Част 1. Формализми за представяне и използване на знания

1.1. Представяне и използване на знания в системите с изкуствен интелект – основни понятия

Работата със знания е една от съществените отличителни черти на програмните системи с изкуствен интелект (СИИ). Според едно от популярните определения *интелектът е способност за формулиране, натрупване и използване на знания* (Intelligence is applied knowledge). Терминът "знания" в СИИ се използва за означаване на кодирания опит на агентите. Опитът е източникът на знания за решаването на задачи. Чрез определението "кодиран" се означава обстоятелството, че знанията са формулирани, записани и готови за използване.

Съществена за СИИ е възможността за преход от работа с данни към работа със знания. Традиционните програмни системи работят с информация, организирана във вид на бази от данни (БД), докато за системите с изкуствен интелект и по-точно за т. нар. ***системи, основани на знания*** (СОЗ; Knowledge-based Systems, KBS) е характерна работата с ***бази от знания*** (БЗ). Има фундаментална разлика между възможностите, които предоставят тези два типа системи за съхранение и предоставяне на информация.

При БД е възможно извличане само на такава информация, която е представена в явен вид в базата. При БЗ е възможно извършването на разсъждения (извод), в резултат на което *може да се генерира нова информация, която не присъства в явен вид в базата*. В този смисъл по принцип БД не могат да представят и системите за управление на БД не могат да обработват непълна информация, докато за БЗ такова ограничение не съществува.

1.1.1. Същност на знанията в СОЗ

В общ *енциклопедичен план* знанието обикновено се определя като система от съждения с принципна и единна организация, основана на обективни закономерности. В *обективен смисъл* (т.е. като резултат от определен познавателен процес) знанието се противопоставя на заблуждението. В *субективен смисъл* знанието представлява мнение, вяра, убеждение (по реални причини) в истинността на наблюдението.

В СОЗ знанията представляват ***съвкупност от твърдения, представящи мнението, вярата и убежденията на т. нар. когнитивен агент***.

Характерна за СОЗ е възможността за едновременна работа със знания (най-често знания за областта/областите на приложение на системата) и данни (данни за задачата, решавана текущо от системата).

Различия между данните и знанията в СОЗ:

- в степента на общност (и постоянност/изменяемост);
- по начина на използване;
- наличие на класификационно-йерархична структура на знанията (основана на родово-видови отношения от тип обект – клас, клас – суперклас, тип – подтип, ситуация – подситуация и т.н.).

