Les sentències judicials sobre el sistema algorítmic administratiu BOSCO: més enllà que el codi font

Joost J. Joosten

Universitat de Barcelona

Gener, 2025

1 Reflexions d'un no-legalista sobre el cas Bosco

En aquest primer apartat comparteixo algunes reflexions meves sobre la pregunta si és desitjable o no revelar el codi font d'un programari informàtic usat dins de l'administració pública. Les meves reflexions vindràn des de la meva disciplina: la lògica en general i la lògica matemàtica en particular. Abans d'entrar en més detalls de la visió d'un lògic és convenient adinsar-nos una mica en la lògica formal.

1.1 Lògica formal

Una part capdal de la lògica matemàtica tracta amb llenguatges formals i la seva expresivitat. Els llenguatges que s'estudien en la lògica són llenguatges formals i no pas llenguatges naturals. Un llenguatge natural fa referència al món real exterior. Per exemple, la frase "Està plovent" fa referència a un fet metereològic al món exterior. Per avaluar si frase és veritat o no, hem de entendre que *ara i aquí* hem d'avaluar si efectivament està plovent.

Els llenguagtes formals s'asemblen molt a les matemàtiques: hi ha un conjunt de regles ben estrictes que diuen quina frase és correcta i quina no. Per exemple, dins del llenguatge de la aritmètica la frase $2 > 3$ és sintacticament correcte (tot i que el seu significat estandard és falç) mentres que la frase $23 >$ no ho és. El llenguatge formal fa referència a un món formal com ara nombres, vectors, etc.

Es poden fer servir llenguatges formals per descriure part del món real mitjançant ontologies formals i la seva traducció.

Considerem un exemple[1]. Suposem que el llenguatge formal és el llenguatge simple de la lògica proposicional. Per tant, el nostre llenguatge parla de proposi-

[1]En aquest exemple i alguns exemples més endavant, ens basem en gran part sobre *Le nozze di Giustizia, Interactions between Artificial Intelligence, Law, Logic, Language and Computation with some case studies in Traffic Regulations and Health Care.* J. J. Joosten and M. Montoya García. Preprint.

cions $P, Q, R, S, \ldots$ que es poden combinar mitjançant connectius com "i" (escrivim $\wedge$), "si aleshores" (escrivim $\rightarrow$), etc. Les proposicions es poden veure com a contenidors d'informació. La seva semàntica formal diu que el valor d'una proposició pot ser *vertader* o *fals*. Sovint, 1 s'escriu per cert i 0 per fals en la semàntica formal.

Ara considereu una estipulació legal que diu

> Si conduïu en una zona industrial i plou, la vostra velocitat màxima permesa és de 35 quilòmetres per hora. (†)

Podem triar la nostra ontologia formal com

$$\begin{aligned} P &:= \text{Conduïu en una zona industrial} \\ Q &:= \text{Plou} \\ R &:= \text{La velocitat de conducció no és superior a 35 km/h} \end{aligned}$$

Sota aquesta traducció podem expressar l'estipulació legal com

$$(P \wedge Q) \rightarrow R. \tag{1}$$

Clarament, ens agradaria acceptar que (1) és cert si i només si el comportament de conducció respecte a aquesta clàusula (†) és legal.

Podem imaginar que P (conduir en una zona industrial) es pot relacionar amb l'observable d'una ubicació GPS. De la mateixa manera, R es pot relacionar amb els valors proporcionats per un dispositiu de velocitat. No obstant això, Q (està plovent) sembla essencialment subjecte a un judici: una humitat elevada amb partícules d'aigua en suspens a l'aire o remolins molt fluix i lent comptaria com a pluja? En la semàntica formal, no hi ha cap problema d'ambigüitat. El valor pot ser 1 o 0 i quin dels dos serà proporcionat per les dades que alimentem al raonament formal. Serem nosaltres que hem de decidir si plou o no plou: en la lògica formal es treballa amb les dades ja interpretades.

Però independentment d'aquest vincle de les variables proposicionals P, Q i R amb el món real, en el vessant formal podem veure amb total rigor matemàtic que tot el següent[2] són vàlides:

$$\begin{aligned} R &\rightarrow ((P \wedge Q) \rightarrow R), \\ \neg Q &\rightarrow ((P \wedge Q) \rightarrow R), \\ \neg P &\rightarrow ((P \wedge Q) \rightarrow R), \\ (\neg P \vee \neg Q) &\rightarrow ((P \wedge Q) \rightarrow R), \\ &\text{etc.} \end{aligned}$$

Aquest exemple hauria de mostrar i aclarir que la precisió matemàtica només es pot obtenir per a la relació entre llenguatge formal i semàntica formal. Il·lustrem aquesta discussió i observació a la figura 1 a continuació.

