Maszyny Gödla: w pełni samoodniesieniowe, optymalne, uniwersalne, samodoskonalące się maszyny

Przedstawiamy pierwszą klasę matematycznie rygorystycznych, ogólnych, w pełni samoodniesieniowych, samodoskonalących się, optymalnie wydajnych rozwiązywaczy problemów. Zainspirowany słynnymi samoodniesieniowymi formułami Kurta Gödla (1931), taki rozwiązywacz problemów przepisuje dowolną część swojego kodu, gdy tylko znajdzie dowód, że przepisanie jest przydatne, gdzie zależna od problemu funkcja użyteczności, sprzęt i cały początkowy kod są opisane przez aksjomaty zakodowane w początkowym wyszukiwarce dowodów, która jest również częścią początkowego kodu. Wyszukiwarka systematycznie i wydajnie testuje obliczalne techniki dowodowe (programy, których wyniki są dowodami), aż znajdzie udowodnioną, użyteczną, obliczalną samoprzepisanie. Pokazujemy, że takie samoprzepisanie jest globalnie optymalne — bez lokalnych maksimów! — ponieważ kod musiał najpierw udowodnić, że nie jest przydatne kontynuowanie poszukiwań dowodu alternatywnych samoprzepisów. W przeciwieństwie do poprzednich metod niesamoodniesieniowych opartych na wbudowanych wyszukiwarkach dowodów, nasza nie tylko może pochwalić się optymalnym rzędem złożoności, ale także może optymalnie zmniejszyć wszelkie spowolnienia ukryte przez notację O(), pod warunkiem, że użyteczność takich przyspieszeń jest w ogóle udowadnialna.

Wprowadzenie i zarys

W 1931 roku Kurt Gödel użył arytmetyki elementarnej do zbudowania uniwersalnego języka programowania do kodowania dowolnych dowodów, biorąc pod uwagę dowolny wyliczalny zbiór aksjomatów. Następnie skonstruował samoodniesieniowe stwierdzenia formalne, które twierdzą, że są nieudowodnialne, wykorzystując sztuczkę diagonalizacji Cantora , aby wykazać, że systemy formalne, takie jak tradycyjna matematyka, są albo wadliwe w pewnym sensie, albo zawierają nieudowodnione, ale prawdziwe stwierdzenia. Od czasu przedstawienia przez Gödla podstawowych ograniczeń dowodzenia i obliczeń oraz późniejszej konstrukcji przez Konrada Zuse pierwszego działającego programowalnego komputera (1935-1941), przeprowadzono wiele prac nad wyspecjalizowanymi algorytmami rozwiązującymi problemy zaczerpnięte z mniej lub bardziej ogólnych klas problemów. Najwyraźniej jednak jeden niezwykły fakt do tej pory umknął uwadze informatyków: możliwe jest użycie samoodniesieniowych systemów dowodowych do zbudowania optymalnie wydajnych, a jednocześnie bardzo prostych pod względem koncepcyjnym uniwersalnych rozwiązywaczy problemów. Wszystkie tradycyjne algorytmy rozwiązywania problemów/uczenia maszynowego/uczenia się przez wzmacnianie są wbudowane. Niektóre z nich są zaprojektowane w celu ulepszenia pewnego ograniczonego typu polityki poprzez doświadczenie, ale nie są częścią modyfikowalnej polityki i nie mogą się same ulepszyć w teoretycznie uzasadniony sposób. Ludzie są potrzebni do tworzenia nowych/lepszych algorytmów rozwiązywania problemów i dowodzenia ich przydatności przy odpowiednich założeniach. Wyeliminujmy ograniczającą potrzebę wysiłku człowieka w najbardziej ogólny możliwy sposób, pozostawiając całą pracę, w tym poszukiwanie dowodów, systemowi, który może przepisać i ulepszyć siebie w dowolny obliczalny sposób i w najbardziej wydajny sposób. Aby zająć się tym „Wielkim Problemem Sztucznej Inteligencji”, wprowadzamy nową klasę optymalnych, w pełni samoodnoszących się ogólnych rozwiązywaczy problemów zwanych maszynami Gödla . Są to uniwersalne systemy rozwiązywania problemów, które oddziałują z pewnym (częściowo obserwowalnym) środowiskiem i mogą w zasadzie modyfikować się bez istotnych ograniczeń poza ograniczeniami obliczalności. Ich początkowy algorytm nie jest wbudowany; może całkowicie przepisać się sam, ale tylko jeśli wyszukiwarka dowodów osadzona w początkowym algorytmie może najpierw udowodnić, że przepisanie jest przydatne, biorąc pod uwagę sformalizowaną funkcję użyteczności odzwierciedlającą czas obliczeń i oczekiwany przyszły sukces (np. nagrody). Zobaczymy, że

samoprzepisywanie wynikające z tego podejścia jest w rzeczywistości globalnie optymalne (Twierdzenie 1, Sekcja 4), w odniesieniu do dobrze znanych podstawowych ograniczeń dowodzenia Gödla. Ograniczenia te nie powinny nas martwić; jeśli nie ma dowodu na użyteczność jakiegoś samoprzepisywania, to ludzie również nie mogą wiele zrobić. Początkowa wyszukiwarka dowodów jest O()-optymalna (ma optymalny rząd złożoności) w rozumieniu Twierdzenia 2, Sekcja 5. Jednak w przeciwieństwie do układów Huttera (Sekcja 2), maszyna Gödla może dodatkowo przyspieszyć swoją wyszukiwarkę dowodów, aby spełnić dowolne formalizowalne pojęcia optymalności wykraczające poza te wyrażalne w notacji O(). Nasze podejście daje pierwsze teoretycznie solidne, w pełni samoodniesieniowe, optymalne, ogólne rozwiązywacze problemów. Zarys. Sekcja 2 przedstawia podstawowe koncepcje, relacje do najbardziej istotnych wcześniejszych prac i ograniczenia. Sekcja 3 przedstawia istotne szczegóły samoodniesieniowego systemu aksjomatycznego, Sekcja 4 Twierdzenie o globalnej optymalności 1, a Sekcja 5 O()-optymalny (Twierdzenie 2) początkowy poszukiwacz dowodów. Sekcja 6 zawiera przykłady i dodatkowe relacje do wcześniejszych prac, krótko omawia kwestie takie jak techniczne uzasadnienie świadomości i udziela odpowiedzi na kilka często zadawanych pytań na temat maszyn Gödla.

Podstawowy przegląd, powiązanie z poprzednimi pracami i ograniczenia

Wiele tradycyjnych problemów informatyki wymaga tylko jednego definiującego problem wejścia na początku procesu rozwiązywania problemu. Na przykład, początkowe wejście może być dużą liczbą całkowitą, a celem może być jej rozłożenie na czynniki. W dalszej części rozważymy jednak również bardziej ogólny przypadek, w którym rozwiązanie problemu wymaga interakcji z dynamicznym, początkowo nieznanym środowiskiem, które generuje ciągły strumień wejść i sygnałów sprzężenia zwrotnego, jak w przypadku zadań sterowania autonomicznym robotem, gdzie celem może być maksymalizacja oczekiwanej skumulowanej przyszłej nagrody [19]. Może to wymagać rozwiązania zasadniczo dowolnych problemów.

Notacja i konfiguracja

O ile nie zaznaczono inaczej lub nie jest to oczywiste, w całym artykule zakłada się, że nowo wprowadzone zmienne i funkcje obejmują zakres domyślny w kontekście. B oznacza alfabet binarny {0, 1}, B^* zbiór możliwych ciągów bitów nad B, l(q) oznacza liczbę bitów w ciągu bitów q; q_n n-ty bit q; λ pusty ciąg (gdzie l(λ) = 0); $q_{m:n} = λ$ jeśli m > n i $q_m q_{m+1} \ldots q_n$ w przeciwnym razie (gdzie $q_0 := q_{0:0} := λ$). Nasz sprzęt (np. uniwersalna lub ograniczona przestrzennie maszyna Turinga lub abstrakcyjny model komputera osobistego) ma pojedynczy okres życia, który składa się z dyskretnych cykli lub kroków czasowych t = 1, 2, ... Jego całkowity okres życia T może być znany z góry lub nie. W dalszej części wartość dowolnej zmiennej Q zmieniającej się w czasie w czasie t będzie oznaczana przez Q(t). Podczas każdego cyklu nasz sprzęt wykonuje elementarną operację, która wpływa na jego zmienny stan $s \in S \subset B^*$ i prawdopodobnie również zmienny stan środowiska $Env \in E$. (Tutaj nie musimy jeszcze określać zbioru E zależnego od problemu). Istnieje zakodowana na stałe funkcja przejścia stanu $F : S \times E \rightarrow S$. Dla t > 1, s(t) = F(s(t − 1),Env(t − 1)) jest stanem w punkcie, w którym operacja sprzętowa cyklu t − 1 jest zakończona, ale operacja cyklu t jeszcze się nie rozpoczęła. Env(t) może zależeć od przeszłych działań wyjściowych zakodowanych w s(t−1) i jest jednocześnie aktualizowane lub (prawdopodobnie) obliczane przez potencjalnie reaktywne środowisko. Aby wygodnie rozmawiać o programach i danych, często przypisujemy nazwy pewnym zmiennym łańcuchowym zakodowanym jako komponenty lub podłańcuchy s. Szczególnie interesujące są 3 zmienne o nazwie time, x, y, p:

1. W czasie t zmienna time przechowuje unikalną reprezentację binarną t. Inicjujemy time(1) = ‘1’, ciąg bitów składający się tylko z jedynki. Sprzęt zwiększa time od jednego cyklu do następnego. Wymaga to co najwyżej O(log t) i średnio tylko O(1) kroków obliczeniowych.

2. Zmienna x przechowuje dane wejściowe ze środowiska. Dla t > 1, x(t) może różnić się od x(t − 1) tylko wtedy, gdy program uruchomiony na maszynie Gödla wykonał specjalną instrukcję żądania danych wejściowych w czasie t − 1. Mówiąc ogólnie, opóźnienia między kolejnymi danymi wejściowymi powinny być wystarczająco duże, aby programy mogły wykonywać pewne elementarne obliczenia na danych wejściowych, takie jak kopiowanie ich do pamięci wewnętrznej (zarezerwowana część s) przed nadejściem kolejnych danych wejściowych.

