Language for description of worlds

Dimiter Dobrev
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
d@dobrev.com

We will reduce the task of creating AI to the task of finding the right language for description of the world. This language will not be a programming language because the programming languages describe only computable functions, while this language will describe a slightly wider class of functions. Another feature of this language will be that the description can be divided into separate modules. This will allow us to search the world description automatically by detecting it module by module. Our approach to creating this new language will be to start from one particular world and write a description of that particular world. Our idea is that the language that can describe this particular world will be appropriate to describe arbitrary world.

Keywords: Artificial Intelligence, Language for description of worlds, Event-Driven Model, Definition of Property, Definition of Algorithm.

Въведение

Задачата е да разберем света. За да го разберем, трябва да го опишем, а за да го опишем ни е нужен специален език за описание на светове.

За да създадем този специален език, първо трябва да си отговорим на въпроса: „Какво представлява светът?“ Ще видим, че света може да си го мислим като някаква функция от $\mathbb{R}$ в $\mathbb{R}$. Ако тази функция беше изчислима, тогава езикът за описание на светове щеше да е някакъв език за програмиране. Ще се окаже, че изчислимите функции не са ни достатъчни и затова ще опишем един малко по-широк клас от функции.

Търсеният от нас език няма да е език за програмиране, но подхода ни да го намерим ще е същият като подхода при създаването на нов език за програмиране. Ще започнем от една конкретна задача (в случая един конкретен свят) и ще се опитаме да напишем програма за тази задача (в случая ще напишем описание на конкретния свят).

Светът е една функция, която ще наречем f, а описанието на f е функция, която ще наречем g. Това, което ще търсим, е функцията g. По-точно ще търсим описание написано на нашия нов език за описание на светове. Разликите между f и g са:
1, f може да е неописуема.
2. f може да е много по-сложна от g.
3. f работи с конкретни състояния на света, а g работи с представата на агента за състоянието на света.
4. f е детерминирана, докато g в общия случай не е.

Представата за състоянието на света ще се състои от няколко множества, всяко с различна вероятност. Представата за наблюдението ще се състои от няколко възможности, всяка с различно наблюдение, различна вероятност и различна нова представа за състоянието на света.

Ще кажем кога g е коректна и кога е минимална. Ще казваме, че сме разбрали света, когато сме намерили коректна и минимална функция g.

Колко агента ще има в света, който ще разгледаме? Във всеки свят има поне един агент и това е главният герой. Ще разгледаме два варианта. При първия вариант главният герой е сам, а при втория, освен него, има още един агент, който е неговият противник.

Ще покажем, че от езика за описание на светове, лесно можем да направим Изкуствен Интелект. За да разберем света, трябва да намерим неговото описание. Бихме могли да търсим, като изброим всички възможни описания (exhaustive search or brute-force search), но това търсене не би успяло. Ние ще подходим по-интелигентно като предположим, че описанието на света е разделено на отделни модули, които могат да бъдат откривани един по един.

Конкретният свят, който ще опишем, ще бъде светът, в който агентът играе шах. Написали сме програмата [7], която описва този свят. Това е компютърна програма написана на езика [8]. Можете да стартирате тази програма и да видите колко просто се е получило описанието на този свят (на играта шах). Описанието се състои от 24 модула, които представляват Event-Driven модели (това са ориентирани графи с по десетина състояния всеки). Освен ED моделите имаме още две подвижни следи (т.е. два масива). ED моделите са три вида (5 зависимости, 9 алгоритъма и 10 свойства, което прави общо 24). Освен 24-те модула и двата масива ни се е наложило да добавим още седем прости правила, които допълнително описват света.

В света, който ще разгледаме, агентът няма да вижда всичко. Той няма да вижда цялото табло, а само едно квадратче от таблото. Тоест, ще разглеждаме свят, в който има Partial Observability. Това е интересният случай, защото, когато агентът вижда всичко, задачата не е чак толкова интересна. Когато агентът не вижда всичко, на него му се налага да си представи това, което не вижда в момента.

За да ограничим броя на командите на агента ще използваме едно просто кодиране. Ще разделим стъпките на три (първа, втора и трета). Коя ще е командата ще зависи от това на коя стъпка агентът я е подал.

Това кодиране ще ни даде и първата зависимост. Това ще е зависимостта 1, 2, 3. Тази зависимост, както и всички други зависимости в света, ще бъде представена с Event-Driven модел. ED моделите представляват ориентиран граф с няколко състояния и стрелки между състоянията. На стрелките съответстват събития, които от своя страна се представят като конюнкции. Събитията са истина в едни моменти и лъжа в други.

Ще предполагаме, че в някои от състоянията на Event-Driven моделите се случва нещо специално. Това специалното, което се случва, ще го наричаме „следа“. Следата е смисълът на модела, защото, ако всички състояния бяха еднакви, тогава щеше да е без значение в кое състояние се намираме.

Следващите две зависимости ще са Horizontal и Vertical. Тези зависимости ще ни дадат $\boldsymbol{x}$ и $\boldsymbol{y}$ координатите на наблюдаваното квадратче. Декартовото произведение на тези два модела ще е модел на цялото шахматно табло.

Следата на това декартово произведение ще бъде позицията на шахматната дъска. Тази следа ще е „подвижна“, защото позицията на дъската не е постоянна, а се променя.

След като описахме шахматната дъска, трябва да опишем движението на фигурите. За целта са ни нужни алгоритмите. Какво е алгоритъм? Зависимостите ги представихме като Event-Driven модели. Алгоритмите ще ги представим по същият начин. За нас алгоритмът ще е последователност от действия (събития). Програмите изчисляващи функции от $\mathbb{N}$ в $\mathbb{N}$, ще са алгоритми според нашата дефиниция, но освен това дефиницията ще включва и други алгоритми, като готварските рецепти например.

Ще опишем алгоритмите на движение на различните шахматни фигури. Например, алгоритмът на царя е по-прост, защото е детерминиран, докато алгоритмът на топа е по-сложен, защото в него има недетерминирани разклонения.

Много малко са авторите, които се опитват да дефинират понятието алгоритъм. Единствените опити за дефиниция на алгоритъм, които са ни известни са направени от Moschovakis [1, 2].

За да покажем, че нашата дефиниция на алгоритъм е смислена, трябва да представим машината на Тюринг като Event-Driven модел. По този начин ние ще покажем, че нашата дефиниция покрива това, което другите автори влагат в понятието алгоритъм. Разбира се, нашата дефиниция ще даде повече и ще излезе извън класа на функциите от $\mathbb{N}$ в $\mathbb{N}$.

Ще представим машината на Тюринг като отделен свят. Описанието на този свят ще се състои от два Event-Driven модела. Първият модел ще опише безкрайната лента. Вторият модел ще опише самата машина на Тюринг. Алгоритмът на практика ще е вторият ED модел, защото първият просто описва безкрайната лента. За да имаме алгоритъм, той трябва да има нещо върху което да се прилага. За да имаме машина на Тюринг, трябва да имаме безкрайна лента, върху която да работи машината.

Другото фундаментално понятие, което се въвежда в тази статия, това е понятието свойство. Това понятие също ще е зависимост и също ще се представя с Event-Driven модел. Ще използваме свойствата, за да кодираме входа на агента. Входът се състои само от три букви, но с тези три букви ние можем да представим безбройно много свойства.

Първият свят, който описахме, е детерминиран и ние го описахме с една функцията g, която е детерминирана. Повечето светове не са детерминирани и затова нашият език за описание на светове трябва да може да описва и недетерминирани функции. Въвеждането на втори агент в света веднага го прави недетерминиран, защото не се знае как точно ще се държи този втори агент.

В описанието на света ще има различни събития. Повечето ще са свързани с входа и изхода. Някои събития ще са свързани със състоянието на някой ED модел. Ще има и случайни събития, които се случват с някаква вероятност. Ще има дори и невъзможни събития. Това ще са събития, които никога не се случват, но които може да се случат в нашето въображение. Тези събития ще ни помогнат, за да опишем света. Макар и невъзможни, тези събития ще участват в различни алгоритми.

В първия свят, който описахме, има само един агент, който играе сам срещу себе си. Ще опишем втори вариант на играта шах, при който, освен главния герой, има още един агент. Важното е, че състоянието на света ще е различно за различните агенти. Състоянието на света е контекстът на агента. Контекстът е: „кой“, „къде“, „кога“. Може да предположим, че за всички агенти контекстът „кога“ е един и същи, но „къде“ може да е различно за различните агенти. „Кой“ също може да е различно, защото един агент може да е мъж, а друг да е жена. Един може да играе с белите, а друг с черните.

Написахме емулираща програма за първия свят. Можем ли да напишем такава програма и за втория свят? В общия случай не можем, защото в този свят има твърдения от вида „този алгоритъм може да бъде изпълнен“ и операции от вида „агент изпълнява алгоритъм“. Това в общия случай са неразрешими твърдения и неизчислими операции. Казахме, че светът в общия случай може да бъде неизчислима функция и затова понякога светът може да се опише, но не може да се направи програма, която да емулира описаното. В частния случай с играта шах нещата са различни. В този частен случай може да се направи емулираща програма, защото в шаха всичко е крайно и всичко е изчислимо.

Какво представлява светът?

Първо ще кажем какво вижда агентът. Той вижда един поток от информация или една редица от вход и изход (наблюдения и действия). Тази редица изглежда така:

$$v_0, a_1, v_1, a_2, v_2, a_3, v_3, \ldots$$

Този поток от информация е проявлението на света, но зад това проявление се крие същността, която агентът трябва да разбере.

Какво представлява тази същност? Това е една функция, която генерира този поток от информация. По-точно функция, която взима като аргумент действията (изходите) и връща като стойност наблюденията (входовете).

$$V_i = f(a_i)$$

Тук описахме нещата твърде опростено, като предположихме, че функцията *f* няма памет и не зависи от това, което се е случило до момента. Предположихме, че функцията *f* зависи единствено и само от последното действие на агента. Това опростяване е прекалено и с него се губи същността на задачата. Въпреки това ние споменаваме за това възможно опростяване, защото то се използва в почти всички статии за ИИ. В литературата, когато е на лице това опростяване се казва, че имаме Full Observability, а когато го нямаме се казва, че разглеждаме случая на Partial Observability. В тази статия ние ще предполагаме, че функцията *f* има памет. Тоест, ще предполагаме, че разглеждаме случая на Partial Observability.

Сега функцията *f* ще изглажда така:

$$<V_i, s_{i+1}> = f(a_i, s_i)$$

Тук s_i е състоянието на света. Състоянието на света е паметта на света. Но функцията f е самият свят, което означава че s_i е паметта на функцията f. Тоест, s_i е това, което светът е запомнил в момента i.

Функцията f ще взима като аргумент действието на агента и текущото състояние на света и ще върне две неща (наредена двойка). Първото, което ще върне, е новото наблюдение, а второто е новото състояние на света (новата стойност на паметта).

Можем ли да кажем, че функцията f описва света и че тя е неговата същност? Не съвсем, трябва да добавим още и s_0. Тоест, трябва да добавим и началното състояние на света (или началната стойност на паметта). Сега вече можем да кажем, че двойката $<f, s_0>$ е светът или поне, че това го описва достатъчно точно.

За да стане картината по-цветна и по-интересна ще допуснем, че не всички действия на агента са разрешени. Тест, ще допуснем, че в един момент са разрешени едни действия, а в друг момент са разрешени други. За целта ще предположим, че функцията f не е тотална и за някои двойки от действие и състояние на света не е дефинирана.