[2] Animem al lector a traduir-los anomenades *tautologies* un altre cop a les estipulacions legals en llenguatge natural.

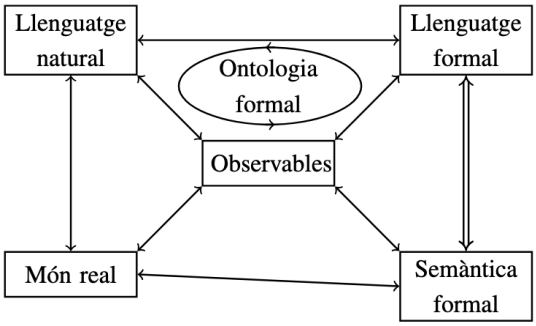

Figure 1: Les ontologies formals s'han de triar amb cura. El rigor matemàtic només es pot obtenir per a la relació entre expressions en un *Llenguatge formal* respecte a la *Semàntica formal* d'aquest llenguatge formal. Aquesta relació és l'única representada amb una doble fletxa. Totes les altres relacions tenen sentit sobre la base de la confiança i el sentit comú i els mètodes formals en la seva totalitat són impossibles.

1.2 Llenguatge formal versus natural

El llenguatge natural serveix bé en uns contextos i el llenguatge formal es presta millor per altres contextos. Entre els que treballem en lògica formal i programació és ben conegut que el llenguatge natural és molt dolent per a descriure algorismes o per a descriure matemàtiques: no ens serveix pas. Atributs típics del llenguatge natural és que sovint és difusa, requereix una interpretació per obtenir precisió i que sovint el llenguatge natural és ambigu.

Per tant, si una llei pretén estipular un algorisme, per als matemàtics i lògics ens és obvi que el llenguatge natural dona molts problemes. A sota veurem un parell d'exemples. Per ara, només voldria mencionar que des d'una perspectiva tècnica fer servir el llenguatge natural per a descriure com ha de funcionar un algorisme és demanar problemes. I en realitat hi ha moltes lleis on la llei fa exactament això: descriure com les dades d'un comportament (a sota: velocitat, temps de conducció, et cetera) s'han de transformar a dir que el comportament ha sigut legal o no.

No proposem que per a lleis computacionals[3] s'ha de fer servir un llenguatge formal. Una llei escrita en llenguatge formal seria incomprensible per a la gran part dels ciutadans. Una solució fàcil seria escriure una llei computacional amb una combinació de llenguatge natural i formal: el llenguatge natural per a poder entendre i el formal per a poder implementar sense ambigüitats ni errors. Observem que llenguatges de programació també són llenguatges formals. Per tant,

[3]S'anomena una llei *computacional* quan la llei està pensada per ser implementada mitjançant un programari.

des d'un punt de la lògica observem que revelar el codi font d'una implementació d'una llei computacional seria el més natural.

1.3 Revelar el codi font

Com hem dit abans, des de la perspectiva de la programació, la lògica i de les matemàtiques és imprescindible revelar el codi font d'una especificació o implementació d'una llei computacional. No volem minimitzar la complexitat legal que podria comportar això: només és una constatació des del nostre àmbit.

També sembla obvi (a una persona que desconeix tots el detalls legals) que revelar el codi font o al menys les especificacions formals d'aquests codi és el que requereix la

> Ley 19/2013, de 9 de diciembre, de transparencia, acceso a la información pública y buen gobierno.

Dins d'aquesta llei existeixen excepcions per tal d'eximir l'obligació de publicar el codi font. L'article 14 parla d'això:

> **Jurisprudencia Artículo 14. Límites al derecho de acceso.**
> 1. El derecho de acceso podrá ser limitado cuando acceder a la información suponga un perjuicio para:
>
> a La seguridad nacional.
>
> b La defensa.
>
> c Las relaciones exteriores.
>
> d La seguridad pública.
>
> e La prevención, investigación y sanción de los ilícitos penales, administrativos o disciplinarios.
>
> f La igualdad de las partes en los procesos judiciales y la tutela judicial efectiva.
>
> g Las funciones administrativas de vigilancia, inspección y control.
>
> h Los intereses económicos y comerciales.
>
> i La política económica y monetaria.
>
> j El secreto profesional y la propiedad intelectual e industrial.
>
> k La garantía de la confidencialidad o el secreto requerido en procesos de toma de decisión.
>
> l La protección del medio ambiente.

