La incidencia filosófica de la programación lógica.[1]

L. M. Pereira[†]

1. Introducción

Empezaremos este trabajo tratando la incidencia de la inteligencia artificial (IA) sobre el campo de la filosofía de la lógica. Para ello examinaremos las exigencias que desde la IA se le plantean a la lógica en los ámbitos de la representación del conocimiento y del estudio de las formas de razonamiento. Después expondremos algunas de las contribuciones que ha aportado la IA desde el campo de la programación lógica (PL) a formas menos estáticas de lógica a fin de tratar con conocimiento dinámico, esto es, con información contradictoria e incompleta, hipótesis que se obtienen por abducción, argumentación, diagnóstico y depuración, actualización o aprendizaje. Durante este recorrido ahondaremos en las implicaciones que esto tiene para la filosofía del conocimiento.

La intención es atraer tanto a los lectores con inclinaciones filosóficas como a los veteranos de la PL a nuevos territorios recién establecidos en el área de la lógica, mostrando la investigación en curso en IA a través de la PL, ofreciendo una panorámica personal de este ámbito. Los apartados técnicos de la exposición aspiran a facilitar una comprensión intuitiva de las cuestiones mediante ejemplos sencillos que muestren cómo se tratan y se superan. Para presentaciones formales detalladas se remite a la abundante bibliografía.

La estructura de este trabajo es como sigue: en la primera sección se repasa y enfatiza el papel principal de la lógica como herramienta para el razonamiento y la representación del conocimiento humano. La sección siguiente es una extensión de la anterior, donde se resaltan las ventajas de mecanizar la lógica en un ordenador. Después, se relacionan las diferentes incursiones de la IA en la lógica y se selecciona, para un ulterior análisis, la última de ellas,

---

[1] Artículo publicado originariamente en inglés en la revista portuguesa *Intelectu*, nº 6, Dec. 2001. Editado posterioremente en D. Gabbay et al. (eds.), *Handbook of the Logic of Argument and Inference*. Studies in Logic and Practical Reasoning series, vol I, pp 425-448. Elsevier Science, 2002. Versión actualizada del artículo "The Logical Impingement of AI", referenciado en [41]. Permiso del autor y de la revista *Intelectu*.

[†] Centro de Inteligência Artificial (CENTRIA), Departamento de Informática. Universidade Nova de Lisboa, P-2829-516 Caparica, Portugal. Email: lmp@di.fct.unl.pt, URL:http://centria.di.fct.unl.pt/~lmp/.

atendiendo a su novedosa importancia para la filosofía. A continuación se resaltan algunas de las herramientas de la PL relacionadas con el marco propuesto. Seguidamente, se subraya el uso de tales herramientas para diagnosticar una teoría lógica. En la sección posterior se esboza un marco de lógica dinámica que se explicita mediante una breve descripción del proyecto MENTAL. Finalmente, se ofrecen las conclusiones y, para acabar, se propone un desafío a los filósofos.

2. Evolución y razonamiento

La evolución ha dotado a los seres humanos de pensamiento simbólico y de habilidades para comunicarse con lenguaje simbólico.

El conocimiento común objetivo necesita que el pensamiento siga reglas de argumentación y razonamiento independientes de contenidos concretos, reglas que no deben ser totalmente subjetivas, so pena de hacer imposible la comunicación precisa y el trabajo racional colectivo. Dichas reglas se han integrado en el pensamiento humano y suponen un elemento de enorme valor para la supervivencia.

En la evolución cognitiva del ser humano, tanto el conocimiento mimético (el que permite simular la realidad usando mapas o modelos), como el conocimiento imitativo (el que está presente en las representaciones rituales o en la reproducción de artefactos), representaron pasos esenciales en la socialización, tal como se puede ver, p. ej., en los planes de caza. En consecuencia, podemos decir que en todas las culturas humanas han surgido juegos sociales que siguen reglas abstractas.

Las reglas de juego encapsulan situaciones concretas definiendo patrones, estableciendo además secuencias causales situación-acción-situación que reflejan la causalidad a la que obedece la realidad física. De los juegos sobrevinieron abstracciones y, finalmente, emergieron los conceptos que definen situaciones, reglas generales del pensamiento y su encadenamiento, argumentos legítimos y contra-argumentos. Todos ellos reunidos componen un meta-juego cognitivo.

La idoneidad de la lógica informal para captar el conocimiento y para razonar, constituyéndose en una *lingua franca* verificable y accesible desde todos los lenguajes y culturas, descansa en su habilidad para fomentar la comprensión racional y la objetividad

común. La dinámica misma de la evolución del conocimiento objetivo, sea individual o colectivo, sigue leyes y patrones de raciocinio.

Además, y más recientemente, las mismas reglas de razonamiento pueden y de hecho están siendo usadas para razonar sobre el razonamiento. Aunque algunos de los métodos de razonamiento son bien conocidos, otros se usan de manera inconsciente, pero, como ocurre con las reglas de la gramática, pueden ser descubiertos a través de la investigación. Nuevos métodos de razonamiento pueden y han sido inventados y perfeccionados a través de la historia. Algunos ejemplos son la inducción transfinita, la *reductio ad absurdum* (prueba por contradicción), la recursión, la abducción y la eliminación de contradicciones, por citar sólo algunos.

Se pueden discutir estos nuevos métodos de razonamiento como cualesquiera otros. Pero el razonamiento, si es que queremos llegar a algún consenso, sólo puede ser cuestionado con otro razonamiento. Algunos argumentan que "discusión científica y filosófica" es sinónimo de argumentación informal *ad hoc* persuasiva, emotiva y culturalmente relativa, afín a algo como la retórica. Ignoran que la argumentación es precisamente una forma de razonamiento que se ha convertido a si misma en el objeto de la formalización lógica, y no son conscientes de que la retórica está bien para los predicadores, pero no favorece una comunicación de ida y vuelta necesaria para alcanzar el acuerdo común en la rigurosa praxis científica que conduce al conocimiento acumulativo.

3. La lógica y la computadora

La lógica, como hemos visto, ha surgido como la formulación de las leyes del pensamiento independientemente de su contenido. Los predicados expresan cualquier relación conceptual que deseemos, tanto sobre el mundo exterior como interior. Estos predicados pueden combinarse usando cuantificadores y conectivas lógicas para formar fórmulas, y pueden manipularse según reglas de inferencia que caracterizan a las diversas formas de razonamiento.

La lógica es capaz de articular simultáneamente una visión extensional del comportamiento de los predicados (concebidos como cajas negras o relaciones *input/output*), con una visión intensional de los mismos, (viéndolos como susceptibles de descomponerse

funcionalmente en otros predicados, como cajas negras escondidas dentro de otras cajas negras).

Dado que el lenguaje de la lógica es simbólico y sus reglas independientes del contenido, se puede especificar su funcionamiento usando procedimientos que siguen reglas generales, abstractas. Y esto es algo que se puede programar en un ordenador. A través de su mecanización, las teorías lógicas y las formas de razonamiento alcanzan, por primera vez, una existencia *in vitro*, la habilidad y la disposición para ejecutarse de forma repetitiva sin necesidad de contar con el soporte material de la mente humana.

Los dos puntos de vista de los que hablábamos arriba, el extensional y el intensional, se reconcilian en la computadora en tanto que la semántica declarativa (el "qué" se va a computar), es equivalente a la semántica computacional procedural (el "cómo" es computado). Ésta es la piedra angular del paradigma de la programación lógica, del que trataremos más adelante.

4. La filosofía y la IA

La filosofía ha sido cuna del pensamiento racional en la historia intelectual humana. Su estudio del lenguaje fomentó el refinamiento de la lógica como abstracción del lenguaje natural y como herramienta de investigación lingüística. Las investigaciones acerca de los fundamentos de las matemáticas también asignaron a la lógica un papel meta-instrumental que llevaría a usarla como la herramienta por excelencia de la manipulación simbólica de propósito general. La ciencia de la computación también ha adoptado a la lógica como su herramienta fundacional, al tiempo que la IA ha hecho posible su conversión en un lenguaje de programación.

Simultáneamente, la IA ha desarrollado la lógica más allá de los confines de la acumulación monotónica, típica de los dominios matemáticos precisos, completos, perdurables, condensados y cerrados, con el propósito de expandirla al ámbito no-monotónico del conocimiento impreciso, incompleto, contradictorio, discutible, revisable, distribuido y cambiante propio del mundo real. En resumen, la IA ha dinamizado lo que antes era estático.

En IA, así como en cualquier otra ciencia, la lógica tradicional juega un papel importante. De hecho, cualquier ciencia, implícita o explícitamente, usa la lógica y el

razonamiento para construir argumentos, contra-argumentos y para resolver disputas. Pero aún se puede esperar un papel mayor de la IA en la lógica.

Primero, la IA supone la mecanización de la lógica, resultando así una herramienta central para gran cantidad de actividades racionales.

Segundo, la IA pretende hacer explícito y bien definido el uso inconsciente de la lógica, y someterlo a pruebas objetivas automatizándola.

Tercero, la IA emplea la lógica como un lenguaje genérico de comunicación, de conocimiento y de procedimientos entre humanos y ordenadores y entre ordenador y ordenador.