Ще предполагаме, че действието и наблюдението на агента са букви от крайна азбука, а състоянието на паметта е реално число. Тоест, ще предполагаме, че информацията която се приема и предава за една стъпка е ограничено количество, а паметта на света е с размер не по-голям от континуум. Размерът на континуума ни е нужен, за да може в паметта на света да се запише една безкрайна редица от нули и единици (броят на тези редици е континуум). По-голям размер от континуум не ни е нужен, защото всяка функция f, чиято памет е повече от континуум, може да се представи като функция с континуум памет. Това представяне, ще представлява една функция h, която изпраща множеството от състоянията на света S в подмножество на S, което е с размер не повече от континуум. Това представяне ще има свойството, че няма да има разлика дали сме тръгнали от s_0 или от $h(s_0)$. Няма да има разлика от гледна точка на редицата от вход и изход, макар и редицата от състоянията на света, през които ще се премине, формално да е друга. Тоест, това представяне ще е рестрикцията на f върху $h(S)$.

Как ще опишем света?

Както казахме, светът е една функция f. Ние няма да търсим директно функцията f, а ще търсим нейно приближение, което я описва. Има пет причини, за това да търсим приближение:

1. Функцията f може да не е описуема. Тоест, може тази функция да си няма описание, което ние да намерим.
2. Функцията f може да е описуема, но да е твърде сложна и да не можем да намерим нейното описание. Затова ще търсим нейно приближение, което да е по-просто.
3. Освен функцията f ние трябва да опишем и s_0 (началното състояние на света). Състоянията на света може да са континуум много, а нещата, които можем да опишем, са изброимо много. Затова ще заменим състоянието на света с описание на състоянието на света.
4. Функцията f взима като аргумент текущото състояние на света. Тоест, за да използваме тази функция, ние трябва да знаем точно кое е текущото състояние на света, но ние това не

го знаем. Ние имаме само някаква бегла представа за текущото състояние и ни трябва функция, която ще се задоволи с тази бегла представа.
5. Можем да приемем, че функцията *f* е детерминирана, но не можем да се ограничим само с детерминистични описания на света.

Изчислимост на света

Първият въпрос, който ще си зададем, когато описваме света, е дали функцията *f* е изчислима?

Можем да си представим света и агента като две черни кутии, които си обменят информация. Функцията, която описва агента трябва да е изчислима, защото, ако не е изчислима, няма как да напишем програма, която да я изчислява. Тоест, няма да можем да създадем ИИ. От друга страна функцията *f* (тази, която описва света) може спокойно да е неизчислима. Ние света няма да го създаваме, защото той вече е създаден и на нас ни остава задачата само да го опишем. Въпросът е можем ли да опишем неизчислима функция? Да, можем да опишем такава функция, но разбира се няма да можем да изчислим описаното.

Ние самите, когато си представяме света, си го представяме неизчислим. Вземете като пример правилото: „Ако съществува доказателство, тогава твърдението е вярно“. Това правило е неизчислимо, защото въпросът дали съществува доказателство е полуразрешим.

Ако трябваше да създаваме света, щяхме да се интересуваме не само от това дали функцията *f* е изчислима, а дали е изчислима за разумно време. Например, в играта шах всяка позиция е или печеливша или не е печеливша. Дали позицията е печеливша е изчислимо, но ако искаме да изчислим печеливша ли е началната позиция, тогава няма да ни стигне нашият живот, за да го изчислим (дори и животът на нашата галактика няма да е достатъчен). Това е при предположение, че използваме най-бързия компютър, с който разполагаме. Този феномен се нарича „комбинаторна експлозия“ и означава, че някои функции са на теория изчислими, но на практика не са.

Тоест, ние няма да се интересуваме изчислима ли е функцията *f* и доколко лесно изчислима е тя. Това означава, че езикът за описание на светове ще може да описва дори и неизчислими светове. Въпросът е, ще може ли да се описват неописуеми светове? Може ли светът да е неописуем? Да може да е неописуем, но, ако е такъв, тогава ние няма как да го опишем. Какъвто и език за описание да изберем, описанията ще са изброимо много, докато световете са много повече. Затова очевидно ще има неописуеми светове, но ние ще се ограничим с описуемите и ще предполагаме, че за всеки неописуем свят има описуем, който е достатъчно подобен на него.

Функцията *g*

Както видяхме, функцията *f* може да не е изчислима и дори и да не е описуема. Затова ние няма да търсим функцията *f*, а нейно добро приближение. Ще търсим една функция *g*, която също може да е неизчислима, но задължително ще бъде описуема. Всъщност, няма да търсим директно функцията *g*, а нейното описание написано на нашия нов език за описание на светове, който ще създадем.

Дори и функцията f да е описуема, може тя да е много сложна и да не успеем да я намерим и затова ще се задоволим с функцията g, която е нейно приближение.

Има още един проблем. Освен, че трябва да опишем функцията f, ние трябва опишем и началното състояние на света. По-точно на нас не ни е нужно началното, а текущото състояние на света, защото нас не ни интересува състоянието на света при неговото сътворение, а текущото му състояние (в [5, 6] е обяснено защо текущото състояние на света е по-важно от началното.)

Казахме, че състоянията на света може да са континуум много. За съжаление описанията са изброимо много, което значи, че можем да опишем не повече от изброимо много състояния. В общия случай няма как да опишем всички възможни състояния. Затова функцията g няма да работи със състояния на света, а с описуемите множества от състояния (някои от тези множества може да са едноелементни).

Дори и състоянията на света да бяха изброимо много, пак щяхме да предпочетем нашето описание да работи с множества от състояния. Причината е, че ние не знаем точно кое е текущото състояние на света, а имаме само някаква приблизителна представа. Тази представа ще се изобрази с множество от състояния. Когато множеството има повече от един елемент, тогава ние не знаем точно кое е състоянието на света.

Ето как ще изглежда това приблизително описание на състоянието на света.

Представа за състоянието на света

Представата за състоянието на света ще бъде приблизително описание на състоянието на света, което ще се състои от няколко (крайно много) множества от състояния, като за всяко множество ще имаме някаква вероятност. Смисълът на представата е, че състоянието е в първото множество със съответната вероятност (първата), състоянието е във второто множество с втората вероятност и т.н.

Тук ще заменим вероятностите с интервали. Смисълът е, че не можем да предполагаме, че за всяко множество знаем точната вероятност състоянието да е в това множество. Въпреки това, можем да предположим, че знаем интервала, в който тази вероятност се намира. Когато си нямаме понятие каква е вероятността, тогава интервалът ще е $[0, 1]$.

Дефиниция: Представата за състоянието на света ще бъде:

$$\{ \langle S_1, [a_1, b_1] \rangle, \dots, \langle S_n, [a_n, b_n] \rangle \}$$

S_i – това са описуеми множества от състояния на света.
$[a_i, b_i]$ – това са вероятностни интервали.

Кои ще са описуемите множества от състояния на света ще зависи от езика за описание на световете, който ще изберем, но който и език да изберем тези множества ще са изброимо много.

Функцията g, която търсим, няма да работи със състоянията на света, а с представата за това кое е състоянието на света. Това, което ще върне g е представа за наблюдението.

Представа за наблюдението

Представата за наблюдението ще ни даде очакваните наблюдения с вероятността, с която ги очакваме. За всяко очаквано наблюдение ще имаме една нова представата за състоянието на света. За различните очаквани наблюдения ще имаме различна нова представа за състоянието на света, защото новата представа зависи от това кое ще е наблюдението.

Дефиниция: Представа за наблюдението ще бъде:

$$\{ <v_1, [a_1, b_1], idea_1>, \dots , <v_n, [a_n, b_n], idea_n>, <incorrect, [a_0, b_0], idea_0> \}$$

v_i – това са възможни наблюдения (някои от възможните наблюдения).
$[a_i, b_i]$ – това са вероятностни интервали (вероятността i-тото наблюдение да бъде видяно).
$idea_i$ – това са различни представи за състоянието на света.
Последната наредена тройка ни дава вероятността това да е некоректен ход. И тук има нова представа за състоянието на света, защото, когато ходът е некоректен, състоянието на света остава същото, но представата на агента за състоянието на света се променя.

Не е необходимо всички възможни наблюдения да участват в представата за наблюдението. Ще участват само тези наблюдения, чиято вероятност е различна от нула.

Функцията g няма да ни каже точно кое ще е следващото наблюдение, а ще ни даде някаква представа за това. (Ще ни даде няколко възможности, всяка от които със съответната вероятност.) Функцията g няма да ни даде директно и новата представа за състоянието на света, защото това ще зависи от следващото наблюдение. Ако знаем, че следващото наблюдение е v_i, тогава следващата представа за състоянието на света ще бъде $idea_i$. Какво ще правим, ако следващото наблюдение го няма като вариант в представата, която ни дава функцията g? Тогава ще трябва да променим функцията g, за да я направим по-адекватна.

Общият вид на g

Общият вид на функцията g е следният:

$$Idea\_for\_view = g(a, idea)$$

Тоест, g ще взима като аргументи действие и представа за състоянието на света и ще върне като резултат представа за наблюдението.

Функцията f беше толкова проста и ясна, а функцията g, която трябва да бъде нейно приближение, я дефинирахме прекалено сложно. Не може ли f и g да са една и същата функция? Има един случай когато двете функции могат да съвпаднат. Това е когато g е детерминистична функция. Тоест, това е когато g ни казва точно кое ще е новото наблюдение и точно кое ще е новото състояние на света. Хубаво е, ако успеем да намерим такава детерминистична функция, но не винаги ще можем да го направим.

Дефиниция: Детерминистична представа за състоянието на света ще бъде:

$$\{ \langle \{s\}, [1, 1] \rangle \}$$

Тоест, има само една възможност и тя е с вероятност единица и при тази възможност множеството от състоянията на света е едноелементно (синглетон, unit set).

Дефиниция: Детерминистична представа за наблюдението ще бъде:

$$\{ \langle v, [1, 1], idea \rangle \}, \text{където } idea = \{ \langle \{s\}, [1, 1] \rangle \}$$

Тоест, има само едно възможно наблюдение и то е с вероятност единица. Освен това, следващата представа за състоянието на света е детерминистична.

Дефиниция: Функцията g ще е детерминистична, ако за всяко действие и всяка детерминистична представа за състоянието на света връща детерминистична представа за наблюдението.

Дефиниция: Детерминистично описание на света ще наричаме детерминистична функция g и детерминистична представа за началното състояние на света (или за текущото).

За да съвпадне g с f, трябва тя да е детерминистична. Това е така, защото дефинирахме f като детерминистична функция. В [4] доказахме, че всеки недетерминистичен свят може да се представи като детерминистичен. (Това го направихме, като представихме случайността като една зависимост, която е толкова сложна, че не може да бъде разбрана.) Можем да се ограничим с детерминистични светове, но не можем да се ограничим само с детерминистични описания, защото, когато една зависимост не може да бъде разбрана, тя трябва да може да се опише приблизително чрез вероятности и недетерминизъм.

Представата за състоянието на света ни дава едно недетерминистично описание на състоянието на света. Представата за наблюдението ни дава едно недетерминистично описание на следващото наблюдение и на следващото състояние на света. Тоест, чрез тази дефиниция на функцията g ние си подсигурихме тя да е недетерминистична.

Ако имаме детерминистично описание на света, тогава ще знаем точно какво ще се случи. Ще знаем точно кое ще е следващото наблюдение и точно кое ще е следващото състояние на света. Това е едно прекалено силно е предположението. В общия случай ще предполагаме, че няма детерминистично описание или поне, че такова описание не може да бъде намерено.

Кога сме разбрали света?

