanne
        - Bruker "ab" eller "abnormal" for å si at noe ikke stemmer
        - Tillater entailed sentences å fjernes etter at nye setninger blir lagt til i KB.
        - Bruker model preference
            - I en slik logikk er en setning entailed hvis den er sann i alle foretrukne modeller i KB, i motsetning til i klassisk logikk der kravet er sannhet for alle modellene
    - Default logic
        - Default regler som produserer betingede konklusjoner
        - Eksempel - Bird(x) : Flies(x) \ Flies(x)
            - Betyr at hvis Bird(x) er true og Flies(x) er konsistent med KB så kan Flies(x) bli konkludert av default
        - Default regel har 3 komponenter
            - Prerequisite (P) (forutsetning)
            - Justification (J)

        - Conclusion (C)
        - P: J1,...,Jn \ C
        - Hvis P og J1,...,Jn ikke kan bevises False, så kan konklusjonen C bli trukket
- Truth maintenance systems
    - Mange av inferences har default status heller enn å være absolutt sikker
    - Inferred facts kan være feile og trenger å bli trukket tilbake - belief revision
    - Anta KB inneholder setning P og vi vil kjøre Tell(KB, -P)
        - For å unngå kontradiksjon: Retract(KB, P)
        - Men hva med setninger som er inferred fra P?
        - Truth maintenance systems er designet for å håndtere disse komplikasjonene

Chapter 22 - Natural Language Processing

Anbefaler TDT4117 Informasjonsgjenfinning for dypere forståelse rundt temaet

22.1 Language Model

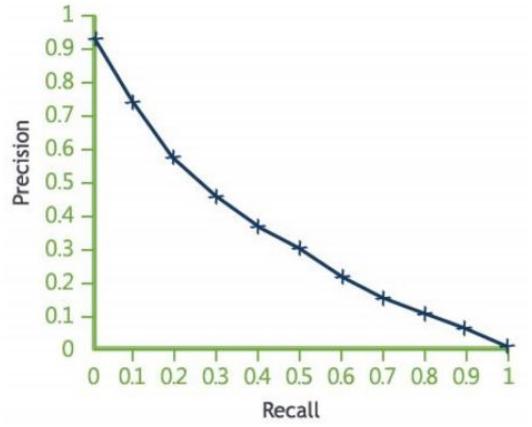
- Språkmodell som ser på sannsynlighetsfordelingen til en tekst, hvor sannsynlig er det at en random sekvens av en tekst er ord
- Naturlige språk er ambiguous (tvetydig), en type tekst kan betyr flere ting, slik at det er vanskelig for en datamaskin å skulle forstå.
- Språk endrer seg også hele tiden, slik at språkmodellene alltid vil være en tilnærming
- N-gram character model
    - Sannsynlighet at en sekvens av n karakterer vil dukke opp i et gitt dokument
    - Markov chain - sannsynligheten i steg k er bare avhengig av de karakterene som kom umiddelbart før, og ingen andre
    - Vi kaller kroppen til en tekst for corpus
    - Bra for language identification - gitt et tekst, hvilket natural language er det
- Smoothing n-grams
    - Ord som ikke forekommer i en språkmodell får lav sannsynlighet
    - Kan ikke garantere at alle ord som den leser er i språkmodellen, disse vil da få sannsynlighet = 0, som gjør at hele teksten får sanns = 0
    - Smoothing fikser dette ved å jevne ut - gir ukjente ord en lav sannsynlighet og jevner ut sannsynlighetene slik at det fortsatt går opp i 1.
- Model evaluation
    - Bruk treningssett for å velge beste modellen for nåværende case
- N-gram word models
    - Som for karakterer, bare for ord

22.2 Text classification

- Kategorisering - gitt en tekst, klassifiser hvilken klasse den tilhører
    - Kan feks oppdage spam
- Sentiment analysis - klassifisere feks en filmanalyse som positiv eller negativ