Cuarto, en la IA, incluso cuando los procedimientos y mecanismos no son implementados usando la lógica de manera directa, ésta puede jugar el papel de un lenguaje de especificación preciso de requisitos, así como el de un formalismo con el cual estudiar sus propiedades semánticas.

Quinto, la IA ha contribuido significativamente a la formulación y examen del problema de identificar los límites del razonamiento simbólico en los ordenadores, así como a determinar si estos límites también son aplicables a los humanos.

Sexto y último, la IA ha ayudado a los investigadores a explorar nuevos problemas y métodos de razonamiento y a combinar modalidades dispares de razonamiento en un marco uniforme y unificado, para tratar información incompleta, imprecisa, contradictoria y cambiante.

Este último papel, en contraste con el que le antecede, más mimado por los filósofos, todavía no ha captado la atención de los mismos. La epigénesis de última hora de la lógica, con su andamiaje computacional, apunta a un papel innovador para la misma: a saber, el de soportar, justificar y mecanizar las dinámicas de los procedimientos epistémicos y argumentativos abstractos subyacentes a la práctica científica misma. Y esto, creo, son buenas noticias para los filósofos de la ciencia.

En consecuencia, éste es el papel en el que nos vamos a centrar, con el deseo de que sean los filósofos quienes ayuden a madurarlo para conseguir una epistemología computacional en toda regla.

5. Una herramienta lógica para la I.A.

La lógica clásica fue desarrollada para estudiar objetos matemáticos bien definidos, consistentes e inmutables. Por eso adquirió un carácter estático. La IA necesita trabajar con conocimiento dinámico, en condiciones no totalmente definidas, usando formas más dinámicas de lógica. Muchas de éstas han sido objeto central de investigación en programación lógica, un campo de la IA que usa la lógica directamente como lenguaje de programación [2], proporcionando métodos específicos de implementación y sistemas eficientes [3]. La programación lógica se utiliza, además, como vehículo de implementación habitual en otras aplicaciones de la lógica a la IA.

¿Cuáles son los problemas y requisitos de esta concepción dinámica de la lógica que la IA, a través de la programación lógica, ha contribuido a resolver?[4]

5.1 Programación Lógica

Hay una estrecha conexión entre la implicación lógica y la causalidad física. Después de todo, la lógica permite modelar los acontecimientos del mundo. Esto es así aún más en el caso de la programación lógica, donde todas las conclusiones verdaderas deben apoyarse, "estar causadas", por algunas premisas, y donde la implicación es unidireccional, es decir, no contrapositiva: las "causas" no son reversibles. Se usa la notación de cláusulas de Horn para expresar esta direccionalidad. La habilidad de la programación lógica para modelar acciones mediante actualizaciones se explicará en la sección dedicada a este tema.

Para producir un resultado o conclusión C se necesita una conjunción de condiciones positivas P suficientes, conjuntadas con comas (,), que la sustente, unida a la ausencia o negación de la conjunción de todas las condiciones negativas $\neg N$ que, dado P, la evitarían:

[2] Cf. [28] para los fundamentos y [7] para las recientes innovaciones.
[3] Para los más avanzados, incorporando desarrollos teoréticos más recientes, ver el sistema XSB en://www.cs.sunysb.edu/sbprolog/xsb-page.html/.
[4] Para un tratamiento técnico detallado de los siguientes problemas cf. nuestro libro [7],y referencias que en él aparecen, así como las referencias más recientes incluidas en este texto.

$$C \leftarrow P, \neg N$$

Si hay más que una forma de obtener *C*, entonces tendremos varias reglas de esta forma:

$$C \leftarrow P_i, \neg N_i$$

Si la información sobre cada regla, y sobre todas y cada una de las reglas para C, es completa, entonces tendremos:

$$C \leftrightarrow \bigvee_i P_i \neg N_i$$

Pero, ¿qué ocurriría si no fuera así? En primer lugar, puede que no tengamos suficiente información sobre cada $\neg N_i$. Quizás estemos en condiciones de saber todo lo requerido sobre cada P_i, sin alguno de los cuales no podría obtenerse *C*. Pero con respecto a $\neg N_i$, es decir, con respecto a todas las condiciones que, de estar presentes, nos impedirían concluir *C* aún conociendo cada P_i, es más difícil.

5.2 Mundos abiertos

Hay demasiadas cosas que pueden fallar en un mundo no-matemático abierto, algunas de las cuales ni siquiera las sospecharíamos. Podría estallar una bomba (a una distancia de seguridad) y destruir todo el sistema físico que estamos tratando de modelar lógicamente. ¿Deberíamos modelar tal posibilidad? En el mundo real, cualquier escenario es demasiado complejo como para definirlo exhaustivamente a cada momento. Tenemos que permitir la ocurrencia de situaciones imprevistas, basadas en información nueva. Así, en lugar de tener que asegurarnos de probar que alguna condición N_i no se da, podemos asumirlo con la condición de que, en caso contrario, estemos preparados para asumir la información que se seguiría. Es decir, podemos asumir una regla más general, pero debemos estar preparados para tratar con excepciones.

Tomemos, por ejemplo, esta porción de conocimiento, expresándola en el formato habitual de cláusulas de Horn:

$$fiel(H, K) \leftarrow casado(H, K), \neg amante(H, L)$$

Normalmente no disponemos de información explícita acerca de quién es o no el amante de quién, aunque este tipo de información puede sobrevenir inesperadamente,

especialmente si se trata de presidentes[5]. De esta manera, podemos codificar nuestro conocimiento sobre la presunción de que si un individuo *H* se casa con un individuo *K*, y no hay información ni bases para concluir *amante(H, L)*, entonces *H* es fiel a *K*:

$$fiel(H, K) \leftarrow casado(H, K), not\ amante(H, L)$$

Esta es la vía que utiliza la llamada negación por defecto *not P*, que se puede leer como "no hay una prueba para *P*". Esto es, no exigimos pruebas de que ¬*amante(H, L)* para cualquier *L*, dado un *H*, pero lo asumiremos así a menos que podamos establecer que *amante(H, L)* para algún *L*, dado *H*. En otras palabras, si no tengo evidencia para concluir *amante(H, L)* para algún *L*, dado *H*, puedo asumir que es falso para todo *L*, dado *H*.

5.3 Asunciones rebatibles

Los investigadores de la IA introdujeron la negación por defecto a través de la programación lógica, y se puede leer, indistintamente, como: "no hay una prueba para P", "la falsedad de P es asumible", "la falsedad de P es abducible", "no hay evidencia para P" o "no hay argumentos para P". La negación por defecto permite tratar con falta de información, situación común en el mundo real. Introduce la no-monotonía en la representación del conocimiento. Si posteriormente se obtiene información de que hay un par de amantes, debemos volver sobre la conclusión previa acerca de cualquier fidelidad que se hubiera supuesto. De hecho, puede que las conclusiones no sean sólidas aunque lo sean las reglas que las sostienen. Los textos legales, los reglamentos, y los tribunales emplean esta forma de negación de manera recurrente, pues tratan por fuerza con mundos abiertos, aunque no había sido formalizada en lógica hasta que la IA descubrió la necesidad de hacerlo.

5.4 La hipótesis del mundo cerrado

Obsérvese que *not* debería garantizar información tanto positiva como negativa. Esto es, deberíamos ser capaces de escribir:

$$\neg fiel(H, K) \leftarrow casado(H, K), not\ \neg amante(H, L)$$

[5] Cuando escribí este ejemplo, estaba en USA, y el presidente norteamericano estaba siendo insistentemente acosado debido a un supuesto *affair* amoroso.

para representar un mundo donde la gente es infiel por defecto o costumbre y donde se requiere probar explícitamente que alguien no tiene un amante para concluir que esa persona no es infiel. Esto ocurre a menudo en bases de datos y de conocimientos donde se adopta la hipótesis del mundo cerrado (HMC[6]). Esto es, la HMC se da cuando suponemos que la bases de datos o de conocimientos poseen toda la información positiva pertinente, de modo que sólo lo establecido de forma explícita, o lo que se deriva explícitamente de ello, es verdadero, tomando todo lo demás como falso. Pero, ¿cómo disponer de toda la información en un mundo cambiante?

La información normalmente se expresa de forma positiva atendiendo a la economía lingüística y mental. Así que, basándose en la HMC, se considera como la negación de la información positiva tanto la información ausente como la que no se puede obtener de forma explícita. Esto quiere decir que, cuando no hay información alguna sobre amantes, ¬*amante(H; L)* es verdad por la HMC y, en cambio, no lo es *amante(H, L)*. Esta asimetría es indeseable, dado que los nombres de los predicados son puramente convencionales y podemos, con razón, usar el predicado *not amante(H, L)* y su negación *not* ¬*amante(H, L)* para modelar nuestro conocimiento. Aunque en general se tomará el predicado afirmativo P si su extensión es mayor que su opuesto negativo, es probable que de antemano ignoremos cual tiene la extensión mayor. A un lenguaje lógico no deberían afectarle esta clase de particularidades de la representación del conocimiento.

5.5 Negación explícita

Además, aún cuando no dispongamos de información factual o derivable, ni negativa ni positiva, querremos ser capaces de decir que tanto una como otra pueden ser, según el conocimiento que dispongamos, epistemológicamente falsas. Por tanto no podemos aceptar el principio del tercio excluso, porque tanto un predicado como su negación pueden ser simultáneamente falsos.