Ще кажем, че g е коректна, когато са изпълнени две неща. Първо, стойностите на f трябва да са вътре в резултата на g. Второ, да можем да дадем такива стойности на вероятностите, че изискванията (неравенствата) да се удовлетворяват.

Дефиниция: Функцията g е коректна, когато:
1. За всяко състояние на света s, което е в представата за състоянието на света *idea* и за всяко действие a, ако $\langle v, s' \rangle = f(a, s)$ тогава v е в представата за наблюдението $g(a, idea)$ и s' е в съответната представа за състоянието на света.

2. Има решение една конкретна система от неравенства. Системата ще се получи, като вземем *g(a, idea)*. Нека *idea* да е съставено от *n* описуеми множества, а *g(a, idea)* да е съставено от *N* описуеми множества. Нека множествата във всяка представа за състоянието на света да са непресичащи се. Тогава можем да разбием всяко от *n*-те множества на *N* парчета. Разбиването ще стане по това къде функцията *f* ще изпрати елементите на множеството (за всеки от елементите *s*, кое ще е очакваното наблюдение *v* и в кое множество ще отиде състоянието *s'*). Получаваме *N.n* парчета и на всяко трябва да сложим някаква вероятност. Тези вероятности първо трябва да удовлетворяват едни неравенства описани в [4]. Това условие трябва да бъде изпълнено независимо то това, коя е функцията *f*. По-интересно е условието, което зависи, от това коя е функцията *f*. Ще добавим изискването празните множества да имат вероятност нула. Ако функцията *f* е такава, че всичките *N.n* множества не са празни, то тогава системата от неравенства ще има решение, но в общия случай при конкретна функция *f*, някои от множествата може да са празни и системата може да няма решение.

Дефиниция: Ще казваме, че *g* е минимална, когато няма друга коректна функция, която да е по-малка (т.е. никое от множествата в резултата да не може да се намали, както и никой от вероятностните интервали.)

Коректна функция винаги имаме. Нека вземем функцията *g*, която ни връща винаги:

$$\{ \langle v_1, [0, 1], idea \rangle, \dots, \langle v_n, [0, 1], idea \rangle, \langle incorrect, [0, 1], idea \rangle \},$$
$$\text{където } idea = \{ \langle S, [1, 1] \rangle \}$$

Нека тук участват всички наблюдения. Тоест, *g* е функцията, която ни казва, че понятие си няма кое ще е следващото наблюдение, а следващото състояние ще бъде някое от състоянията на света. Тоест, и за следващото състояние на света *g* си няма идея, кое ще е то.

Няма да е проблем да намерим коректна функция *g*, но ще е трудно да намерим коректна и минимална функция *g*.

Дефиниция: Ще казваме, че сме разбрали света, ако успеем да намерим коректна и минимална функция *g*, която го описва.

Агентът никога няма да знае дали е успял да разбере света, защото неговата информация за функция *f* ще бъде събрана на базата на крайно много наблюдения. Само един външен наблюдател, който знае точно коя е функцията *f*, ще може да каже дали сме разбрали света. Затова понятието „разбрали сме света“ е по-скоро теоретично, без практическо приложение.

Детерминистичната функция *g* е минимална. Ако тази функция е и коректна, тогава значи сме разбрали света и то така сме го разбрали, че за нас няма никаква случайност. Например, ако хвърлим зар трябва да знаем какво ще се падне. Това изискване е прекалено силно и затова няма да изискваме детерминистична функция, за да предполагаме, че сме разбрали света.

Има ли описание?

Ако животът е краен (т.е. ако имаме крайна редица от наблюдения и действия), тогава има описание на света g, което на сто процента описва този живот (има дори и изчислимо описание, което напълно го описва). Когато животът е безкраен (т.е. редицата е изброима) тогава може да няма изчислимо описание, което напълно да описва този живот. Светът може да не е описуем и сто процентово описание може въобще да няма, но ще предполагаме, че има описание, което описва живота достатъчно добре. Това е така, първо защото за всяко крайно начало на живота има описание, което го описва напълно и второ защото не е казано, че описанието трябва да е сто процентово.

Какво значи да имаме сто процентово описание на живота? Това означава да имаме детерминирана функция g и детерминирана $idea_0$ (началната представа за състоянието на света), които описват живота точно. Т.е., ако се вземат като аргументи действията a_i да се получат наблюденията v_i.

Описанието на света може да описва живота приблизително, като казва, че нещо ще се случи с определена вероятност и дори с вероятност, която не е точно определена, а е в интервал (виж в [6] за интервалите от вероятности).

Като пример за приблизително описание на живота ще дам това, което ми каза баща ми: „В живота ти ще има и хубави и лоши неща!“ Това описание е приблизително, защото той дори не ми каза, какъв ще е процента (вероятността) на добрите неща. Тоест, вероятността би трябвало да е в интервала $[0, 1]$.

Ако животът е безкраен, тогава може да нямаме сто процентово описание, но ще имаме много описания, които го описват приблизително. Бихме могли да предположим, че сме избрали най-доброто от приблизителните описания, но тук няма най-добро, защото за всяко приблизително описание ще има и по-добро приблизително описание. Въпреки това можем да предположим, че сме избрали едно достатъчно добро приблизително описание на света.

Колко ще са агентите?

Във всеки свят има поне един агент и това е главният герой. Това е агентът, който получава входа от света и който със своя изход влияе на света. Във вашия свят вие сте главният герой. Вие имате модел на света и този модел описва още много други агенти. Това са хора, животни, божества, дори предмети, защото предполагаме понякога, че предметите също извършват действия. Например, колата ви се разваля. Вие може да предположите, че това е действие извършено от колата. Когато колата се движи вие не предполагате, че тя прави това по собствена воля, но кучетата така възприемат света. Кучето лае по колата, а не по шофьора. Тоест, кучето приема колата като отделен агент и не разбира, че всъщност шофьора е този, който я управлява.

Можете да предположите, че във вашият свят има само един агент и това сте вие. Може да предполагате, че всички останали хора не съществуват реално, а са само плод на вашето въображение. Всеки свят може да бъде представен като едно-агентен или като мулти-агентен, но мулти-агентният модел е много по-естествен и разбираем. Затова повечето от нас си мислят, че не са сами на света и че в техния свят има и други хора (агенти).

Конкретният свят, който ще разгледаме ще бъде играта шах и ще разгледаме два варианта на този свят. В първият вариант агента ще е един и ще играе сам срещу себе си. При втория вариант агентите ще са двама. Главният герой ще играе с белите фигури, но ще има втори агент (противник), който ще мести черните фигури.

Вторият вариант е по-интересен и по-сложен, но е и по-труден за описване.

Какво ще представлява ИИ?

Казахме, че свеждаме задачата за създаването на ИИ до задачата за намирането на език за описание на светове. Ако имаме търсения от нас език за описание, как от този език ще направим програмата ИИ?

В [5] казахме, че ИИ трябва да отговори на два въпроса: „Какво става?“ и „Какво да правя?“. В тази статия няма да се занимаваме с втория въпрос, но ако сме разбрали света, то бихме могли мислено да направи няколко хода напред и с алгоритъм подобен на Min-Max да изберем най-доброто действие, което се очаква да доведе до най-доброто възможно бъдещо развитие.

По-голяма трудност представлява отговорът на първия въпрос. Този въпрос е свързан с разбирането на света. Тоест, намирането на функцията *f*, която е същността на света. Намирането на функцията *f* е намирането на нейното описание написано на някакъв език за описание на светове. Както казахме, освен функцията трябва да намерим и моментното състояние на света (текущата стойност на паметта). За да намерим тази текуща стойност, ние отново се нуждаем от езика за описание на светове, защото този език ще ни даде и формàта на паметта. Тоест, за да намерим текущата стойност на паметта ние трябва да знаем, в кой формат е записана тази стойност.

Ако знаем езика за описание на световете, ние ще знаем какво търсим и лесно ще направим програма, която да търси такова описание. Разбира се, няма да е лесно да направим това търсене ефективно.

Нужно ще ни е описанието да бъде откриваемо. За целта, то трябва да бъде разделено на отделни модули. Тези модули трябва да представляват зависимости, които могат да бъдат открити една по една. Например, ако вие се опитате да отгатнете паролата на моя имейл, това ще ви бъде трудно, защото възможните комбинации са твърде много. Ако можете да отгатвате паролата буква по буква, тогава задачата ви би била много по-проста. За първата буква има малко възможности и вие можете лесно да я отгатнете стига аз да съм достатъчно добър, за да ви казвам дали сте познали. Тоест, нужно е описанието да се разбие на отделни прости модули и всеки от тях да има по нещо специфично, което да го направи откриваем.

Защо конкретен свят?

Когато искаме да създадем нов език за програмиране, ние не го създаваме от веднъж, а първо избираме една конкретна задача и пишем програма, която да решава тази задача. После взимаме друга задача и пишем друга програма и така постепенно усъвършенстваме езика като го пригаждаме към задачите, които искаме да решим.

Аналогично ще постъпим, когато искаме да създадем език за описание на светове. Ще вземем един конкретен свят и ще се опитаме да го опишем на някакъв език. Този език първоначално няма да е универсален и да може да опише произволен свят, но ако успее да опише първия свят, това ще е успех. Прилагайки този език към различни светове постепенно ще го усъвършенстваме докато не се получи универсалният език, който ни трябва.

Първият свят, с който ще започнем, това ще е играта „шах“. По-точно това ще е свят, в който агентът играе шах. Дали агентът ще е сам в този свят или вътре в света ще има още един агент, който да е противникът му и който ще мести черните фигури?

Ще започнем с по-простия вариант, когато агентът (главният герой) е сам и мести и белите и черните фигури. Когато агентът играе с бяла фигура той ще смени цвета си и следващият му ход ще бъде с черните фигури. Така играят хора поставени в изолация, като затворници и други. Много по-интересно е да се играе с противник, но тогава светът е много по-сложен. Във втория случай светът ще стане неизчислим, защото ще имаме агент, който мести черна фигура, а това е агент, който изпълнява алгоритъм.

В първия случай светът ще бъде изчислим с изключение на едно правило. Това е правилото, че нямаме право да играем ход, след който сме шах или че не можем да играем ход, ако на следващият ход противникът може да ни вземе царя. Тоест, ако съществува алгоритъм, чрез който противникът може да ни вземе царя.

Сега ще ми кажете, че това е напълно изчислимо и че ако ви дам определен ход и позиция, вие веднага ще ми кажете дали ще сте шах след този ход. Да това е изчислимо в случая с играта шах, защото там всичко е крайно (крайна дъска 8 x 8), но в общия случай това дали съществува алгоритъм е неизчислимо.

Кой конкретен свят?

Ще предположим, че агентът играе шах и мести фигури по шахматното табло.

```
views     -> ....x.y...x..x.........................yyx.yxx...
incorrect -> ..6..4..6..4.46..6..6..62.6..62.6..42.4244.466.6.6
actions   -> ab.a..b..bb.aa....aa.a..ba.ab.b..ab..b.b.a.a..a.a.
```

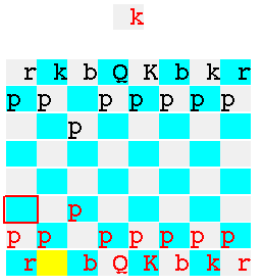

Figure 1

В лявата част на фигура 1 се вижда потока входно-изходна информация. Това не е целият поток, а само последните 50 стъпки. На първия ред са наблюденията на агента, а на третия ред са действията му. Възможните наблюдения са три {0, x, y}. Възможните действия също са три {0, a, b}. Нулата е заменена с точка за по-добра четливост. На втория ред се вижда кой действия са позволени в конкретния момент (2 означава, че действието ***a*** е некоректно, 4 означава, че ***b*** е некоректно, 6 означава, че и ***a*** и ***b*** са некоректни).

