3-дәріс: Интеллектуалды жүйелердегі білімдерді ұсыну

Пән: Интеллектуалды ақпараттық жүйелер

Алдыңғы тақырыптар: Интеллектуалды ақпараттық жүйелерге кіріспе; Интеллектуалды ақпараттық жүйелер архитектурасы

Дәрістің мақсаты мен күтілетін нәтиже

Параметр	**Мазмұны**
Дәрістің мақсаты	Бүгін біз білімді компьютерге қалай «түсінетін» тілде жеткізуге болатынын, оның қандай негізгі модельдері бар екенін және қай жағдайда қайсысын таңдау керектігін бірге қарастырамыз.
Күтілетін нәтиже	Дәріс соңында сіз факт, дерек және білімді ажырата аласыз; білім базасының құрылымын түсінесіз; логикалық, өнімдік, семантикалық, фреймдік, онтологиялық және векторлық модельдерді салыстыра аласыз; қарапайым мысалды бірнеше модельмен ұсына аласыз.
Негізгі ұғымдар	білім (knowledge), білім базасы (knowledge base), онтология (ontology), факт (fact), ереже (rule), фрейм (frame), семантикалық желі (semantic network), предикат (predicate), инференция (inference), білім графы (knowledge graph), embedding
Оқыту форматы	Түсіндіру, шағын мысалдар, салыстыру, аудиторияға бағытталған сұрақтар және қорытынды бақылау сұрақтары.

1. Кіріспе және тақырыптың орны

Өткен дәрісте біз интеллектуалды ақпараттық жүйенің архитектурасын қарастырдық. Есіңізде болса, онда пайдаланушы интерфейсі, білімді алу модулі, білім базасы, жұмыс жады, логикалық қорытындылау механизмі және түсіндіру модулі бір-бірімен тығыз байланыста жұмыс істейтінін айттық. Енді сол архитектураның жүрегіне — **білім базасына** — тереңірек үңілейік.

Осы жерде бір қарапайым, бірақ өте маңызды сұрақ туындайды: **адамның тәжірибесі мен пәндік сала заңдылықтарын компьютер өңдей алатын түрде қалай жазамыз?** Міне, **білімдерді ұсыну** дәл осы сұраққа жауап береді.

Мысал келтірсек. Дәрігер «**науқастың** дене қызуы жоғары, жөтел бар және өкпеде қабыну белгісі байқаса, **пневмония** ықтималдығын тексеру қажет» деп ойлайды. Бірақ компьютер **табиғи тілдегі** осы ойды өздігінен дәл түсінбейді. Жүйеге симптомдардың атауы, олардың байланысы, ереже, ерекшелік, сенімділік деңгейі және қорытынды жасау тәртібі **формалды түрде** берілуі тиіс. Яғни, біз дәрігердің **басындағы ойды** компьютерге «**аударып**» беруіміз керек.

Аудиторияға сұрақ: «Excel кестесіне барлық белгілерді енгізсек, ол білім базасы бола ма?»

Жауап қарапайым: **жоқ.** Кесте **деректерді** тек сақтайды, ал **білім базасы ұғымдардың мағынасын**, **байланыстарын** және **қорытынды жасауға қажет заңдылықтарын** да сипаттайды.

Сақтау мен түсіну арасындағы айырмашылық осы.

2. Білім, дерек және ақпарат

Бұл үшеуін ажырату — бүгінгі дәрістің негізі болып келеді. Күнделікті өмірде біз оларды араластырып жатамыз, бірақ интеллектуалды жүйе үшін бұл айырмашылық өте маңызды.

Дерек (data) — объект **немесе** оқиға туралы тіркелген **жеке мән**. Мысалы, «температура = 39 °C». Бұл жерде әзірге ешқандай мағына жоқ — **тек сан**.

Ақпарат (information) — **контекст** берілген және мағынасы ашылған дерек: «науқастың температурасы қалыпты деңгейден жоғары». Енді бізде салыстыруға болатын, мағына пайда болды.

Білім (knowledge) — **шешім қабылдауға** немесе **жаңа қорытынды алуға** мүмкіндік беретін **түсіндірілген заңдылық**: «жоғары температура басқа белгілермен бірге **инфекциялық** процесстің белгісі болуы мүмкін». Енді жүйе тек салыстырып қана қоймай, **әрекетке көше алады**.

Деңгей	**Мысал**	**Жүйе не істей алады**
Дерек	Температура: 39 °C	Мәнді сақтайды және көрсетеді.
Ақпарат	39 °C — жоғары температура	Мәнді норма және контекстпен салыстырады.
Білім	Егер жоғары температура және жөтел болса, инфекция гипотезасын тексер	Жаңа гипотеза жасап, келесі әрекетті ұсынады.

Диаграмма ұсынысы: DIKW пирамидасы (Data(дерек) → Information(ақпарат) → Knowledge(білім) → Wisdom(даналық))

Білімнің негізгі түрлері

Білім де біртекті емес. Оны бірнеше түрге бөліп қарастырайық:

- **Декларативтік білім** — «не бар?» деген сұраққа жауап береді. Мысалы: **пневмония** — өкпенің қабыну ауруы. Бұл — фактілер мен ұғымдар жиынтығы.
- **Процедуралық білім** — «қалай орындау керек?» деген сұраққа жауап береді. Бұл — әрекеттер, алгоритмдер және ережелер.
- **Эвристикалық білім** — тәжірибеге негізделген, абсолютті кепілдік бермейтін практикалық қағидалар. Мысалы: « **егер науқастың қызуы ұзақ уақыт түспесе, алдымен инфекцияны тексер**». Бұл әрдайым дұрыс бола бермейді, бірақ көбіне көмектеседі.
- **Метабілім** — жүйенің өзі туралы білімі: қандай білім бар, қай ережені қашан қолдану керек және қай дереккөз сенімді.
- **Құрылымдық білім** — объектілердің бөліктері, иерархиясы және өзара байланысы туралы білім.

3. Білімдерді ұсыну ұғымы және талаптары

3.1. Білімдерді ұсыну дегеніміз не?

Білімдерді ұсыну (knowledge representation) — пәндік сала туралы білімді символдар, құрылымдар және **формалды тілдер** арқылы кодтау тәсілі. Бұл жерде екі міндет қатар тұр: **білімді сақтау** және сол білімнің негізінде **ой қорыту**. Сондықтан жақсы модель тек «**жазып қоюға**» емес, **сұраққа жауап беруге**, **сәйкессіздікті табуға** және **жаңа фактылар алуға** мүмкіндік береді.