La HMC, y los requisitos epistemológicos así como los de simetría, pueden reconciliarse leyendo el símbolo `¬´ no como la negación clásica, que supone el principio de tercio excluso según el cual todo predicado es verdadero o falso, sino como una nueva forma de negación, denominada en programación lógica “negación explícita” (que ignora el principio

[6] Nota de los traductores: en inglés CWA, *Closed Word Assumption*

del tercio excluso). Ahora podemos establecer la HMC para todos los predicados P o ¬*Q* que queramos simplemente escribiendo:

$$\neg P \leftarrow not\ P \text{ o } Q \leftarrow not\ \neg Q$$

Alternativamente, puede hacerse uso de un supuesto *not P* o de un supuesto *not* ¬*Q* en la medida en que lo requieran esas ocurrencias de predicado, como se ha visto anteriormente.

5.6 Revisión de supuestos

Pasemos a examinar la necesidad de revisar los supuestos y de introducir en nuestro marco un tercer valor de verdad llamado 'indefinido'.

Es inevitable que surja cierta confusión cuando combinamos los dos puntos de vista mencionados antes:

$$fiel(H, K) \leftarrow casado(H, K), not\ \neg amante(H, L)$$
$$\neg fiel(H, K) \leftarrow casado(H, K), not\ amante(H, L)$$

Asumiendo *casado(H, K)* para algún *H* y *K*, encontramos que *fiel(H, K)* y ¬*fiel(H, K)* son contradictoriamente verdaderos. Dado que no tenemos evidencia para *amante(H, L)* ni para ¬*amante(H, L)*, sencillamente no disponemos de información sobre ellos y asumimos su falsedad en los dos casos. Pero cuando un supuesto conduce a una contradicción, entonces hay que eliminarlo. Este es el venerable principio de reducción al absurdo, o "razonamiento por contradicción". En nuestro situación, los dos son supuestos que conducen a una contradicción. ¿Cuál habría que rechazar? ¡Están en pie de igualdad!

5.7 Indefinición

A falta de información que nos permita inclinarnos por alguno, vamos a rechazar los dos. Es decir, asumimos que ni *amante(H, L)*, ni ¬*amante(H, L)* son falsos. Y puesto que no se puede probar que ninguno de los dos sean verdaderos, consideramos a ambos indefinidos. Así, introducimos un tercer valor de verdad para caracterizar de manera más adecuada la falta de información sobre los amantes, haciendo de este modo que *fiel(H, K)* y ¬*fiel(H, K)* sean indefinidos. La obtención del estatus de indefinición puede obtenerse fácilmente añadiendo a nuestro conocimiento:

$$\neg amante(H, L) \leftarrow not\ amante(H, L)$$
$$amante(H, L) \leftarrow not\ \neg amante(H, L)$$

Sin más información no se puede probar ni que *amante(H, L)* ni que ¬ *amante(H, L)* sean verdaderos o falsos. Cualquier intento de hacerlo aboca a una circularidad auto-referencial propiciada por la negación por defecto. Sin embargo, si tomamos hipotéticamente por verdadero cualquiera de ellos, el otro se convierte automáticamente en falso y viceversa. Esto no descarta las dos posibles situaciones. Pero la opción más segura y escéptica es no tomar partido en la discusión marital y abstenerse de creer en alguna de ellas.

En efecto, la semántica bien fundada (WFS) de los programas lógicos asigna a los literales de las dos cláusulas de arriba el valor *indefinido* en su modelo bien fundamentado de conocimiento escéptico, aunque también permite establecer para los otros dos, modelos más crédulos y no minimales.

Pero, ¿por qué no podemos añadir simplemente:

$$amante(H, L) \leftarrow \neg amante(H, L)?$$

Porque no funcionará. Seguiríamos sin poder probar que *amante(H, L)* o ¬*amante(H, L)* son definitivamente verdaderos, y la contradicción persistiría.

Cuando se trata con aquello que no se puede probar, se necesita un tercer valor de verdad para expresar la incapacidad epistémica para aportar información. Incluso asumiendo que el mundo es ontológicamente bivalente, nuestro acceso a información codificada sobre él puede ser en general trivalente. Además, el mundo bien podría no ser bivalente o, en todo caso, no capturable en un esquema bivalente. ¿Podemos decir si una onda-partícula está o no aquí?, ¿está en verdad aquí o no-aquí aunque no podamos decirlo? Aparentemente, esta disputa no puede dirimirse experimentalmente.

En todo caso, hay otras razones a favor del uso de un tercer valor de verdad lógico. Como nuestro conocimiento del mudo real se construye sobre una base imperfecta, podría ser que estableciéramos, involuntaria e inconscientemente, dependencias circulares como la anteriormente aludida. Por ejemplo, un legislador bien puede promulgar leyes circulares, conflictivas. Sin embargo querremos seguir razonando, expresen o no legítimamente tales circularidades lo que se pretende modelar. Y cuando no lo hacen, querremos detectarlo y operar con ellas en vez arrojar todo por la borda.

Pero la indefinición también puede aprovecharse para expresar conocimiento común, sea desde el escepticismo o desde la credulidad. Aún ayer leí en un libro de proverbios del viejo marinero que decía: “Hay que capear las tormentas que no se pueden evitar y evitar las que no se pueden capear”, es decir:

$$capear \leftarrow tormenta, no\ evitar$$

$$evitar \leftarrow tormenta, no\ capear$$

¿Qué le sucederá a un experto marinero en una tormenta? ¿La capeará o la evitará? Pueden pasar tres cosas, incluyendo la indecisión sobre el resultado. En nuestro caso, se trata de capear el conocimiento circular, no de evitarlo en las aguas tranquilas de los paraísos matemáticos artificiales...

5.8 Expresividad

La introducción de esta semántica trivalente en la programación lógica es una importante innovación de la IA y comporta ciertas consecuencias. A saber, como hemos visto, nos permite cierto escepticismo y suponer no más que lo que está garantizado.

Otra innovación, recapitulando, es que la negación ¬ que usábamos antes, la negación explícita, es también trivalente, y en eso difiere de la negación clásica. La negación explícita, decíamos, no se adecua al principio de tercio excluso. Un predicado y su negación pueden ser simultáneamente falsos (“la silla está enfadada”, “la silla no está enfadada”). En una base de conocimientos es posible que no seamos capaces de probar algo ni su negación de forma definitiva, de modo que ambas deben considerarse falsas por defecto.

Además, la introducción de la negación explicita y, en particular, su combinación con la negación por defecto, aumenta nuestra capacidad expresiva para representar conocimiento. Pensemos por ejemplo en la orden

$$\neg cruzar \leftarrow tren$$

frente a la orden contraria

$$\neg cruzar \leftarrow not\ \neg tren$$

La última es claramente la opción más segura. De acuerdo con ésta, hay que probar que no viene ningún tren antes de que $\neg cruzar$ se convierta en falsa, mientras que con la primera opción $\neg cruzar$ es falsa siempre que no se pueda probar que se acerca un tren.

Otra innovación es que la flecha implicacional, combinada con la negación por defecto o con la negación explicita, no es la implicación material, ya que para ella no vale la ley de contraposición. De

$$A \leftarrow B$$

$$\neg A$$

no se sigue $\neg B$, dado que no puede presuponerse la contraposición $\neg B \leftarrow \neg A$. Siempre que se desee la contraposición del condicional, ha de declararse de forma explícita. Además, $\leftarrow$ debe verse como una regla de inferencia, que puede usarse proceduralmente de 'abajo a arriba' para concluir *A* dado *B*, o de 'arriba a abajo', como cualquier procedimiento invocado, para intentar probar el cuerpo *B* con el fin de probar la cabeza *A*. Los hechos son sólo cláusulas con el cuerpo vacío. Esta lectura inferencial de las cláusulas como reglas convierte a $\leftarrow$ en un operador direccional, propiedad que lo hace más adecuado, no sólo para modelar la inferencia, sino también la causalidad.

De hecho, las semánticas de la programación lógica enfatizan exactamente esto con su insistencia en aceptar sólo modelos minimales. Los modelos minimales son aquellos para los cuales todo literal positivo o explícitamente negado *verdadero* está soportado por alguna regla cuya cabeza o conclusión es un literal y cuyo cuerpo o conjunto de premisas está a su vez soportado. Los hechos, puesto que tienen cuerpo vacío, se soportan automáticamente.

Además, cualquier literal positiva o explícitamente negado es *falso* sólo en el caso de que todas las reglas para él tengan un cuerpo falso. En consecuencia, si no hay reglas para un literal, éste es inmediatamente falso.

Todos los otros literales positiva o explícitamente negados son indefinidos (esto ocurre sólo si tienen que ver con bucles auto-referenciales sin salida usando la negación por defecto). Finalmente, un literal negado por defecto, *not P*, es *verdadero/ falso/ indefinido* sólo en caso de que P sea, respectivamente, *falso/ verdadero/ indefinido*.

5.9 Diagnóstico de teorías

Hay muchos ejemplos del uso de la negación por defecto, de la negación explícita, de la resolución de contradicciones y de la trivalencia en el marco de la programación lógica; principalmente en las áreas de la abducción, la argumentación, la revisión de creencias, la actualización de conocimiento, el aprendizaje y el diagnóstico. Vamos a hablar del último de

ellos, esto es, del diagnóstico o depuración de una base de conocimiento genérica. Sus conexiones con las áreas previamente mencionadas nos permitirán acercarnos también un poco a ellas.