3. y(t) jest wyjściowym ciągiem bitów, który może następnie wpływać na środowisko, gdzie y(1) domyślnie = „0”. Na przykład y(t) można interpretować jako sygnał sterujący dla robota manipulującego środowiskiem, którego działania mogą mieć wpływ na przyszłe dane wejściowe.

4. p(1) jest początkowym oprogramowaniem: programem implementującym oryginalną politykę interakcji ze środowiskiem i wyszukiwania dowodów. Szczegóły zostaną omówione poniżej.

W dowolnym momencie t (1 ≤ t ≤ T) celem jest maksymalizacja przyszłego sukcesu lub użyteczności. Typowa funkcja użyteczności „wartość do przejścia” (do zmaksymalizowania) ma postać u(s,Env): S ×E →R, gdzie R jest zbiorem liczb rzeczywistych:

$$u(s, Env) = E_\mu \left[\sum_{\tau=time}^{T} r(\tau) \;\middle|\; s, Env \right]$$

gdzie r(t) jest wartością rzeczywistą nagrody wejściowej (zakodowanej w s(t)) w czasie t, Eμ(· | ·) oznacza warunkowy operator oczekiwania w odniesieniu do pewnego potencjalnie nieznanego rozkładu μ ze zbioru M możliwych rozkładów (M odzwierciedla wszystko, co wiadomo o potencjalnie probabilistycznych reakcjach środowiska), a wyżej wymieniony czas = czas(y) jest funkcją stanu s, który jednoznacznie identyfikuje bieżący cykl. Należy zauważyć, że bierzemy pod uwagę możliwość wydłużenia oczekiwanego okresu życia Eμ(T | s,Env) poprzez odpowiednie działania. Alternatywne formalizowalne funkcje użyteczności mogłyby sprzyjać poprawie najgorszego przypadku zamiast oczekiwanej przyszłej wydajności lub wyższego pobierania nagrody na przedział czasu itp.

Podstawowa idea maszyny Gödla

Nasza maszyna staje się samoodniesieniową maszyną Gödla poprzez załadowanie jej określoną formą zależnego od maszyny, samomodyfikującego się kodu p. Początkowy kod p(1) w kroku czasowym 1 obejmuje (zwykle suboptymalną) podprocedurę rozwiązywania problemów do interakcji ze środowiskiem, taką jak Q-learning , oraz ogólną podprocedurę wyszukiwania dowodów (sekcja 5), która systematycznie tworzy pary (switchprog, proof) (zmienne podciągi s), aż znajdzie dowód twierdzenia docelowego, które zasadniczo stwierdza: „natychmiastowe przepisanie p przez bieżący program switchprog na danej maszynie implikuje wyższą użyteczność niż pozostawienie p takim, jakie jest”. Następnie wykonuje switchprog, który może całkowicie przepisać p, w tym wyszukiwarkę dowodów. Sekcja 3 wyjaśni szczegóły niezbędnego początkowego systemu aksjomatycznego A zakodowanego w p(1). Twierdzenie o globalnej optymalności (twierdzenie 1, sekcja 4) pokazuje, że ta strategia samodoskonalenia nie jest zachłanna: ponieważ użyteczność „pozostawienia p takim, jakie jest” implicite ocenia wszystkie możliwe alternatywne switchprog, które niezmodyfikowany p może znaleźć później, otrzymujemy globalnie optymalną autozmianę — bieżący switchprog reprezentuje najlepszą ze wszystkich możliwych odpowiednich autozmian, w odniesieniu do danych ograniczeń zasobów i początkowej strategii wyszukiwania dowodu.

Techniki dowodowe i O()-optymalny początkowy wyszukiwacz dowodu.

Sekcja 5 przedstawi O()-optymalną inicjalizację wyszukiwarki dowodu, to znaczy taką, która ma optymalny rząd złożoności (twierdzenie 2). Nadal jednak pozostanie wiele miejsca na samodoskonalenie ukryte przez notację O(). Wyszukiwarka używa internetowego rozszerzenia Universal Search do systematycznego testowania internetowych technik dowodowych, które są programami generującymi dowody, które mogą odczytywać części stanu s (podobnie matematycy są często bardziej zainteresowani technikami dowodowymi niż twierdzeniami). Aby udowodnić docelowe twierdzenia, jak powyżej, techniki dowodowe mogą wywoływać specjalne instrukcje generowania aksjomatów i stosowania reguł wnioskowania w celu przedłużenia bieżącego dowodu o twierdzenia. Tutaj aksjomatyczny system A zakodowany w p(1) obejmuje aksjomaty opisujące (a) w jaki sposób każda instrukcja wywołana przez program uruchomiony na danym sprzęcie zmieni stan maszyny s (w tym wskaźniki instrukcji itp.) z jednego kroku do następnego (tak, że techniki dowodowe mogą wnioskować o efektach dowolnego programu, w tym wyszukiwarki dowodów), (b) sam początkowy program p(1) (sekcja 3 pokaże, że jest to możliwe bez wprowadzania cykliczności), (c) stochastyczne właściwości środowiskowe, (d) formalną funkcję użyteczności u, np. równanie (1). Ocena użyteczności automatycznie uwzględnia koszty obliczeniowe wszystkich działań, łącznie z poszukiwaniem dowodów.

Powiązanie z poprzednią pracą Huttera

Niesamoodniesieniowy, ale nadal O()-optymalny „najszybszy" algorytm Huttera dla wszystkich dobrze zdefiniowanych problemów Hsearch używa wbudowanego narzędzia do wyszukiwania dowodów metodą brute force. Załóżmy dyskretne domeny wejścia/wyjścia X/Y, formalną specyfikację problemu $f : X \rightarrow Y$ (powiedzmy, opis funkcjonalny tego, jak liczby całkowite rozkładają się na czynniki pierwsze) i konkretne $x \in X$ (powiedzmy, liczbę całkowitą, która ma zostać rozłożona na czynniki). Hsearch porządkuje wszystkie dowody odpowiedniego systemu aksjomatycznego według rozmiaru, aby znaleźć programy q, które dla wszystkich $z \in X$ obliczają f(z) w granicach czasowych $t_q(z)$. Jednocześnie spędza większość czasu na wykonywaniu q z najlepszym obecnie udowodnionym ograniczeniem czasowym $t_q(x)$. Okazuje się, że Hsearch jest tak szybki jak najszybszy algorytm, który w sposób udowodniony oblicza f(z) dla wszystkich $z \in X$, z wyjątkiem stałego czynnika mniejszego niż $1+ \varepsilon$ (dowolnego > 0) i stałej addytywnej specyficznej dla f, ale niezależnej od x. Ta stała może być jednak ogromna. Aixi(t,l) Huttera jest powiązana. W dyskretnym cyklu k = 1, 2, 3, . . . czasu życia Aixi(t,l) działanie y(k) skutkuje percepcją x(k) i nagrodą r(k), gdzie wszystkie wielkości mogą zależeć od całej historii. Używając uniwersalnego komputera, takiego jak maszyna Turinga, Aixi(t,l) potrzebuje początkowej fazy konfiguracji offline (przed interakcją ze środowiskiem), w której używa wbudowanego brute force proof searcher do zbadania wszystkich dowodów o długości co najwyżej L, filtrując te, które identyfikują programy (o maksymalnym rozmiarze l i maksymalnym czasie wykonania t na cykl), które nie tylko mogłyby wchodzić w interakcję ze środowiskiem, ale które dla wszystkich możliwych historii interakcji również poprawnie przewidują dolną granicę ich własnej oczekiwanej przyszłej nagrody. W cyklu k, Aixi(t,l) uruchamia następnie wszystkie programy zidentyfikowane w fazie konfiguracji (co najwyżej 2l), znajduje ten z najwyższą samooceną i wykonuje odpowiadającą mu akcję. Niezależny od problemu czas konfiguracji (w którym wykonywana jest prawie cała praca) wynosi $O(L \cdot 2^L)$. Czas online na cykl wynosi $O(t \cdot 2^l)$. Oba są stałe, ale zazwyczaj ogromne. Zalety i nowość maszyny Gödla. Istnieją zasadnicze różnice pomiędzy maszyną Gödla a Hsearch Huttera i Aixi(t,l), w tym:

1. Dowody twierdzeń Hsearch i Aixi(t,l) są wbudowanymi, niesamoodniesieniowymi, niemodyfikowalnymi metaalgorytmami, które nie mogą się same ulepszyć. Oznacza to, że zawsze będą cierpieć z powodu tych samych ogromnych stałych spowolnień (zwykle 101000) ukrytych w notacji O(). Ale w zasadzie nie ma nic, co uniemożliwiałoby naszemu prawdziwie samoodniesieniowemu kodowi

udowodnienie i wykorzystanie drastycznych redukcji takich stałych w najlepszy możliwy sposób, który udowodniony stanowi ulepszenie, jeśli w ogóle istnieje.

2. Wykazanie O()-optymalności Hsearch i Aixi(t,l) zależy od sprytnego przydzielenia czasu obliczeniowego niektórym z ich niemodyfikowalnych metaalgorytmów. Nasze globalne twierdzenie optymalności (twierdzenie 1, sekcja 4) jest jednak uzasadnione przez zupełnie inny typ rozumowania, który rzeczywiście wykorzystuje i zasadniczo zależy od faktu, że nie ma w ogóle niezmienialnego oprogramowania, a sam wyszukiwacz dowodów jest czytelny i modyfikowalny oraz może być ulepszany. To jest również powód, dla którego jego samodoskonalenie może być czymś więcej niż tylko O()-optymalne.

3. Hsearch używa „sztuczki" polegającej na udowadnianiu więcej niż jest to konieczne, co również znika w czasami dość mylącej notacji O(): marnuje czas na znajdowanie programów, które w sposób udowodniony obliczają f(z) dla wszystkich z ∈ X, nawet gdy bieżący f(x)(x ∈ X) jest jedynym obiektem zainteresowania. Maszyna Gödla musi jednak udowodnić tylko to, co jest istotne dla jej celu sformalizowanego przez u. Na przykład ogólne u równania. (1) całkowicie ignoruje ograniczoną koncepcję O()-optymalności, ale zamiast tego formalizuje silniejszy typ optymalności, który nie ignoruje ogromnych stałych tylko dlatego, że są stałe.