Кез келген **білімдерді ұсыну жүйесі** әдетте үш құрамнан тұрады:

1. **Синтаксис** — бұл қандай таңбалар мен құрылымдар рұқсат етілгені;
2. **Семантика** — бұл таңбалардың нені білдіретіні;
3. **Инференция ережелері** — бұл бар білімнен жаңа қорытындыны қалай шығару керектігі туралы.

Бұны тілге теңеуге болады: **синтаксис** — грамматика, **семантика** — мағына, ал **инференция** — сол тілде ойлау қабілеті.

3.2. Модель не үшін қажет?

Біз «**модель**», «**білімдерді ұсыну моделі**» деген сөздерді жиі қолданамыз. Бірақ бұл модель бізге не үшін керек?

Модель — бұл шындықтың жеңілдетілген, мақсатты көрінісі. Біз нақты әлемді толық көшіріп ала алмаймыз — ол тым күрделі. Сондықтан біз тек **маңызды** нәрселерді алып, қалғанын елемейміз. Міне, осы «іріктеу» — модельдің міндеті.

Модель не үшін қажет — төрт негізгі себеп:

1. Сақтау үшін. Білімді **ретсіз мәтін түрінде** сақтауға болмайды. Модель оған **құрылым** береді. Мысалы, «Айжан — студент, ол AI101 оқиды» деген сөйлемді компьютер түсінбейді. Ал RDF үштігі түрінде:

Aizhan — type — Student

Aizhan — studiesCourse — AI101

— бұны компьютер нақты өңдей алады.

2. Қорытынды жасау үшін. Модель **жаңа білімді** шығаруға мүмкіндік береді. Мысалы, егер біз «**барлық студент — адам**» және «**Айжан — студент**» деп жазсақ, жүйе өзі «**Айжан — адам**» деген қорытындыға келеді. Бұл — модельдің ең басты **күші**.

3. Түсіндіру үшін. Адам «неге бұлай?» деп сұрағанда, жүйе қай **ережеден**, қай **фактіден** осы шешімге келгенін көрсете алады. Модельсіз бұл мүмкін емес — тек «**қара жәшік**» болады.

4. Байланыс үшін. Модель — адам мен компьютер арасындағы ортақ тіл. Сарапшы, әзірлеуші және пайдаланушы бір модельге қарап, бір нәрсені түсінеді.

Мысал келтірейік

Елестетіңіз, сіз — дәрігерсіз. Сіздің басыңызда мыңдаған ереже бар: «қызуы бар + жөтелі бар = инфекция», «қызуы бар + басы ауырады = мигрень» деген сияқты. Бірақ бұл білімді компьютерге айтып бере алмайсыз — ол тым көп әрі ретсіз.

Модельсіз:

Сіз компьютерге **әр науқас үшін** бөлек айтуыңыз керек: «мына науқастың қызуы бар, жөтелі бар, сондықтан оған антибиотик бер». Компьютер **тек осы бір** науқастың жағдайын орындайды. Ертең басқа науқас келсе — қайтадан айтуыңыз керек. Яғни, компьютер ештеңе үйренбейді, тек айтқаныңызды қайталайды.

Модельмен:

Сіз ережені бір рет қана жазасыз:

text

ЕГЕР температура > 38 ЖӘНЕ жөтел бар

ОНДА инфекция гипотезасын тексер

Енді компьютер **осы ережені мыңдаған** науқасқа өзі қолданады. Сізге әр науқас үшін қайта айтудың қажеті жоқ — компьютер **өзі шешім** шығарады.

Қорытынды: Модель — білімді бір рет жазып, оны мыңдаған жағдайда қолдануға мүмкіндік беретін құрал.

Яғни, модель неғұрлым жақсы болса, соғұрлым жүйе **ақылдырақ** болады.

3.3. Модельге қойылатын талаптар

Біз «**модельге қойылатын талаптарды**» тізіп қана қоймай, олардың **не үшін** қажет екенін түсінуіміз керек. Әр **талап** — нақты бір мәселенің шешімі болып келеді.

1. Ұсыну жеткіліктілігі — не үшін?

Егер модель пәндік саланың маңызды бөліктерін қамтымаса, жүйе дұрыс жауап бере алмайды. Мысалы, дәрігерлік жүйеде: науқаста **«қызу» бар, бірақ «жөтел» жоқ болса** — жүйе пневмонияны таба алмайды.

Мысал: айтайық Сіздің жүйеңізде тек «**температура**» мен «**жөтел» бар**. Ал науқас «**басым ауырады**» деп шағымданса, жүйе оны елемейді. Себебі — модельде бұл ұғым жоқ. Сондықтан модель **толық** болуы керек.

2. Қорытындылау жеткіліктілігі — не үшін?

Модель тек сақтап қана қоймай, **жаңа білім** шығара алуы керек. Әйтпесе ол — жай ғана деректер базасы.

Мысал: Сізде «Айжан — студент» және «барлық студент — адам» деген фактілер бар. Егер жүйе «Айжан — адам» деп қорытынды жасай алмаса, онда модель нашар. Жақсы модель **өзі ойлайды**.

3. Қорытындылаудың тиімділігі — не үшін?

Үлкен білім базасында сұранысқа жауап тез болуы қажет. Пайдаланушы 5 минут күтпейді.

Мысалға: Google-да іздегенде 0.5 секундта жауап келеді. Егер жүйе 30 секунд ойланса, пайдаланушы оны қолданбайды. Сондықтан модель **жылдам** болуы керек.

4. Білімді алу тиімділігі — не үшін?

Айтайық, ауруханада жаңа ереже пайда болды: «**егер науқастың қызуы 3 күннен артық түспесе**, оны қосымша тексеруге жібер». Енді осы ережені жүйеге қосу керек.

Нашар жағдай:

Жаңа ереже қосу үшін программистті шақыру керек. Ол кодты ашады, өзгертеді, тексереді, қайта іске қосады. Бұл — бірнеше күн, кейде апта. Ал осы уақытта дәрігерлер ескі жүйемен жұмыс істеуге мәжбүр. Жүйе ескіреді — себебі өмір өзгерді, бірақ жүйе соңынан үлгермейді.