Supongamos que tenemos las siguientes reglas, que expresan un programa lógico que nos permite computar el predicado C basándonos en los predicados *P* y *N*:

$$C(X) \leftarrow P(X), \neg N(X)$$

$$P(a) \quad \neg N(a)$$

$$P(b)$$

$$P(c) \quad \neg N(c)$$

Ahora anticipemos la posibilidad de que la regla para C pueda ser falsa, i.e. que pueda tener excepciones, escribiéndola así:

$$C(X) \leftarrow P(X), \neg N(X), not\ excepcion(C(X))$$

Si no especificamos nada más sobre el predicado *excepción*, las conclusiones del programa lógico permanecerán invariables, de manera que el programa puede contener reglas como esa desde el principio para cualquier predicado que consideremos.

Si después descubrimos que *C(a)* es falsa, se puede salvar la regla para C, puesto que ya no vale para todo X=a, simplemente añadiendo a nuestro conocimiento:

$$excepcion(C(a))$$

La regla para C seguirá funcionando en todos los demás casos, aunque se pueden añadir más excepciones.

Este método se ocupa de las reglas que no son sólidas, i.e., que producen resultados erróneos.

¿Qué pasa con los otros casos problemáticos; aquellos afectados de incompletitud, esos en los que están ausentes resultados que podrían ser correctos? Supongamos que nos damos cuenta de que *C(b)* debiera ser verdadero, ¿qué regla se aplicará? Bien, es suficiente con introducir, para todo conjunto de reglas de un predicado, una regla que sirva para todo (digamos, todo terreno). En este caso:

$$C(X) \leftarrow ausente(C(X))$$

$$P(X) \leftarrow ausente(P(X))$$

$$\neg N(X) \leftarrow ausente(\neg N(X))$$

Si no especificamos nada más sobre el predicado *ausente*, las conclusiones del programa lógico siguen iguales, así que el programa puede contener esas reglas desde el principio. Ahora, para hacer a *C(b)* verdadero, basta con abducir o adoptar una de estas dos hipótesis: que *ausente(C(X))* o que *ausente(¬N(b))* es verdadera.

Este método se ocupa de las reglas que son incompletas, es decir, que fallan a la hora de proveer resultados.

Hemos logrado generalizar. Haciendo abducibles a los dos predicados, *excepcion(_)* y *ausente(_)*, podemos diagnosticar y depurar una base de conocimiento expresada en términos lógicos.

La abducción es un conocido proceso de razonamiento mediante el cual uno puede adoptar hipótesis que hacen verdaderas (o falsas) a ciertas instancias de un predicado, a fin de probar alguna conclusión. Consiste en razonar desde los objetivos hasta sus requisitos, y ha sido ampliamente estudiada en el contexto de la programación lógica. Desde luego, cuando conducen a contradicciones, las asunciones o hipótesis adoptadas de esta forma pueden revelarse falsas. Tenemos por tanto que prepararnos para la aplicación de un proceso de resolución de contradicciones basado en la revisión de supuestos, posiblemente adoptando a su vez otros supuestos. Esto también ha sido ampliamente estudiado en IA y en programación lógica; como resultado, los sistemas de razonamiento automatizados son ahora capaces de hacer ese trabajo por nosotros y han sido aplicados a dominios reales.

5. 10. Actualización.

Por último, pero no por ello menos importante, el trabajo en programación lógica se ha preocupado por la actualización de una base de conocimiento en términos de otra. Esta noción de actualización del conocimiento, al ser opuesta a la de la simple actualización de hechos, abre otra dimensión hacia el carácter dinámico de la lógica, en contraposición a la naturaleza estática de la lógica clásica. Dada una base de conocimiento, que contiene hechos y reglas, y dado también algún conocimiento nuevo con hechos y reglas, que posiblemente sea contradictorio con el conocimiento previo al que se añade, ¿cuál es la base de conocimiento actualizada resultante? ¿Puede ser repetido ese proceso?

La mayor parte del trabajo llevado a cabo en el campo de la programación lógica se ha centrado en la representación de conocimiento estático; es decir, conocimiento que no evoluciona con el tiempo. Éste es un serio inconveniente cuando se trata con bases de conocimiento dinámico, en las cuales no sólo la parte extensional (los hechos) cambia dinámicamente, sino también la parte intensional (las reglas). Trabajos recientes han mostrado cómo se puede conseguir esto en áreas de aplicación que incluyan dos bases de conocimento sobre partes de una legislación, sobre reglamentos, sobre procedimientos de seguridad o sobre reglas de comportamiento de un robot.

La introducción de actualizaciones de programas dinámicos extiende las actualizaciones a conjuntos ordenados de programas lógicos (o módulos). Cuando este orden es interpretado como orden temporal, la actualización dinámica de programas describe la evolución de un programa lógico que experimenta una secuencia de modificaciones. Esto abre la posibilidad de un diseño incremental y evolutivo de los programas lógicos, conduciendo al paradigma de la programación lógica dinámica (PLD). Este nuevo paradigma facilita significativamente la modularización de la programación lógica y, de esta manera, la modularización del razonamiento no-monótono como un todo.

Específicamente, supongamos que se da un conjunto de módulos de programas lógicos, y cada uno describe un estado diferente de nuestro conocimiento del mundo. Diferentes estados pueden representar diferentes puntos temporales o diferentes conjuntos de prioridades, o quizás incluso diferentes puntos de vista. El papel de la actualización dinámica de programas es el de emplear las relaciones mutuas existentes entre los diferentes módulos para determinar precisamente, en cualquier escenario de composición modular, las semánticas declarativas tanto como las procedurales del programa combinado resultante de los módulos. Sin embargo, no hay porque ver únicamente a las partes ordenadas de los módulos como una evolución temporal en el programa. Módulos diferentes también pueden representar diferentes conjuntos de prioridades, de puntos de vista o distintos agentes. En el caso de las prioridades, una actualización de un programa dinámico especifica el significado exacto de la “unión” de los módulos, sujeta a esas prioridades.

Con el propósito de representar información negativa, se emplean en la actualización programas lógicos generalizados que admiten la negación por defecto no sólo en los cuerpos de las reglas, sino también en sus cabezas. Esto es necesario, en particular, para especificar

que aquellos átomos que devinieran falsos fuesen borrados. Sin embargo, tales actualizaciones son mucho más expresivas que una mera inserción y eliminación de hechos. Las actualizaciones se pueden especificar por medio de reglas arbitrarias; por tanto, ellas mismas son programas lógicos. En consecuencia, se puede actualizar un programa lógico generalizado P (el programa inicial) por otro programa lógico generalizado U (el programa actualizador), obteniéndose como resultado un programa lógico $P \oplus U$ nuevo y actualizado. Como consecuencia, los módulos del programa pueden contener tanto información contradictoria como superpuesta.

La intuición ordinaria detrás de la actualización de un modelo del mundo se ha basado en la ley de la inercia, es decir, en el hecho de que las cosas no cambian a menos que se le obligue a ello. Supongamos, por ejemplo, que tenemos un modelo en el cual “soleado” es verdadero y “lluvia” es falso; si después recibimos la información de que el sol ya no brilla, concluiremos que “soleado” es falso por mor de la actualización, y que “lluvia” sigue siendo falsa por inercia.

Supongamos ahora, en cambio, que nuestra visión del mundo es descrita por un programa lógico que queremos actualizar. ¿Es suficiente con actualizar cada uno de sus modelos? ¿Está contenida toda la información soportada por un programa lógico en su conjunto de modelos? La respuesta a ambas preguntas es negativa. Un programa lógico codifica más que el conjunto de sus modelos individuales. Codifica las relaciones entre los elementos de un modelo, que se pierden si consideramos las actualizaciones simplemente ateniéndonos a un modelo base, como se propone por parte de los programas de revisión. De hecho, mientras que la semántica de la base de conocimiento resultante después de una actualización representa el significado pretendido cuando sólo la parte extensional de la base de conocimiento original (el conjunto de hechos) está siendo actualizada, se obtienen resultados fuertemente *anti-intuitivos* cuando también la parte intensional de la base de datos (el conjunto de reglas) sufre cambios, como muestra el siguiente ejemplo:

Consideremos el programa lógico P:

dormir ← *not tv_encendida*

tv_encendida ←

ver_tv ← *tv_encendida*

Claramente M=*{tv_encendida, ver_tv}* es su único modelo estable. Supongamos ahora que la actualización U declara que hay un corte de corriente, y que si hay un fallo de corriente entonces la televisión ya no funciona, como se representa con el siguiente programa lógico U:

not tv_encendida ← *corte_de_corriente*

corte_de_corriente ←

Con la actualización de modelos sencillos, tendríamos M_U=*{corte_de_corriente, ver_tv}* como la única actualización de M por U. Esto es así porque hay que añadir *corte_de_corriente* al modelo y su adición nos fuerza a hacer falsa *tv_encendida*. Pero, aún habiendo un corte de corriente, seguimos viendo la televisión.