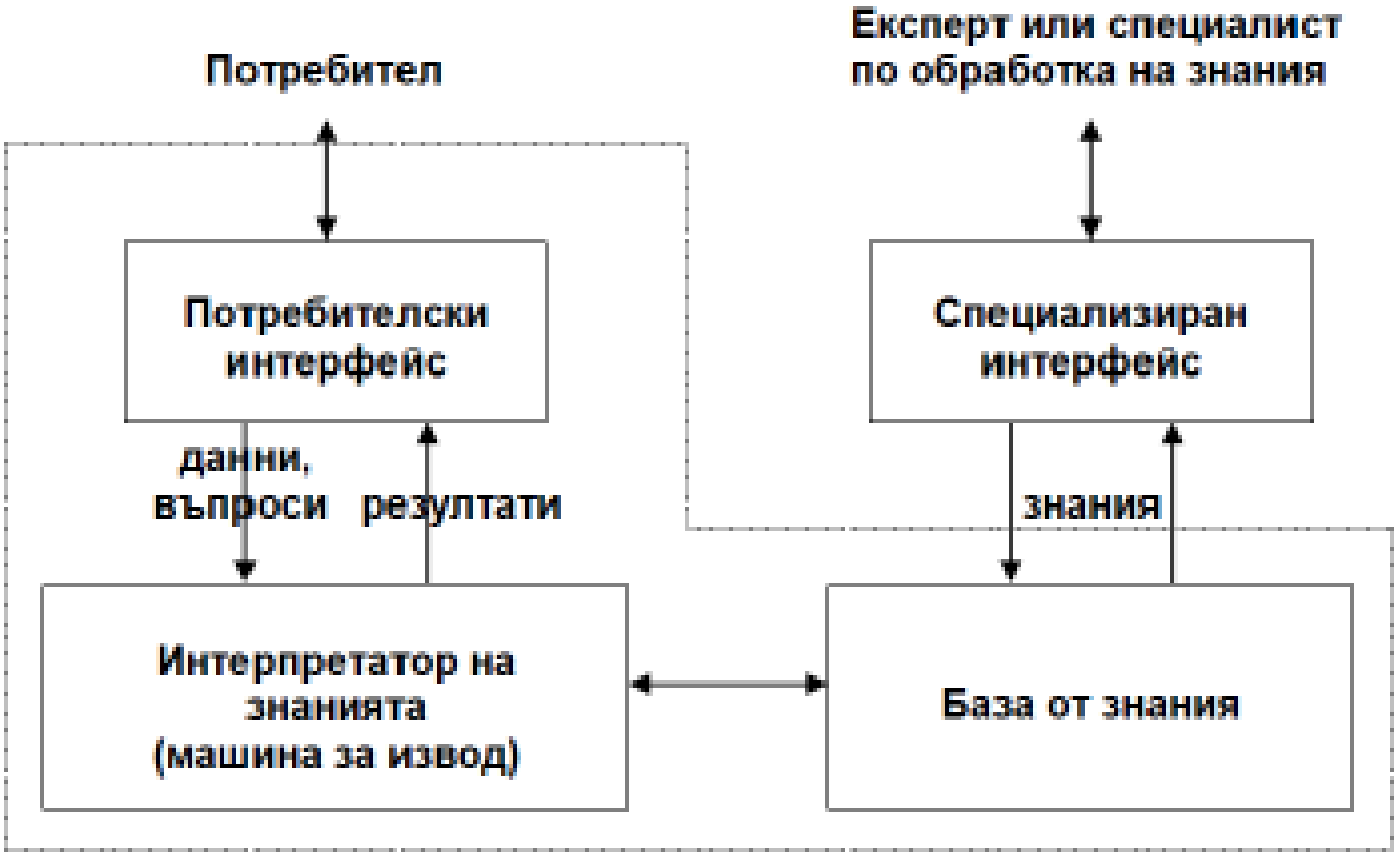
В общия случай не може да се посочи точна граница между данните и знанията в СОЗ.

1.1.2. Архитектура на СОЗ

Основава се на приемане (символен подход) или отхвърляне (конекционистки подход) на т. нар. *хипотеза за представянето на знанията* (knowledge representation hypothesis).

Основно за системите, следващи идеологията на символния подход, е изискването за максимално отделяне на знанията на системата от програмите, които ги използват (интерпретират). Те включват в състава си два обособени компонента (Фигура 1):

- ***база от знания*** (knowledge base);
- ***машина за извод*** (inference engine) или интерпретатор на знанията (knowledge interpreter).



Фигура 1. Архитектура на СОЗ (символен подход)

От съдържателна гледна точка могат да бъдат обособени следните типове явно представяни знания в СОЗ:

- знания за обектите и фактите в предметната област (фактологически знания);
- знания за връзките (релациите) между обектите и фактите;
- метазнания (поддържащи знания – знания/обяснения за основанията за формулирането на определени „порции“ от знанията за предметната област (предметните знания); стратегически знания – знания за приложимите методи за използване/интерпретиране на предметните знания и др.).

Изборът или конструирането на конкретен формализъм за представяне и използване на знания (ПИЗ) обикновено се съобразява със следните изисквания:

- коректност на правилата за извод (правилата за извод да бъдат такива, че ако явно зададените знания в БЗ са верни, то всички извлечени (изведени) от тях знания също да са верни);
- естественост на представянето (по отношение на терминологията и възприетата в специализираната литература структура на знанията за съответната предметна област);
- ясна семантика (значението на правилно построените изрази да бъде добре определено, т.е. всеки правилно построен израз да може да се интерпретира еднозначно);
- модулност на представянето;
- ефективност (по отношение на изискванията за памет и време).

1.1.3. Основни типове формализми за ПИЗ

Съществуват два основни типа формализми за ПИЗ: *формализми от декларативен тип* (декларативни формализми – позволяват естествено представяне на знания, с чиято помощ да се отговори на въпроса „*какво?*“ – какво е известно за съответната предметна област, задача и т.н.) и *формализми от процедурен тип* (процедурни формализми – насочени са към отговор на въпроса „*как?*“ – как се решават задачи от даден тип и т.н.). При декларативните формализми основната тежест пада върху представянето на знанията; при процедурните формализми съществен е начинът на използване на знанията. При декларативните формализми знанията се представят явно, в декларативен формат, а при процедурните формализми знанията се съдържат в процедурите (частите) на някаква програмна система. Процедурните формализми дават възможност за по-голяма ефективност при използването на знанията, но при тях измененията в БЗ стават по-трудно, отколкото при декларативните.