Това, което е в лявата част на фигура 1, е това, което вижда агентът. Това, което е в дясната част на фигурата, агентът не го вижда, но трябва да си го представи, за да разбере света. В дясно се вижда позицията на таблото, вижда се коя фигура агентът е вдигнал (коня), вижда се и от къде я е вдигнал (жълтото квадратче), вижда се и кое е наблюдаваното в момента квадратче (ограденото с червена линия).

Partial Observability

Ограничили сме агента да не вижда цялото табло, а да вижда само едно от квадратчетата (това, което е наблюдаваното в момента). Агентът може да мести поглед и да променя квадратчето, което вижда, като по този начин може да огледа цялото табло. Въпреки това, важно е, че предполагаме, че агентът не вижда цялото табло и че имаме Partial Observability. По този начин от агента се изисква да може да си представи тази част от света, която той не вижда в момента. Агентът ще вижда само едно от квадратчетата, а цялото шахматно табло ще бъде в неговото въображение.

За да има въображение агентът, трябва да има идея за текущото състояние на света (за s_i). Какво значи да има идея? Най-добре да знае точно какво е текущото състояние на света. Ако не знае точно, добре е да знае какви са възможните състояния и каква е вероятността за всяка от тези възможности. Например, когато агентът мести фигура и виртуалният му противник му отговаря, тогава агентът още не знае какво е местил противникът му. Може да има някаква идея, а може и да огледа таблото и да разбере точно какво е играл противникът. Тоест, агентът няма да знае точно какво е текущото състояние на света, но ще има някаква представа.

Предполагаме, че агентът е разбрал света, тоест намерил е функция f, която го описва и има идея за текущото състояние на света (за s_i).

Кодиране

Агентът ще може да извършва 7 действия. Той ще може да мести поглед (квадратчето, което наблюдава) в четирите посоки. Ще може да вдигне фигурата, която вижда и да пусне вдигната фигура в квадратчето, което вижда в момента. Седмото действие е да не прави нищо.

Ще ограничим действията на агента до три букви {0, a, b}. Символа 0 ще използваме да действието „не върша нищо“. Как с оставащите два символа ще опишем 6 действия? Това ще стане чрез кодиране. Ще разделим стъпките на три. На всяка първа стъпка ще казваме как движим квадратчето по хоризонтала (т.е. как движим прозореца ни на наблюдение). На всяка втора стъпка ще казваме как движим квадратчето по вертикала, а на всяка трета стъпка ще казваме дали вдигаме фигура или пускаме вдигнатата фигура.

В [3] споменахме, че трябва да избягваме излишното кодиране, защото светът е достатъчно сложен и няма нужда допълнително да го усложняваме. Тук обаче не става дума за излишно кодиране, защото с това кодиране светът не се усложнява, а става по-прост, защото заменяме седем действия с три.

1, 2, 3

Първата зависимост, която ще съществува в този конкретен свят идва от това, че разделихме стъпките на три. Тази зависимост ще наречем „1, 2, 3“. Моделът на тази зависимост е изобразен на фигура 2.

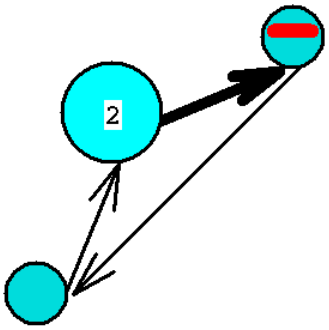

Figure 2

Какво представлява тази зависимост? Тя брои: едно, две, три.

Тази зависимост се представя с един Event-Driven модел. Тези модели са описани в [4]. В конкретния случай това е модел с три състояния. В този модел имаме само едно събитие и това е събитието „винаги“ (тоест, „истина“ или „на всяка стъпка“).

Следа

Дали в състоянията на гореописания модел се случва нещо специално, което да можем да забележим и което да ни помогне да открием този модел? Тоест, има ли „следа“ (това е терминология въведена в [6]).

Да, в третото състояние задължително едно от действията ***a*** или ***b*** е некоректно (или и двете). Това е така, защото в третото състояние ние казваме дали вдигаме фигурата, която виждаме или пускаме вдигната вече фигура. Няма как тези две действия да са възможни едновременно.

Можем и без тази следа да опишем света, но без нея зависимостта „1, 2, 3“ би била много по-трудно откриваема. Затова е добре, че имаме някаква следа в този модел.

Следата е това специалното, което дава смисъла на модела. Например, в хладилника има студена бира и това прави хладилника един по-специален шкаф. Ако във всички шкафове имаше студена бира, тогава хладилника нямаше да е по-специален и щеше да е все едно кой шкаф ще отворим.

Следата ни дава възможност да предскажем какво ще се случи. Освен това, следата ни помага да познаем в кое състояние сме и да ограничим недетерминираността. Например, по това че в един шкаф има студена бира, а в друг няма, ще ни помогне да различим шкафовете. Нека имаме два бели шкафа. Искаме да отворим бял шкаф, в който има студена бира. Отваряме двата бели шкафа, но ако само в единия има студена бира, затваряме другия и по този начин ограничаваме недетерминираността.

Ще разглеждаме два вида следа – постоянна и подвижна. Постоянна следа ще са специалните неща (явления), които всеки път се случват, а подвижна следа ще са нещата, които се случват временно.

Например, да си представим една къща като един Event-Driven модел. Състоянията на този модел ще са стаите. Нещо постоянно за стаите ще е броят на вратите. Временни явления, които се явяват и изчезват са „светла“ и „топла“. Тоест, постоянната следа може да ни каже коя стая е преходна, а подвижната следа ще ни каже, коя стая в момента е топла.

Стаите могат да са свързани с различни обекти. Тези обекти си имат свойства (явленията, които се наблюдават, когато наблюдаваме съответния обект). Обектите могат да са постоянни и подвижни и съответно техните свойства ще са постоянни и временни явления. Пример за постоянни обекти са мебелите (особено по-тежките). Пример за подвижни обекти са хората и животните. Тоест, постоянната следа ще описва това което е постоянно, а подвижната следа ще описва това, което е временно.

Horizontal и Vertical

Следващият Event-Driven модел, който ще ни е нужен за описанието на света е моделът „Horizontal“ (фигура 3).

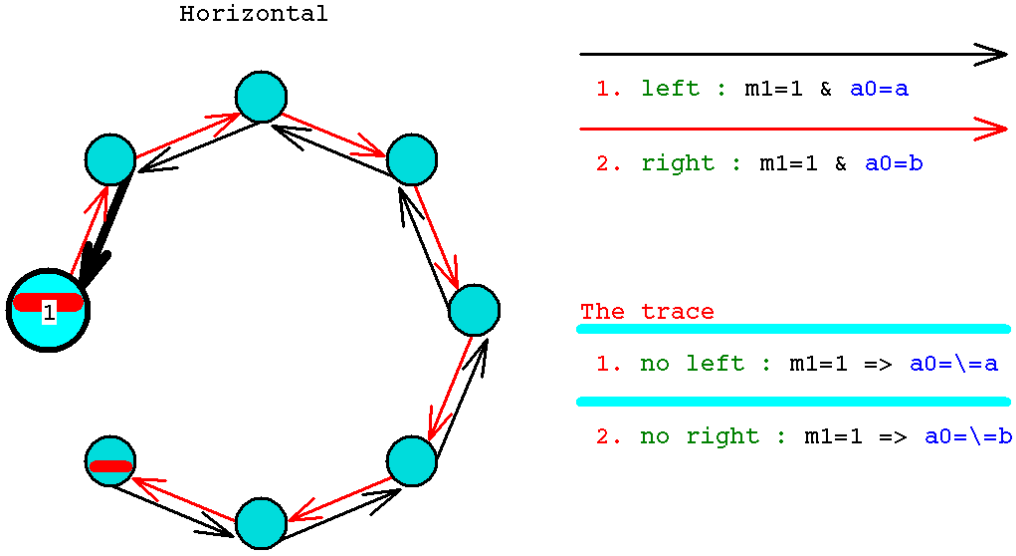

Figure 3

Този модел ще ни даде информация в коя колона на шахматната дъска се намира квадратчето, което наблюдаваме.

Тук имаме две събития и това са събитията „наляво“ и „надясно“. Тоест, агентът мести поглед наляво или надясно. Тоест, той извършва действията ***a*** и ***b***, когато моделът 1 е в състоянието 1. Имаме и две следи. В състоянието 1 не може да се играе наляво. Тоест, когато сме в състояние 1 събитието „наляво“ не може да се случи. Аналогично, е със състоянието 8 и следата, че там не може да се играе надясно. Тези две следи ще направят моделът откриваем. Например вие, ако сте в тъмна стая широка 7 стъпки, ще установите,

че след седем стъпки наляво по-наляво не може. Ще го установите, защото ще се блъснете в стената. Тоест, сблъсъка със стената е следата в случая. Такъв сблъсък ще има само на първата и на последната позиция.

Тази следа, освен че ще ни помогне да открием модела, тя ще е полезна още за да ни обясни света. Как иначе бихте си обяснили защо в най-лявата колона не можете да играете „наляво“?

Съвсем аналогичен на модела „Horizontal“ е моделът „Vertical“ (фигура 4).

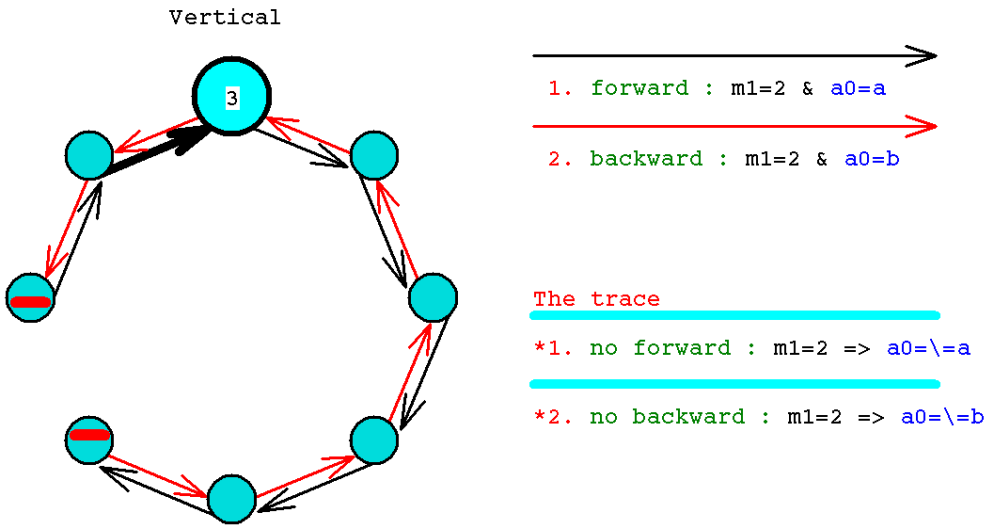

Figure 4

Този модел ще ни каже в кой ред се намира квадратчето, което наблюдаваме. Аналогично имаме две събития („напред“ и „назад“), както и две следи („не може напред“ и „не може назад“)

Логично е да направи декартовото произведение на горните два модела и ще получим модел с 64 състояния, който ще отговаря на шахматното табло.

Лошото е, че в това декартово произведение няма постоянна следа. Тоест, нищо специално не се случва в някое от квадратчетата. Случват се разни работи, но те не са постоянни, а временни. Например, в едно квадратче може да виждаме бяла пешка и това да е сравнително постоянно, но не е напълно постоянно, защото пешката може да се премести.