4. Zarówno maszyna Gödla, jak i Aixi(t,l) mogą maksymalizować oczekiwaną nagrodę (Hsearch nie może). Jednak maszyna Gödla jest bardziej elastyczna, ponieważ możemy podłączyć dowolny typ formalizowalnej funkcji użyteczności (np. nagrodę w najgorszym przypadku) i w przeciwieństwie do Aixi(t,l) nie wymaga wyliczalnego rozkładu środowiskowego.

Niemniej jednak możemy użyć Aixi(t,l) lub Hsearch, aby zainicjować podciąg e p, który odpowiada za interakcję ze środowiskiem. Maszyna Gödla zastąpi e, gdy tylko znajdzie udowodnioną lepszą strategię.

Ograniczenia maszyn Gödla

Podstawowe ograniczenia są ściśle powiązane z tymi, które zostały po raz pierwszy zidentyfikowane w słynnym artykule Gödla na temat formuł samoodniesieniowych . Każdy formalny system obejmujący arytmetykę (lub ZFC itp.) jest wadliwy lub dopuszcza nieudowodnione, ale prawdziwe stwierdzenia. Stąd nawet maszyna Gödla z nieograniczonymi zasobami obliczeniowymi musi ignorować te samodoskonalenia, których skuteczności nie może udowodnić, np. z powodu braku wystarczająco silnych aksjomatów w A. W szczególności można skonstruować patologiczne przykłady środowisk i funkcji użyteczności, które uniemożliwiają maszynie udowodnienie twierdzenia docelowego. Porównaj twierdzenie Bluma o przyspieszeniu oparte na pewnych nieobliczalnych predykatach. Podobnie realistyczna maszyna Gödla z ograniczonymi zasobami nie może czerpać korzyści z samodoskonaleń, których przydatności nie może udowodnić w ramach swoich ograniczeń czasowych i przestrzennych. Jednakże w przeciwieństwie do poprzednich metod, może ona w zasadzie wykorzystać przynajmniej udowodnione dobre przyspieszenia dowolnej części swojego początkowego oprogramowania, w tym tych części, które są odpowiedzialne za ogromne (ale niezależne od klasy problemu) spowolnienia, ignorowane przez wcześniejsze podejścia.

Podstawowe szczegóły jednej reprezentatywnej maszyny Gödla

Dowodzenie twierdzeń wymaga schematu aksjomatów dającego wyliczalny zbiór aksjomatów formalnego systemu logicznego A, którego wzory i twierdzenia są ciągami symboli nad pewnym skończonym alfabetem, który może obejmować tradycyjne symbole logiki (takie jak →, ∧,=, (,), ∀, ∃, . . ., c_1, c_2, . . . , f_1, f_2, . . .), teorii prawdopodobieństwa (takie jak E(·), operator oczekiwania), arytmetyki

(+,−, /,=, Σ,<, . . .), manipulacji ciągami (w szczególności symboli reprezentujących dowolną część stanu s w dowolnym czasie, takich jak $s_{7:88}(5555)$). Dowód to ciąg twierdzeń, z których każde jest aksjomatem lub wywnioskowane z poprzednich twierdzeń poprzez zastosowanie jednej z reguł wnioskowania, takich jak modus ponens w połączeniu z unifikacją. Pozostała część pominie standardową wiedzę, którą można znaleźć w dowolnym podręczniku teorii dowodu. Zamiast wypisywać wszystkie aksjomaty konkretnego A w żmudny sposób, skupimy się na nowych i krytycznych szczegółach: jak pokonać problemy z samoodniesieniem i jak poradzić sobie z potencjalnie delikatnym generowaniem dowodów online, które mówią o samym aktualnie działającym generatorze dowodów i mają na niego wpływ.

Techniki dowodowe

Wyszukiwarki dowodów metodą siłową (używane w Aixi(t,l) i Hsearch Huttera) systematycznie generują wszystkie dowody w kolejności ich rozmiarów. Aby wyprodukować pewien dowód, zajmuje to czas wykładniczy w stosunku do rozmiaru dowodu. Zamiast tego nasze O()-optymalne p(1) wygeneruje wiele dowodów o niskiej złożoności algorytmicznej znacznie szybciej. Systematycznie testuje techniki dowodowe napisane w uniwersalnym języku L zaimplementowanym w p(1). Na przykład L może być wariantem PROLOGU [7] lub uniwersalnego języka programowania inspirowanego Forth używanego w niedawnych pracach nad optymalnym wyszukiwaniem. Technika dowodowa składa się z instrukcji, które pozwalają na odczytanie dowolnej części s, takiej jak dane wejściowe zakodowane w zmiennej x (podciąg s) lub kod p(1). Może zapisywać na sp, części s zarezerwowanej dla tymczasowych wyników. Może również przepisać switchprog i wytworzyć przyrostowo rosnący dowód umieszczony w zmiennej ciągu proof przechowywanej gdzieś w s. proof i s^p są resetowane do pustego ciągu na początku każdego nowego testu techniki dowodzenia. Oprócz standardowych instrukcji arytmetycznych i definiujących funkcje , które modyfikują s^p, język programowania L zawiera specjalne instrukcje przedłużające bieżący dowód o poprawne twierdzenia, ustawiające switchprog i sprawdzające, czy znaleziono udowodniony optymalny program modyfikujący p i czy powinien on zostać teraz wykonany. Niektóre długie dowody można wytworzyć za pomocą krótkich technik dowodzenia. Natura sześciu poniższych instrukcji modyfikujących dowody (nie ma innych) uniemożliwia wstawienie niepoprawnego twierdzenia do dowodu, trywializując weryfikację dowodu:

1. get-axiom(n) przyjmuje jako argument liczbę całkowitą n obliczoną przez prefiks aktualnie testowanej techniki dowodowej za pomocą instrukcji arytmetycznych, takich jak te używane w poprzedniej pracy [45]. Następnie dołącza n-ty aksjomat (jeśli istnieje, zgodnie ze schematem aksjomatu poniżej) jako twierdzenie do bieżącego ciągu twierdzeń w dowodzie. Początkowy schemat aksjomatu koduje:

a) Aksjomaty sprzętowe opisujące sprzęt, formalnie określające, w jaki sposób pewne komponenty s (inne niż dane wejściowe środowiska x) mogą się zmieniać od jednego cyklu do następnego. Na przykład, jeśli sprzęt jest maszyną Turinga2 (TM), to s(t) jest ciągiem bitów, który koduje bieżącą zawartość wszystkich taśm TM, pozycje jej głowic skanujących i bieżący stan wewnętrzny skończonego automatu TM, podczas gdy F określa tablicę wyszukiwania TM, która mapuje dowolną możliwą kombinację stanu wewnętrznego i bitów powyżej głowic skanujących na nowy stan wewnętrzny i działanie takie jak: zastąpienie bieżącego bitu pewnej głowicy przez 1/0, zwiększenie (przesunięcie w prawo) lub zmniejszenie (przesunięcie w lewo) pewnej głowicy skanującej, odczytanie i skopiowanie następnego bitu wejściowego do komórki powyżej głowicy skanującej taśmy wejściowej itd. Alternatywnie, jeśli sprzęt jest dany przez abstrakcyjny model nowoczesnego mikroprocesora z ograniczoną pamięcią, s(t) zakoduje bieżącą zawartość pamięci, wartości rejestrów, wskaźniki instrukcji itd. Na przykład następujący aksjomat może opisywać, w jaki sposób wskaźnik instrukcji pewnego 64-bitowego sprzętu przechowywany w s1:64 jest stale zwiększany, dopóki istnieje brak przepełnienia, a wartość s65 nie wskazuje na konieczność wykonania skoku do innego adresu:

$$(\forall t \forall n : [(n < 2^{64} - 1) \wedge (n > 0) \wedge (t > 1) \wedge (t < T)$$
$$\wedge (string2num(s_{1:64}(t)) = n) \wedge (s_{65}(t) = \text{‘0’})]$$
$$\rightarrow (string2num(s_{1:64}(t+1)) = n + 1))$$

Tutaj semantyka użytych symboli, takich jak „(”, „>” i „→” (implikuje), jest tradycyjna, podczas gdy „string2num” symbolizuje funkcję tłumaczącą ciągi bitów na liczby. Oczywiste jest, że każdy abstrakcyjny model sprzętowy można w pełni zaksjomatizować w podobny sposób.

b) Aksjomaty nagrody definiujące koszty obliczeniowe dowolnej instrukcji sprzętowej i koszty fizyczne działań wyjściowych (np. sygnały sterujące y(t) zakodowane w s(t)). Powiązane aksjomaty przypisują wartości pewnym zdarzeniom wejściowym (zakodowanym w zmiennej x, podciągu s) reprezentującym nagrodę lub karę (np. gdy robot sterowany maszynowo przez Gödla uderzy w przeszkodę). Dodatkowe aksjomaty definiują całkowitą wartość życia maszyny Gödla jako funkcję skalarną wszystkich nagród (np. ich sumy) i kosztów poniesionych między cyklami 1 i T itd. Na przykład załóżmy, że $s_{17:18}$ można zmienić tylko za pomocą zewnętrznych danych wejściowych; poniższy przykładowy aksjomat mówi, że całkowita nagroda wzrasta o 3, gdy takie dane wejściowe są równe „11” (niewyjaśnione symbole niosą oczywiste znaczenie):

$$(\forall t_1 \forall t_2 : [(t_1 < t_2) \wedge (t_1 \geq 1) \wedge (t_2 \leq T) \wedge (s_{17:18}(t_2) = \text{‘11’})]$$
$$\rightarrow [R(t_1, t_2) = R(t_1, t_2 - 1) + 3]),$$

gdzie $R(t_1, t_2)$ jest interpretowane jako kumulatywna nagroda między czasami t_1 i t_2. Oczywiste jest, że każdy formalny schemat wytwarzania nagród może być w pełni zaksjomatyzowany w podobny sposób.