1.2. Представяне и използване на знания чрез системи от продукционни правила

Формализмът на **системите от продукционни правила** по същество е декларативен, с елементи на процедурност на по-ниско равнище. Негова основна характеристика е декомпозирането на знанията на малки части (продукционни правила) от типа условие – следствие (ситуация – действие).

Системите, основани на правила, включват в състава си три основни компонента:

- работна памет (контекст) – съдържа в подходящ вид данните за конкретната задача (входните данни, евентуално изменени в процеса на решаване на задачата);

- база от правила – съхранява знанията за предметната област, представени под формата на импликации от вида (<име_на_правило> <лява_страна> <дясна_страна>);
- интерпретатор на правилата – машината за извод на съответната СОЗ. Представлява програмна система, чието основно предназначение е да приложи описаните чрез правилата знания върху данните за конкретната задача.

1.2.1. Работна памет

Работната памет (РП) съдържа данните за решаваната задача, които са установени (известни) към текущия момент. Тези данни се формират от два източника:

- от потребителя (зададени като условие на задачата или получени като отговор на допълнителен/уточняващ въпрос на интерпретатора);
- от интерпретатора (получени в процес на логическия извод). Съдържанието на РП обикновено се модифицира по време на работния цикъл на интерпретатора на правилата, като се добавят резултати от работата на интерпретатора или се изключват или модифицират данни от РП.

Работната памет е списък от т. нар. *елементи на работната памет* (ЕРП). Структурата на ЕРП е близка до (дори обикновено съвпада със) структурата на елементите на левите страни на правилата (т. нар. *елементи на условието* в левите страни на правилата). Най-често ЕРП описват отделни обекти от предметната област чрез принадлежността им към съответни типове и множеството от стойности на определени атрибути (свойства/характеристики). В такъв случай всеки ЕРП е вектор от вида

$$(type \ \ attribute_1: value_1 \ \ . \ . \ . \ \ attribute_n: value_n),$$

където *type* и всички $attribute_i$ и $value_i$ са атоми.

Примери за ЕРП:

(person name: peter age: 23 hometown: sofia)

(student name: peter department: computerScience)

(goal task: putDown importance: 5 urgency: 1)

Декларативното разбиране на всеки ЕРП е като твърдение, което се отнася до квантор за съществуване:

$$\exists x[type(x) \wedge attribute_1(x) = value_1 \wedge attribute_2(x) = value_2 \wedge \cdots \wedge attribute_n(x) = value_n].$$

Обектите (индивидите), за които са валидни тези твърдения, не се посочват явно в ЕРП.

1.2.2. Продукционни правила

Базата от правила е линейна поредица от продукционни правила, всяко от които има свое уникално име, лява страна и дясна страна. По същество правилото описва импликация между лявата и дясната си страна, с помощта на която се моделира причинно-следствена връзка между двете страни. Възможно е продукционните правила да имат и други компоненти като приоритет, цена на изпълнението на дясната страна на правилото и др.

Лявата страна (условието, антецедентът) на едно продукционно правило представлява поредица от „елементарни“ условия (наричани още елементи на условието, ЕУ), като по подразбиране се предполага конюнктивна връзка между отделните елементи на условието. ЕУ могат да бъдат положителни или отрицателни (ще смятаме, че отрицателните ЕУ имат вида *~cond*, където *cond* е положителен ЕУ).

Положителните ЕУ обикновено са вектори от вида

(*type* *attribute*$_1$: *specification*$_1$ . . . *attribute*$_n$: *specification*$_n$),

където всеки спецификатор *specification*$_i$ може да бъде:

- атом;
- променлива;
- израз, който може да бъде оценен (такива изрази ще заграждаме в квадратни скоби “[ ]”);
- тест (сравнение, заградено в “[ ]”);
- конюнкция ($\wedge$) на спецификатори, дизюнкция ($\vee$) на спецификатори или логическо отрицание ($\neg$) на спецификатор.