Sin embargo, mediante la inspección del programa inicial y las reglas de actualización, es probable que concluyamos que, como *ver_tv* era verdadero únicamente debido a que *tv_encendida* era verdadero, el cambio de *tv_encendida* debería hacer a *ver_tv* falsa por defecto. Además, se esperaría que *dormir* también se convirtiera en verdadera. En consecuencia, el modelo pretendido de la actualización de P por U es el modelo $M_{U'}$=*{corte_de_corriente, dormir}*. Supongamos ahora que hay otra actualización U_2, descrita por el programa lógico:

not corte_de_corriente ←

que asegura que ya se ha restablecido la corriente. Ahora deberíamos esperar que la televisión se encendiese de nuevo. Ya que ha vuelto la corriente, esto es, que *corte_de_corriente* es falso, la regla *not tv_encendida* ← *corte_de_corriente* de *U* debería no tener efecto, y obtenerse así el valor de verdad de *tv_encendida* por inercia de la regla del programa original *P tv_encendida* ←.

Este ejemplo ilustra que, a la hora de actualizar bases de conocimiento, no es suficiente considerar únicamente los valores de verdad del contenido de los literales de las cabezas de sus reglas, porque los valores de verdad de los cuerpos de sus reglas pueden

también verse afectados por las actualizaciones de otros contenidos. En otras palabras, sugiere que debe ser aplicado el *principio de inercia* no sólo a los contenidos individuales en una interpretación, sino a la *totalidad de las reglas de la base de conocimiento*.

La primera ley de Newton, también conocida como la ley de la inercia, afirma que: *"todo cuerpo permanece en reposo o en movimiento rectilíneo con velocidad uniforme, a menos que sea forzado a cambiar ese estado por la acción de una fuerza desequilibrante sobre él"*. Se da la tendencia a interpretar esta ley en términos de sentido común: las cosas se mantienen como están a menos que se le aplique algún tipo de fuerza. Esto es verdadero pero no agota el significado de la ley. Son todas las fuerzas aplicadas las que determinan el resultado. Tomemos un cuerpo al cual se le aplican varias fuerzas, y que está en estado de equilibrio porque esas fuerzas se anulan entre sí. Si, posteriormente, se quitase una de esas fuerzas, el cuerpo empezaría a moverse.

El mismo tipo de comportamiento se presenta en la actualización de programas. Antes de obtener el valor de verdad, por inercia, de los elementos no afectados directamente por el programa de actualización, debemos verificar si la verdad de tales elementos no está indirectamente afectada por la actualización de otros elementos. Esto es, el cuerpo de una regla puede actuar como una fuerza que sostiene la verdad de su cabeza, pero esta fuerza puede dejar de serlo si el cuerpo se vuelve falso. De acuerdo con esto, la manera correcta de ver la actualización de programas, y en particular el papel de la inercia, es decir que son las reglas del programa inicial las que se mantienen en el programa actualizado debido a la inercia (en vez de a la verdad de los literales) y sólo en caso de que éstas no sean rechazadas por el programa de actualización. Esto debería ser así porque las reglas codifican más información que sus modelos.

Este enfoque fue adoptado primero en [34], cuando los autores presentaron la transformación de un programa en el cual, dado un programa inicial y un programa de actualización, se producía un programa actualizado obedeciendo la regla de la inercia. La postura de la aproximación de la programación lógica dinámica es precisamente la de asegurar que cualquier regla de actualización añadida está, de hecho, en vigor, y que las reglas del programa previo continúan en activo (por inercia) sólo en la medida de lo posible, es decir, se mantienen mientras no entren en conflicto con ninguna nueva adición.

Para representar información negativa en programas lógicos y sus actualizaciones, la PLD (programación lógica dinámica) permite la presencia de la negación por defecto en las cabezas de las reglas. Vale la pena señalar por qué es más adecuado en las actualizaciones generalizar el lenguaje para permitir la negación por defecto en las cabezas de las reglas que introducir la negación explícita en los programas (tanto en cabezas como en cuerpos). Supongamos que se nos da una regla que asegura que *A* es verdadero cuando se da alguna condición *Cond*. Esto se representa de manera natural por la regla *A*←*Cond*. Ahora supongamos que decimos, y es una actualización, que *A* ya no debe ser el caso (esto es, que debe ser eliminada o retractada), si se da una condición *Cond'*. ¿Cómo representar este nuevo conocimiento?

Mediante el uso de la programación lógica extendida (con la negación explícita) esto puede representarse por ¬*A*←*Cond'*. Pero esta regla dice más de lo que queremos. Afirma que *A* es falso dado *Cond'*, pero simplemente deseamos, en tal caso, que se Anule la verdad de *A*. Todo lo que se quiere decir es que si *Cond'* es verdadera entonces *not A* debe ser el caso, es decir, *not A*←*Cond'*. La diferencia entre la negación explícita y la negación por defecto es fundamental en tanto no esté accesible de forma completa la información sobre el átomo *A*, como hemos visto anteriormente. Bajo estas circunstancias, la primera formulación indica que hay evidencia para la falsedad de *A*, mientras que la última significa que no hay evidencia para la verdad de *A*. En el ejemplo de la eliminación, esta última opción es la deseable.

En otras palabras, una cabeza *not A* significa que se elimina *A* si el cuerpo se mantiene. Eliminar *A* significa que *A* ya no es verdadera, no necesariamente que es falsa. Cuando también se adopta la HMC, entonces esta eliminación conlleva la falsedad de *A*. En el proceso de actualización, la HMC debe estar codificada explícitamente desde el comienzo mediante la conversión de todos los literales *not A* en falsos en el programa inicial que está siendo actualizado. Esto es, los dos conceptos, eliminación e HMC, son ortogonales y hay que incorporarlos por separado. En los *Modelos Estables* y las semánticas *Bien Fundadas* de los programas únicos generalizados, la HMC se adopta *ab initio*, y la negación por defecto en las cabezas se combina con la ausencia de prueba porque no hay actualización y, de esa manera, no hay eliminación. Nótese que, en las actualizaciones, la cabeza *not* no se puede mover libremente al interior del cuerpo para obtener negaciones simples por dos razones:

primero, esta maniobra, aunque permitida en una semántica bivalente, no se admite en una trivalente; segundo y mas importante, hay información pragmática que no se puede eludir a la hora de especificar exactamente cual de los *not* figuran en la cabeza, digamos aquella que se elimina cuando el cuerpo es verdadero y, así, no es indiferente para la negación que otro literal de cuerpo positivo se traslade a la cabeza.

En [3], se ha propuesto un Lenguaje de actualizaciones (LACS)[7] expresamente diseñado para especificar secuencias de actualizaciones de programas lógicos. Aún cuando la principal motivación subyacente a la introducción de programación lógica dinámica (PLD)[8] era representar la evolución del conocimiento en el tiempo, la relación entre los diferentes estados puede codificar otros aspectos de un sistema de conocimiento, como se ha señalado más arriba. De hecho, desde su introducción, se han empleado tanto PLD como LACS para representar distintos aspectos; principalmente como una manera de:

- representar y razonar acerca de la evolución del conocimiento en el tiempo
- combinar reglas aprendidas por agentes
- razonar acerca de actualizaciones de creencias de agentes
- modelar interacción entre agentes
- modelar y razonar acerca de acciones
- resolver inconsistencias en el razonamiento metafórico
- combinar actualizaciones y preferencias

El aspecto común entre estas aplicaciones de la PLD es que los estados asociados con el conjunto dado de teorías sólo representan una de las varias dimensiones representacionales posibles (tiempo, jerarquías, dominios, …). Esto es así porque la PLD se define únicamente en términos de secuencias lineales de estados. Por ejemplo, se puede usar la PLD para modelar la relación de un grupo jerárquicamente relacionado de agentes, y también se puede usar para modelar la evolución de un único agente en el tiempo. Pero la PLD, tal y como es, no puede arreglárselas con ambas actualizaciones al unísono y modelar la evolución de un grupo tal de agentes en el tiempo.

---

[7] Nota de los traductores. En el original inglés LUPS *(Language of Updates)*
[8] Nota de los traductores. En el original inglés, DLP (Dynamic Logic Programming)

Un ejemplo de un escenario multidimensional tal puede encontrarse en el razonamiento legal, donde la legislación se atiene al principio *Lex Superior* (*Lex Superior Derogat Legi Inferiori*) de acuerdo con el cual la regla promulgada por una autoridad superior en la jerarquía anula una establecida por una autoridad inferior, mientras que la evolución de la ley en el tiempo se gobierna por el principio *Lex Posterior* (*Lex Posterior Derogat Legi Priori*), de acuerdo con el cual una regla promulgada en un momento posterior invalida a la anterior.

En efecto, la actualización del conocimiento no puede ser simplemente interpretada como situándose únicamente en la dimensión del tiempo. Varias dimensiones de actualización pueden combinarse simultáneamente, incluyendo la temporal o no, tales como la especificidad (p. ej. en las taxonomías), la fuerza de la instancia de actualización (p. ej. en el dominio legislativo), la posición jerárquica de la fuente de conocimiento (p. ej. en las organizaciones), la credibilidad de la fuente (p. ej. en el conocimiento incierto, extraído o aprendido), o la preferencia de la opinión (p. ej. en una sociedad de agentes). Para que sea posible esto, se ha extendido la PLD para permitir un estructura más general de estados.