Стигаме до извода, че следата може и да не е постоянна.

Подвижна следа

Както казахме, „подвижна следа“ ще са специалните неща, които се случват в едно състояние, но не се случват постоянно, а само временно.

Как да изобразим подвижната следа? Постоянната следа изобразявахме, като отбелязвахме върху състоянието дали някакво събитие винаги се случва в това състояние (винаги отбелязваме с червено, а със синьо отбелязваме, когато никога не се случва).

Подвижната следа ще изобразим като масив, който има толкова клетки, колкото състояния има съответния модел. Във всяка клетка ще запишем подвижните следи, които в момента са в съответното състояние. Тоест, масивът на подвижната следа ще мени стойностите си.

Ето как ще изглежда масива на подвижната следа на декартовото произведение на втория и третия модел:

	1	2	3	4	5	6	7	8
8	black rook unmov	black knight	black bishop	black queen	black king unmov	black bishop	black knight	black rook unmov
7	black pawn unmov	black pawn unmov		black pawn unmov	black pawn unmov	black pawn unmov	black pawn unmov	black pawn unmov
6			black pawn					
5								
4								
3			white pawn					
2	white pawn unmov	white pawn unmov		white pawn unmov	white pawn unmov	white pawn unmov	white pawn unmov	white pawn unmov
1	white rook unmov	lift	white bishop	white queen	white king unmov	white bishop	white knight	white rook unmov

Figure 5

Тази подвижна следа е много сложна, защото това е подвижната следа на модел с 64 състояния. Нека да вземем подвижната следа на модел с две състояния (фигура 6). Това е моделът 4, който помни дали сме вдигнали фигура. Неговата подвижна следа ще помни коя е вдигнатата фигура. Разбира се, този модел освен подвижна следа си има и постоянна, която казва, че в състоянието 2 не може „нагоре“, докато в състоянието 1 не може „надолу“.

Подвижната следа на този модел представлява масив с две клетки, които съответстват на двете състояния на ED модела. Клетката, която съответства на текущото състояние е отбелязана, като е оградена с червена линия. Не е толкова важно какво има в клетката съответстваща на текущото състояние, а това което е в другите клетки, защото те ни казват какво ще се случи, когато текущата клетка стане друга. В случая, ако пуснем вдигната фигура ще отидем в състоянието 1 и там ще видим вдигната фигура. (Ще видим това, което сме пуснали. В случая ще видим „бял кон“.)

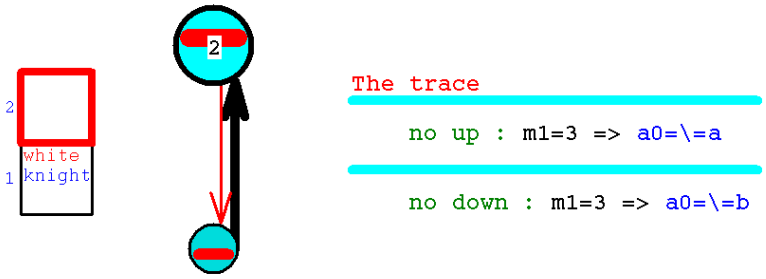

Figure 6

Казахме, че езикът за описание на светове ще ни каже как изглежда паметта на функцията *f*. Къде се записва вътрешното състояние на света? Записва се на две места. Първо, това е текущото състояние на всеки от моделите и второ, това е подвижната следа. Например на фигура 5 виждате как чрез подвижната следа се представя позицията на шахматната дъска.

Ако езика за описание на светове беше стандартен език за програмиране, неговата памет щеше да е стойността на променливите и на масивите. Тук можем да направим аналогията, че текущото състояние на един ED модел е стойността на една променлива, а стойността на една подвижна следа е стойността на един масив.

Стойността на текущото състояние на един ED модел обикновено е едно число, ако моделът е детерминиран, но може да е няколко числа, ако ED моделът има няколко текущи състояния. Стойността на всяка от клетките на подвижната следа ще се състои от няколко числа, защото в едно състояние може да има много подвижни следи. Разбира се, постоянните следи също може да са повече от една.

Алгоритми

След като описахме основните правила на играта шах и позицията на таблото, следващата стъпка е да кажем как се движат фигурите. За целта ни е нужно понятието алгоритъм.

За повечето хора алгоритмът това е машина на Тюринг. Това е така, защото те разглеждат само функциите от $\mathbb{N}$ в $\mathbb{N}$ и за тях алгоритъм е нещо което изчислява такава функция. За нас алгоритмът ще описва последователност от действия в произволен свят. Например за нас алгоритми ще са готварските рецепти, танцовите стъпки, умението да се хване топка и т.н. Казахме последователност от действия. Нека се коригираме и да стане последователност от събития. Действието е събитие, но не всяко събитие е действие или поне не е наше действие, а може да е действие на някой друг агент. В описанието на алгоритъма освен наши действия има и други събития. Например, чакаме докато водата кипне. Кипването на водата е събитие, което не е наше действие.

При нашата дефиниция алгоритмът може да се осъществи въобще без нашето участие. Да вземем като пример Лунната соната. Това е алгоритъм, който ще изпълним, ако изсвирим Лунната соната, но ако я изсвири някой друг, тогава това пак ще е алгоритъм, но изпълнен от някой друг. Ако разпознаем Лунната соната, ние ще сме разпознали този алгоритъм, нищо че не го изпълняваме.

Няма да е много важно кой изпълнява алгоритъма. Нормално е един алгоритъм първо да ни го покаже някой друг, после да го изпълним и ние.

Ще разгледаме три варианта на алгоритъм:
1. Релсов път.
2. Планинска пътека.
3. Отивам си вкъщи.

При първия вариант ще предполагаме, че имаме ограничения, които не ни позволяват да се отклоним от изпълнението на алгоритъма. Например, когато се качим на рейса, ние пътуваме по маршрута и не можем да се отклоним, защото друг кара рейса. Когато слушаме Лунната соната, отново нищо не можем да променим, защото не свирим ние.

При втория вариант ние можем да се отклоним, но има последствия, ако се отклоним. Планинската пътека минава покрай пропаст. Ако се отклоним, ще паднем в пропастта. При третия вариант можем да се отклоним от пътя. След отклонението можем отново да се върнем в пътя, а може и да минем по друг път. Алгоритъма за прибиране у дома ни казва, че ако го изпълним, ще сме си вкъщи, но по никакъв начин не сме задължени да го изпълним или да го изпълним точно по този начин.

Обикновено, когато говорим за алгоритъм предполагаме детерминираност. Представяме си компютърна програма, при която за всеки следващ момент се знае точно кое ще е действието, което ще бъде извършено. Вече дори и компютърните програми не са еднонишкови. При многонишковите програми не е съвсем ясно кое ще е следващото действие, което ще бъде извършено. Още по-ясен е примерът с готварските рецепти. Когато правим палачинки, не е казано дали първо да сложим яйцата и после млякото или обратното. И в двата случая ще изпълним един и същи алгоритъм.

Представете си алгоритъма като движение в пещера. Можете да вървите напред, но можете да се върнете и назад. Галерията има разклонения и вие имате избор на къде да завиете. Само, ако излезете от пещерата, ще сте прекратил изпълнението на алгоритъма „движа се в пещерата“. Тоест, ще си представяме алгоритъма като ориентиран граф с много разклонения, а не като път без разклонения.

Алгоритъма на фигурите

С алгоритмите ще опишем движението на фигурите. Ние ще изберем варианта „релсов път“ (първият от разгледаните варианти). Тоест, когато вдигнете фигура ще се включва съответният алгоритъм, който няма да ви позволи да направите грешен ход.

Можеше да изберем и варианта „планинска пътека“. Тоест, да може да се отклоните от алгоритъма, но това да е с последствия. Например, вдигате фигурата и започвате да изпълнявате алгоритъма, но ако го нарушите, ще изпуснете вдигната фигура и тя ще се върне на мястото си.

Можеше да изберем и варианта „отивам си вкъщи“. При този вариант се движите както пожелаете, но можете да поставите фигурата само на тези места, където алгоритмът би могъл да я поставил, ако беше изпълнен. Тоест, имате пълна свобода на движението, а алгоритмът само ви дефинира кои ходове са коректните.

Ще изберем първият вариант главно защото сме пуснали агента да играе случайно и ако не го вкараме в релси, за него ще е много трудно да изиграе коректен ход. Освен това трябва да си помислим за това как агентът ще разбере света. Как ще ги открие тези алгоритми? Ако го вкараме в релси, той ще научи алгоритъма по неволя, но ако го оставим свободно да се движи за него ще е много трудно да отгатне какви са тези правила на движение (какви са тези алгоритми). Например, ако покажете на един ученик алгоритъма за намиране на корен квадратен, то на него ще му е сравнително лесно да го научи. Много по-трудно би му било, ако му обясните какво е корен квадратен го оставите сам да намери алгоритъма за изчисляването му. Можете да покажете на ученика какво е корен квадратен с дефиниция или с примери, но по-лесно ще ви разбере, ако му покажете директно алгоритъма.

Какво ще представляват алгоритмите? Това ще са Event-Driven модели. Ще има някакво събитие, което ще е вход и което ще стартира алгоритъма и още някакво събитие, което ще е изход и след което алгоритъма ще престане да се изпълнява. По-нататък ще направим изходите да са два (успешен и неуспешен изход).

Всяка фигура ще си има алгоритъм:.

Алгоритмите на царя и на коня

Най-простият алгоритъм ще бъде алгоритъма на царя (фигура 7). Входящото събитие ще бъде вдигам фигурата цар. Входящата точка ще бъде състоянието 1 (при всичките алгоритми това ще бъде входящата точка). Събитията ще са 4 (наляво, надясно, напред и назад).

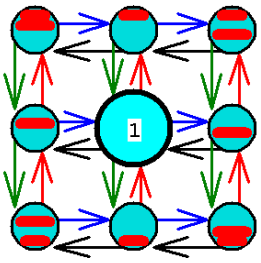

Figure 7

Следата ще се състои от четири събития (не може наляво, не може надясно и т.н.) Тези четири събития (следи) ще ограничат движението до девет квадратчета. Тези 4 събития (следи) ще са релсите, в които ще влезем и които няма да ни позволят да напуснем деветте квадратчета докато изпълняваме алгоритъма. На фигура 7 четирите следи са отбелязани с червени хоризонтални линии. Например, горните три състояния имат първата следа, което значи, че от тези три състояния не може напред.

Ще можем да пуснем вдигната фигура (царят) във всеки момент, когато пожелаем. Разбра се, може да има други правила или алгоритми, които да ни ограничават. Например, не можем да вземем собствена фигура, тоест има и други ограничения, но те не идват от този алгоритъм. Ако пуснем фигурата в състоянието 1, тогава ходът ни няма да е истински, а ще е фалшив. Ако пуснем фигурата в някое от другите състояния, тогава ще сме изиграли един истински ход.

Малко по-сложен е алгоритмът на коня (фигура 8). Основната разлика с алгоритъма на царя е, че тук има още една следа. Тази следа ни ограничава и в някои от състоянията няма да можем да спускаме вдигната фигура. (Тази следа е отбелязана на фигура 8, а другите 4 следи не са отбелязани.) Спазвайки този алгоритъм ние имаме само две възможности. Първата е да изиграем коректен ход с коня, а втората е да изиграем фалшив ход като върнем коня там откъдето сме го взели.