Жақсы жағдай:

Сарапшы (мысалы, тәжірибелі дәрігер) өзі жүйеге кіріп, ережені жазады:

text

ЕГЕР қызу > 3 күн

ОНДА қосымша_тексеру = иә

Программист керек емес. Ереже бірден жұмыс істей бастайды. Жүйе өмірмен бірге жаңарып отырады. Яғни, бағдарламаның негізгі коды — бір жерде, ал ережелер — бөлек жерде. Ережені өзгерту керек болса, кодқа тимейсіз — тек ереженің өзін өзгертесіз. Бұл — ыңғайлы, қауіпсіз және жылдам.

Қорытынды: Егер сарапшы ережені өзі қоса алса — жүйе тез жаңарады, ұзақ өмір сүреді. Ал егер әр ереже үшін программист керек болса — жүйе тез ескіреді, ақыры қолданыстан шығады.

5. Түсіндірілушілік — не үшін?

Адам «неге?» деп сұрауға құқылы. Әсіресе медицина, қаржы, заң салаларында.

Мысалы: Дәрігерлік жүйе «пневмония» деп шешім шығарды. Дәрігер «неге?» деп сұрайды. Жүйе: «Себебі температура > 38, жөтел бар, өкпеде қабыну белгісі бар» деп жауап береді. Егер жүйе түсіндіре алмаса — дәрігер оған сенбейді.

6. Белгісіздік пен қайшылықты өңдеу — не үшін?

Өмірде біздің біліміміз әрдайым толық бола бермейді. Кейде «иә» немесе «жоқ» деп нақты айту мүмкін емес. «Белгі болса, ауру бар» деген қатаң ереже жиі дұрыс болмайды — себебі бір белгі бірнеше нәрсенің белгісі болуы мүмкін, ал кейде белгі мүлде басқа нәрсені білдіреді.

Өмірден мысал:

Елестетіңіз, сіз банкте жұмыс істейсіз. Клиенттің транзакциясы келіп түсті. Бұл — алаяқтық па, әлде қалыпты жағдай ма?

Міне, қиындық осында:

- Клиент таңғы 9-да өз қаласында 50 000 теңге шешті — қалыпты
- Клиент түнде 3-те шет елден 500 000 теңге шешті — күдікті
- Клиент түнде 2-де өз қаласында 30 000 теңге шешті — шекарада

Үшінші жағдай — дәл ортасында. Оны «алаяқтық» деуге де, «қалыпты» деуге де болады. Нақты жауап жоқ.

Қатаң ереже не істейді?

text
ЕГЕР транзакция түнде болса
ОНДА алаяқтық

Бұл ереже түндегі кез келген транзакцияны «алаяқтық» дейді. Ал шын мәнінде адам шетелде демалып, отбасына ақша жіберуі мүмкін. Қатаң ереже — өмірдің икемділігін ұстай алмайды.

Икемді ереже не істейді?

Жүйе «бұл транзакция алаяқтық жиынына 0.6 дәрежеде жатады» дейді. Яғни:

- 0.0 — мүлде жатпайды (күндіз, өз қаласы, аз сома)
- 0.6 — орташа жатады (түнде, өз қаласы, орташа сома)
- 1.0 — толық жатады (түнде, шет ел, үлкен сома)

Бұл — **айқын емес логика (fuzzy logic)** деп аталады. Ол «иә/жоқ» деп қана қоймай, «қаншалықты?» деген сұраққа да жауап береді.

Қорытынды: Нақты өмірде білім әрдайым толық емес, шекаралар бұлыңғыр. Сондықтан модель «иә/жоқ» деп қатаң айтудың орнына, «**қаншалықты**?» деген сұраққа жауап бере алуы керек. Міне, сондықтан ықтималдық пен айқын емес логика қажет**.**

Қорытынды ой: Әр талап — бір нақты мәселенің шешімі:

Талап	**Қандай мәселені шешеді**
Ұсыну жеткіліктілігі	Білімнің толық болмауы
Қорытындылау жеткіліктілігі	Жаңа білім шығара алмау
Қорытындылаудың тиімділігі	Баяу жауап
Білімді алу тиімділігі	Жүйенің тез ескіруі
Түсіндірілушілік	Пайдаланушының сенімсіздігі
Белгісіздікті өңдеу	Нақты өмірдің күрделілігі

Маңызды ой: Модельді таңдағанда, осы талаптардың қайсысы сіздің міндетіңіз үшін ең маңызды екенін анықтау керек. **Медицинада — түсіндірілушілік, іздеу жүйесінде — жылдамдық, заң жүйесінде — дәлдік.**

4. Логикалық модель

Енді нақты модельдерге көшейік. Ең алдымен — **логикалық модель**. Мұнда білім формулалар түрінде беріледі, ал қорытынды **формалды логика** ережелерімен алынады. Бұл тәсілдің басты артықшылығы — **дәлдік**: формуланың мағынасы және дұрыс қорытындының шарттары анық белгіленеді.

4.1. Пікірлер логикасы (Propositional Logic)

Қарапайым пікірлер ақиқат немесе жалған мән қабылдайды. Белгілеулер: **P** — «температура жоғары», **Q** — «жөтел бар», **R** — «инфекцияға күдік бар». Ереже былай жазылады:

(P∧Q)→R

Егер P және Q ақиқат болса, **modus ponens** ережесі арқылы R алынады.

Шектеуі: пікірлер логикасы объектінің ішкі құрылымын көрсетпейді. «Айданың температурасы жоғары» және «Бектің температурасы жоғары» жеке символдар ретінде жазылады; олардың екеуіндегі «пациент» ұғымының ортақтығы айқын емес. Яғни, логика «не» екенін біледі, бірақ «кім» екенін білмейді.

Ереженің құрылымы (Modus Ponens)

Логикада бұл ереже екі **алғы**шарттан және бір қорытындыдан тұрады:

1. **Басты алғышарт (импликация):** Егер **A** болса, онда **B** болады (A → B).
2. **Қосымша алғышарт: A** орындалды (ақиқат).
3. **Қорытынды:** Демек, **B** де орындалады.

Қарапайым мысал:

- 1-алғышарт: Егер бүгін жаңбыр жауса (A), онда жер су болады (B).
- 2-алғышарт: Бүгін жаңбыр жауып тұр (A).
- Қорытынды: Демек, жер су (B).