22.3 Information retrieval

- Kategorisert av
    - En corpus av dokumenter
    - Queries på et query-språk
    - Et resultatsett
    - En presentasjon av resultatsettet
- Oppgaven å finne dokumenter som er relevante for en brukers behov for informasjon
    - Beste eksemplet er søkemotorer
- Moderne IR for det meste basert på statistikk av ordforekomster (termer)
    - Hvis det viktige ordet fra query forekommer ofte i et dokument, så er det sannsynligvis relevant
- Websøk er en special case for IR
    - Kan utnytte hyperlink struktur (PageRank algoritme)
    - Kan utnytte statistikk på brukeroppførsel
- IR-scoring
    - Gitt dokument d og query q, så har man en scoring function S(d,q) = s, som er en numerisk relevanse-score
    - Term frequency: TF(t,d) - antall ganger term t finnes i dokument d
    - Stoppord-filter - fjerner ord som forekommer ofte, men som ikke gir oss noe informasjon (a, this, the, then, from, to, but)
    - Hvis et ord forekommer i mange dokumenter, så er det mest sannsynlig ikke relevant, altså er det ikke en diskriminerende faktor
        - IDF=log(D/d,t) der D er antall dokumenter og d,t er antall dokumenter med termen t i seg
        - $$S(d, D, q) = \sum_{t_i \in q} TF(t_i, d) * IDF(t_i, D)$$
    - Mange varianter for å regne TF-IDF, feks BM25
- Evaluering av IR systemer
    - Precision - hvor stor del av de returnerte dokumentene som er relevante
        - Regnes ut ved å ta antall relevante dokumenter returnert delt på antall dokumenter returnert totalt
        - Viktigste for websøk
    - Recall - hvor stor del av alle relevante dokumenter ble returnert
        - Regnes ut ved å ta antall relevante dokumenter returnert delt på totalt antall relevante dokumenter som finnes (også de som ikke ble returnert)
        - Viktig hvis feks du er en advokat og må få tilgang til alle relevante dokumenter som finnes for en rettssak
    - Vanligvis en tradeoff mellom P og R
        - Ved å returnere alle dokumenter så får du 100% recall (fordi du får alle relevante docs), men lav precision (fordi du får mange irrelevante også)
        - Motsatt, ved å returnere feks 1 dokument, så kan precision være 100% hvis dette er relevant, men recall være lav



        - F-score (F-measure)
            - Harmonisk middelverdi for P og R, som later til å være nærmere den laveste av de to
            - 2PR / (P+R)
        - Andre measures
            - R-precision, Mean Average Precision (MAP)
- Normalisering - lange vs korte dokumenter, man evaluerer de forskjellig. Normalisering retter opp dette, ved å jevne ut
- Stemming - Fjerner prefixes og suffixes, står igjen med stamordet
    - Feks connect -> kan være connected, connects, connection, connecting
    - Alle er ord med ish samme betydning, men skrives forskjellig
    - Stemming fjerner slik at ordet kun er connect og man kan telle forekomster av dette
- PageRank og HITS
    - PageRank ser ikke på user query, men på sider som blir pekt på av andre viktige sider. Disse er mest sannsynlig viktige. Utviklet av Google og brukes der
    - HITS ser på sider som blir pekt på av mange (authority), og også sider som peker på mange andre sider (hub), disse blir vurdert som viktige

22.4 Information extraction

- Oppgaven å automatisk hente ut strukturert informasjon fra ustrukturert eller semi-strukturert maskin-leselige dokumenter
    - Fra tekst til database
- Named Entity Recognition (NER) - finne "tingene" som er nevnt i teksten
    - Henter ut navn, priser, organisasjoner, datoer, forskjellige substantiv egentlig som plasseres i ulike "kategorier"
- Relation Extraction - finne binære relasjoner mellom to entiteter
    - Feks navn og land -> nasjonalitet, eller aliaser
    - Feks ser man en pris med NER, men hvilket objekt har den prisen? Forhold mellom ord
- Event extraction - Finne komplekse relasjoner mellom flere entiteter og/eller hendelser (events)
    - Feks et objekt som utfører en handling som fører til noe