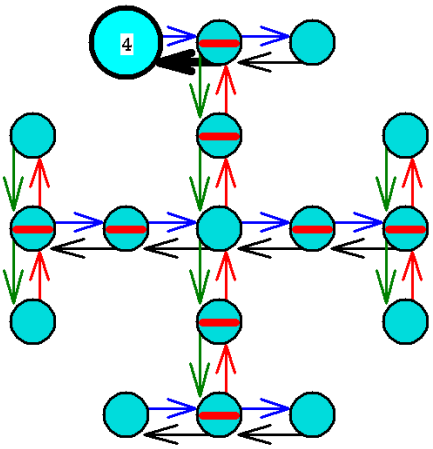

Figure 8

Алгоритмите на топа и на офицера

Макар че има малко състояния алгоритмът на топа е по-сложен (фигура 9). Причината за това е, че този алгоритъм е недетерминиран. Например в състоянието 3 когато играем „напред“, тогава има две стрелки които отговарят на това събитие. Съответно има две състояния, които могат да са следващите. Тази недетерминираност веднага се разрешава, защото в състоянието 1 задължително трябва да се вижда, че от това квадратче е вдигната фигура, докато в състоянието 3 задължително това не трябва да се вижда. Тоест, имаме следа която веднага разрешава тази недетерминираност.

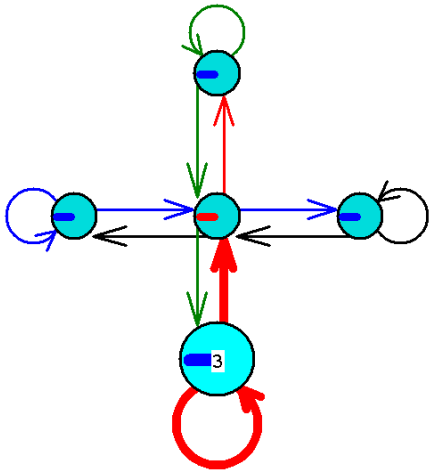

Figure 9

Алгоритмът на офицера е още по сложен (фигура 10). Основната причина за това е, че не можем да се придвижим директно по диагонала, а за целта трябва да направим две стъпки

(първата по хоризонтала и втората по вертикала). Когато в състоянието 1 се случи събитието „наляво“, тогава ние не знаем дали сме тръгнали по диагонала „наляво и напред“ или по диагонала „наляво и назад“. Тогава се получава недетерминираност, която не може да бъде разрешена незабавно. Все пак, тази недетерминираност ще се разреши когато дойде едно от събитията „нанапред“ или „назад“. В двете възможни състояния имаме следи, които ни казват, че в състояние 8 не може „напред“, а в състояние 2 не може „назад“. Ако и в двете състояния не можеше „напред“, то тогава събитието „напред“ би нарушило алгоритъма. В случая, в едното състояние може, а в другото не може. Тоест, събитието „напред“ е разрешено, но когато то се случи състоянието 8 ще престане да бъде активно и недетерминираността ще се разреши. (На фигура 10 сме отбелязали само следите „не може напред“ и „не може назад“.)

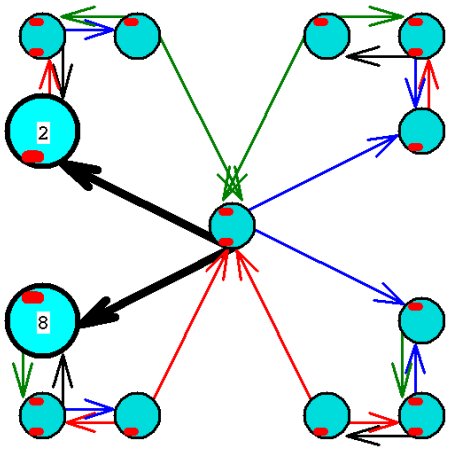

Figure 10

Най-сложен е алгоритмът на царицата, защото той е съчетание от алгоритмите на топа и на офицера. Алгоритмът на пешката не е сложен, но имаме четири такива алгоритъма, защото имаме различни алгоритми за бяла и за черна пешка, като и за преместена и за непреместена пешка.

Понятието алгоритъм

Важно е, че в тази статия е дефинирано понятието алгоритъм. Много малко са хората, които въобще си задават въпроса какво е алгоритъм. Единствените опити за дефиниция на алгоритъм, които са ми известни са направени от Moschovakis [1, 2]. В тези трудове Moschovakis казва, че повечето автори дефинират алгоритъма чрез някаква абстрактна машина и отъждествяват алгоритмите с програмите за тази абстрактна машина. Moschovakis формулира каква дефиниция на алгоритъм на нас ни е необходима. Той иска да създаде едно общо понятие, което да не зависи от конкретната абстрактна машина. Такова понятие е изчислимата функция, но това понятие е твърде общо за Moschovakis и той иска да направи по-специализирано понятие, което да отразява това, че една изчислима функция може да се изчисли от много принципно различни алгоритми. В [1] не е постигната високата цел поставена от Moschovakis. Това, което той е направил може да се приеме за една нова абстрактна машина. Наистина тази машина е много интересна и е по-абстрактна от повечето известни машини, но отново имаме недостатъка, че програмата на машината може безсмислено да се усложни като се получи друга програма реализираща същия алгоритъм. Макар, че в [1] не се постига целта да бъде създадена обща дефиниция

на алгоритъм, самият Moschovakis казва, че за него по-важното е да постави въпроса, дори и да не успее да му отговори. Точните думи на Moschovakis са: „my chief goal is to convince the reader that the problem of founding the theory of algorithms is important, and that it is ripe for solution.“

Машината на Тюринг

Описахме алгоритмите на движение на шахматните фигури като Event-Driven модели. Можем ли да приемем, че всеки алгоритъм може да се представи като Event-Driven модел? Дали машината на Тюринг може да се представи по този начин?

Ще опишем един свят, който представлява машина на Тюринг. Първото нещо, което трябва да опишем в този свят е безкрайната лента. В играта шах описахме шахматното табло като подвижната следа на някакъв Event-Driven модел с 64 състояния. Тук отново ще използваме подвижната следа, но ще ни е нужен модел с изброимо много състояния. Да вземем модела от фигура 3. Това е модел на лента с осем клетки. Трябва ни същият модел, пак да има две събития (наляво и надясно), но да не е ограничен отляво и отдясно. Получава се Event-Driven модел с безкрайно много състояния. Досега използвахме ED модели само с крайно много състояния. Сега ще ни се наложи да добавим и някои безкрайни ED модели, но които имат проста структура като този. В случая моделът представлява просто един брояч, който помни едно цяло число (т.е. елемент на $\mathbb{Z}$). Броячът има две операции (минус едно и плюс едно) или (наляво и надясно). Добавянето на този безкраен брояч разширява езика, но както казахме ние ще разширяваме езика, за да покрием световете, които искаме да опишем.

Каква ще е паметта на този свят? Трябва да запомним стойността на брояча (коя клетка от лентата гледа главата на машината). Това е произволно цяло число. Тоест, вече имаме безкрайна памет, макар и изброима. Освен това ще трябва да запомним и какво има записано върху лентата. За целта ще ни трябва безкрайна последователност от букви което е равномощно на континуум. Обикновено използваме Машините на Тюринг, за да изчисляваме функции от $\mathbb{N}$ в $\mathbb{N}$. В този случай бихме могли да се ограничим само с конфигурации, при които е използвана само крайна част от лентата, тоест бихме могли да се ограничим с изброима памет, но на безкрайната лента могат да са записани безкрайно много неща и ако искаме да опишем произволен свят, то трябва да приемем, че на лентата може да е записана произволна конфигурация.

Няма да разглеждаме случая когато паметта е повече от континуум, защото всеки свят с памет повече от континуум може да се представи като такъв, който е с континуум памет.

Представата на агента за състоянието на света ще е изброима дори ако паметта на света е континуум (реално число). Казано по друг начин, агентът няма как да си представи всички възможни конфигурации върху лентата, а само изброима част от тези конфигурации.

При горното разсъждение използваме това, че си мислим агента като абстрактна машина с безкрайна памет. Ако агента си го мислим като реален компютър с крайна памет, тогова в горното разсъждение трябва да заменим „изброима“ с „крайна“. Все пак, ако агентът е програма на реален компютър, но тази крайна памет е толкова голяма, че за по-просто си я мислим за изброима.

Описахме безкрайната лента на машината на Тюринг с един безкраен ED модел. За да опишем главата на машината (самият алгоритъм) ще ни трябва още един ED модел. Втория ED модел ще го построим използвайки машината на Тюринг.

Нека предположим, че машината използва две букви {0, 1}. Ще направим Event-Driven модел с четири събития:
write(0),
write(1),
move left,
move right.

Тогава всяка от командите на машината ще изглежда така:

if observe(0) then write Symbol_0, move Direction_0, goto Command_0
if observe(1) then write Symbol_1, move Direction_1, goto Command_1

Тук Symbol_i, Direction_i и Command_i са заменени с конкретни стойности. Например:
if observe(0) then write(1), move left, goto s_3
if observe(1) then write(0), move right, goto s_7

Всяка команда ще заменим с четири състояния, които ще я опишат. Горната команда ще изглежда така:

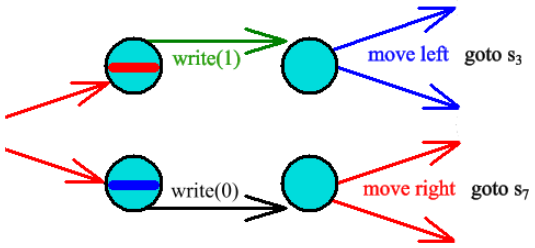

Figure 11

На фигура 11 входа е по събитието „move right“. Всъщност, ще се влиза от много места, понякога по събитието „move left“, а понякога по събитието „move right“. Важното е, че входът ще е недетерминиран, но веднага тази недетерминираност ще се разреши, защото първите две състояния имат следа. В горното задължително трябва да се случи събитието „observe(0)“, а в долното задължително това събитие не трябва да се случва.

Тоест, всяко от състоянията на автомата се заменя с четири състояния, както е показано на фигура 11, след това отделните четворки се свързват помежду си. Например, четворката от фигура 11 се свързва с четворката съответстваща на s_3 чрез стрелки по събитието „move left“ и с четворката съответстваща на s_7 чрез стрелки по събитието „move right“.

Трябва да добавим още малко следа. За всяко едно от състоянията е възможно само едно от четирите събития. Трябва да добавим като следа, че другите три събития са невъзможни. Това е, ако искаме алгоритмът да е от тип „релсов път“. Ако предпочитаме да е от тип „планинска пътека“ трябва добавим следа, която да казва че ако се случи някое от другите три събития, ще настъпят съответните последствия. Ако искаме типът да е

„отивам си вкъщи“, тогава другите три събития трябва да са условие за излизане от алгоритъма.

По този начин представихме машината на Тюринг с Event-Driven модел. По-точно с два ED модела, първия с безкрайно много състояния и втория с краен брой (четири пъти повече от състоянията на машината).

Кой изпълнява алгоритъма на машината на Тюринг? Може да предположим, че четирите събития са действия на агента и че той е този, който изпълнява алгоритъма. Може да предположим, че тези събития ги изпълнява друг агент или че те просто се случват. Тогава агента не изпълнява алгоритъма, а е само наблюдател. В общият случай, една част от събитията на алгоритъма ще са действия на агента, а останалата част няма да са. Например, „сипвам вода“ е действие на агента, а „водата завира“ не е негово действие. Агентът може да влияе и на събитията, които не са негови действия. Това е описано в [5]. Спрямо тези събития той може да има някакво „предпочитание“ и чрез това „предпочитание“ той би могъл на влияе на това дали тези събития ще се случат.