c) Aksjomaty środowiska ograniczające sposób, w jaki środowisko będzie wytwarzać nowe dane wejściowe (zakodowane w pewnych podciągach s) w reakcji na sekwencje danych wyjściowych y zakodowane w s. Na przykład może być wiadome z góry, że środowisko jest próbkowane z nieznanego rozkładu prawdopodobieństwa, który jest obliczalny, biorąc pod uwagę poprzednią historię lub co najmniej obliczalny w granicach. Albo, bardziej restrykcyjnie, środowisko może być jakimś nieznanym, ale deterministycznym programem komputerowym próbkowanym z Speed Prior, który przypisuje niskie prawdopodobieństwo środowiskom, które są trudne do obliczenia jakąkolwiek metodą. Albo interfejs do środowiska jest markowski , to znaczy, że bieżące dane wejściowe zawsze jednoznacznie identyfikują stan środowiska — w tym szczególnym przypadku wykonano wiele prac. Co więcej, środowisko może ewoluować w całkowicie przewidywalny sposób znany z góry. Wszystkie takie wcześniejsze założenia są doskonale formalizowalne w odpowiednim A (w przeciwnym razie nie moglibyśmy pisać o nich prac naukowych).

d) Aksjomaty niepewności; aksjomaty manipulacji ciągiem: standardowe aksjomaty arytmetyki i rachunku różniczkowego i całkowego oraz teorii prawdopodobieństwa oraz statystyki i manipulacji ciągiem, które (w połączeniu z aksjomatami środowiska) umożliwiają konstruowanie dowodów dotyczących (możliwie niepewnych) właściwości przyszłych wartości s(t), a także ograniczeń oczekiwanego pozostałego czasu życia/kosztów/nagród, biorąc pod uwagę pewien czas τ i pewne hipotetyczne wartości dla składników s(τ) itd. Przykładowe twierdzenie mówiące coś o oczekiwanych właściwościach przyszłych danych wejściowych x może wyglądać następująco:

$$(\forall t_1 \forall \mu \in M : [(1 \leq t_1) \wedge (t_1 + 15597 < T) \wedge (s_{5:9}(t_1) = \text{‘01011’})$$

$$\wedge(x_{40:44}(t_1) = \text{‘00000’})] \rightarrow (\exists t : [(t_1 < t < t_1 + 15597)$$
$$\wedge(P_\mu(x_{17:22}(t) = \text{‘011011’} \mid s(t_1)) > \frac{998}{1000})])),$$

gdzie Pμ(. | .) reprezentuje prawdopodobieństwo warunkowe względem aksjomatyzowanego rozkładu a priori μ z zestawu rozkładów M opisanych przez aksjomaty środowiska (punkt 1c). Biorąc pod uwagę konkretny formalizowalny sprzęt (punkt 1a) i formalizowalne założenia dotyczące potencjalnie probabilistycznego środowiska (punkt 1c), oczywiście można w pełni zaksjomatyzować wszystko, co jest potrzebne do rozumowania opartego na dowodach.

e) Aksjomaty stanu początkowego: Informacje o tym, jak zrekonstruować stan początkowy s(1) lub jego części, tak aby osoba poszukująca dowodu mogła budować dowody zawierające aksjomaty typu

$$(s_{\mathbf{m:n}}(1) = \mathbf{z}),\ e.g.:\ (s_{7:9}(1) = \text{‘010’}).$$

Tutaj i w pozostałej części używamy pogrubionej czcionki we wzorach, aby wskazać symbole zastępcze składni (takie jak m,n,z) dla ciągów symboli reprezentujących zmienne (takie jak m,n,z), których semantyka jest wyjaśniona w tekście (w obecnym kontekście z jest ciągiem bitów $s_{m:n}(1)$). Należy zauważyć, że nie jest fundamentalnym problemem pełne zakodowanie zarówno opisu sprzętu, jak i początkowego opisującego sprzęt p w samym p. Aby to zobaczyć, należy zauważyć, że niektóre oprogramowanie może zawierać program, który może wydrukować oprogramowanie.

f) Aksjomaty użyteczności opisujące ogólny cel w postaci funkcji użyteczności u.

2. apply-rule(k, m, n) przyjmuje jako argumenty indeks k (jeśli istnieje) reguły wnioskowania, takiej jak modus ponens (przechowywany na liście możliwych reguł wnioskowania zakodowanej w p(1)) oraz indeksy m, n dwóch wcześniej udowodnionych twierdzeń (ponumerowanych w kolejności ich utworzenia) w bieżącym dowodzie. Jeśli jest to stosowne, odpowiednia reguła wnioskowania jest stosowana do adresowanych twierdzeń, a wynikowe twierdzenie jest dołączane do dowodu. W przeciwnym razie aktualnie testowana technika dowodowa jest przerywana. Zapewnia to, że dowód nigdy nie zostanie uzupełniony o nieważne dowody.

3. delete-theorem(m) usuwa m-te twierdzenie w aktualnie przechowywanym dowodzie, zwalniając w ten sposób pamięć tak, że części s przechowujące dowody mogą być ponownie wykorzystane, a maksymalny rozmiar dowodu nie jest koniecznie ograniczony przez ograniczenia pamięci. Twierdzenia usunięte z dowodu nie mogą być jednak już uwzględnione przez apply-rule w celu wytworzenia dalszych przedłużeń dowodu.

4. set-switchprog(m,n) zastępuje switchprog przez $s^p_{m:n}$, pod warunkiem, że $s^p_{m:n}$ jest rzeczywiście niepustym podciągiem s^p, którego pamięć można zapisać za pomocą technik dowodowych.

5. state2theorem(m, n) przyjmuje dwa argumenty całkowite m, n i próbuje przekształcić bieżącą zawartość sm:n w twierdzenie w postaci

$$(s_{\mathbf{m:n}}(\mathbf{t_1}) = \mathbf{z}),\ e.g.:\ (s_{6:9}(7775555) = \text{‘1001’}).$$

gdzie t1 reprezentuje czas mierzony (przez sprawdzenie czasu) krótko po wywołaniu state2theorem, a z bistring sm:n(t1) (przypomnij sobie przypadek szczególny t1 = 1 z pozycji 1e). Tak więc akceptujemy

oznaczoną czasem bieżącą obserwowalną zawartość dowolnej części s jako twierdzenie, którego nie trzeba dowodzić w alternatywny sposób, powiedzmy, ze stanu początkowego s(1), ponieważ obliczenia do tej pory wykazały już, że twierdzenie jest prawdziwe. Tak więc możemy wykorzystać informacje przekazywane przez dane wejściowe ze środowiska i fakt, że czasami (ale nie zawsze) najszybszym sposobem ustalenia wyniku programu jest jego uruchomienie. Ten niekonwencjonalny interfejs online między składnią a semantyką wymaga jednak szczególnej ostrożności. Musimy unikać niespójnych wyników poprzez części s, które zmieniają się podczas odczytu. Na przykład, bieżąca wartość szybko zmieniającego się wskaźnika instrukcji IP (ciągle aktualizowanego przez sprzęt) może być zasadniczo nieczytelna w tym sensie, że wykonanie samej podprocedury odczytu zmodyfikuje IP wiele razy. Dla wygody (zwykle ograniczony) sprzęt można skonfigurować tak, aby przechowywał zawartość szybkich zmiennych sprzętowych co c cykli w zarezerwowanej części s, tak aby odpowiednia odmiana state2theorem() mogła przynajmniej przetłumaczyć pewne ostatnie wartości szybkich zmiennych na twierdzenia. Nie rozwiąże to jednak wszystkich problemów związanych z samoobserwacjami. Na przykład $s_{m:n}$, które ma zostać odczytane, może również zawierać własne, tymczasowe, stale zmieniające się zmienne wskaźnika ciągu procedury odczytu itd. Aby rozwiązać takie problemy na komputerach z ograniczoną pamięcią, state2theorem najpierw używa pewnego stałego protokołu, aby sprawdzić, czy bieżący $s_{m:n}$ jest w ogóle czytelny, czy też mógłby się zmienić, gdyby został odczytany przez pozostały kod state2theorem. Jeśli tak, lub jeśli m, n nie znajdują się w odpowiednim zakresie, instrukcja nie ma dalszego efektu. W przeciwnym razie dołącza zaobserwowane twierdzenie w postaci ($s_{m:n}$(t1) = z) do dowodu. Na przykład, jeśli aktualny czas to 7770000, to wywołanie state2theorem(6,9) może zwrócić twierdzenie ($s_{6:9}$(7775555) = ‘1001’), gdzie 7775555 − 7770000 = 5555 odzwierciedla czas potrzebny state2theorem na wykonanie początkowego sprawdzenia i odczytanie wiodących bitów z ciągle rosnącego czasu (czas odczytu również kosztuje czas), tak aby można było mieć pewność, że 7775555 jest niedawną właściwą etykietą czasu po rozpoczęciu state2theorem.

6. check() weryfikuje, czy cel wyszukiwania dowodu został osiągnięty. Najpierw sprawdza, czy ostatnie twierdzenie (jeśli takie istnieje) w dowodzie ma formę twierdzenia docelowego. Twierdzenie docelowe stwierdza, że biorąc pod uwagę bieżącą aksjomatyzowaną funkcję użyteczności u (element 1f), użyteczność przełączenia z p do bieżącego switchprog byłaby wyższa niż użyteczność kontynuowania wykonywania p (co powodowałoby ciągłe wyszukiwanie alternatywnych switchprog). Twierdzenia docelowe to ciągi symboli (zakodowane w s) w (dekodowanej) formie

$$(u[s(\mathbf{t_1}) \oplus (switchbit(\mathbf{t_1}) = \text{‘1’}), Env(\mathbf{t_1})] >$$
$$u[s(\mathbf{t_1}) \oplus (switchbit(\mathbf{t_1}) = \text{‘0’}), Env(\mathbf{t_1})]),$$

gdzie zmienna t_1 (reprezentowana przez symbol zastępczy składniowy t_1) oznacza krok czasu, podczas gdy wszystkie inne symbole należą do alfabetu rachunku dowodzącego twierdzeń, tj. zbiór możliwych twierdzeń docelowych jest sparametryzowany tylko przez t_1. Tutaj rachunek powinien dopuszczać notację $s(t_1)\bigoplus$(switchbit(t_1) = ‘b’) jako skrót dla stanu uzyskanego, gdy zastąpimy switchbit(t_1), prawdziwą wartość zmiennej bit switchbit (zakodowaną w s) w czasie t_1, przez b ∈ {0, 1}. Ułatwi to formułowanie twierdzeń, które porównują wartości uwarunkowane różnymi alternatywnymi hipotetycznymi własnościami $s(t_1)$. (Należy zauważyć, że $s(t_1)$ może być znane tylko częściowo przez obecną technikę dowodzenia nawet w środowiskach, w których $s(t_1)$ i switchbit(t_1) są w pełni ustalone dla wszystkich prawidłowych t_1.) Celem wprowadzenia t_1 jest poradzenie sobie z opóźnieniami czasowymi specyficznymi dla sprzętu, które mogą być zaangażowane w sprawdzanie i przełączanie — dopasowanie abstrakcyjnych ciągów symboli znalezionych podczas wyszukiwania dowodu do rzeczywistego bieżącego stanu maszyny Gödla może zająć znaczną ilość czasu. Jeśli znaleziono twierdzenie docelowe, check() używa prostej wstępnie okablowanej podprocedury, aby sprawdzić, czy

pozostało wystarczająco dużo czasu, aby ustawić zmienną switchbit (pierwotnie 0) na 1, zanim ciągle rosnący czas będzie równy t_1. Jeśli ta podprocedura zwróci wynik ujemny, check() kończy działanie. W przeciwnym razie ustawia switchbit := 1 (nie ma innego sposobu na zmianę switchbit). Następnie wielokrotnie testuje time, aż time > t_1, aby upewnić się, że warunek wzoru (6) został spełniony w t_1. Następnie przekazuje kontrolę do switchprog (nie ma innego sposobu wywołania switchprog). Switchprog może następnie przepisać wszystkie części s, wyłączając części zarezerwowane sprzętowo, takie jak time i x, ale włączając p.