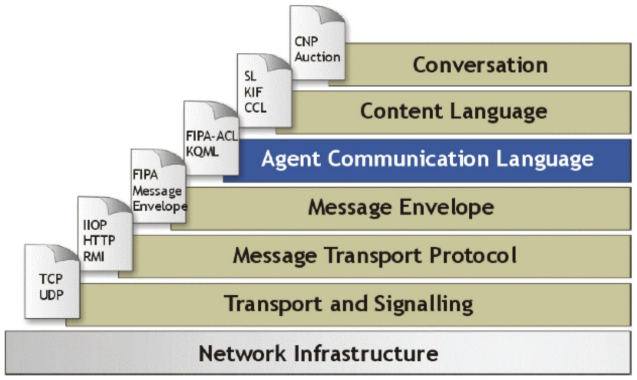
- Knowledge-based approach
  - Ekspert skriver extraction algorithm basert på kunnskap om språk - feks pattern matching
- Data-driven approach
  - Extraction algorithm kommer fra dataene (eksempler på relasjoner, eniteter), fra feks supervised machine learning
- Pattern matching
  - String matching er en type pattern matching
  - En regular expression (pattern) beskriver et sett av strenger
  - Eniteter og relasjon kan dermed bli beskrevet av regexps
  - Å matche disse regexps til tekst tillatter extraction
- Regexps
  -

Regexp	Matching strings
.*	Flemming, a, %(&%!
[A-Z].+	"Flemming", "XML"
[A-Z][a-z]+	"Flemming"
[A-Z][a-z]+ [A-Z][a-z]	"John Flemming"
(doctor |dr.)? [A-Z][a-z]+ [A-Z][a-z]	"dr. John Flemming", "doctor John Flemming"

- Data-driven
  - Maskinlæring og sannsynlighetsmodeller
    - Hidden Markov Models (HMM)
    - Conditional Random Fields (CRF)
  - Fordeler
    - Robust: tolererer støy, grasiøs nedbrytning
    - Bærbar til andre domener/språk uten ekspertkunnskap
  - Ulemper
    - Trening krever markerte eksempler → Merknad flaskehals
    - Ikke transparent

Multiagent-systems

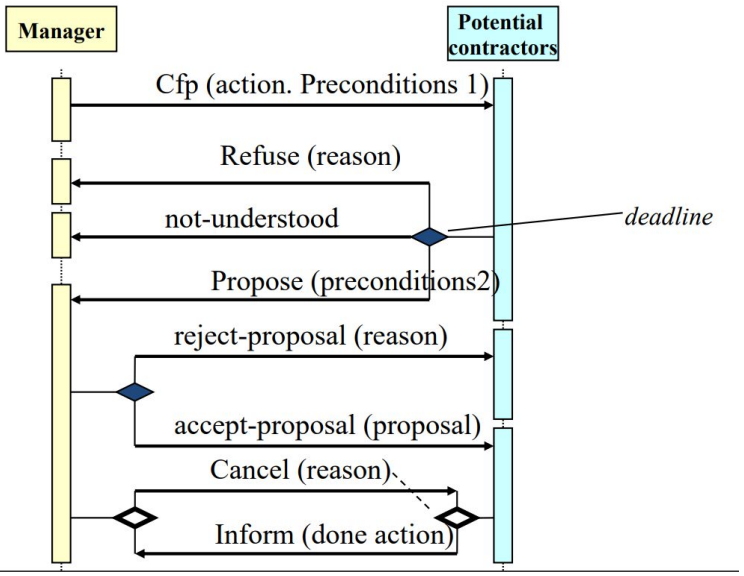
- Agent design (Micro)
  - Hvordan bygger vi agenter som er kapable til uavhengige, autonome handlinger for å suksessfull utføre oppgavene som deles ut til dem?
- Society design (Macro)
  - Typer og karakteristikker av interaksjoner mellom agenter, interaksjonsprotokoller
  - Samarbeid for å nå samme mål
  - Samarbeid i et samfunn av selv-interesserte (egen vinning) agenter?
  - Språk for agent-agent-kommunikasjon og agent-bruker kommunikasjon
  - Koordinasjon
- Modell for agent-basert system



- Communication as an Action
  - Handlinger har så langt endret miljøet, tilstanden til verden
  - Konsekvensene til kommunikative handlinger
    - Påvirke andre agenter, endre deres tilstand (oppfatninger, kunnskap, intensjoner)
    - Ingen agent kan tvinge en annen agent til å gjøre noe eller endre data på internal state
- Agent Communication Languages
  - Conversation and content languages
    - Content language: Prolog, SQL etc
    - Conversation language: KQML, FIPA-ACL
- FIPA-ACL performatives:

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

- Interaction protocol for task allocation
  - Contract Net (uses FIPA-ACL as conversation language)



- Other agents can answer back and do the task for the asking agent
- Contract Net Protocol (CNP) er en kjent protokoll for task sharing/allocation, har tre steg
  - Task announcement: agent bestemmer om den er kapabel for oppgave ved å evaluere kvalifikasjonskriterier til oppgaveannonseringen. Hvis den er kvalifisert så lagres detaljer rundt oppgavene
  - Bid processing: Etter announcement om oppgave, så mottar manager bud for en begrenset tidsperiode
  - Award processing: Tapere sletter detaljer om oppgavene, vinneren begynner å utføre oppgaven

- Andre interaksjonsmekanismer
  - Negotiation
    - Enkel issue: forhandle om prisen, prisen er en issue
    - Flere issues: Feks når man kjøper en bil, forhandle om attributter i tillegg til pris
    - Bilateral/mutlilateral - 2 eller flere agenter involvert
  - Auctions
    - Sealed/not-sealed - alle agenter gir bud samtidig og alle agenter ser hverandre sine bud
    - First price/second price - hva vinneren betaler, feks høyeste bud

Game Theory

- Studerer valgene til agenter i en multiagent enivronmen
  - Handlinger av hver agent har en effekt på utfallet
  - Strategical decision making
- Strategisk reasoning
  - En agent sitt valg avhenger av hva den tenker den andre agenten vil gjøre
    - Agenten er uncertain om valget til andre deltakende agenter
    - Prøver å forutse neste valg
    - Regner ut hvordan disse potensielle valgene har innvirkning på seg
    - Gjør valg etter dette

- Different types of games
    - 1. simultaneous/strategic normal form games
        - Representert som en payoff matrise
        - Agenten velger sin strategi kun en gang i begynnelsen av spillet, og alle agenter gjør valget sitt likt. One-shot
        - Utfordringen er å forutse hva motstanderen vil gjøre
        - Prisoners dilemma
    - 2. Sequential games
        - Representert som et tre
        - I hvert punkt må agenten velge strategi
        - Minimax og alphabeta funker på slike spill, der det er "zero sum games with perfect information"
        - Sjakk, Go
    - 3. Repeated games
        - Fokus: Huske tidligere oppførsler til andre, stategier basert på tidligere oppførsler
        - Main issue: evolusjon av samarbeid, læring og opponent modeling
        - Prisoners dimella spilt flere ganger
- Simultaneous (normal) games
    - Travelers dilemma - how will the agent think and choose?
    - Prisoners dilemma - To menn er i fengsel i separate celler
    - Stag hunt - trust dilemma, jakte stort bytte sammen eller lite alene
    - Battle of Sexes - et kjærestepar med hver sine preferanser, hvem innfrir den andres ønske
- Assumptions of classical game theory
    - Agenter er rationelle
        - De har veldefinerte oppgaver over et sett med utfall
        - De velger handlingen som leder de til de ønskede utfallene
    - Agents har vanlig kunnskap
        - Kan reglene for spillet
        - Vet at andre agenter er rasjonelle og at de andre vet at de vet at de er rasjonelle
- Defining strategic environments
    - Three main components
        - Player
        - Strategies and actions
        - Payoffs (utilities)

Representation of payoffs:

agent j

agent i	action1	action2
action1	1 1	4 1
action2	1 4	4 4

- Strategiske spill
    - Et spill i strategisk/normal form er definert av
        - Mer enn én agent