Бұл — адам баласының күнделікті ойлау тәсілінің формалды көрінісі.

4.2. Бірінші ретті предикаттар логикасы (First-Order Logic)

Предикаттар логикасы объектілерді, қасиеттерді және қатынастарды көрсетеді. Мысалы:

```
Студент(Аружан)
Оқиды(Аружан, AI)
∀x(Студент(x) ∧ Емтиханнан_өтті(x, Exam) → Қабылданды(x))
```

Мұндағы:

- **∀** — «барлық» (universal quantifier);
- **∃** — «кемінде бір» (existential quantifier);
- **→** — импликация;
- **∧** — конъюнкция.

Мысал:

∀x(Bird(x) → HasWings(x))

Bird(Torgai)

HasWings(Torgai) % қорытынды

Бірақ **«барлық құс ұшады» ережесіне** пингвин секілді ерекшеліктерді енгізу классикалық логикада қосымша формализацияны талап етеді (default reasoning, non-monotonic logic). Бұл — логиканың «қатаңдығы» мен нақты өмірдің «икемділігі» арасындағы табиғи шиеленіс.

Қолданылатын құралдар:

- **Prolog** — логикалық бағдарламалау тілі; фактілер мен ережелерді сипаттап, сұраныс бойынша қорытынды жасайды;
- **Datalog** — деректер базалары мен ережелік сұраныстарда қолданылады;
- **Z3** — SMT шешушісі; логикалық және арифметикалық шектеулердің орындалуын тексереді.

Prolog мысалы:

```
bird(torgai).

has_wings(X) :- bird(X).

?- has_wings(torgai).

% нәтиже: true
```

5. Өнімдік модель және ережелер жүйесі

Өнімдік модель — тәжірибеде ең кең тараған тәсілдердің бірі. Мұнда білім **«ЕГЕР шарт, ОНДА әрекет немесе қорытынды»** түріндегі ережелермен ұсынылады. Ереже екі бөліктен тұрады:

- **Антецедент** — іске қосылу шарты;
- **Консеквент** — шарт орындалғанда жасалатын қорытынды немесе әрекет.

Мысал:

ЕГЕР студенттің қорытынды балы ≥ 50

ЖӘНЕ емтихан балы ≥ 20

БОЛСА, ОНДА пән меңгерілді.

Факт ретінде баллдар жұмыс жадысына енгізіледі. Инференция механизмі шарттары орындалған ережелерді табады және жаңа факт қосады. Бұл — адамның «егер... онда...» деп ойлауының тура көшірмесі.

5.1. Тура және кері тізбектеу

Бұл екі механизм — өнімдік жүйенің «ойлау тәсілдері». Олардың қайсысын қолдану — міндеттің табиғатына байланысты.

Механизм	Бастапқы нүкте	Жұмыс қағидасы	Қолайлы жағдай	Мысал
Тура тізбектеу (forward chaining)	Белгілі фактілер	Фактілерге сәйкес ережелер іске қосылып, жаңа қорытындылар алынады.	Мониторинг, көптеген ықтимал қорытынды, оқиғаға реакция.	Медициналық мониторинг: симптомдардан диагнозға қарай.
Кері тізбектеу (backward chaining)	Мақсат немесе гипотеза	Мақсатты дәлелдеу үшін қандай шарттар қажет екені кері бағытта тексеріледі.	Диагностика, нақты сұраққа жауап беру, Prolog сұранысы.	«Пневмония ма?» деген сұрақтан симптомдарға қарай.

Тура тізбектеу (Forward Chaining) — не үшін қажет?

Мәні: Бар фактілерден бастап, ережелерді қолданып, жаңа фактілер шығарамыз. Яғни, **деректерден қорытындыға** қарай жылжимыз.

Не үшін қажет: Жағдай өзгеріп отырғанда, жүйе дереу реакция жасауы керек. Мониторинг, бақылау, оқиғаға жауап беру — осының бәрі тура тізбектеуге жатады.

мысал — Медициналық мониторинг:

Елестетіңіз, ауруханада науқастың жағдайы үздіксіз бақыланады. Сенсорлар әр секунд сайын деректер жібереді.

Фактілер (бастапқы):

Температура = 39.2

Жөтел = бар

Қан қысымы = төмен

Ережелер:

ЕГЕР температура > 38 ЖӘНЕ жөтел бар

ОНДА инфекция_күдігі = бар

ЕГЕР инфекция_күдігі = бар ЖӘНЕ қан_қысымы төмен

ОНДА шұғыл_көмек_қажет = иә

ЕГЕР шұғыл_көмек_қажет = иә

ОНДА дәрігерді_шақыр

Тура тізбектеу жұмысы:

1. Температура 39.2 → «инфекция_күдігі = бар» қосылады
2. Инфекция + қысым төмен → «шұғыл_көмек_қажет = иә» қосылады
3. Шұғыл көмек → «дәрігерді_шақыр» әрекеті орындалады

Нәтиже: Жүйе өзі дәрігерді шақырады. Адам араласпайды.

Тағы бір мысал — Банктегі алаяқтықты анықтау:

Факті:

Клиент 3 минутта 5 рет шет елден ақша шешті

Ереже:

ЕГЕР 5 минутта 3+ транзакция шет елден болса → күдікті

Ереже:

ЕГЕР күдікті → картаны блокта

Жүйе автоматты түрде картаны блокадайды. Бұл — тура тізбектеу.

Қашан қолданылады:

- Мониторинг жүйелері (науқас, желі, қауіпсіздік)
- Оқиғаға реакция (дабыл, ескерту)
- Көптеген ықтимал қорытынды бар жағдайлар

- Деректер үнемі өзгеріп отыратын жүйелер

Кері тізбектеу (Backward Chaining) — не үшін қажет?

Мәні: Мақсаттан бастап, оны дәлелдеу үшін қандай шарттар қажет екенін кері бағытта тексереміз. Яғни, **қорытындыдан деректерге** қарай жылжимыз.

Не үшін қажет: Нақты бір сұраққа жауап беру керек болғанда. Барлық мүмкін қорытындыларды шығарып жатпай, тек қажеттісін тексереміз. Бұл — тиімдірек.

мысал — Дәрігерлік диагностика:

Науқас келіп: «Менде пневмония ма?» деп сұрайды.

Мақсат: пневмония = бар ма?