La programación lógica dinámica multidimensional (PLDM[9]) generaliza la PLD para permitir colecciones de estados representados por dígrafos acíclicos arbitrarios, y no sólo secuencias de estados. La PLDM asigna semánticas a conjuntos y subconjuntos de programas lógicos dependiendo de cómo se relacionan unos con otros como se define por el dígrafo acíclico (DA)[10] que representa a los estados y su configuración. A través de tal generalización natural, la PLDM permite una mayor expresividad, aumentando de ese modo el conjunto de las aplicaciones de la programación lógica unificables bajo un único marco. La generalidad y flexibilidad proporcionada por un DA aseguran un mayor alcance y nuevas posibilidades. En virtud de las características recientemente añadidas de multiplicidad y composición, la PLDM proporciona un punto de vista “social” a la programación lógica, importante en estos tiempos de redes y agentes, para combinar el conocimiento en general.

La actualización inevitablemente hace emerger cuestiones sobre la revisión y la preferencia, y algún trabajo se ha perfilado sobre la articulación de estos aspectos distintos, aunque bastante complementarios. Y el aprendizaje se ve normalmente como cambios consecutivos aproximados, opuesto al cambio exacto; y combinar los resultados del

[9] Nota de los traductores. En el original inglés, MDLP *(Multi-Dimensional Dynamic Logic*

aprendizaje mediante múltiples agentes, múltiples estrategias, o múltiples conjuntos de datos, inevitablemente plantea problemas en el terreno de la actualización. Finalmente, la revisión de creencias dirigida por metas puede ser fructíferamente interpretada como actualización abductiva.

Así pues, no sólo se entremezclan de manera natural los susodichos temas (requiriéndose medios y herramientas precisas y formales para hacerlo), sino que su combinación da lugar a su vez a una compleja arquitectura que sirve de base para agentes racionales con programación lógica, que se pueden renovar entre sí y a agentes con pizarra actualizables, estructurados y comunes. Puede suponerse, en consecuencia, que el fomento de esta red de temas en la comunidad de la programación lógica es oportuno y fructífero. De hecho, se están indagando áreas de aplicación tales como el desarrollo de software, el aprendizaje multi-estratégico, la revisión abductiva de creencias, los diagnósticos basados en modelos, la arquitectura de agentes, y otros, desde esta perspectiva.

6. Un marco para la lógica dinámica.

No es demasiado difícil imaginar cómo puede usarse un proceso combinado de generación de reglas, de diagnóstico sistemático, y de revisión de reglas mediante actualización, para conseguir aprendizaje automático, de manera integrada, dentro del marco uniforme de la programación lógica.

Para iniciar el aprendizaje, se empieza con algún conocimiento ya adquirido, fijado, en forma de regla; esto es, con una teoría, y con un generador de reglas para añadirle nuevo conocimiento pretendido, con el objetivo de explicar observaciones abductivamente conocidas, ya sean positivas o negativas, en forma de hechos y hechos negados explícitamente.

La meta es generar reglas que definan un concepto positivo tanto como su negación, de manera que cubran todas las instancias de observación conocidas. Esta generación automática de nuevas reglas está sujeta a un sesgo predefinido: sólo se permiten en el proceso de generación algunos formatos de reglas y los predicados que las comprenden. Las reglas generadas pueden contradecirse entre ellas en algunas de las instancias de observación, y por

[10] Nota de los traductores. En el original inglés, DAG *(Acyclic Digraph)*

ello deben ser sometidas a diagnóstico para identificar posibles revisiones mínimas alternativas.

Para decidir qué revisión adoptar a continuación, concebimos nuevas observaciones posibles con respeto a la teoría disponible en curso, cuyos resultados, si se supieran, permitirían decidir entre las revisiones en conflicto. Se obtienen los resultados de esas observaciones denominadas cruciales, bien por el programa que los solicita, o por un proceso de revisión de creencias ejecutado posteriormente para llevar a cabo las acciones que conduzcan a los resultados de observación.

Una vez que se seleccionan las revisiones deseadas sobre las bases de los resultados, así como de los criterios de preferencia programados, las revisiones se validan por un procedimiento de actualización. Nótese que las reglas que han sido revisadas pueden estar ellas mismas sujetas a revisiones posteriores si fuese necesario.

De hecho, se repetirá la totalidad del proceso sobre las bases de nuevo conocimiento entrante, o confrontando el conocimiento con teorías desarrolladas automáticamente de diferente manera, con distintos bagajes, sesgos, generadores de reglas, diagnosticadores, revisores, preferencias, planificadores, observaciones y procedimientos de actualización, que caracterizan a un agente racional.

La confrontación de agentes epistémicos descansa en la argumentación y en la depuración mutua. Además del dominio legislativo y legal, la esfera de la discusión científica también depende de tales procedimientos, y puede beneficiarse de su automatización.

Abordar la argumentación conlleva un conjunto de herramientas similares a las del diagnóstico y la depuración. Los argumentos pueden atacar las hipótesis de otro argumento directamente, probando la negación de sus hipótesis, o indirectamente, contradiciendo una conclusión del otro argumento que descanse sobre sus supuestos. Sin embargo, tales ataques de un argumento a otro pueden, a su vez, ser contraatacados de la misma manera, y pueden en respuesta contra-contraatacar, etc. La programación lógica ha mostrado cómo se puede estudiar este proceso y como sacar las conclusiones de la confrontación de argumentos contradictorios, además de cómo se han de revisar para alcanzar un acuerdo.

El esbozo de un marco lógico dinámico está siendo llevado a cabo en el marco de nuestro proyecto de agentes MENTAL, en el Centro de Inteligencia Artificial de la Universidade Nova de Lisboa.

Tanto en la academia como en la industria, se tiene cada vez más la sensación de que los agentes artificiales serán una tecnología clave en tanto aumente la distribución, interconexión y apertura de los sistemas de computación. En tales entornos, la habilidad de los agentes para planificar autónomamente y perseguir sus acciones y metas, para cooperar, coordinar y negociar con otros, y para responder flexible e inteligentemente a situaciones dinámicas e impredecibles, conducirá a mejoras significativas en la calidad, así como a la sofisticación en los sistemas de software que puedan ser concebidos e implementados, además de en las áreas de aplicación y en los problemas que se puedan entonces abordar.

El propósito del proyecto MENTAL es el de establecer, sobre bases teóricas sólidas, el diseño de una arquitectura global para agentes mentales (es decir, incluyendo conocimiento, creencias e intenciones) basada y construida sobre la fortaleza de la programación lógica.

Los agentes comparten un medio ambiente común y cambiante, modelado él mismo en programación lógica. La totalidad del marco está siendo conceptualizado en programación lógica y verificado con programación lógica distribuida. El uso de agentes permite distribuir y compartir la información únicamente cuando surge la necesidad. De este modo, se puede reducir la complejidad de la manipulación de conocimiento. También se puede mejorar la eficiencia mediante la ejecución simultánea de agentes.

Un agente debe ser capaz de organizar su conocimiento, sus creencias, sus intenciones (metas), sus planes, al tiempo que recibe nueva información e instrucciones, y debe poder también reaccionar a las condiciones cambiantes del entorno. Debe ser también capaz de interactuar con otros agentes mediante el intercambio de conocimiento y creencias, además de reaccionar a las solicitudes de otros agentes. Dos agentes tales serán capaces de cooperar diagnosticando errores o ausencia de información en la base de información del otro, o de cooperar en el uso de recursos comunes, eludiendo interferencias mutuas no deseadas en sus planes, así como en la revisión mutua de creencias para alcanzar una meta común.

Cada agente se compone de subagentes especializados relacionados funcionalmente y posiblemente concurrentes, que ejecutan diferentes metas e instrucciones que les son asignadas por el agente. Ejemplos de subagentes tales son los que implementan las funcionalidades del razonamiento reactivo, la revisión de creencias, la argumentación, la explicación, el aprendizaje, la gestión del diálogo, la información concurrente, la evaluación

de preferencias, la estrategia y el diagnóstico. A los subagentes les coordina una capa de meta-nivel, que se asegura de la distribución interna de tareas y de la revisión de creencias, de la comunicación entre subagentes, de las decisiones finales y de toda la interacción con el exterior; esto es, tanto con el entorno como con otros agentes, incluyendo la comunicación, la gestión de interrupciones, las peticiones y las observaciones.

Aunque cada agente tiene una coordinación de meta-nivel, una colección de agentes interactuantes no tiene un soporte tal, y su comportamiento colectivo manifestará propiedades (emergentes) que son difíciles de prever.

Debido a que el conocimiento y las creencias son en general incompletas, contradictorias y propensas a errores, y más aún en conjuntos de multi-agentes, hacemos uso de semánticas, de procedimientos e implementaciones para tratar con la abducción, con la revisión de creencias, con la depuración, con la argumentación y con información contradictoria, incompleta, errónea, imperfecta, vaga y por defecto.

Se logra el refuerzo del aprendizaje usando técnicas de revisión de creencias para conseguir un efecto similar a la retropropagación. Un aprendizaje tal permite el refuerzo de de la evidencia asociada al conocimiento y a las creencias de un agente, como resultado de su contribución a conclusiones correctas o incorrectas. Además, los agentes pueden comparar y combinar sus grados de evidencia para discutir, para adquirir información o para alcanzar el consenso. El uso de algoritmos genéticos para desarrollar “genes” (o “memes”) de creencia es también una técnica que hemos explorado para el aprendizaje, y abre un nuevo campo emergente: el de vincular los algoritmos genéticos con la evolución de conocimiento en un agente basado en la lógica. Además, el intercambio de material genético permite a los agentes tener experiencias de fertilización cruzada.