        - Handlinger
            - Hver agent velger en handling fra sitt egne arsenal av handlinger
            - En vektor av individuelle handlinger kalles en joint action
        - Utility function (nyttefunksjon)
            - Hver agent har sin egen nyttefunksjon som måler hvor godt de joint actions er
            - Hver agent kan ha forskjellige preferanser for ulike joint actions
            - En payoff matrix representasjon viser nytten til joint actions for hver agent
- Et enkelt interasjonsmiljø
    - Anta 2 agenter Agent = {i,j}
    - Alle handlinger som en agent kan utføre Action = {a1, a2…}
    - Alle mulige utfall til systemet er omega= {w1, w2…} hvor hver w representerer et utfall som er korresponderende til en samling av handlinger for hver agent
    - Miljø-oppførsel er gitt av state performer function: Ac x Ac → omega
    - Dermed er faktisk utfall w avhengig av kombinasjonen av handlinger gjort av agent i og j
- Utilities and preferences
    - Agenters preferanser over utfall er gitt av utility funksjon u: omega → R
    - Utility funksjoner leder til preference ordering over utfall
        - w >i w' iff ui(w) > ui(w') where ui is the utility fn w of agent i.
        - Hence, w >j w' iff uj (w) > uj (w')
- Rasjonell handling
    - Anta vi har en case der begge agentene påvirker utfallet og handling Action={C,D}, altså hver agent har to ulike handlinger de kan utføre
        - Det gir fire mulige utfall
        - t(D,D) = w1 t(D,C) = w2 t(C,D) = w3 t(C,C)= w4
    - Anta agentene har utility functions som følger
        - ui (w1) = 1 ui (w2) = 1 ui (w3) = 4 ui (w4) = 4
          uj (w1) = 1 uj (w2) = 4 uj (w3) = 1 uj (w4) = 4
    - Vi sier at preferansene til i og j er som følger
        - i's: w4 ≥i w3 >i w2 ≥i w1 ((C,C) ≥i (C,D) >i (D,C) ≥i (D,D))
          j's: w4 ≥j w2 >j w3 ≥j w1 ((C,C) ≥j (D,C) >j (C,D) ≥j (D,D))
    - Hvis du er agent i, hva vil du foretrekke å gjøre og hvorfor?
    - Dette gir payoff matrix

agent j

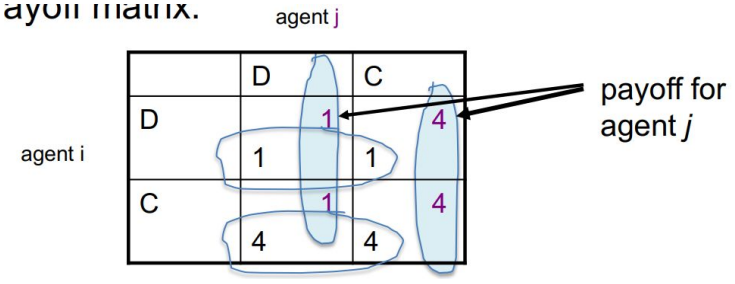
agent i	action c	action d
action a	u_2 u_1	u_4 u_3
action b	u_6 u_5	u_8 u_7

OR

a_j

a_i	action c	action d
action a	u_1, u_2	u_3, u_4
action b	u_5, u_6	u_7, u_8

- Payoff u1 er utility for agent i når agent i velger handling a og agent j velger handling c
- Payoff u2 er utility for agent j når agent i velger handling a og agent j velger c
- Kan kategoriserer dette



- C er det rasjonelle valget til i fordi i foretrekker alle utfall som går gjennom C over alle utfall som kommer gjennom D, uansett hva j gjør. Begge agentene foretrekker faktisk C uavhengig fra den andre
- I denne situasjonen så trenger ikke agentene å tenke på hva den andre agenten vil gjøre - ingen strategic thinking

- Solution concepts
  - En løsning til et spill er en spådom av utfallet til spillet basert på at alle agenter er rasjonelle og strategiske
  - Hvordan spår game theory?
    - Sterk/svak dominant likevekt
    - Iterativ eliminasjon av dominerte handlinger
    - Nash equilibrium
- Solution concept - Strictly Dominant Strategy (SDS)
  - En handling er strengt dominerende hvis den strengt dominerer hver handling i Ai, der Ai er sett av alle joint actions for en agent i
- Strictly Dominant Strategy Equilibrium
  - En dominant handling må være unik, og når den eksisterer, så vil en rasjonell agent velge den
  - Strictly dominant strategy equilibrium er strategien hvor alle agenter velger den strengt dominerende handlingen
  - SDS E eksempel

		agent j	
		defect	cooperate
agent i	defect	2, 2	4, 1
	cooperate	1, 4	3, 3

Preferences:

- i's: **D**,D > **C**,D and **D**,C > **C**,C (compare rows)
- j's: D,**D** > D,**C** and C,**D** > C,**C** (compare columns)