Дясната страна (следствието, консеквентът) на едно продукционно правило обикновено описва поредица от действия и има процедурна интерпретация. Списъкът на допустимите действия в десните страни правилата обикновено включва поне следните видове действия:

- ADD *pattern*: означава добавяне към работната памет на нов ЕРП, специфициран от образеца *pattern*;
- REMOVE *i*: означава изключване от работната памет на ЕРП, който е съпоставен успешно с *i*-тото ЕУ в лявата страна на правилото (виж т. 1.2.3);
- MODIFY *i* (*attribute specification*): означава модифициране на ЕРП, който е съпоставен успешно с *i*-тото ЕУ в лявата страна на правилото, като съществуващата стойност на атрибута *attribute* в този ЕРП се заменя със *specification*.

1.2.3. Интерпретатор на правилата

Алгоритъмът, по който работи интерпретаторът на правилата, е итеративен, като всяка стъпка от работния цикъл на интерпретатора включва две фази:

- *избор на правило*, което да се изпълни (първа фаза);
- *изпълнение на избраното правило* (втора фаза).

По същество задачата за избор на правило за изпълнение се свежда до търсене в базата от правила по някакъв образец. Според образеца на търсене и съответно според начина на интерпретация се говори за две основни стратегии на работа на интерпретатора на правилата:

- *прав извод* (forward chaining) или също *извод, управляван от данните* (data-driven inference);
- *обратен извод* (backward chaining) или *извод, управляван от целите* (goal-driven inference).

При правия извод дадено продукционно правило с лява страна A и дясна страна C се интерпретира по следния начин:

(assert A) $\Rightarrow$ (assert C),

т.е. „ако твърдението A е вярно, то е вярно също и твърдението C“.

При обратния извод едно продукционно правило с лява страна A и дясна страна C се интерпретира по следния начин:

(goal C) ⇒ (goal A),

т.е. „ако трябва се покаже, че е изпълнена целта C, достатъчно е да се покаже, че е изпълнена целта A“.

При извода, управляван от данните, в първата фаза на всяка стъпка от работния си цикъл интерпретаторът на правилата търси правило, чиято лява страна се удовлетворява от данните в работната памет. Лявата страна на едно продукционно правило се удовлетворява от работната памет, когато за всеки от положителните ЕУ в нея съществува поне един ЕРП, съпоставим с този ЕУ, и за всеки от отрицателните ЕУ не съществува съпоставим с него ЕРП. В такъв случай правилото е изпълнимо на текущата стъпка от работния цикъл на интерпретатора.

Съгласно предположенията за синтаксиса на ЕУ и ЕРП, които направихме по-горе, един положителен ЕУ е съпоставим с даден ЕРП, когато ЕРП и ЕУ имат (са от) един и същ тип и стойностите на атрибутите в ЕРП удовлетворяват спецификациите (ограниченията) на същите атрибути в ЕУ. Променливите, участващи в едно правило, обикновено имат област на действие, съвпадаща с това правило. В началото на процеса на съпоставяне със съдържанието на работната памет всички променливи в лявата страна на разглежданото правило са свободни (т.е. не означават конкретни стойности) и стойностите на съответните атрибути от ЕРП са съпоставими с тях (удовлетворяват ограниченията, определени от тях). При съпоставяне с някаква стойност *value* променливата *v* се свързва с *value* и всички включвания на *v* в рамките на областта й на действие означават тази стойност и са съпоставими само с нея. Отрицателният ЕУ *~cond* е съпоставим с даден ЕРП точно когато ЕУ *cond* не е съпоставим с този ЕРП.

Множеството от продукционни правила, чиито леви страни се удовлетворяват от работната памет на дадена стъпка от работния цикъл на интерпретатора на правилата, образуват т. нар. *конфликтно множество*. Ако конфликтното множество е празно, работата на интерпретатора се прекратява. В такъв случай потребителят може да разгледа текущото съдържание на работната памет, която включва в частност резултатите от работата на интерпретатора. Ако конфликтното множество съдържа точно един елемент, този елемент се избира за изпълнение на втората фаза от текущата стъпка от работния цикъл на интерпретатора на правилата.

Ако конфликтното множество съдържа повече от един елемент, се прилага текущо валидната *стратегия за разрешаване на конфликтите* – метод за избор на правило от конфликтното множество (например: първото правило от конфликтното множество, което не е изпълнявано до момента; правило, което използва най-скоро записани в работната памет ЕРП и др.). Изпълнението на избраното правило (втората фаза от текущата стъпка от работния цикъл на интерпретатора на правилата) най-често е свързано с изменение на съдържанието на работната памет – добавяне на нови ЕРП, изключване или модификация на съществуващи ЕРП. Това от своя страна прави възможно избирането и изпълнението на друго правило и т.н.