Кері тізбектеу жұмысы:

1. «Пневмония» деген қорытынды қандай шарттардан шығады?
   → ЕГЕР қызу > 38 ЖӘНЕ жөтел ЖӘНЕ өкпеде_қабыну ОНДА пневмония
2. Енді осы шарттарды тексереміз:
   - Қызу > 38? → Иә (39.2)
   - Жөтел? → Иә
   - Өкпеде_қабыну? → Бұл үшін рентген керек
3. Рентген сұраймыз. Нәтиже: қабыну бар.
4. Барлық шарт орындалды → «Пневмония = бар» қорытындысы.

Нәтиже: Жүйе тек қажетті сұрақтарды қойды. Басқа ауруларды тексермеді. Бұл — уақытты үнемдеу.

Тағы бір мысал — Техникалық қолдау жүйесі:

Пайдаланушы: «Интернет жұмыс істемейді» дейді.

Мақсат: интернет_жұмыс_істемейді = себебін_тап

Кері тізбектеу:

1. Интернет неге істемейді? → Мүмкін себептер: кабель, роутер, провайдер
2. Кабельді тексер → Жарамды
3. Роутерді тексер → Жарамды
4. Провайдерді тексер → Ақаулық бар!
5. Шешім: «Провайдерде ақаулық, күте тұрыңыз»

Жүйе тек қажетті тексерулерді жасады.

Қашан қолданылады:

- Диагностика (медицина, техника)
- Нақты сұраққа жауап беру
- Prolog сұраныстары
- Эксперттік жүйелерде «неге?» деген сұраққа жауап

Екеуін салыстыру — қайсысын қашан қолдану керек?

Критерий	**Тура тізбектеу**	**Кері тізбектеу**
Бағыт	Дерек → Қорытынды	Қорытынды → Дерек
Бастапқы нүкте	Бар фактілер	Мақсат/гипотеза
Не үшін	Жағдайды бақылау, реакция	Нақты сұраққа жауап
Тиімділігі	Көп қорытынды шығарғанда	Аз қорытынды керек болғанда
Мысал	Мониторинг, дабыл	Диагностика, Prolog

Жүйе түрі	Реактивті жүйе	Консультативті жүйе

Өмірлік аналогия:

Тура тізбектеу — бұл **қауіпсіздік камерасы**. Ол үнемі бақылайды, бірдеңе болса — дабыл қосады. Сіз одан «не болды?» деп сұрамайсыз — ол өзі айтады.

Кері тізбектеу — бұл **дәрігер**. Сіз оған «менің не болды?» деп сұрайсыз. Ол сізге сұрақтар қойып, тек қажетті тексерулерді жасайды.

Қорытынды ой: Екі механизм — бір мақсаттың екі жолы. Қайсысын таңдау — **міндеттің табиғатына** байланысты:

- Егер жүйе **үнемі бақылап, дереу реакция жасауы** керек болса → **тура тізбектеу**
- Егер жүйе **нақты сұраққа жауап беруі** керек болса → **кері тізбектеу**

Көптеген заманауи жүйелер **екеуін де** қолданады: тура тізбектеу — мониторинг үшін, кері тізбектеу — сұранысқа жауап беру үшін.

Конфликт жиыны (conflict set): Бір уақытта бірнеше ереже іске қосылуға дайын болса, жүйе басымдық, нақтылық, жаңалық немесе ереженің салмағы бойынша біреуін таңдайды. Бұл басқару стратегиясы өнімдік жүйенің нәтижесі мен жылдамдығына тікелей әсер етеді. Яғни, «қай ережені бірінші қолдану керек?» деген сұрақ — жүйенің тағдырын шешеді.

Құралдар:

- **CLIPS** — сараптамалық жүйелерге арналған ережелік орта;
- **Drools** — Java экожүйесіндегі Business Rules Management System (BRMS);
- **Jess** — Java негізіндегі ережелік қозғалтқыш;
- **Experta** — Python-дағы шағын оқу жобаларына арналған ережелік кітапхана.

Drools туралы қысқаша: Drools — Java тілінде жазылған, күрделі логикалық ережелерді басқаруға арналған ашық бастапқы кодты бағдарламалық жүйе. Қарапайым тілмен айтқанда, ол бағдарлама ішіндегі «егер мынау орындалса, онда ананы істе» деген сияқты бизнес-ережелерді кодтан бөлек, ыңғайлы форматта басқаруға мүмкіндік береді. Бұл — әзірлеуші мен бизнес-сарапшы арасындағы «тіл» рөлін атқарады.

Drools мысалы:

ЕГЕР клиенттің сатып алу сомасы 50 000 теңгеден асса

ЖӘНЕ ол тұрақты тұтынушы болса,

ОНДА оған 10% жеңілдік жасалсын.

6. Семантикалық желі

Семантикалық желі білімді **граф** ретінде көрсетеді. Бұл — адам миының ассоциативті ойлауына ең жақын модель.

Мысал:

Торғай —IS-A→ Құс

Құс —HAS-PART→ Қанат

Торғай —LIVES-IN→ Қазақстан

Негізгі қатынас түрлері:

Қатынас	Мағынасы	Мысал
IS-A	класс иерархиясы	Торғай IS-A Құс
PART-OF	бөлік-бүтін	Қанат PART-OF Құс
HAS-PART	бүтін-бөлік	Құс HAS-PART Қанат
INSTANCE-OF	экземпляр-класс	Торғай INSTANCE-OF Құс

Мұрагерлік (inheritance): Егер «Құстың қанаты бар» және «Торғай құсқа жатады» делінсе, жүйе «Торғайдың қанаты бар» деген қасиетті мұраға алады. Бұл — адамның «торғай — құс, құстың қанаты бар, демек торғайдың да қанаты бар» деп ойлауының тура формалды көрінісі.

Артықшылығы: адамға көрнекі, графтық навигация ыңғайлы.

Шектеуі: қатынастардың мағынасы қатаң анықталмаса, әр әзірлеуші бір байланысты әртүрлі түсіндіруі мүмкін. Мысалы, біреу «IS-A»-ны класс иерархиясы деп түсінсе, екіншісі экземпляр қатынасы деп түсінуі мүмкін.

Құралдар:

- **Neo4j**, **Amazon Neptune**, **ArangoDB** — графтық деректер базалары;
- **Cypher** — Neo4j-дің сұраныс тілі;
- **Gephi** — желіні визуализациялауға арналған құрал (бірақ толық инференция механизмі емес).