  - Defect er strengt dominerende for begge agenter, og er dermed SDS likevekt (equilibrium)
- Weakly dominant strategy WDS
  - Kalles svakeste dominante hvis den svakt dominerer hver handling i Ai

		agent j	
		action a	action b
agent i	action a	2, 1	0, 2
	action b	2, 3	4, 3

  -
  - Ingen sterke dominante strategier for noen av agentene
  - Handling b er weakly dominant strategy for både i og j
  - Derfor er (b,b) en svak dominant strategi
  - Rasjonelle agenter vil vanligvis spille denne strategien
- Solution concept - Iterated elimination of dominated strategies (IEDS)
  - Dette løsningskonseptet kan bli brukt i spill hvor det er ingen dominante strategi likevekter
  - Required Common knowledge
    - Hver agent vet at den andre er rasjonell
    - Alle spillere vet alle sine payoff funksjoner
  - En rasjonell agent velger aldri en suboptimal handling, aka en dominert handling
  - IESD er en solution technique som iterativt eliminerer SDA fra alle agenter helt til ingen flere handlinger er strictly dominated
  - Eksempel

agent j

agent i		L	M	R
	U	1,0	1,2	0,1
	D	0,3	0,1	2,0

agent j

agent i		L	M	R
	U	1,0	1,2	0,1
	D	0,3	0,1	2,0

agent j

agent i		L	M	R
	U	1,0	1,2	0,1
	D	0,3	0,1	2,0

    -
    - R er strengt dominert for agent j (av handling M)
    - Vi fjerner R fordi i er rasjonell og vet at j ikke spiller R
    - j oppdager nå at D er strengt dominert for agent i. Agent j eliminerer dermed D for i
    - Eliminerer L for j
    - Rekkefølgen av eliminasjon har ikke noe å si for IESD
- Iterated elimination of weakly dominated actions IEWDS
  - Hvis det ikke er noen strongly dominated actions, bruk weakly for å eliminere
- IEDS/IEWDS problems

    - Forskjellige utfall kan komme med utfall som overlever, avhengig av rekkefølgen til eliminasjon i IEWDS
    - Det kan være flere enn én løsning etter eliminasjon
    - Trenger derfor en sterkere solution concept
        - Nash equilibrium
- Pareto optimalt og social welfare
    - To viktige notasjoner for sammenligning og seleksjon for løsninger - hvor god er en løsning
    - To kriterier: Pareto optimalitet og social welfare maximation
- Pareto optimalitet
    - En løsning Sp er pareto optimal hvis det er ingen løsning S' med
        - ∃ agent i : ui (S') > ui (SP)
        - ∀ agent j: uj (S') ≧uj (SP).
        - Altså: det er ingen andre utfall hvor en agent kan øke sin payoff uten å minske payoff til den andre agenten

		A	B
Agent1	A	9,9	7, 10
	B	10, 7	8, 8

    - AA er optimal fordi ingen andre utfall gjøre en agent bedre uten å gjøre den andre verre
    - AB optimal fordi 7 kan bli bedre for agent 1 men da blir agent 2 verre
    - BA optimal
    - BB ikke optimal, siden begge agenter er bedre med AA
- Social welfare
    - Er summen av utilities til alle agenter for denne strategiprofilen
    - Hvis en løsning maksimerer sosial velferd (altså det sosiale optimum), så er de tilgjengelige utilities ikke bortkastet
    - Hvis en løsning er et sosialt optimum, så er den Pareto efficient
        - Motsatt holder ikke
- Repeated/evolutionary games
    - Iterativ prisoners dilemma
        - Hvordan samarbeide? Feks spille spillet flere ganger
    - Hvis du vet at du vil møte motstanderen din igjen, virker indensivet til å snitche å bli borte
    - Samarbeid er det rasjonelle valget i en evig prisoners dilemma for to grunner
        - Hvis man snitcher nå, kan motstanderen straffe deg ved å snitche neste gang
        - Hvis du samarbeider fra starten, og taper så kan man utjevne tap av utility over fremtidige runder
- Backward induction problem
    - Hvis du vet at du skal spille spillet n ganger, så vil du på runde n ha insentiv til å innrømme for å få ekstra payoff.