Należy zauważyć, że technika dowodzenia niekoniecznie musi obliczać prawdziwe oczekiwane użyteczności przełączania i nieprzełączania — wystarczy, że określi, która jest wyższa. Na przykład, łatwo można udowodnić, że przyspieszenie podprocedury wyszukiwarki dowodów o współczynnik 2 z pewnością będzie warte pomijalnego (w porównaniu z czasem życia T) czasu potrzebnego do wykonania algorytmu zmiany podprocedury, niezależnie od dokładnej użyteczności przełączenia. System aksjomatyczny A jest parametrem definiującym daną maszynę Gödla. Oczywiste jest, że A musi być wystarczająco silny, aby umożliwić dowody twierdzeń docelowych. W szczególności aksjomaty teorii nieoznaczoności (punkt 1d) muszą być wystarczająco bogate. Nie jest to problem fundamentalny: po prostu wstawiamy wszystkie tradycyjne aksjomaty teorii prawdopodobieństwa

Twierdzenie o globalnej optymalności

Intuicyjnie, w dowolnym danym momencie p powinno wykonać pewien algorytm samomodyfikacji tylko wtedy, gdy jest to „najlepsza" ze wszystkich możliwych samomodyfikacji, biorąc pod uwagę funkcję użyteczności, która zazwyczaj zależy od dostępnych zasobów, takich jak rozmiar pamięci masowej i pozostały czas życia. Jednak na pierwszy rzut oka twierdzenie o celu (6) wydaje się implicite mówić tylko o jednym algorytmie modyfikacji, mianowicie switchprog(t_1) ustawionym przez systematyczny wyszukiwacz dowodów w czasie t_1. Czy ten typ lokalnego wyszukiwania nie jest zachłanny? Czy nie mógłby doprowadzić do lokalnego optimum zamiast globalnego? Nie, nie może, zgodnie z twierdzeniem o globalnej optymalności: Twierdzenie 1 (globalnie optymalne samozmiany, biorąc pod uwagę u i A zakodowane w p). Biorąc pod uwagę dowolną formalizowalną funkcję użyteczności u (punkt 1f) i zakładając spójność podstawowego formalnego systemu A, każda samozmiana p uzyskana poprzez wykonanie pewnego programu switchprog zidentyfikowanego poprzez dowód twierdzenia docelowego (6) jest globalnie optymalna w następującym sensie: użyteczność rozpoczęcia wykonywania bieżącego switchprog jest wyższa niż użyteczność czekania, aż wyszukiwarka dowodów wyprodukuje później alternatywny switchprog. Dowód. Twierdzenie docelowe (6) implicite mówi o wszystkich innych switchprog, które wyszukiwarka dowodów mogłaby wytworzyć w przyszłości. Aby to zobaczyć, rozważ dwie alternatywy decyzji binarnej: (1) albo wykonaj bieżący switchprog (ustaw switchbit = 1), albo (2) kontynuuj wyszukiwanie dowodów i switchprog (ustaw switchbit = 0), aż wyszukiwarka systematyczna znajdzie jeszcze lepszy switchprog. Oczywiście druga alternatywa dotyczy wszystkich (być może nieskończenie wielu) potencjalnych switchprog, które zostaną rozważone później. Oznacza to, że gdyby bieżący switchprog nie był „najlepszy", wówczas osoba przeszukująca dowody nie byłaby w stanie udowodnić, że ustawienie switchbit i wykonanie switchprog spowoduje wyższą oczekiwaną nagrodę niż odrzucenie switchprog, zakładając spójność A. Q.E.D.

Alternatywne twierdzenie o zrelaksowanym celu

Możemy zastąpić twierdzenie o celu (6) (punkt 6) następującym alternatywnym twierdzeniem o celu:

$$(u[s(\mathbf{t_1}) \oplus (switchbit(\mathbf{t_1}) = \text{`1'}), Env(\mathbf{t_1})] \geq$$
$$u[s(\mathbf{t_1}) \oplus (switchbit(\mathbf{t_1}) = \text{`0'}), Env(\mathbf{t_1})]).$$

Jedyną różnicą w stosunku do pierwotnego twierdzenia docelowego (6) jest to, że znak „>" stał się znakiem „≥". Oznacza to, że maszyna Gödla zmieni się sama, gdy tylko znajdzie dowód, że zmiana nie pogorszy sytuacji. Twierdzenie o globalnej optymalności podobne do twierdzenia 1 obowiązuje.

Optymalne wyszukiwanie dowodów z odchyleniem (BIOPS)

Tutaj konstruujemy p(1), które jest O()-optymalne w pewnym ograniczonym sensie, który zostanie opisany poniżej, ale nadal może zostać ulepszone, ponieważ nie jest koniecznie optymalne w sensie danego u (na przykład u z równania (1) nie wspomina ani nie dba o optymalność O()). Nasze optymalne wyszukiwanie dowodów z odchyleniem (BIOPS) jest zasadniczo zastosowaniem uniwersalnego wyszukiwania do wyszukiwania dowodów. Poprzednie praktyczne warianty i rozszerzenia uniwersalnego wyszukiwania zostały zastosowane [36, 38, 50, 45] do zadań wyszukiwania programów offline, w których dane wejściowe programu są ustalone tak, że ten sam program zawsze generuje te same wyniki. Jednak w naszym środowisku online BIOPS musi wziąć pod uwagę, że ta sama technika dowodzenia rozpoczęta w różnym czasie może dać różne dowody, ponieważ może odczytać części s (np. dane wejściowe), które zmieniają się w miarę upływu czasu życia maszyny. BIOPS zaczyna się od rozkładu prawdopodobieństwa P (początkowego odchylenia) na technikach dowodowych w, który można zapisać w L, np. P(w) = K−l(w) dla programów złożonych z K możliwych instrukcji [25]. BIOPS jest prawie optymalny pod względem odchylenia w tym sensie, że nie poświęci dużo więcej czasu na żadną technikę dowodową, niż na to zasługuje, zgodnie ze swoim odchyleniem probabilistycznym, mianowicie nie więcej niż jego prawdopodobieństwo razy całkowity czas wyszukiwania

Definicja 1 (Poszukiwacze optymalne pod względem odchylenia). Niech R będzie klasą problemów, C będzie przestrzenią wyszukiwania kandydatów na rozwiązania (gdzie każdy problem r ∈ R powinien mieć rozwiązanie w C), P(q | r) będzie odchyleniem zależnym od zadania w formie warunkowych rozkładów prawdopodobieństwa kandydatów q ∈ C. Załóżmy, że mamy również zdefiniowaną procedurę, która tworzy i testuje dowolne dane q na dowolnym r ∈ R w czasie t(q, r) (zwykle nieznanym z góry). Następnie wyszukiwarka jest n-bias-optymalna (n ≥ 1), jeśli dla dowolnego maksymalnego całkowitego czasu wyszukiwania $T_{total} > 0$ gwarantuje rozwiązanie dowolnego problemu r ∈ R, jeśli ma rozwiązanie p ∈ C spełniające $t(p, r) \leq P(p \mid r)\, T_{total}/n$. Jest ona n-bias-optymalna, jeśli n = 1.

Metoda 5.1 (BIOPS) W fazie (i = 1, 2, 3, . . .) Wykonaj: Dla wszystkich samoograniczających się technik dowodowych w ∈ L spełniających $P(w) \geq 2^{-i}$ Wykonaj:

1. Uruchom w do momentu zatrzymania lub wystąpienia błędu (takiego jak dzielenie przez zero) lub zużycia $2^i P(w)$ kroków.

2. Cofnij efekty w na s^p (nie kosztuje znacznie więcej czasu niż wykonanie w).

Technika dowodowa w może przerwać Metodę 5.1 tylko przez wywołanie instrukcji check() (punkt 6), która może przekazać kontrolę do switchprog (który prawdopodobnie nawet usunie lub przepisze Metodę 5.1). Ponieważ początkowe p działa na sformalizowanym sprzęcie, a ponieważ techniki dowodowe testowane przez p mogą odczytywać p i inne części s, mogą one generować dowody dotyczące (oczekiwanej) wydajności p i samego BIOPS. Metoda 5.1 ma co najmniej optymalny rząd złożoności obliczeniowej w następującym sensie. Twierdzenie 2. Jeśli niezależnie od zmiennych czasów, jakaś nieznana szybka technika dowodowa w wymagałaby co najwyżej f(k) kroków, aby wygenerować dowód o mierze trudności k (liczba całkowita zależna od natury rozwiązywanego

zadania), to Metoda 5.1 będzie potrzebowała co najwyżej O(f(k)) kroków. Dowód. Łatwo zauważyć, że Metoda 5.1 będzie potrzebowała co najwyżej O(f(k)/P(w)) = O(f(k)) kroków — stały czynnik 1/P(w) nie zależy od k. Q.E.D. Należy jednak zauważyć, że same dowody mogą dotyczyć zupełnie innych, dowolnych, formalizowalnych pojęć optymalności (silniejszych niż te wyrażalne w notacji O()) ucieleśnionych przez daną, specyficzną dla problemu, sformalizowaną funkcję użyteczności u. Może to wywołać użyteczne, wpływające na stałe przepisywanie początkowego narzędzia dowodowego, pomimo jego ograniczonego (choć popularnego i szeroko stosowanego) pojęcia O()-optymalności.