El uso de la programación lógica para esta empresa se justifica sobre la base de que proporciona un riguroso fundamento teorético que, de modo unitario, abarca los temas mencionados, así como un vehículo de implementación para el procesamiento paralelo y distribuido. Adicionalmente, la programación lógica proporciona un instrumento formal altamente flexible para la especificación y la experimentación rigurosas con diseños computacionales, haciéndolos extremadamente útiles para el prototipado, incluso se prevén otros lenguajes de implementación específicos, posiblemente de menor nivel.

7. Observaciones finales.

¿Cómo se puede abordar los temas generales del razonamiento explorados aquí en un marco uniforme, sino en el contexto de la lógica? Sugiero, de hecho, que la lógica es el único espacio concebible como núcleo de esta empresa. ¿Y no deberíamos hacerlo? ¿No deberíamos mecanizar los procedimientos de razonamiento de manera que los robots y los ordenadores pudieran llevar a cabo, para nosotros o con nosotros, el a veces aburrido, a veces demasiado complejo trabajo relacionado con ellos? Seguramente, sólo perseverando podremos afrontar los cada vez más exigentes problemas de razonamiento que nos salen al paso.

Espero haberos convencido de que la IA, especialmente a través de la programación lógica, continuará cumpliendo un buen papel en la identificación, formalización e implementación de las leyes del pensamiento. Más notablemente, la IA se ha impuesto el reto de abrir el terreno de la lógica a la dinámica del conocimiento en cambio continuo. Y de paso, están consiguiendo satisfacer nuestras expectativas y requisitos. Continuar este trabajo es esencial si queremos encarar, si bien con la ayuda de los ordenadores, los retos de un conocimiento cada vez más acumulado y distribuido en un mundo cambiante.

Se ha atesorado una gran riqueza en la investigación en programación lógica en forma de resultados publicados y sistemas construidos a lo largo de los más de 25 años que lleva en marcha, aunque algunos de ellos no pudieron ser realizados totalmente en su momento debido a los costes tecnológicos y a las limitaciones de financiación, o simplemente debido a la falta de disponibilidad de equipamiento barato y eficiente, situación afortunadamente superada en nuestros días. Así que, aunque la teoría y los conocimientos implementacionales estaban ahí en su momento, su realización plena no era entonces viable.

Sin embargo, actualmente, los investigadores jóvenes no están lo suficientemente al tanto de esos resultados del pasado, algunos de principios de los años ochenta, y de su potencial, incluso cuando la tecnología ha evolucionado como para que ahora se puedan poner en funcionamiento. Paradójicamente ahora que la complejidad de los sistemas actuales hace a esas tecnologías teóricas aún más deseables. Por otro lado, el coste de la memoria y la aparición en escena de nuevas “tecnologías de implementación”, como la tabulación

(dirigida por la teoría)[11], hacen que la inversión anterior en teoría no sólo sea realizable ahora, sino también razonable y provechosa.

Por otra parte, no deberíamos ni detenernos ni abstenernos de investigar ahora para el futuro, porque las "tecnologías teóricas" de hoy que afrontan problemas reales, serán las proveedoras de las cosechas implementacionales y aplicacionales del mañana.

Por ejemplo, si no fuera por el impresionante desarrollo de las semánticas en programación lógica en los últimos 12 años y sus extensiones al razonamiento no-monotónico y a otras formas del mismo, no tendríamos hoy las impresionantes, por robustas y eficientes, implementaciones de sistemas de tales semánticas (ya sean las variedades estables y bien delimitadas o su combinación híbrida), las cuales han estado abriendo toda una gama de áreas de aplicación así como una serie de sofisticadas capacidades de razonamiento acerca de ellas. Esto fue posible, únicamente, por los sucesivos y prolongados esfuerzos en la generalización y la síntesis teorética, y por la integración combinada del desarrollo teórico y procedural, así como por la implementación práctica.

La lección, entonces, para las investigaciones que están por venir en esta todavía emergente tecnología de la información, es que debe ser continua, tanto para completar la inversión del pasado como para promover futuras ventajas.

El paradigma de la programación lógica proporciona un marco riguroso, abarcante, integrador, general y bien definido para el estudio sistemático de la computación, ya sea en cuanto a sintaxis, semántica, procedimientos, o en cuanto a implementaciones, entornos, herramientas y normas. La Programación Lógica aborda problemas y proporciona soluciones en un nivel suficiente de abstracción de manera que generaliza los dominios del problema. Esto se ve posibilitado por su fuerte fundamentación en la lógica, tanto en sustancia como en método, y eso constituye una de sus principales ventajas.

De hecho, las capacidades del razonamiento computacional tales como las asunciones por defecto, la abducción, la revisión de creencias, la eliminación de contradicciones, la actualización, el aprendizaje, las restricciones, etc., una vez desarrolladas en términos de generalidad y caracterización abstracta, pueden ser adoptadas e integradas en distintas áreas de aplicación.

---

[11] Nota de los traductores: en ingles, *theory-driven tabling.*

La programación lógica es, sin duda, el privilegiado crisol para la integración articulada de funcionalidades y técnicas, pertenecientes al diseño y la mecanización de sistemas complejos y dirigidos a las cada vez más exigentes y sofisticadas capacidades computacionales. Por ejemplo, consideremos de nuevo las características de los razonamiento mencionadas en secciones previas. Los agentes racionales futuros, si somos realistas, necesitarán combinarse para llevar a cabo sus tareas. Ningún otro paradigma computacional nos proporciona los medios para su integración conceptual coherente. Y, en todo caso, es el mismo vehículo que permite verificar su especificación, cuando no su completa implementación. ¿O no?

Además, la cuestión no es únicamente la de la integración conceptual, sino también la de la fertilización cruzada. ¿Cómo puede usarse el aprendizaje en la tarea de la revisión de creencias? ¿Cómo eliminamos las contradicciones en las actualizaciones? ¿Cómo pueden combinarse las preferencias aprendidas con las actualizaciones de las preferencias para guiar la revisión de creencias en un agente racional? ¿Cómo se pueden revisar las preferencias borrosas a la luz de las restricciones o supuestos abducidos en curso? ¿Cómo pueden combinarse los programas lógicos probabilísticos y multi-valuados?

La tarea de los filósofos en esta empresa puede consistir en participar en el apuntalamiento conceptual de las herramientas lógicas implementadas computacionalmente, y de emplear tales herramientas, mediante el aprendizaje automático, para la reconstrucción racional y funcional del argumento epistemológico historicista, así como de la pluralidad de la teoría científica de la evolución. La clasificación taxonómica cambiante de tipos naturales podría ser un buen punto de partida.

8. Agradecimientos.

Este trabajo ha sido financiado en su mayoría por el proyecto PRAXIS XXI MENTAL (2/2.1/TIT/1593/95) y por una beca de la NATO mientras el autor disfrutaba de una estancia sabática en el Departamento de Ciencias de la Computación de la Universidad de California, Riverside. Gracias a los coautores de los trabajos conjuntos que han sido mencionados.

Referencias.

[1] José Júlio Alferes, Carlos V. Damásio y Luís Moniz Pereira, "A logic programming system for non-monotonic reasoning". Journal of Automated Reasoning, 14:93-147, 1995.

[2] José Júlio Alferes, Joâo A. Leite, Luís Moniz Pereira, Halina Przymusinska y Teodor C. Przymusinski, "Dynamic logic programming", Int. Conf. on Knowledge Representation and Reasoning (KR'98), Trento, Italia. Stuart Shapiro (ed.), Morgan Kaufmann, 1998.

[3] José Júlio Alferes, Luís Moniz Pereira, H. Przymusinska y T.C. Przymusinski, "LUPS- a language for updating logic programs". En M. Gelfond, N. Leone y G. Pfeifer (eds.), Procs. de la 5ª Int. Conf. on Logic Programming and Nonmonotonic Reasoning, El Paso, Texas, USA, págs. 162-176, LNAI 1730, Springer, 1999. Una versión actualizada, de título: "LUPS - a language for updating logic programs", fue aceptada para su publicación por la revista AI.

[4] José Júlio Alferes, Joâo A. Leite, Luís Moniz Pereira, Halina Przymusinska y Teodor C. Przymusinski, "Dynamic Updates of Non-Monotonic Knowledge Bases", The Journal of Logic Programming 45(1-3): 43-70, September/October 2000.

[5] José Júlio Alferes, Joâo A. Leite, Luís Moniz Pereira, Paulo Quaresma, "Planning as Abductive Updating". En D. Kitchin (ed.), Procs. AISB´00 Symposium on AI Planing and Intelligent Agents, pags. 1-8, AISB, Birmingham, England, April 2000.

[6] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinka, Teodor C. Przymusinka y Paulo Quaresma, "Dynamic Knowedge Representation and its Applications". En Stefano Cerri y Danail Dochev (eds.), Procs. 9 Int. Conf. on Artificial Intelligence - Methodology, Systems, Applications (AIMSA'00), Varna, Bulgaria, LNAI 1904, Springer, pags. 1-10, 2000.