    - Dette gjør runde n-1 til siste "ekte" runde, som gjør at du vil ha intensiv til å innrømme her også
    - Dette er backward induction problem
    - Å spille prisoners dilemma med en bestemt og kjent antall runder, så er innrømmelse beste strategi
- Axelrod's torunament
    - Anta at du spiller iterativ prisoners dilemma mot en rekke motstandere
    - Hvilken strategi velger du, for å maximize payoff
    - Axelrod undersøkte dette problemet med en computer turnering for programmer som spilte prisoners dilemma
    - Programmene spilte mot hverandre og seg selv flere ganger
        - ALLD:
            - Always defect - the hawk strategy.
        - RANDOM
            - Selects either cooperate or defect on random.
        - TIT-FOR-TAT
            - On round u = 0, cooperate
            - On round u > 0, do what your opponent did on round u-1.
        - TESTER
            - On 1st round defect. If the opponent ever retaliated with defection, then play TIT-FOR-TAT. Otherwise play a repeated sequence of cooperating for two rounds, then defecting.
        - JOSS
            - As TIT-FOR-TAT, except periodically defect.
    -
    - Oppskrift for å gjøre det bra i turneringen
        - Ikke vær misunnelig - spill som om det ikke var score
        - Vær snill - start med å samarbeide, og gjengjeld samarbeid
        - Slå tilbake riktig - alltid straff snitch umiddelbart, men ikke overdo it
        - Ikke bær nag - alltid gjengjeld samarbeid umiddelbart
- Nash equilibrium
    - Strategiprofil slik at hver strategi er beste svar (maksimerer payoff) til alle andre strategier
    - Beste respons for hver agent gir settet av payoff maximizing strategies for hver strategiprofil hos andre spillre
    - En måte å finne Nash eq i to-person strategisk form spill er å utilize de beste svar korresponderingene i payoff matrise
    - Marker den beste responsen til hver agent gitt handlingsvalg til annen agent. En strategisk profil hvor beste responsen til alle agenter rkyser er et Nash eq (ikke nødvendigvis et sett med kun ett element)
    -

		agent j	
		defect	cooperate
agent i	defect	2, 2	4, 1
	cooperate	1, 4	3, 3

    - Her er (defect, defect) en Nash eq fordi
        - Hvis agent i endrer strategi, så blir den dårligere

        - Hvis agent j endrer strategi, så blir den dårligere
        - Som i sjakk, der begge står så godt at de ikke vil flytte fordi da står de dårligere, såkalt trekkvang.
- Nash eq of prisoners dilemma
    - Pareto optimal: CD, DC, CC
    - Social welfare max: CC
    - Løsningen er DD
        - Unique nash (også strengt dominant strategiprofil) er ikke pareto optimal
        - Heller ikke social max
    - Intuisjonen sier at det ikke er det beste utfallet
    - Dette paradokset er det fundamentale problemet til multi-agent interaksjoner
- Multiple Nash eq
    - Ikke alle interaksjonsscenarioer har en enkelt Nash eq

		agent j	
		l	r
agent i	l	1, 1	0, 0
	r	0, 0	2, 2

    - Anta to agenter som hver velger enten l eller r, hvis valgene deres ikke matcher så er det payoff på null
    - Det er to nash eq, ll og rr
    - De to er ikke like, vil agentene bli enige om en øsning?
        - rr er social optimum
    - Slike type spill er kjent som pure coordination game
- Multiple Nash eq - battle of sexes

		agent j	
		movie	opera
ent i	movie	2, 1	0, 0
	opera	0, 0	1, 2

    - Hvis i velger m, så burde j velge m. Samme for o
    - Nash eq er mm og oo
    - Dermed er de enige om at samarbeid er bedre, men er uenige om det beste utfallet
- Multiple Nash eq - Stag hunt

agent j

agent i	Hare	Stag
Hare	1, 1	2, 0
Stag	0, 2	3, 3

-
- Har to Nash eq - hh og ss
- SS er pareto optimal/payoff dominant mens HH er risk dominant
- Høyrere payoff vs tryggere?
- Hvis en agent stoler på den andre så kan ingen av de gjøre det bedre enn ved å samarbeide
-