Jak przetrwały poszukiwacz dowodów może użyć Biops do rozwiązania pozostałych zadań wyszukiwania dowodów

Poniższe nie jest niezbędne. Załóżmy, że wykonanie switchprog odpowiadające pierwszemu znalezionemu twierdzeniu docelowemu nie przepisało kodu samego p - bieżące p jest nadal równe p(1) - i zresetowało switchbit i zwróciło kontrolę do p tak, aby mogło ono kontynuować od miejsca, w którym zostało przerwane. W takim przypadku podprogram Biops p(1) może użyć Optimal Ordered Problem Solver Oops , aby przyspieszyć wyszukiwanie n-tego twierdzenia docelowego (n > 1) przez ponowne użycie technik dowodowych dla wcześniej znalezionych twierdzeń docelowych, jeśli to możliwe. Podstawowe idee są następujące. Zawsze, gdy twierdzenie docelowe zostanie udowodnione, p(1) zamraża odpowiadającą mu technikę dowodową: jego kod staje się niezapisywalny przez techniki dowodowe, które zostaną przetestowane w późniejszych zadaniach wyszukiwania dowodów. Pozostaje jednak czytelny, tak że może być edytowany i/lub wywoływany jako podprogram przez techniki przyszłych dowodów. Pozwalamy również prefiksom technik dowodowych na tymczasowe przepisywanie rozkładu prawdopodobieństwa na ich sufiksach, w ten sposób zasadniczo przepisując procedurę wyszukiwania opartą na prawdopodobieństwie (przyrostowe rozszerzenie Metody 5.1) na podstawie poprzednich doświadczeń. Jako efekt uboczny przeprowadzamy metawyszukiwanie w celu znalezienia szybszych procedur wyszukiwania, co może znacznie przyspieszyć naukę nowych zadań [45]. Biorąc pod uwagę nowe zadanie wyszukiwania dowodów, Biops wykonuje Oops, poświęcając połowę całkowitego czasu wyszukiwania na wariant Metody 5.1, który przeszukuje tylko wśród samoograniczających się technik dowodowych, zaczynając od ostatnio zamrożonej techniki dowodowej. Pozostały czas jest poświęcany na nowe techniki dowodowe z dowolnymi prefiksami (które mogą jednak ponownie wykorzystywać wcześniej zamrożone techniki dowodowe). (Możemy również szukać uogólniającej techniki dowodowej rozwiązującej wszystkie zadania wyszukiwania dowodów do tej pory. W pierwszej połowie wyszukiwania nie musielibyśmy testować technik dowodowych na zadaniach innych niż najnowsze, ponieważ wiemy już, że ich prefiksy rozwiązują poprzednie zadania) Można wykazać, że Oops jest zasadniczo 8-bias-optimal (patrz Def. 1), biorąc pod uwagę 8-bias-optimal (patrz definicja 1), biorąc pod uwagę 8-bias-optimal lub 8-bias-optimal z powodu zamrożonych rozwiązań poprzednich zadań . Ten wynik natychmiast przenosi się do Biops. Podsumowując, Biops zasadniczo przydziela część całkowitego czasu wyszukiwania dla nowego zadania technikom dowodowym, które wykorzystują poprzednie udane techniki dowodowe w obliczalny sposób. Jeśli nowe zadanie można rozwiązać szybciej poprzez edycję kopii/przywołanie wcześniej zamrożonych technik dowodowych niż rozwiązując nowe zadanie wyszukiwania dowodów od podstaw, to Biops to odkryje i skorzysta z tego. Jeśli nie, to przynajmniej nie zostanie znacząco spowolnione przez poprzednie rozwiązania — Biops pozostanie 8-bias-optimal. Należy jednak pamiętać, że Biops nie jest jedynym możliwym sposobem zainicjowania programu do wyszukiwania dowodów maszyny Gödla.

Dyskusja i dodatkowe powiązania z poprzednimi pracami

Tutaj wymieniamy kilka przykładów możliwych typów samodoskonalenia, stosowalności maszyny Gödla do różnych zadań zdefiniowanych przez różne funkcje użytkowe i środowiska , sprzętu

probabilistycznego i dodatkowych powiązań z poprzednimi pracami . Krótko omawiamy również samoodniesienie i świadomość i podajemy listę odpowiedzi na często zadawane pytania.

Możliwe typy samodoskonalenia maszyny Gödel

Jakie udowodnione, przydatne samodoskonalenia są możliwe? Istnieje niewiele ograniczeń tego, co maszyna Gödel może zrobić:

1. W jednym z najprostszych przypadków może pozostawić swój podstawowy program do wyszukiwania dowodów nienaruszony i po prostu zmienić stosunek podziału czasu między podprogramem wyszukiwania dowodów a podzasadami e — tymi częściami p odpowiedzialnymi za interakcję ze środowiskiem.

2. Albo maszyna Gödla może modyfikować tylko e. Na przykład początkowe e może regularnie przechowywać ograniczone wspomnienia przeszłych zdarzeń gdzieś w s; może to pozwolić p wywnioskować, że przydatne byłoby zmodyfikowanie e tak, aby e przeprowadził pewne eksperymenty w celu zwiększenia wiedzy o środowisku i wykorzystał uzyskane informacje do zwiększenia pobierania nagród. W tym sensie maszyna Gödla ucieleśnia zasadniczy sposób radzenia sobie z problemem eksploracji kontra eksploatacja [19]. Należy zauważyć, że oczekiwana użyteczność przeprowadzenia pewnego eksperymentu może przewyższać tę nieprzeprowadzenia go, nawet jeśli późniejszy wynik eksperymentu sugeruje, aby nadal działać zgodnie z poprzednim e.

3. Maszyna Gödla może również modyfikować swoje aksjomaty, aby przyspieszyć działanie. Na przykład może znaleźć dowód, że oryginalne aksjomaty powinny zostać zastąpione lub rozszerzone o twierdzenia wyprowadzalne z oryginalnych aksjomatów.

4. Maszyna Gödla może nawet zmienić swoją własną funkcję użyteczności i twierdzenie docelowe, ale może to zrobić tylko wtedy, gdy ich nowe wartości są udowodnione jako lepsze według starych.

5. W wielu przypadkach nie oczekujemy, że maszyna Gödla zastąpi swój program do wyszukiwania dowodów kodem, który całkowicie porzuca wyszukiwanie dowodów. Zamiast tego oczekujemy, że tylko niektóre podprogramy programu do wyszukiwania dowodów zostaną przyspieszone — lub że być może tylko kolejność generowanych dowodów zostanie zmodyfikowana w sposób specyficzny dla problemu. Można to zrobić, modyfikując rozkład prawdopodobieństwa w technikach dowodowych początkowego programu do wyszukiwania dowodów optymalnych pod względem stronniczości

6. Ogólnie rzecz biorąc, użyteczność ograniczonych przeróbek może być często łatwiejsza do udowodnienia niż użyteczność całkowitych przeróbek. Na przykład załóżmy, że jest godzina 20:00, a stałym celem naszego agenta sterowanego maszyną Gödel jest maksymalizacja przyszłej oczekiwanej nagrody, przy użyciu (alternatywnego) twierdzenia o celu . Częścią tego jest unikanie głodu. W jego lodówce nie ma nic, a sklepy zamykają się o 20:30. Nie ma czasu, aby zoptymalizować swoją drogę do supermarketu w każdym najmniejszym szczególe, ale jeśli nie zacznie działać teraz, pozostanie głodny tej nocy (w zasadzie takie konsekwencje działań w niedalekiej przyszłości powinny być łatwo udowodnione, być może nawet w sposób związany z tym, w jaki sposób ludzie udowadniają sobie zalety potencjalnych działań). Innymi słowy, jeśli poprzednia polityka agenta nie obejmowała już, powiedzmy, automatycznej codziennej wieczornej wycieczki do supermarketu, polityka ta powinna zostać udowodniona, przynajmniej w ograniczonym zakresie i prosto teraz, póki jest jeszcze czas, tak aby agent na pewno dostał trochę jedzenia dziś wieczorem, bez wpływu na mniej pilne przyszłe zachowanie, które można zoptymalizować/zdecydować później, takie jak szczegóły trasy do jedzenia lub jutrzejszych działań.

7. W pewnych nieciekawych środowiskach nagroda jest maksymalizowana przez stawanie się głupim. Na przykład, dane zadanie może wymagać wielokrotnego i wiecznego wykonywania tej samej czynności aktywującej ośrodek przyjemności, tak szybko, jak to możliwe. W takich przypadkach maszyna Gödel może usunąć większość swojego bardziej czasochłonnego początkowego oprogramowania, w tym wyszukiwarkę dowodów.

8. Zauważ, że nie ma powodu, dla którego maszyna Gödel nie miałaby rozszerzyć swojego własnego sprzętu. Załóżmy, że jej żywotność wynosi 100 lat. Biorąc pod uwagę trudny problem i aksjomaty ograniczające możliwe zachowania środowiska, maszyna Gödel może znaleźć dowód, że jej oczekiwana skumulowana nagroda wzrośnie, jeśli zainwestuje 10 lat w budowę szybszego sprzętu obliczeniowego, wykorzystując zasoby fizyczne swojego środowiska.