[7] José Júlio Alferes y Luís Moniz Pereira, "Reasoning with logic programming", Lecture Notes in Artificial Inteligence, vol. 1111, Springer, 1996.

[8] José Júlio Alferes y Luís Moniz Pereira, "Updates plus Preferences". En M. O. Aciego, I. P. de Guzman, G. Brewka y L, M, Pereira (eds.), Logic in AI, Procs. JELIA'00, Málaga, LNAI 1919, pags, 345-360. Springer, 2000.

[9] José Júlio Alferes y Luís Moniz Pereira, "Logic programming updating: a guide tour". En "Essays in honour of Robert Kowalski", Fariba Sadri and Antonis Kakas (eds.), Springer, forthcoming 2001.

[10] José Júlio Alferes, Luís Moniz Pereira, y Teodor C. Przymusinski, "`Classical' Negation in Nonmonotonic Reasoning and Logic Programming", Journal of Automated Reasoning, 200:107-142, 1998.

[11] José Júlio Alferes, Luís Moniz Pereira, y Terrance L. Swift, "Wellfounded Abduction via Tabled Dual Programs". En D. De Schreye (ed.), Procs. 16ª Int. Conf. on Logic Programming, Las Cruces, New Mexico, pags. 426-440, MIT Press, 1999.

[12] John Barth, "On with the story", pag 55, Back Bay Books, Little Brown and Company, 1996.

[13] Carlos V. Damásio, Wolfgang Nejdl, Luís Moniz Pereira y Michael Schroeder, "Model-bases Diagnosis Preferences and Strategies Representation with Logic Meta-Programming". En K. Apt and F. Turini (eds.), "Meta-Logics and Logic Programming", pp. 267-308, MIT Press, 1995.

[14] Carlos V. Damásio y Luís Moniz Pereira, "A Survey on Paraconsistent Semantics for Extended Logic Programs". En D. M. Gabbay and Ph. Smets (eds.), "Handbook of Defeasible Reasoning and Uncertainty Management Systems", vol. 2, pp 241-320, Kluwer Academic Publishers, 1998.

[15] Carlos V. Damásio y Luís Moniz Pereira, "Hybrid probabilistic Logic Programs as Residuated Logic Programs". En M. O. Aciego, I. P. de Guzmán, G. Brewka and L. M.

Pereira (eds.), Logic in AI, Procs. JELIA'00, Málaga, LNAI 1919, pags. 57-72, Springer 2000.

[16] Carlos V. Damásio y Luís Moniz Pereira, "Monotonic and Residuated Logic Programs". En Procs. 6ª European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01), S. Benferhat and Ph. Besnard, Springer, LNAI 2001.

[17] Carlos V. Damásio y Luís Moniz Pereira, "Antitonic Logic Programs". En Procs. 6ª Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR"01), T. Eiter and M. Truszczynski (eds), Springer LNAI 2001.

[18] Carlos V. Damásio, Luís Moniz Pereira y Terrance L. Swift, "Coherent Well-founded Annotated Logic Programs". En M. Gelfond, N. Leone and G. Pfeifer (eds.), Procs. 5ª Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), El Paso, Texas USA, pags. 262-276, LNAI-1730, Springer, 1999.

[19] Terrence Deacon, "The Symbolic Species", W. W. Norton, 1997.

[20] Pearangelo Dell'Acqua, Luís Moniz Pereira, "Updating Agents" En S. Rochefort, F. Sadri and F. Toni (eds.), Procs. ICLP'99 Workshop on Multi-Agent Systems in Logic (MASL'99), Las Cruces, New Mexico, 1999.

[21] Pierangelo Dell'Acqua, Luís Moniz Pereira, "Preferring and Updating in Abductive Multi-Agent Systems". En Procs. Workshop on "Engineering Societies in the Agent's World 2001" (ESAW'01), Andrea Omicini et al. (eds.), Prague.

[22] Merlin Donald, "Origins of the Human Mind", Harvard University Press, 1991.

[23] Meter Froehlich, Carlos V. Damásio, Wolfgang Nejdl, Luís Moniz Pereira, Michael Schroeder, “Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks. En Procs. of the 12th International Conference on Industrial & Engineering

Applications of Artificial Intelligence & Expert Systems IEA/AIE-99. Cairo, Egypt. LNAI, Springer 1999.

[24] Joseph Gartner, Terrance L. Swift, Allen Tien, Carlos V. Damásio, Luís Moniz Pereira, "Psychiatric Diagnosis from the Viewpoint of Computational Logic". En J. Lloyd et al. (eds.), Procs. of First Int. Conf. on Computational Logic (CL 2000), London, UK, pages 1362-1376, LNAI 1861, Springer, July 2000.

[25] Paul R. Gross and Norman Levitt, "Higher Superstition", The Johns Hopkins University Press 1994.

[26] John Holland, "Emergence", Addison-Wesley, 1998.

[27] Anthony C. Kakas, Robert A. Kowalski and Francesca Toni, "Abductive Logic Programming", Journal of Logic and Computation, 2:19-770, 1993.

[28] Robert A. Kowalski, "Logic for Problem Solving", North-Holland, 1979.

[29] Imre Lakatos, "Proofs and Refutations: the Logic of Mathematical Discovery", Worrall, J. and Zahar, E. G. (eds), Cambridge University Press, 1976.

[30] E. Lamma, F. Riguzzi, L. M. Pereira, "Belief Revision by Lamarckian Evolution". En Procs. 1st European Workshop on Evolutionary Learning (EvoLEARN2001), E. J. W. Boer et al (Eds.) "Applications of Evolutionary Computing", Springer-Verlag, LNCS 1037, pp. 404-416, 2001.

[31] Evelina Lamma, Luís Moniz Pereira and Fabricio Iguzzi, "Logic aided Lamarckian Evolution". En P. Brazil and R. Michalski (eds.), Procs. Multi-Strategy Learning Workshop (MSL'00), Guimaraes, Portugal, publicado por LIAAC – Porto University, June 2000.

[32] Evelina Lamma, Fabricio Riguzzi and Luís Moniz Pereira, "Strategies in Combined

Learning via Logic Programs". Machine Learning 28 (1/2): 3-87, January 2000.

[33] Joao A. Leite, José Júlio Alferes, Luís Moniz Pereira, "Minerva-A Dynamic Logic Programming Agent Architecture". En Proc. Agent Theories, Architectures, and Languages (ATAL'01), John-Jules Meyer (ed.), Springer LNAI, 2001.

[34] Joao A. Leite and Luís Moniz Pereira, "Generalizing updates: from models to programs". En ILPS'97 Workshop on Logic Programming and Knowledge Representation, LPKR'97, Port Jefferson 1997.

[35] Joao A. Leite, José Júlio Alferes and Luís Moniz Pereira, "Multi-dimensional Dynamic Logic Programming". In F. Sauri and K. Satoh (eds.), Procs. CL-2000 Workshop on Computational Logia in Multi-Agent Systems (CLIMA'00), London, England, July 2000. Una version actualizada y extendida apareció como: "Multi-dimensional Dynamic Knowledge Representation", Procs. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'01), T. Ester and M. Truszczynski (eds), Springer LNAI 2001.

[36] Joao A. Leite, Francisco C. Pereira, Amílcar Cardoso and Luís Moniz Pereira, "Metaphorical Mapping consistency via Dynamic Logic Programming". En G. Wiggins (ed.), Procs. AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science, pags 41-50, AISB, Birmingham, England, April 2000.

[37] Luís Monteiro and Luís Moniz Pereira, "Aspectos Cibernéticos da Epistemología". En "Novas Perspectivas das Ciencias do Homem", pp. 13-27, Biblioteca de Ciencias Humanas 4, Editorial Presença, Lisboa, 1970.

[38] Thomas Nagel, "The Last Word", Oxford University Press, 1997.

[39] Isaaco Newtono, "Philosophiae Naturalis Principia Mathemathica". Editio tertia & aucta emendate. Apud Guil & Joh. Innys, Regiae Societatis typographos, 1762. Cita original: *"Corpus omne perseverare in statu suo quiescendi vel movi uniformiter in directum, nisi*

quatenus illud a viribus impressis cogitur statum suum mutare."

[40] Luís Moniz Pereira, "Inteligência Artificial: Mito e ciência". In Revista Coloquio-Ciências", 3:1-13, October 1988, Fundaçao Calouste Gubenkian, Lisboa.

[41] Luís Moniz Pereira, "The Logical Impingement of Artificial Intelligence", Grazer Philosophische Studien (Internationale Zeitschrift Für Analytishe Philosophie), Amsterdam/Atlanta, vol. 56, pp. 183-204, 1999.

[42] Steven Pinker "How the mind works", W. W. Norton, 1997.

[43] Paulo Quaresma and Luís Moniz Pereira, "Modeling Agent Interaction in Logic Programming", en O. Barenstein (ed.), Procs. of the 11th Int. Conf. on Applications of Prolog (INAP'98), Tokyo, Japan, September 1998.

[44] Paulo Quaresma and Irene P. Rodrigues, "A collaborative legal information retrieval system using dynamic Logic Programming". En Procs. 7th Int. Conf. on AI and Law (ICAIL-99), pages 190-191. ACM Press 1999.

Traducción de Efrén Poveda García y Carlos Sobral Areán. Supervisión de Alejandro Sobrino Cerdeiriña.