7. Фреймдік модель

Фрейм (frame) — типтік объектіні, жағдайды немесе оқиғаны сипаттайтын құрылым. Фрейм **атаудан** және **слоттардан** тұрады. Слот — қасиет немесе қатынас; оның мәні, әдепкі мәні, шектеуі және қажет болса процедурасы болады. Бұны «объектінің сауалнамасы» деп ойлауға болады.

Фрейм	Слот	Мән мысалы
СТУДЕНТ	аты	Аружан
СТУДЕНТ	білім беру бағдарламасы	Ақпараттық жүйелер
СТУДЕНТ	курс	2
СТУДЕНТ	мәртебе	әдепкі мәні: белсенді
СТУДЕНТ	оқиды	ЖОО фрейміне сілтеме

Мұрагерлік: «Адам» фрейміндегі туған күні мен байланыс деректері «Студент» фреймін беріледі, ал «Студент» оған оқу тобы мен GPA слоттарын қосады. Бұл — семантикалық желідегі мұрагерліктің фреймдік көрінісі.

Процедуралық тіркеме (procedural attachment): Слот өзгергенде есептеу немесе тексеру орындай алады, мысалы GPA жаңартылғанда академиялық мәртебені қайта есептеу. Бұл — фреймге «тірі» қасиет береді.

Фрейм vs класс: Фреймдік тәсіл объектіге бағытталған бағдарламалауға ұқсайды, бірақ мақсаты бөлек. Класс объектінің бағдарламалық құрылымы мен әдістерін ұйымдастырады; фрейм пәндік мағынаға, әдепкі білімге, мұрагерлікке және инференцияға бағытталады.

Құралдар:

- **Protégé** — класстар мен қасиеттер арқылы фреймге жақын құрылымдар құрылады;
- **CLIPS** — deftemplate конструкциясы слоттары бар фактілерді сипаттайды.

Қазіргі жүйелерде фрейм идеялары объектілік модельдер мен онтологияларда жалғасады.

8. Онтологиялық модель

Онтология (ontology) — пәндік саланың ортақ және формалды концептуалдық моделі. Ол класстарды, нақты экземплярларды, қасиеттерді, қатынастарды, шектеулерді және аксиомаларды анықтайды. Онтологияның мақсаты тек граф салу емес, **терминдердің бір мағынада қолданылуын қамтамасыз ету** және автоматты қорытындыға жағдай жасау. Бұл — ұйымдағы «ортақ тіл» рөлін атқарады.

Университет онтологиясының шағын бөлігі:

- Person және Course — класстар;
- Student — Person класының ішкі класы;
- studiesCourse — Student пен Course арасындағы объектілік қасиет;
- hasGPA — сандық қасиет;
- Aizhan — Student класының экземпляры.

«Әр студент кемінде бір курс оқиды» деген шектеу аксиома ретінде берілуі мүмкін.

8.1. RDF, RDFS және OWL — қысқаша

Бұл — тілдер **(формалды тілдер). Яғни, ереже емес, принцип емес —** білімді жазуға арналған стандартты тілдер**.**

Тіл	Не үшін қажет	Мысал
RDF	Фактілерді «субъект — предикат — объект» үштігі түрінде жазады	Айжан — type — Student
RDFS	Құрылым береді: класс, ішкі класс, домен, диапазон	Student subClassOf Person
OWL	Күрделі логика: қайшылық, кардиналдық, эквиваленттілік	Professor disjointWith Student

RDF — «Айжан — студент» деп жазады.
RDFS — «Студент — Адамның бір түрі» деп құрылым қосады.
OWL — «Бір адам әрі Профессор, әрі Студент бола алмайды» деп логика қосады.

Бір сөйлеммен: RDF — сақтайды, RDFS — құрылымдайды, OWL — ойлайды.

SPARQL (SPARQL Protocol and RDF Query Language)

RDF графынан мәліметтерді іздеп табуға арналған сұраныс тілі. SQL-ге функционалдық тұрғыдан ұқсайды, бірақ үштіктер үлгілерімен жұмыс істейді.

Тапсырма: «Жасанды интеллект» (AI101) курсында оқитын барлық студенттердің есімін табу керек.

SPARQL сұранысы:

PREFIX ex: <http://example.org/>

```
SELECT ?studentName

WHERE
{
  ?studentName ex:studiesCourse ex:AI101 .
}
```

SQL-мен салыстырғанда: SQL-де бұл үшін екі немесе үш кестені бір-біріне қосу (JOIN) керек болар еді, ал SPARQL-де тек ізделіп отырған үштіктің үлгісін (?studentName – studiesCourse – AI101) көрсету жеткілікті.

Reasoner (қорытындылау модулі)

Reasoner жасырын білімді табады. Мысалы, Student ⊑ Person және Aizhan:Student берілсе, reasoner Aizhan:Person қорытындысын шығарады. **HermiT**, **Pellet** және **FaCT++** — кең тараған OWL reasoner үлгілері.

Құралдар:

- **Protégé** — негізгі құрал; класстар иерархиясын, object property және data property түрлерін, individuals пен аксиомаларды графикалық интерфейсте жасауға болады;
- **Apache Jena** және **RDF4J** — Java қолданбаларында RDF/OWL өңдейді;
- **GraphDB** және **Stardog** — семантикалық графтарды сақтайтын платформалар.

9. Білім графтары

Білім графы (knowledge graph) — объектілер мен олардың семантикалық байланыстарын граф түрінде біріктіретін практикалық архитектура. Онтология графтың **схемасы мен мағынасын** анықтай алады, ал білім графы көбіне **үлкен көлемдегі нақты фактілерді** қамтиды.

Мысал: Университет графында студенттер, пәндер, оқытушылар, кафедралар және аудиториялар бір желіге біріктіріледі.

Қолданылуы:

- Іздеу жүйелері (Google Knowledge Graph, Wikidata);
- Ұсыныс жүйелері;
- Ұйым деректерін интеграциялау;
- Алаяқтық байланыстарын анықтау;
- Генеративті модельге нақты контекст беру (RAG).

Дереккөздер әртүрлі болғандықтан **entity resolution**, яғни бір объектінің әр жазбадағы атауларын сәйкестендіру маңызды.

10. Белгісіз білімді ұсыну

Нақты өмірде білім әрдайым толық және детерминдік емес. «Белгі болса, ауру бар» деген қатаң ереже жиі дұрыс болмайды. Сондықтан жүйе белгісіздіктің табиғатына сәйкес модель қолданады.