При извода, управляван от целите, интерпретаторът на правилата търси правило, чиято дясна страна съдържа действие, изпълнението на което би довело до записване в работната памет на ЕРП, съпоставим с някаква зададена (най-често от потребителя) цел. След това интерпретаторът анализира лявата страна на намереното правило, като тези ЕУ в лявата страна на правилото, които в текущия момент не се удовлетворяват от работната памет, се разглеждат като нови цели (подцели), за проверката на всяка от които се извършва съответен обратен извод. Този процес продължава, докато текущите цели се удовлетворят от данните в работната памет или докато се изчерпат възможностите за тяхното удовлетворяване. При необходимост се осъществява възврат (backtracking). Възможно е също така да бъдат конструирани и задавани на потребителя уточняващи въпроси относно липсващи стойности на атрибути в някои ЕРП.

1.2.4. Примери

Тук ще разгледаме два примера за прав извод върху системи, основани на продукционни правила, при които не е необходимо определяне и използване на стратегия за разрешаване на конфликтите.

Пример 1

Имаме три тухли с различни размери, поставени на купчина една върху друга. Определени са три различни позиции, в които искаме да поставим тухлите с помощта на ръката на интелигентен робот. Означаваме тези позиции като 1, 2 и 3. Нашата цел е да поставим тухлите в тези позиции по реда на техния размер, като най-голямата е в позиция 1, а най-малката е в позиция 3.

Приемаме, че при стартиране на интерпретатора на правилата работната памет съдържа следните елементи:

```
(counter value:1)
(brick name:A size:10 position:heap)
(brick name:B size:30 position:heap)
(brick name:C size:20 position:heap)
```

В този случай желаният резултат е: тухла B се намира на позиция 1, тухла C – на позиция 2 и тухла A – на позиция 3. Целта може да бъде постигната с използване на две подходящи продукционни правила.

Първото правило ще се грижи да постави най-голямата текущо налична тухла в ръката на робота, а второто ще постави тухлата, която в момента се намира в ръката на робота, на първата (по реда на номерата) свободна позиция:

```
1. IF (brick position:heap name:n size:s)
      ~(brick position:heap size:{> s})
      ~(brick position:hand)
   THEN MODIFY 1 (position hand)

2. IF (brick position:hand)
      (counter value:i)
   THEN MODIFY 1 (position i)
        MODIFY 2 (value [i+1])
```

В този пример на всяка стъпка от работния цикъл на интерпретатора е изпълнимо не повече от едно от двете правила – първото правило изисква ръката на робота да е празна, а второто изисква в нея да има тухла.

Проследяване на работата на интерпретатора на правилата при извод, управляван от данните:

1. Правило 2 не е изпълнимо, тъй като в ръката на робота не държи тухла (няма тухла с позиция hand). Правило 1 е изпълнимо само за тухла B, тъй като тя е единствената, за която не съществува по-голяма тухла в купчината. При съпоставяне на Правило 1 с работната памет променливата n се свързва с B и s се свързва с 30. ЕРП, описващ тухлата B, се модифицира и получава следния вид:

```
(brick name:B size:30 position:hand)
```

2. Сега вече Правило 1 не е изпълнимо, но е изпълнимо Правило 2. В резултат на неговото изпълнение настъпват промени в ЕРП, описващи тухлата B и брояча (номера на първата свободна позиция) – те стават съответно

(brick name:B size:30 position:1) и
(counter value:2)

3. Тухла B вече не е в купчината, а заема позиция 1, затова сега Правило 1 е изпълнимо за тухла C, чиято позиция се променя:

(brick name:C size:20 position:hand)

4. Тази стъпка е подобна на стъпка 2. Правило 2 сега променя позицията на тухла C на 2 и увеличава брояча до 3:

(brick name:C size:20 position:2) и
(counter value:3)

5. Сега в купчината е останала само тухла A, така че Правило 1 е изпълнимо за нея и в резултат тя се премества в ръката на робота:

(brick name:A size:10 position:hand)

6. Правило 2 се изпълнява отново и този път тухла A се поставя на позиция 3:

(brick name:A size:10 position:3) и

(counter value:4)