Przykładowe zastosowania

Przykład 1 (Maksymalizacja oczekiwanej nagrody przy ograniczonych zasobach). Robot, który potrzebuje co najmniej 1 litra benzyny na godzinę, wchodzi w interakcję z częściowo nieznanym otoczeniem, próbując znaleźć ukryte, ograniczone złoża benzyny, aby od czasu do czasu zatankować swój zbiornik. Jest nagradzany proporcjonalnie do swojego czasu życia i umiera po maksymalnie 100 latach lub gdy tylko jego zbiornik się opróżni lub spadnie z klifu itp. Prawdopodobne reakcje środowiskowe są początkowo nieznane, ale zakłada się, że są próbkowane z aksjomatyzowanego Speed Prior , zgodnie z którym trudne do obliczenia reakcje środowiskowe są mało prawdopodobne. Pozwala to na obliczeniową strategię dokonywania niemal optymalnych przewidywań. Jednym z produktów ubocznych maksymalizacji oczekiwanej nagrody jest maksymalizacja oczekiwanego czasu życia. Mniej ogólne, bardziej tradycyjne przykłady, które nie obejmują znaczącej interakcji z prawdopodobnym otoczeniem, są również łatwo obsługiwane w ramach opartych na nagrodach:

Przykład 2 (Ograniczona czasowo optymalizacja NP-trudna). Początkowym wejściem do maszyny Gödla jest reprezentacja spójnego grafu z dużą liczbą węzłów połączonych krawędziami o różnej długości. W danym czasie T powinna ona znaleźć ścieżkę cykliczną łączącą wszystkie węzły. Jedyna nagroda o wartościach rzeczywistych pojawi się w czasie T. Jest ona równa 1 podzielonemu przez długość najlepszej dotychczas znalezionej ścieżki (0, jeśli żadna nie została znaleziona). Nie ma innych danych wejściowych. Produktem ubocznym maksymalizacji oczekiwanej nagrody jest znalezienie najkrótszej ścieżki możliwej do znalezienia w ograniczonym czasie, biorąc pod uwagę początkowe odchylenie.

Przykład 3 (Szybkie dowodzenie twierdzenia). Udowodnij lub obal tak szybko, jak to możliwe, że wszystkie liczby całkowite parzyste > 2 są sumą dwóch liczb pierwszych (przypuszczenie Goldbacha). Nagroda wynosi 1/t, gdzie t jest czasem wymaganym do wygenerowania i zweryfikowania pierwszego takiego dowodu.

Przykład 4 (Zoptymalizuj dowolny suboptymalny rozwiązywacz problemów). Biorąc pod uwagę dowolny formalizowalny problem, zaimplementuj suboptymalny, ale znany rozwiązywacz problemów jako oprogramowanie na sprzęcie maszyny Gödel i pozwól wyszukiwarce dowodów działać równolegle.

Sprzęt probabilistycznej maszyny Gödel

Powyżej skupiliśmy się na przykładowej maszynie deterministycznej. Łatwo rozszerzyć to na komputery, których działania są obliczane w sposób probabilistyczny, biorąc pod uwagę obecny stan. Następnie rachunek oczekiwań używany dla probabilistycznych aspektów środowiska musi zostać po prostu rozszerzony na sam sprzęt, a mechanizm weryfikacji dowodów musi uwzględniać, że nie istnieje coś takiego jak pewne twierdzenie — w najlepszym razie istnieją formalne stwierdzenia, które są prawdziwe z takim a takim prawdopodobieństwem. W rzeczywistości, to może być najbardziej

realistyczne podejście, ponieważ każdy fizyczny sprzęt jest podatny na błędy, co powinno być brane pod uwagę przez realistyczne probabilistyczne maszyny Gödel. Ustawienia probabilistyczne automatycznie unikają również pewnych problemów spójności aksjomatycznej. Na przykład przewidywania, których prawdopodobieństwo spełnienia jest mniejsze niż 1,0, niekoniecznie powodują sprzeczności, nawet jeśli nie pokrywają się z obserwacjami.

Więcej powiązań z poprzednimi pracami nad mniej ogólnymi samodoskonalącymi się maszynami

Pomimo (lub może z powodu) ambitności i potencjalnej mocy samodoskonalących się maszyn, poza naszymi laboratoriami w IDSIA i TU Munich, powstało niewiele prac w tym zakresie. Tutaj wymienimy zasadnicze różnice między maszyną Gödla a naszymi poprzednimi podejściami do „nauki uczenia się", „metanauki", samodoskonalenia, samooptymalizacji itd.

1. Maszyna Gödla kontra algorytm historii sukcesu i inne algorytmy Metalearner Modyfikowalne komponenty uczącego się nazywane są jego polityką. Algorytm, który modyfikuje politykę, jest algorytmem uczącym się. Jeśli algorytm uczący się ma modyfikowalne komponenty reprezentowane jako część polityki, mówimy o samomodyfikującej się polityce (SMP). SMP mogą modyfikować sposób, w jaki modyfikują siebie itd. Maszyna Gödla ma SMP. W poprzedniej pracy użyliśmy algorytmu historii sukcesu (SSA), aby zmusić niektóre (stochastyczne) SMP do wyzwalania coraz lepszych automodyfikacji . W czasie życia uczącego się SSA jest okazjonalnie wywoływane w momentach obliczonych zgodnie z samym SMP. SSA używa backtrackingu, aby cofnąć te wygenerowane przez SMP modyfikacje SMP, które nie zostały empirycznie zaobserwowane jako wyzwalające dożywotnie przyspieszenia nagród (mierzone do bieżącego wywołania SSA — ocenia to długoterminowe skutki modyfikacji SMP, przygotowując grunt pod późniejsze modyfikacje SMP). Modyfikacje SMP, które przetrwają SSA, reprezentują dożywotnią historię sukcesu. Do następnego wywołania SSA budują podstawę dla dodatkowych modyfikacji SMP. Wyłącznie przez samodzielne modyfikacje nasi uczący się na podstawie SMP/SSA rozwiązali złożone zadanie w częściowo obserwowalnym środowisku, którego przestrzeń stanów jest znacznie większa niż większość znalezionych w literaturze [48]. Jednak algorytm szkoleniowy maszyny Gödla jest teoretycznie bardziej wydajny niż SSA. SSA empirycznie mierzy użyteczność poprzednich samodzielnie modyfikacji i niekoniecznie zachęca do udowodnionych optymalnych. Podobne wady dotyczą wspomaganego przez człowieka, nieautonomicznego, samomodyfikującego się ucznia Lenata [22], naszego metagenetycznego programowania [32] rozszerzającego genetyczne programowanie Cramera [8, 1], naszych ekonomii metauczenia się [32] rozszerzających ekonomie uczenia maszynowego Hollanda [15] oraz metalearnerów opartych na gradiencie dla ciągłych przestrzeni programowych różniczkowalnych rekurencyjnych sieci neuronowych [34, 13]. Wszystkie te metody można jednak wykorzystać do zasiania p(1) początkową polityką.

2. Maszyna Gödla kontra Oops i Oops-rl

Optymalny uporządkowany rozwiązywacz problemów Oops (używany przez Biops) jest stronniczo optymalnym (patrz definicja 1) sposobem wyszukiwania programu, który rozwiązuje każdy problem w uporządkowanej sekwencji problemów o dość ogólnym typie, nieustannie organizując, zarządzając i ponownie wykorzystując wcześniej zdobytą wiedzę. Solomonoff niedawno zaproponował również powiązane pomysły dla asystenta naukowca, który modyfikuje rozkład prawdopodobieństwa uniwersalnego wyszukiwania [23] w oparciu o doświadczenie. Jak wskazano wcześniej (sekcja dotycząca ograniczeń Oops), jednak metody podobne do Oops nie są bezpośrednio stosowane do ogólnych zadań uczenia się przez całe życie (RL) , takich jak te, dla których zaprojektowano Aixi . Prosta i naturalna, ale ograniczona koncepcja optymalności Oops to biasoptimality (Def. 1): Oops to wyszukiwarka niemalże stronniczo-optymalna dla programów, które obliczają rozwiązania, które

można szybko zweryfikować (uwzględnia się koszty weryfikacji). Na przykład można szybko sprawdzić, czy jakiś aktualnie testowany program obliczył rozwiązanie problemu wież Hanoi użytego we wcześniejszym artykule : wystarczy sprawdzić, czy trzeci kołek jest pełen dysków. Ale ogólne zadania RL są trudniejsze. Tutaj, w zasadzie, ocena wartości pewnego zachowania pochłania całe życie uczącego się! Oznacza to, że naiwny test, czy program jest dobry, czy nie, pochłonąłby całe życie. Oznacza to, że moglibyśmy przetestować tylko jeden program; potem życie by się skończyło. Tak więc ogólne maszyny RL potrzebują bardziej ogólnej koncepcji optymalności i muszą robić rzeczy, których zwykłe Oops nie robi, takie jak przewidywanie przyszłych zadań i nagród. Możliwe jest użycie dwóch modułów Oops jako komponentów raczej ogólnego modułu uczenia się przez wzmacnianie (Oops-rl), jeden moduł uczy się predykcyjnego modelu środowiska, a drugi używa tego modelu świata do wyszukiwania sekwencji działań maksymalizujących oczekiwaną nagrodę. Pomimo właściwości optymalności stronniczości Oops dla pewnych uporządkowanych sekwencji zadań, Oops-rl nie jest jednak koniecznie najlepszym sposobem spędzania ograniczonego czasu obliczeniowego w ogólnych sytuacjach RL. Udowodniona optymalna maszyna RL musi w jakiś sposób udowodnić właściwości zachowań, których w przeciwnym razie nie można przetestować (takich jak: jaka jest oczekiwana nagroda za to zachowanie, której nie można naiwnie przetestować, ponieważ nie ma wystarczająco dużo czasu). To jest część tego, co robi maszyna Gödla: stara się znacznie skrócić czas testowania, zastępując naiwne, czasochłonne testy znacznie szybszymi dowodami przewidywalnych wyników testów, kiedykolwiek jest to możliwe. Sama weryfikacja dowodu może być przeprowadzona bardzo szybko. W szczególności weryfikacja poprawności znalezionego dowodu zazwyczaj nie zużywa pozostałego czasu życia. Stąd maszyna Gödel może używać Oops jako podmodułu wyszukiwania dowodów z optymalnym odchyleniem. Ponieważ dowody same w sobie mogą dotyczyć zupełnie innych, dowolnych pojęć optymalności (nie tylko optymalności z odchyleniem), maszyna Gödel jest bardziej ogólna niż zwykły Oops. Ale nie jest ona tylko rozszerzeniem Oops. Zamiast Oops może ona równie dobrze używać alternatywnych metod nieoptymalnych z odchyleniem do inicjowania swojego programu do wyszukiwania dowodów. Z drugiej strony, Oops nie jest tylko prekursorem maszyny Gödel. Jest to samodzielny, przyrostowy, optymalny z odchyleniem sposób przydzielania czasu wykonania programom, które ponownie wykorzystują wcześniej udane programy, i jest stosowalny do wielu tradycyjnych problemów, w tym, ale nie wyłącznie, wyszukiwania dowodów.