Свойства

След понятието алгоритъм ще се опитаме да дефинираме още едно фундаментално понятие. Това ще е понятието свойство. За дефиницията на това понятие отново ще използваме Event-Driven модели. Свойствата са явленията, които се наблюдават когато се наблюдава обект със съответното свойство. Явленията са зависимости, които не се наблюдават постоянно, а само от време на време. Щом другите зависимости се представят с Event-Driven модели, естествено е и свойствата да се представят по същия начин.

Разликата между зависимост и свойство ще бъде, че зависимостта ще е активна постоянно (т.е. ще се наблюдава постоянно) докато свойството ще се наблюдава понякога (когато наблюдаваме съответния обект).

Базовото понятие ще е свойство, а обектът ще е абстракция от по-висок ранг. Например, ако в света на играта шах се наблюдават свойствата „бял“ и „кон“, може да се направи извода, че има обект „бял кон“, който се наблюдава и който има тези две свойства. Може и да не стигаме до тази абстракция и да си мислим, че просто някакви свойства се местят. Тоест, че някакви явления се появяват и изчезват.

Второ кодиране

Изходът на агента се състои само от три букви и затова използвахме кодиране за да представим седемте възможни действия на агента. Входът също е ограничен до три букви. Вярно е, че входът трябва да ни даде информация само за едно от квадратчетата, а не за цялото табло. Въпреки това три букви са твърде малко, защото в квадратчето може да има шест различни фигури с два различни цвята. Освен това, трябва ни допълнителна информация като това дали пешката е местена и дали от този квадрат не е вдигнатата фигура. Как да представим всичката тази информация с три букви?

Тази информация не е задължително да идва до агента само за една стъпка. Той може да постои известно време върху квадратчето и да наблюдава входа. Той може да забележи различни зависимости докато наблюдава квадратчето. Наличието или отсъствието на всяка

от тези зависимости ще е информацията, която ще получи агентът за квадратчето, което наблюдава. Макар буквите на входа да са само три, зависимостите, които могат да се опишат с три букви са безбройно много.

Тези зависимости ще наречем свойства и ще предполагаме, че агента може да разпознава (да хваща) тези зависимости. Ще предполагаме още, че той може да хване няколко зависимости, дори когато те са една върху друга. Например, агентът трябва да може да хване свойствата „бял“ и „кон“ дори когато те се проявяват едновременно.

Как изглеждат свойствата? Зависимостите и алгоритмите за движение на фигурите са написани от човек, който има идея какви са правилата на играта шах и как се движат фигурите. Свойствата не са написани от човек, а са генерирани автоматично. Например на фигура 12 е изобразено свойството „пешка“. Това свойство изглежда доста странно и нелогично. Това е така, защото, както казахме, то е генерирано автоматично по случаен начин. Това свойство не е написано от нас, защото ние не знаем как би изглеждала пешката. Не е важно как изглежда тя. Важното е пешката да изглежда по някакъв начин и да може тя да бъде разпозната от агента. Тоест, пешката трябва да си има лице, но не е важно как ще изглежда нейното лице.

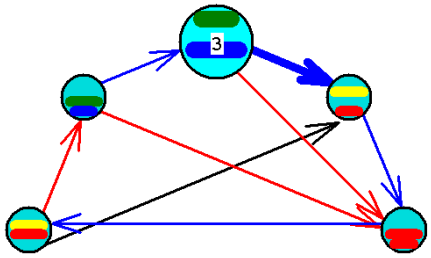

Figure 12

В нашата програма [7] има 10 свойства и всяко едно от тях си има някаква следа. Когато няколко свойства са активни едновременно, тогава всяко едно от тях влияе (чрез следата си) на входа на агента. Понякога тези влияния могат да бъдат противоречиви. Например, едно свойство ни казва, че следващият вход трябва да е буквата ***x***, а друго свойство ни казва обратното (че не трябва да е ***x***). Тогава въпросът се решава с гласуване. Светът брои колко гласа има за всяко решение и избира решението, което е събрало най-много гласове. Що се отнася до противоречивите препоръки, те взаимно се обезсилват.

Детерминираност

Описахме света на играта шах където агентът играе сам срещу себе си. Описанието се състои от 24 модула (Event-Driven модели), два масива (подвижни следи) и още 7 допълнителни правила. Тези правила ни дават допълнителна информация за това как се променя състоянието на света. Например, първото от тези правила ни казва, че ако вдигнем фигура, на нейното място ще се появи свойството „Lifted“. Тези правила ние можем да формулираме благодарение на това, че вече имаме контекста на шахматната дъска (първата подвижна следа). Ако не знаехме за съществуването на тази дъска, нямаше как да формулираме правила за поведението на фигурите върху дъската.
Написахме компютърна програма [7], която илюстрира този свят. В тази демонстрационна програма агентът играе случайно (random). Разбира се, действията на агента не са интересни. Интересното е, че ние сме описали функцията *f*. Тоест, описали сме света.

Описанието, което получихме, е детерминирано. Тоест, функцията *f* е детерминирана. Това означава, че ако знаем какво е състоянието на света и последното действие на агента, тогава знаем точно какво ще е новото състояние на света и точно какво ще е новото наблюдение. (Състоянието на света се състои от текущите състояния на ED моделите и това, което е записано в двата масива.)

Трябва ли описанието на света да е детерминистично? Да се ограничим ли само с такива описания? Въобще не е сигурно, че светът е детерминиран, а дори и да е такъв, не е нужно да се ограничаваме само с детерминистични описания.

Ако опишем недетерминиран свят с детерминистично описание, то много скоро това описание ще покаже своето несъвършенство. Обратното, света може да е детерминиран, но тази детерминираност да е твърде сложна и да не можем да я разберем (да я опишем). Затова може вместо детерминистично описание на света да намерим едно недетерминистично, което да работи достатъчно добре.

Обикновено светът е недетерминиран. Баща ми ми разправяше как е стрелял по бутилки и не е уцелил нито една. Това означава, че не всяко наше действие води до резултат и че резултатите понякога могат да бъдат различни.

Ще допускаме, че функцията *f* може да е недетерминирана. Повечето автори, когато говорят за недетерминираност, предполагат, че всяка възможна стойност на функцията има точно определена вероятност. В [5] и в [6] показахме, че това последното е твърде детерминирано. Би било твърде силно изискването за всяко събитие да можем да кажем точната вероятност, с която то ще се случи. Затова ще предполагаме, че не знаем точната вероятност, а знаем само интервала *[a, b]*, в който е тази вероятност. Обикновено интервалът ще е *[0, 1]* и тогава няма да имаме никаква идея с каква вероятност ще се случи събитието.

Ще усложним света на играта шах, като добавим още един агент. Това ще е противникът, който играе с черните фигури. Това ще доведе до недетерминираност, защото няма да можем да кажем точно как ще играе противникът. Дори противникът да е детерминиран, тази детерминираност може да е прекалено сложна и да не можем да я опишем. Затова ще опишем света с една недетерминистична функция *f*.

Невъзможни събития

Казахме, че светът би бил по-интересен, ако не играем сами срещу себе си, а ако има още един агент, който да мести черните фигури.

За целта ще променим петия Event-Driven модел (този, който ни казва дали играем с белите или с черните фигури). Този модел има две състояния, които се превключват от събитието „change“. Това събитие е дефинирано като събитието „real_move“ (това е реален ход, което е различно от „fake_move“). Ще променим дефиницията на това събитие и ще го дефинираме като „never“ (това е обратното на „every time“).

Има ли смисъл в модела да описваме събития, които няма как да се случат? Отговорът е, че има смисъл, защото тези събития може да се случват мислено. Тоест, тези събития са ни

нужни за да разберем света, макар, че те не се случват. Например, ние не можем да летим и да си сменим пола, но мислено можем да го направим. Примерът не е много добър, защото ние вече можем да летим и да си сменим пола. Тоест, ние може да си мислим за невъзможни събития, освен това в един момент тези събития може от невъзможни да станат възможни.

Ще използваме невъзможното събитие „change“, за да добавим правилото, че не можем да изиграем ход, след който ще сме шах (след който могат да ни вземат царя). На фигура 13 е изобразен алгоритмът, който описва как сменяме (обръщаме дъската) и взимаме царя. Ако този алгоритъм е възможен, тогава ходът не е коректен.

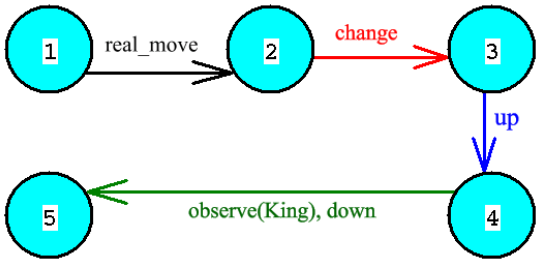

Figure 13

Този алгоритъм в по-голяма степен отговаря на представата ни за това как изглеждат алгоритмите. Докато алгоритмите на фигурите представляваха ориентирани графи с много разклонения, този алгоритъм представлява само един път без разклонения. Тоест, този алгоритъм е просто една последователност от действия без разклонения.

Този алгоритъм се нуждае още от някои ограничения (следи), които не сме отбелязали на фигура 13. Например, в състоянието 1 не можем да се движим в никоя от четирите посоки (иначе бихме могли да се преместим и да изиграем друг ход). Събитието „change“ не може да се случва в никое от състоянията освен състоянието 2. В състоянието 4 имаме ограничението „no observe(King) => no down“. Това последното означава, че единственият ход, който можем да направим, е да вземем цар.

В този алгоритъм участва невъзможното действие „change“. Както казахме, това действие е невъзможно, но можем да го извършим мислено. Това събитие може да участва в дефиницията на алгоритми, които няма да изпълняваме, а за които ще е важно само дали съществуват.

Втори агент

Алгоритмът от фигура 13 би се опростил, ако допуснем съществуването на втори агент. Идеята е вместо да сменяме цвета на фигурите, да сменим агента. Вместо да обръщаме дъската, да сменим агента с такъв, който винаги играе с черните фигури. Ще се получи алгоритъм изпълняван от повече от един агент, но такива алгоритми са естествени. Например: „Дадох пари на един човек и той купи нещо с тези пари“. Това е пример за алгоритъм изпълнен от двама агенти.

По важното е, че ще искаме, когато ние преместим бяла фигура, някой друг (друг агент) да премести черна фигура. В предишния случай си задавахме само въпроса „възможен ли е определен алгоритъм“, а тук ще искаме този алгоритъм реално да бъде изпълнен. Не е все

едно алгоритмът да е възможен и той реално да бъде изпълнен. Не е все едно „може ли някой да направи палачинки“ или „жена ви да ви направи палачинки реално“. В единия случай знаете нещо за света, а във втория случай реално ядете палачинки. Когато някой агент реално изпълнява даден алгоритъм, не е все едно кой е агентът, който ще изпълни алгоритъма. Например, предполагаме, че жена ви ще направи палачинките по-добре отколкото вие бихте ги направили.

Ще предполагаме, че след всеки наш „real_move“ агентът, който играе с черните фигури, ще изпълни алгоритъма от фигура 14.