7. Сега, тъй като няма тухли в купчината и в ръката на робота, не е изпълнимо никое от двете правила. Конфликтното множество е празно и работата на интерпретатора на правилата приключва. Работната памет има следното съдържание:

```
(counter value:4)
(brick name:A size:10 position:3)
(brick name:B size:30 position:1)
(brick name:C size:20 position:2)
```

Пример 2

Ще покажем система от продукционни правила, с помощта на които може се определи колко са дните в дадена година:

```
1. IF (wantDays year:n)
   THEN REMOVE 1
        ADD   (year   mod4:[n%4]   mod100:[n%100]
mod400:[n%400])
2. IF (year mod400:0)
   THEN REMOVE 1
        ADD (hasDays days:366)
3. IF (year mod100:0 mod400:{≠ 0})
   THEN REMOVE 1
        ADD (hasDays days:365)
```

```
4. IF (year mod4:0 mod100:{≠ 0})
   THEN REMOVE 1
        ADD (hasDays days:366)
5. IF (year mod4:{≠ 0})
   THEN REMOVE 1
        ADD (hasDays days:365)
```

Преди стартирането на интерпретатора на правилата работната памет трябва да съдържа един елемент от вида

`(wantDays year:n)`,

където n е годината, броят на дните в която трябва да бъде определен.

Като резултат от работата на интерпретатора в работната памет се записва елемент от вида

`(hasDays days:m)`,

където m е търсеният брой дни.

Проследяването на работата на интерпретатора на правилата оставяме за самостоятелно упражнение.

1.2.5. Обща оценка на системите от продукционни правила като формализъм за представяне и използване на знания

Най-съществените силни страни на системите от продукционни правила като формализъм за ПИЗ могат да бъдат формулирани по следния начин:

(1) естественост на представянето на експертни знания

Много често експертите в различни предметни области формулират знанията, които трупат в резултат на професионалния си опит, във вид на собствени правила, които обикновено не могат да бъдат намерени в учебниците и другите специализирани литературни източници.

(2) модулност на базата от знания

По удобен и естествен начин може да се извършва изграждането на базата от правила на части. Сравнително лесно и бързо се извършва откриването и отстраняването на грешки в базата от правила.

(3) възможност за лесно генериране на обяснения на получените резултати

Може да бъде съхранена т. нар. *история на извода* (поредицата от левите и десните страни на правилата, изпълнени последователно в рамките на работния цикъл на интерпретатора), с помощта на която сравнително лесно може да бъде генериран текст на естествен език, който обяснява *как* е достигнато до полученото решение.

Като недостатъци на този формализъм и най-съществени проблеми при неговото използване следва да бъдат посочени:

(1) недостатъчна изразителна сила

Синтаксисът на продукционните правила позволява с тяхна помощ да бъдат представяни знания, които се описват чрез импликация между две твърдения. Нещо повече, десните страни на правилата не могат да съдържат дизюнкции от действия (дизюнкции от следствия) и логическо отрицание.

(2) потенциална неефективност на работата на интерпретатора на правилата

Както определянето на конфликтното множество, така и разрешаването на конфликтите (изборът на правило от конфликтното множество, което да бъде изпълнено на текущата стъпка от работния цикъл на интерпретатора) са източници на потенциална неефективност на работата на интерпретатора на правилата. Така, както беше описан по-горе, процесът на формиране на конфликтното множество (съпоставянето на ЕУ в левите страни на правилата с ЕРП) е сравнително трудоемък заради необходимостта на многократно извършване на голям брой опити за съпоставяне. В действителност броят на съпоставянията може да бъде намален чувствително с помощта на популярния алгоритъм RETE, предложен и реализиран от Чарлс Форджи (1982 г.) във връзка с разработването на първите среди за програмиране на езика OPS5.

Идеята на алгоритъма RETE се основава на два прости факта. Първо, обикновено много правила имат съвпадащи леви страни или поне съвпадащи ЕУ. Следователно може да се намери начин работата по проверката на такива леви страни или ЕУ да се извършва само веднъж. Второ, при всяко изпълнение на някакво правило работната памет се променя съвсем малко. Следователно голяма част от ЕУ, които са били верни (съпоставими с подходящи ЕРП) на предишната стъпка, ще бъдат верни и на текущата и обратно – повечето от ЕУ, които не са били верни преди, няма да бъдат верни и след съответната промяна. С други думи, усилията по съпоставянето могат да бъдат съсредоточени върху тези ЕРП, които напоследък са били изменени, включени или изключени от работната памет.