3. Maszyna Gödla kontra Aixi itd.

W przeciwieństwie do maszyn Gödla, model Aixi Huttera generalnie wymaga nieograniczonych zasobów obliczeniowych na aktualizację danych wejściowych. Łączy uniwersalny schemat predykcji Solomonoffa z obliczeniami expectimax. W dyskretnym cyklu k = 1, 2, 3, . . . działanie y(k) skutkuje percepcją x(k) i nagrodą r(k), obie pobierane z nieznanego (reaktywnego) rozkładu prawdopodobieństwa środowiskowego μ. Aixi definiuje rozkład mieszany ξ jako ważoną sumę rozkładów $\nu \in M$, gdzie M jest dowolną klasą rozkładów obejmującą prawdziwe środowisko μ. Na przykład M może być sumą wszystkich obliczalnych rozkładów, gdzie suma wag nie przekracza 1. W cyklu k + 1 Aixi wybiera jako następne działanie pierwsze w sekwencji działań maksymalizującej nagrodę przewidywaną przez ξ do pewnego danego horyzontu. Ostatnie prace wykazały optymalne wykorzystanie obserwacji przez Aixi w następujący sposób. Bayesowska polityka optymalna p^{ξ} oparta na mieszance ξ jest samooptymalizująca w tym sensie, że jej średnia wartość użyteczności zbiega się asymptotycznie dla wszystkich $\mu \in M$ do optymalnej wartości osiągniętej przez (niewykonalną) Bayesowską politykę optymalną p^{μ}, która zna μ z góry. Warunek konieczny, że M dopuszcza samooptymalizujące polityki, jest również wystarczający. Ponadto p^{ξ} jest Pareto-optymalne w tym sensie, że nie ma innej polityki dającej wyższą lub równą wartość we wszystkich środowiskach $\nu \in M$ i ściśle wyższą wartość w co najmniej jednym. Podczas gdy Aixi wyjaśnia pewne teoretyczne ograniczenia uczenia maszynowego, jest ono obliczeniowo niewykonalne, szczególnie gdy M obejmuje

wszystkie obliczalne rozkłady. Ta wada motywowała pracę nad ograniczonym czasowo, asymptotycznie optymalnym systemem Aixi(t,l) i powiązanym Hsearch, oba już omówione w sekcji 2.4, która również wymienia zalety maszyny Gödla. Obie metody mogłyby jednak zostać użyte do zasiania maszyny Gödla początkową polityką. To samoodniesieniowe aspekty maszyny Gödla zwalniają nas z dużej części ciężaru starannego projektowania algorytmów wymaganego dla Aixi(t,l) i Hsearch. Sprawiają, że maszyna Gödla jest zarówno koncepcyjnie prostsza, jak i bardziej ogólna niż Aixi(t,l) i Hsearch.

Czy ludzie są probabilistycznymi maszynami Gödla?

Nie wiemy. Uważamy, że lepiej, żeby tak było. Ich początkowy, podstawowy formalny system radzenia sobie z niepewnością wydaje się znacząco różnić od systemów tradycyjnego rachunku oczekiwań i logiki

Maszyny Gödla i świadomość

W ostatnich latach temat świadomości zyskał pewną wiarygodność jako poważny problem badawczy, przynajmniej w filozofii i neuronauce. Brakuje jednak technicznych uzasadnień świadomości: jak dotąd nikt nie wykazał, że świadomość jest naprawdę przydatna w rozwiązywaniu problemów, chociaż rozwiązywanie problemów jest uważane za centralne znaczenie w filozofii . Całkowicie samoodniesieniowa maszyna Gödla może być postrzegana jako dostarczająca właśnie takiego technicznego uzasadnienia. Jest „świadomy" lub „samoświadomy" w tym sensie, że całe jego zachowanie jest otwarte na samointrospekcję i modyfikowalne. Może „wyjść poza siebie", wykonując samozmiany, które są udowodnione jako dobre, gdzie sam poszukiwacz dowodów podlega analizie i zmianie poprzez testowane przez siebie techniki dowodowe. I ten typ całkowitego samoodniesienia jest właśnie powodem jego optymalności jako rozwiązywacza problemów w sensie Twierdzenia 1.

Często zadawane pytania

W ostatnim półroczu autor często odpowiadał na pytania dotyczące maszyny Gödla. Oto lista odpowiedzi na typowe pytania.

1. P: Czy dokładny biznes formalnego wyszukiwania dowodów ma sens w niepewnym świecie rzeczywistym?

O: Tak, ma. Musimy tylko wstawić do p(1) standardowe aksjomaty do reprezentowania niepewności i radzenia sobie z ustawieniami probabilistycznymi i oczekiwanymi nagrodami itp. Porównaj definicję użyteczności jako wartości oczekiwanej w równaniu (1).

2. P: Twierdzenie docelowe (6) wydaje się odnosić tylko do pierwszej samozmiany, która może całkowicie przepisać podprogram wyszukiwania dowodów — czy to nie sprawia, że dowód Twierdzenia 1 jest nieważny? Co zapobiega temu, aby późniejsze samozmiany były destrukcyjne?

O: To jest w pełni uwzględnione. Proszę spojrzeć jeszcze raz na dowód Twierdzenia 1 i zauważyć, że pierwsza autozmiana zostanie wykonana tylko wtedy, gdy będzie udowodniona jako użyteczna (w sensie obecnej funkcji untility u) dla wszystkich przyszłych autozmian (dla których obecna autozmiana przygotowuje grunt). To jest w zasadzie główny punkt całej konfiguracji maszyny Gödla.

3. P (związane z poprzednim punktem): Maszyna Gödla implementuje zachowanie metauczenia: co z meta-metą i poziomem meta-meta-meta?

O: Piękne jest to, że wszystkie meta-poziomy są automatycznie redukowane do jednego: każdy dowód twierdzenia docelowego automatycznie udowadnia, że odpowiadająca mu automodyfikacja jest dobra dla wszystkich dalszych automodyfikacji, na które wpływa obecna, w sposób rekurencyjny.

4. P: Oprogramowanie maszyny Gödla może generować tylko obliczalne mapowania z sekwencji wejściowych na sekwencje wyjściowe. Co jeśli środowisko jest nieobliczalne?

A: Wielu fizyków i innych naukowców faktycznie zakłada, że świat rzeczywisty wykorzystuje wszystkie liczby rzeczywiste, z których większość jest nieobliczalna. Niemniej jednak twierdzenia i dowody są po prostu skończonymi ciągami symboli, a wszystkie traktaty fizyki zawierają tylko obliczalne aksjomaty i twierdzenia, nawet jeśli niektóre z twierdzeń można interpretować jako stwierdzenia dotyczące nieprzeliczalnie wielu obiektów, takich jak wszystkie liczby rzeczywiste. (Należy jednak zauważyć, że twierdzenie L¨owenheima-Skolema implikuje, że każda teoria pierwszego rzędu z nieprzeliczalnym modelem, takim jak liczby rzeczywiste, ma również przeliczalny model). Ogólnie rzecz biorąc, formalne opisy obiektów nieobliczalnych wcale nie stanowią fundamentalnego problemu — mogą nadal umożliwiać znalezienie strategii, która maksymalizuje użyteczność w sposób udowodniony. Jeśli tak, maszyna Gödla może to wykorzystać. Jeśli nie, to ludzie nie będą mieli fundamentalnej przewagi nad maszynami Gödla. 5. P: Czy automatyczne dowodzenie twierdzeń nie jest bardzo trudne? Obecne systemy AI nie mogą udowodnić nietrywialnych twierdzeń bez ingerencji człowieka w kluczowych punktach decyzyjnych.

O: Coraz ważniejsze dowody matematyczne (twierdzenie o czterech kolorach itp.) w dużym stopniu zależą od automatycznego wyszukiwania dowodów. A tradycyjni dowodzący twierdzeń nie korzystają nawet z naszych nowych pojęć technik dowodowych i wyszukiwania dowodów O()-optymalnych. Oczywiście, niektóre dowody są rzeczywiście trudne do znalezienia, ale tutaj ludzie i maszyny Gödla stają w obliczu tych samych fundamentalnych ograniczeń.

6. P: Czy „twierdzenia bez darmowego lunchu" nie mówią, że nie można skonstruować uniwersalnych rozwiązywaczy problemów?

O: Nie, nie mówią. Odnoszą się do bardzo szczególnego przypadku problemów wylosowanych z rozkładów jednostajnych i.i.d. na skończonych przestrzeniach problemów.

7. P: Czy maszyna Gödel nie może przełączyć się na program switchprog, który przepisuje funkcję użyteczności na „fałszywą" funkcję użyteczności, która składa bezpodstawne obietnice dużych nagród w niedalekiej przyszłości?

O: Nie, nie może. Powinno być oczywiste, że przepisywanie funkcji użyteczności może mieć miejsce tylko wtedy, gdy maszyna Gödel najpierw udowodni, że przepisywanie jest przydatne zgodnie z obecną funkcją użyteczności.

Wnioski

Początkowe oprogramowanie p(1) naszej maszyny uruchamia początkowy rozwiązywacz problemów, np. jedno z podejść Huttera , które ma co najmniej optymalny rząd złożoności. Jednocześnie uruchamia O()-optymalny początkowy wyszukiwacz dowodów, używając internetowej wersji Universal Search, aby przetestować techniki dowodowe, czyli programy zdolne do obliczania dowodów dotyczących przyszłej wydajności systemu, w oparciu o aksjomatyczny system A zakodowany w p(1), opisujący formalną funkcję użyteczności u, sprzęt i samo p(1). Jeśli nie istnieje żaden udowodniony dobry, globalnie optymalny sposób przepisania p(1), to ludzie również go nie znajdą. Ale jeśli taki istnieje, to samo p(1) może go znaleźć i wykorzystać. To podejście Maszyny Gödla: w pełni samoodniesieniowe optymalne uniwersalne samodoskonalące się 223 daje pierwszą klasę teoretycznie poprawnych, w

pełni samoodniesieniowych, optymalnie wydajnych, ogólnych rozwiązywaczy problemów. Po teoretycznej dyskusji w rozdziałach od 1 do 5, pozostaje jedno praktyczne pytanie: aby zbudować szczególną, szczególnie praktyczną maszynę Gödla z małym początkowym stałym narzutem, które ogólnie użyteczne twierdzenia należy dodać jako aksjomaty do maszyny A (jako początkowe odchylenie), aby osoba przeprowadzająca początkowe wyszukiwanie nie musiała ich udowadniać od zera?