Des de les ciències computacionals[4] es pot fer una observació bàsica respecte a l'article 14. Si la seguretat nacional es veu compromès al publicar-se el codi

[4] Algunes parts de les ciències computacionals es consideren part de la lògica matemàtica.

font d'un software genèric que ha de decidir si una col·lecció de dades dona dret a obtenir un bono social, llavors aquest codi fa alguna cosa molt malament. Tan malament que el codi mateix suposa un perill per a la seguretat nacional. El mateix aplica a clàusules b,c, d, e, i l.

Concloem aquesta secció amb l'observació que en molts països d'Europa (com ara França o Països Baixos) ja és gairebé obligatori publicar el codi font per a software que fa servir l'administració pública per a la presa de decisions per a ciutadans.

2 Necessari, però no suficient

Com hem observat a dalt, des de la perspectiva de la lògica matemàtica i computació, la publicació del codi font o la seva especificació és necessari per tenir transparència i per erradicar ambigüetats. En aquesta secció enraonem que publicar el codi font no sempre és suficient però. En la ciencia de la computació és ben conegut que tot i que el codi font estigui publicat, es poden donar situacions que això no comporta que es pot entendre el que fa un programari. En els següents apartatsexpliquem el perquè.

2.1 El problema de l'aturada

El problema de l'aturada fou formulat per Alan Turing en 1937. El teorema corresponent diu que és fonamentalment impossible tenir un algorisme que pot saber si donat un programari amb entrada (input), aquest programari amb aquesta entrada atura en algun moment o segueix calculant per sempre. En particular, això implica que publicar el codi font d'un programari no comporta que automàticament podem entendre com es comporta: ni tan sols podem saber si s'atura el programari donada una entrada.

A sota incloem un esborrany d'una demostració d'aquest teorema. En realitat la prova no és gaire important per a juristes. Això no obstant, incloem la prova perque és una joia del pensament modern. Per tant, pel que fa al problema Bosco el lector es pot quedar amb el resultat i saltar-se els detalls.

Per tenir una idea del problema de l'aturada, vegem aquí alguns exemples senzills de programes. Primer, considerem un programa Π que pot prendre un nombre natural (aquests són els nombres $0, 1, 2, \ldots$) com a entrada i que fa el següent:

- Π llegeix un número d'entrada i l'emmagatzema al registre y;
- Fins que no s'aturi, Π farà el següent: Comprova $\$y$ –el valor del registre y– i **si** és més gran que 0, **aleshores** farà que el valor del registre y sigui una unitat més petit.
  (Si $\$y > 0$, llavors $\$y := \$y - 1$)

  en cas contrari (és a dir, si $\$y = 0$) Π s'aturarà.

Vegem, per exemple, com actuarà Π quan li donem l'entrada 3. Aleshores, una execució de $\Pi(3)$ seria el següent:

- En el primer pas Π llegeix l'entrada i l'emmagatzema al registre y de manera que $\$y = 3$;
- En el següent pas, Π comprova si $\$y$ és més gran que 0. Com que $3 > 0$ procedeix a substituir el valor actual 3 del registre per $3 - 1$ de manera que aconseguim $\$y = 2$;
- En el següent pas, Π comprova si $\$y$ és més gran que 0. Com que $2 > 0$ procedeix a substituir el valor actual 2 del registre per $2 - 1$ de manera que aconseguim $\$y = 1$;
- Repetint arribem al següent pas que $\$y = 0$;
- En el següent pas, comprova si $\$y$ és més gran que 0. Com que no és el cas que $0 > 0$, el programa simplement fa HALT (s'atura) per definició del seu codi.

Arribem a la conclusió que el programa Π a l'entrada 3 dóna lloc a una computació que finalitza. És a dir, el càlcul s'atura i escrivim $\Pi(3) \downarrow$ per indicar-ho. Per descomptat, és fàcil veure que aquest programa Π simplement resta 1 de l'entrada fins a tocar 0 quan s'aturarà. Així, Π s'aturarà en qualsevol nombre natural com a entrada. De fet, el Π que hem descrit més amunt s'assembla més a un algorisme que a un programa. Un programa seria un algorisme implementat en un dels llenguatges de programació estàndard com *C*, Pascal o semblant. Aquí ens abstenirem de distingir programes i algorismes.

Vegem ara un programa fàcil $\tilde{\Pi}$ que no s'aturarà en cap entrada de nombre natural x. Escriurem $\tilde{\Pi}(x) \uparrow$ per indicar que el programa $\tilde{\Pi}$ a l'entrada x no s'atura. També diem que $\tilde{\Pi}$ *fa un bucle* a l'entrada x.