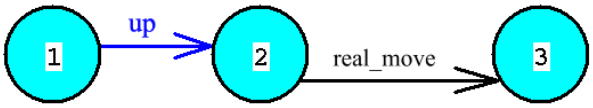

Figure 14

Един алгоритъм не се изпълнява за една стъпка, а за това са нужни много стъпки. Тук обаче ще предполагаме, че противника ще играе с черните фигури веднага (за една стъпка). Хората, когато си мислят, че някой ще направи нещо, обикновено си представят резултата, без да отчитат, че това се извършва в продължение на известно време. Например, когато си мислите: „Днес имам рожден ден и жена ми ще ми направи палачинки“. При това разсъждение вие приемате палачинките за направени без да отчитате, че това отнема време.

Както казахме, не е все едно кой е агентът, който играе с черните фигури. Много важно е дали ни е съюзник или противник (дали ще ни помага или ще ни пречи). Също така, важно е доколко е умен (защото той може да има някакви намерения, но до колко ще ги осъществи зависи от това доколко е умен). Важно е още какво знае и какво вижда агентът. При играта шах предполагаме, че агентът вижда всичко (цялото табло), но в други светове бихме могли да предположим, че агентът знае и вижда само част от информацията. Може да е важно и къде се намира агентът. Тук предполагаме, че това не е важно. Предполагаме, че където и да се намира агентът, той може да се придвижи до произволно квадратче и да вдигне фигурата, която е там. Бихме могли да предположим, че това има значение и че за по-близките фигури е по-вероятно да бъдат преместени от агента, отколкото по-далечните.

Собствено състояние

Тук предположихме, че вторият агент си има собствено състояние на света. Тоест, има си собствена позиция <*x*, *y*> на таблото (квадратчето, което наблюдава). Също така предполагаме, че той играе с черните, за разлика от главния герой, който играе с белите.

Предполагаме, че двамата агенти променят света, чрез една и съща функция *f*, но паметта на функцията (състоянието на света) е различна за двамата агенти. Бихме могли да предположим, че двете състояния на света нямат нищо общо, но тогава действията на втория агент по никакъв начин няма да влияят на света на главния герой. Затова ще предполагаме, че първата следа е обща (т.е. позицията на таблото е обща). Ще предполагаме, че Event-Driven моделите 2, 3 и 5 имат различни активни състояния при двамата агенти. Тоест, че всеки агент си има собствени координати и собствен цвят, с който играе. За останалите ED модели, както и за втората следа също ще предполагаме, че те са отделни за отделните агенти, макар че нищо не пречи да предположим и обратното.

Ако предполагахме, че двамата агенти споделят едно и също състояние на света, тогава алгоритъма от фигура 14 щеше да е много по-сложен. Противника първо щеше да обърне дъската („change“), после щеше изиграе своя ход и пак да обърне дъската, за да остави света на главния герой непроменен. Освен това противникът трябваше да се погрижи да се върне на същите координати <*x*, *y*>, от които е тръгнал (това са координатите на главния герой). Би било много неестествено различните агенти да са съвсем еднакви и да се намират на едно и също място. Много по-естествено е предположението, че агентите са различни и че имат различно състояние на света, но че част от състоянието е общо. Например, „В момента аз правя палачинки и жена ми прави палачинки.“ Може ние да правим едни и същи палачинки, а може моите палачинки да нямат нищо общо с нейните.

Втори свят

Описахме първия свят, в който агентът играеше сам срещу себе си и направихме програмата [7], която илюстрира този сватя. Програмата [7] представлява функцията *f*, която всъщност е емулация на първия свят. Описахме и втори свят, в който агентът играе срещу някакъв противник. Можем ли да направим емулираща програма и за този свят?

Във втория свят добавихме твърдение от вида „този алгоритъм може да бъде изпълнен“ и добавихме операцията от вида „противникът изпълнява този алгоритъм“. Това твърдение и тази операция в общия случай са неразрешими (по-точно те са полуразрешими).

Да вземем например твърдението „този алгоритъм може да бъде изпълнен“. В конкретния случай става дума за това дали противника може да ни вземе царя и това е напълно разрешимо, защото шахматната дъска е крайна и има крайно много позиции и всички алгоритми работещи над шахматната дъска са разрешими. В общия случай алгоритмът може да бъде машина на Тюринг и тогава това твърдение е равносилно на проблема за спирането (halting problem).

Същото може да се каже и за операцията „противникът изпълнява този алгоритъм“. Алгоритмът може да се изпълни по много различни начини, но задачата да намерим поне един от тези начини е полуразрешима. В конкретния случай, когато имаме играта шах, лесно можем да намерим един от начините, по които се изпълнява алгоритмът. Дори можем да намерим всички начини (това са всички възможни ходове). В общия случай това не е така.

Тоест, в конкретния случай ние можем да напишем програма, която емулира този втори свят. Само трябва да изберем поведението на противника, защото за това поведение има много възможности. С други думи казано, за да създадем програма, която да емулира света на играта шах, трябва вътре в нея да вградим програма емулираща шахматен играч.

В общия случай обаче ние няма да можем да напишем програма емулираща описаният от нас свят. Тоест, езика за описание на светове вече описва такива светове, които няма как да бъдат емулирани с компютърна програма. Още в началото казахме, че функцията *f* може да се получи неизчислима. Няма как да напишем програма, която да изчислява неизчислима функция.

Това, че не можем да напишем програма емулираща описаният от нас свят не е голям проблем, защото нашата цел не е да емулираме света, а да напишем програмата ИИ, която на базата на това, че е разбрала света (намерила е описанието му) ще планира успешно бъдещите си ходове. Разбира се, ИИ би могла да процедира като направи една емулация на света и да разиграе няколко от възможните бъдещи развития, като избере това, което е най-доброто. (По същество така работи алгоритмът Min-Max, с който шахматните програми играят.) Тоест, ако можем да направим емулация на света, няма да е лошо, макар и да не е задължително.

ИИ не само, че няма да може да направи пълна емулация на света (когато функцията *f* е неизчислима), но дори ИИ няма да може да разбере кое е текущото състояние на света (когато възможните състояния са континуум много). Въпреки това, ИИ ще може да направи частична емулация. Например, ако в света има безкрайна лента и върху нея има безкрайно много информация, тогава няма как ИИ да разбере текущото състояние на света, но може да опише някаква крайна част от лентата и информацията върху тази крайна част.

Дори и Min-Max алгоритмът не е пълна емулация, заради комбинаторната експлозия. Вместо това, Min-Max прави частична емулация като обхожда само първите няколко хода. Когато в описанието на света има полуразрешимо правило, тогава ИИ ще използва това правило само в едната посока. Например правилото „Ако съществува доказателство, тогава твърдението е вярно“. Хората използват това правило, когато има доказателство и когато те са го намерили. Когато няма доказателство, тогава това правило не се използва, защото няма как да разберем, че доказателство действително няма.

Заключение

Сведохме задачата за създаването на ИИ до една чисто логическа задача. От нас сега се иска да създадем език за описание на светове и този език ще е логически, защото на него ще могат да се опишат неизчислими функции. Ако езика описваше само изчислими функции, тогава това щеше да е език за програмиране, а не логически език.

Основните градивни елементи на нашия нов език това ще са Event-Driven моделите. Това ще са простите модули, които ще откриваме един по един. С тези модули ще представим зависимости, алгоритми и явления.

След това ще направим следващата абстракция като въведем обектите. Обектите няма да могат да бъдат наблюдавани директно, а ще ги засичаме индиректно като наблюдаваме техни свойства. Свойството е специално явление, което се наблюдава когато наблюдаваме обект с това свойство. Тоест, свойството също се представя с ED модел.

Следващата абстракция, това е агентът. Също като обектите, агентите няма да можем да ги засечем директно. Тях ще ги наблюдаваме индиректно чрез техните действия. Откриването на агенти е трудна задача. Хората успяват да открият агенти, но за целта те ги търсят навсякъде. Когато нещо се случи, хората веднага намират обяснение в някакъв агент, който го е извършил. Зад всяко събитие хората виждат като извършител или човек, или животно, или божество. Много рядко приемат, че това се е случило от само себе си. ИИ трябва да подходи по същия начин като хората и да търси агентите навсякъде.

Когато ИИ намери агент трябва да започне да го изучава и да се опитва да се свърже с него. Да намери агент, значи да го измисли. Когато ИИ измисли съществуващ агент, тогава можем да кажем, че го е намерил. Когато си измисли несъществуващ агент, тогава е по-добре да кажем, че си е измислил нещо несъществуващо. Дали агентите са реални или измислени няма голямо значение. Важното е описанието на света получено чрез тези агенти да е адекватно и да дава добри резултати.

ИИ ще изучава агентите като ги класифицира като приятели и като врагове. Ще отбелязва дали са умни и дали са благодарни (съответно отмъстителни). ИИ ще се опитва да се свързва с агентите. За целта първо трябва да разбере към какво се стреми всеки от тях и да му предложи това, което иска агентът и в замяна да се опита да получи нещо полезно за себе си. Тази размяна на блага се нарича изпълнение на коалиционна стратегия. Обикновено се предполага, че агентите се срещат извън света и там се уговарят каква да бъде тяхната коалиционна стратегия. Тъй като няма как агентите да се срещнат извън света, ние ще предполагаме, че те общуват вътре в света. Принципа на общуването е: „Ще ти направя добро и очаквам да ми го върнеш.“ Другият принцип е: „Аз ще се държа предсказуемо и очаквам ти да разбереш какво е моето поведение и да започнеш да изпълняваш коалиционна стратегия (да се държиш така, че и за двама ни да има полза).“

По този начин ние общуваме с кучетата. Даваме им кокал и веднага се сприятеляваме. Какво получаваме в замяна? В замяна те не ни лаят и не ни хапят, а това никак не е малко. По-нататък може да се достигне до по-сложни комуникации. Може да покажем на агента алгоритъм и да искаме от него той да го изпълни. Така може да научим кучето да дава лапа. Още по-нататък може да се достигне до език като се асоциират обекти с явления. Например, произнесената дума е явление и ако това явление се асоциира с даден обект или алгоритъм, тогава агентът може като чуе думата да изпълни алгоритъма. Например, кучето, като си чуе името, може да дойде при вас или ако чуе „чехли“ може да ви донесе чехлите.

Виждате, че езикът за описание на светове може чрез простите модули, от които се състои, да описва доста сложни светове с много агенти и сложни взаимоотношения помежду им. Това, което надграждаме над простите модули не може да виси във въздуха и трябва да стъпи на някаква стабилна основа. Именно Event-Driven моделите са основата, която ще изгради езика за описание на светове и на базата, на които ще изградим всички по-сложни абстракции.

References

[1] Yiannis N. Moschovakis (2001). What is an algorithm? *Mathematics unlimited – 2001 and beyond, edited by B. Engquist and W. Schmid, Springer, 2001, pages 919-936.*

[2] Yiannis N. Moschovakis (2018). Abstract recursion and intrinsic complexity. *Cambridge University Press, Lecture Notes in Logic, Volume 48, ISBN: 9781108415583.*

[3] Dimiter Dobrev (2013). Giving the AI definition a form suitable for the engineer. *arXiv:1312.5713 [cs.AI]*.

[4] Dimiter Dobrev (2017). How does the AI understand what's going on. *International Journal "Information Theories and Applications", Vol. 24, Number 4, 2017, pp.345-369.*

[5] Dimiter Dobrev (2019). Before we can find a model, we must forget about perfection. *arXiv:1912.04964 [cs.AI]*.

[6] Dimiter Dobrev (2019). Event-Driven Models. *International Journal "Information Models and Analyses", Volume 8, Number 1, 2019, pp. 23-58.*

[7] Dimiter Dobrev (2020). AI Unravels the Chess. *http://dobrev.com/software/AI_unravels_the_chess.zip*.

[8] Dimiter Dobrev (2020). Strawberry Prolog, version 5.1. *http://dobrev.com/* .