Тәсіл	**Нені көрсетеді**	**Мысал**
Ықтималдық модель	Оқиғаның ықтималдығы және тәуелділігі	Байес желісі: P(ауру | симптомдар).
Айқын емес логика	Ұғымға мүшелік дәрежесі	Температура 37.5 °C — «жоғары» жиынына 0.4 дәрежеде жатады.

Сенімділік коэффициенті	Сарапшы ережесінің күші	Ереже қорытындысына 0.75 сенімділік беріледі.
Дәлелдер теориясы	Сенім аралығы және бірнеше дереккөз	Әртүрлі сенсор дәлелдерін біріктіру.

Құралдар: Bayesian network үшін **GeNIe/SMILE** және Python-дағы **pgmpy**; айқын емес логика үшін **MATLAB Fuzzy Logic Toolbox** немесе Python-дағы **scikit-fuzzy** қолданылады.

Модельді таңдағанда «белгісіздік ықтимал кездейсоқтық па, әлде ұғым шекарасының көмескілігі ме?» деген сұрақ қойылады.

11. Векторлық ұсыну және embedding

Қазіргі машиналық оқыту жүйелерінде сөз, мәтін, сурет немесе объект жоғары өлшемді сандық вектор ретінде ұсынылады. **Embedding** мағынасы ұқсас объектілерді векторлық кеңістікте жақын орналастыруға үйретіледі. Мысалы, «дәрігер» және «медицина» векторлары «дәрігер» және «геология» векторларына қарағанда жақын болуы мүмкін.

Векторлық ұсынудың күші — мағыналық ұқсастықты табу және құрылымдалмаған мәтінмен жұмыс істеу.

Әлсіздігі — символдық ереже секілді түсінікті логикалық дәлел бермеуі және абсолютті ақиқатқа кепілдік бермеуі. Вектор — фактінің өзі емес, үйренілген сандық репрезентация.

Құралдар: sentence-transformers және әртүрлі embedding модельдері вектор жасайды; **FAISS**, **Milvus**, **Pinecone**, **Weaviate** немесе **pgvector** жақын векторларды іздейді. **RAG (Retrieval-Augmented Generation)** — векторлық іздеуді генеративті модельмен біріктіретін тәсіл.

12. Символдық және нейрондық тәсілдерді біріктіру

Бір модель барлық міндетке бірдей тиімді емес. Қатаң бизнес ережелері мен тексерілетін қорытынды үшін символдық модельдер қолайлы; мәтіннің мағынасына ұқсас іздеу үшін векторлық модель тиімді. **Гибридтік** немесе **нейросимволдық** жүйе осы артықшылықтарды біріктіреді.

Мысал: Университеттің кеңесші жүйесі оқу ережелерін Drools немесе онтологияда сақтайды; пән сипаттамаларын embedding ретінде векторлық базаға орналастырады; студент сұрағын тілдік модель талдайды; соңғы шешім міндетті пререквизиттер мен кредит шектеулеріне қарсы тексеріледі. Осылайша еркін тілмен жұмыс істеу қабілеті формалды бақылаумен толықтырылады.

13. Модельдерді салыстыру

Модель	**Негізгі бірлік**	**Күшті жағы**	**Шектеуі**	**Қашан қолдану**	**Құрал мысалы**
Логикалық	формула, предикат	Дәлелдің қатаңдығы, тексерілетін қорытынды	Ерекшелік пен толық емес білімді беру күрделі	Дәл қорытынды керек болғанда	Prolog, Z3
Өнімдік	ЕГЕР-ОНДА ережесі	Модульдік, түсіндіруге ыңғайлы	Ереже саны артса басқару қиындайды	Бизнес ережелері, диагностика	CLIPS, Drools
Семантикалық желі	түйін және байланыс	Көрнекі қатынастар, графтық навигация	Формалды семантикасы әлсіз болуы мүмкін	Білімді визуалды көрсету	Neo4j, Gephi

Фреймдік	фрейм және слот	Типтік объект құрылымын табиғи береді	Күрделі логикалық қорытынды шектеулі	Объектіні сипаттау	CLIPS, Protégé
Онтологиялық	класс, қасиет, аксиома	Ортақ сөздік, формалды семантика, reasoner	Жобалау құны және есептеу күрделілігі	Үлкен жүйелерде ортақ тіл керек болғанда	Protégé, OWL
Ықтималдық	айнымалы және ықтималдық	Белгісіздікпен негізді жұмыс	Параметрлерді бағалау қажет	Медицина, қаржы	pgmpy, GeNIe
Векторлық	embedding векторы	Мағыналық іздеу, құрылымдалмаған дерек	Түсіндіруі әлсіз, дәл логика емес	Ұқсастық іздеу, RAG	FAISS, pgvector

14. Модельді таңдау алгоритмі

1. **Пәндік міндетті анықтау:** диагностика ма, іздеу ме, жоспарлау ма, әлде деректерді біріктіру ме?
2. **Білім сипатын анықтау:** фактілер, қатаң ережелер, иерархия, ықтимал байланыс немесе мәтіндік мағына.
3. **Қорытынды түрін белгілеу:** дәлелденетін логикалық жауап, ықтимал болжам, ұқсастық бойынша нәтиже немесе әрекет.
4. **Түсіндіру талабы мен қатенің құнын бағалау.** Медицина мен қаржыда тексерілетін ереже және дереккөз маңызды.
5. **Көлем, өзгеру жиілігі, жылдамдық және қолданыстағы технологиялық ортаны бағалау.**
6. **Шағын прототип жасап, дәлдік, толықтық, жауап уақыты, түсіндірілушілік және сүйемелдеу құнын өлшеу.**
7. **Қажет болса бірнеше модельді ортақ архитектурада біріктіру.**

Маңызды қағида: Бағдарламаны танымалдығына қарап емес, білімнің табиғаты мен жүйе міндетіне қарап таңдау керек. Neo4j — граф сақтауға, Protégé — онтология жобалауға, Drools — ережелерді орындауға, FAISS — векторлық іздеуге арналған; олар бір-бірінің толық баламасы емес.

15. Дәрістік кешенді мысал

Міндет: Студентке пән таңдауға көмектесетін интеллектуалды кеңесші құру. Бір пәнді таңдау үшін пререквизит, студент меңгерген пәндер, қызығушылық бағыты, сабақ кестесі және кредит шегі ескеріледі.

Білім бөлігі	**Ұсыну тәсілі**	**Нақты мысал**
Қатаң оқу талабы	Өнімдік ереже	Егер AI101 пәнінің пререквизиті CS102 болса және студент CS102-ден өтпесе, тіркеуге рұқсат берме.
Ұғымдар иерархиясы	Онтология	MachineLearningCourse ≤ AICourse; hasPrerequisite қасиеті.
Студенттің күйі	Фрейм	Student: completedCourses, GPA, interests, creditLimit слоттары.

Білім бөлігі	Ұсыну тәсілі	Нақты мысал
Пән сипаттамасы	Векторлық embedding	«Машиналық оқыту» мен «деректерді талдау» векторлары жақын.

Өңдеу реті:

1. Жүйе пайдаланушы сұрағын түсінеді (тілдік модель);
2. Векторлық іздеу ықтимал пәндерді табады;
3. Онтология пәндердің түрі мен байланысын анықтайды;
4. Ережелік қозғалтқыш пререквизит пен кредит шегін тексереді;
5. Түсіндіру модулі «неге ұсынылды немесе неге рұқсат етілмеді?» деген сұраққа қолданылған фактілер мен ережелерді көрсетеді.

Бұл — **гибридтік жүйенің** нақты көрінісі: символдық (ереже, онтология) және нейрондық (embedding, тілдік модель) тәсілдер бірге жұмыс істейді.

16. Білім базасын әзірлеу кезеңдері

1. Пәндік саланың шекарасын, мақсатты пайдаланушыларды және шешілетін сұрақтарды анықтау.
2. Сарапшыларды, құжаттарды, деректер қорын және басқа білім көздерін белгілеу.
3. Терминдер сөздігін құру, синонимдер мен көпмағыналы атауларды келісу.
4. Ұғымдар, объектілер, қасиеттер, қатынастар, ережелер және ерекшеліктерді бөліп алу.
5. Білім табиғатына сәйкес модель мен бағдарламалық құралды таңдау.
6. Білімді формализациялау және білім базасына енгізу.
7. Синтаксистік, семантикалық және логикалық сәйкестікті тексеру.
8. Бақылау мысалдары арқылы қорытындының дұрыстығын бағалау.
9. Нұсқаларды, авторды, дереккөзді және өзгеру себебін тіркеп, білімді тұрақты жаңарту.

Knowledge acquisition bottleneck — сарапшының жасырын тәжірибесін дәл формалдаудың қиындығы. Сарапшы ережені интуитивті қолдануы мүмкін, бірақ оның барлық шарттары мен ерекшеліктерін бірден атай алмайды. Сондықтан сұхбат, бақылау, хаттамалық талдау, құжаттарды талдау және мысалдарды жіктеу бірге қолданылады.

17. Жиі кездесетін қателер

- **Деректер базасын білім базасымен теңестіру:** сақтау бар, бірақ семантика мен инференция жоқ.
- **Терминдерді анықтамай қолдану:** бір ұғым бірнеше атаумен немесе бір атау бірнеше мағынада беріледі.
- **Барлық білімді бір ғана модельге күштеп орналастыру.**
- **Ережелердің ерекшеліктерін, басымдығын және қайшылығын есепке алмау.**
- **Дереккөз, автор, уақыт және сенімділік туралы метадеректерді сақтамау.**
- **Үлкен тілдік модельдің жауабын тексерілген біліммен шатастыру:** модель ықтимал мәтін құрады, сондықтан маңызды факт сыртқы білім базасымен расталуы тиіс.

18. Қорытынды

Білімдерді ұсыну интеллектуалды жүйенің «не білетінін» және «қалай ой қорытатынын» анықтайды. Логикалық модель дәл формалды қорытындыға, өнімдік модель ережелік шешімге, семантикалық желі қатынастарды көрнекі көрсетуге, фрейм типтік объект құрылымына, онтология ортақ семантика мен

автоматты жіктеуге, ықтималдық тәсіл белгісіздікке, ал векторлық модель мағыналық ұқсастыққа тиімді. Практикалық жүйелер көбіне осы тәсілдердің бірнешеуін біріктіреді.

Дәрістің негізгі тұжырымы: Білімді ұсыну форматын таңдау — жай ғана дерек сақтау технологиясын таңдау емес. Бұл жүйе қандай сұрақтарға жауап бере алатынын, қандай қорытындыны негіздей алатынын және пайдаланушыға шешімін қаншалықты түсіндіре алатынын анықтайтын **жобалық шешім**.

Бақылау сұрақтары

1. Дерек, ақпарат және білім ұғымдарының айырмашылығы неде?
2. Білімдерді ұсыну жүйесіндегі синтаксис, семантика және инференция қандай рөл атқарады?
3. Модель не үшін қажет? Төрт негізгі себебін атаңыз.
4. Модельге қойылатын талаптар қандай мәселелерді шешеді?
5. Пікірлер логикасы мен предикаттар логикасының айырмашылығы қандай?
6. Тура және кері тізбектеу қай жағдайларда қолданылады? Өмірден мысал келтіріңіз.
7. Семантикалық желідегі мұрагерлік қалай жұмыс істейді?
8. Фреймнің слоттары нені сипаттайды?
9. Онтология мен білім графының айырмашылығы қандай?
10. RDF үштігінің үш бөлігі қандай?
11. Ықтималдық модель мен айқын емес логика қандай әртүрлі белгісіздікті көрсетеді?
12. Embedding неліктен формалды логикалық факт болып саналмайды?
13. Неліктен гибридтік білім ұсыну архитектурасы қажет болуы мүмкін?
14. Берілген міндетке модель таңдағанда қандай өлшемдер ескеріледі?

Ұсынылатын әдебиеттер мен ресурстар

1. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Knowledge Representation бөлімдері.
2. Brachman R., Levesque H. Knowledge Representation and Reasoning.
3. Baader F. және т.б. The Description Logic Handbook.
4. W3C құжаттары: RDF 1.1 Concepts, OWL 2 Overview, SPARQL 1.1 Query Language.
5. Protégé ресми пайдаланушы құжаттамасы.
6. CLIPS Reference Manual және Drools Documentation.
7. Neo4j Cypher Manual және Apache Jena Documentation.