Definim el programa $\tilde{\Pi}$ que fa el següent:

- Llegeix un nombre natural d'entrada i l'emmagatzema al registre y;
- Fins que no s'aturi, $\tilde{\Pi}$ farà el següent: Comprova $\$y$ –el valor al registre y– i **if** $\$y = \$y$, **aleshores** farà que el valor al registre y un més gran (Si $\$y = \$y$, aleshores $\$y := \$y + 1$)

  en cas contrari (és a dir, si $\$y \neq \$y$) s'aturarà.

Com que $\tilde{\Pi}$ només pot aturar-se quan algun valor no és igual a si mateix, veiem que el $\tilde{\Pi}$ de fet mai s'aturarà. Una execució de $\Pi(0)$ seria el següent:

- $\$y = 0$;
- $\$y = 1$;
- $\$y = 2$;

- $\vdots$

Fins ara, hem vist un programa Π que s'atura en totes les entrades de nombres naturals i un programa $\tilde{\Pi}$ que fa un bucle a totes les entrades de nombres naturals. A més, en els dos exemples que hem vist fins ara quedava clar què va passar. De vegades això no està tan clar i ara mostrarem un programa $\hat{\Pi}$ amb un comportament menys previsible.

$\hat{\Pi}$ fa el següent:

- Llegeix un nombre natural d'entrada i l'emmagatzema al registre y;
- Es repeteix el següent:
  Si el nombre és parell, el divideix per dos.
  Si el nombre és senar i més gran que 1, el triplica i l'afegeix-ne un.
- I només HALT (atura) quan arribem a un.

Vegem ara què passa quan donem una entrada determinada a $\hat{\Pi}$. Així, una execució de $\Pi(3)$ (començant per $\$y = 3$) seria el següent:

- $\$y = 3$; Com que el valor no és 1, el programa no s'atura. Més aviat comprova si $\$y$ és parell o senar. Com que 3 és senar, el programa procedeix a triplicar-lo i afegint-ne un al resultat ($3 \times 3 + 1 = 10$) fent d'aquest el nou valor del contingut del registre.
- Així, comencem al següent pas $\$y = 10$. Com que aquest no és 1 i com que és parell, el programa reduirà a la meitat el valor.
- $\$y = 5$; Com que aquest no és 1, i com que és senar, el programa el triplicarà i n'hi afegirà un. Així, en el següent pas començarem amb $3 \times 5 + 1 = 16$.
- $\$y = 16$; Com que 16 no és 1, i com que és parell, es dividirà per dos arribant a:
- $\$y = 8$; Com que 8 no és 1, i com que és parell, es dividirà per dos arribant a:
- $\$y = 4$; Com que 4 no és 1, i com que és parell, es dividirà per dos arribant a:
- $\$y = 2$; Com que 2 no és 1, i com que és parell, es dividirà per dos arribant a:
- $\$y = 1$; Com que ara el valor és igual a 1, la instrucció del programa diu al programa que HALT: s'atura.

Per a aquest programa en particular $\hat{\Pi}$ i aquesta entrada particular 3 hem vist que $\hat{\Pi}(3) \downarrow$, és a dir, $\hat{\Pi}$ a l'entrada 3 finalment s'atura. És un problema matemàtic obert (conjectura de Collatz) si $\hat{\Pi}$ s'atura a cada entrada de nombre natural [2]. El programa $\hat{\Pi}$ ha estat provat per a valors d'entrada astronòmicament grans x i sempre s'ha observat que $\hat{\Pi}(x) \downarrow$. Tanmateix, encara no s'ha demostrat que $\hat{\Pi}$ s'aturarà per a totes les entrades possibles.

La indecidibilitat del problema de l'aturada A dalt hem vist un programa Π que s'aturava a totes les entrades, un programa $\tilde{\Pi}$ que feia un bucle a totes les entrades i un programa $\hat{\Pi}$ pel qual actualment encara es desconeix si s'atura a totes entrades o no. Abans de donar un esbós de prova del problema d'indecidibilitat de l'aturada, vegem primer per què és important saber si un programa s'atura en una entrada determinada o no.

Suposem ara que el programa informàtic Π que és el programari informàtic BOSCO i que calcula tenir o no dret a un bono social. Per tant, si al programa Π se li dona algun fitxer d'ingressos x d'un ciutadà, llavors Π hauria de calcular $\Pi(x)$ que diu si té dret al bono social o no en te. Ens agradaria que aquest programa Π s'aturi en un moment determinat i doni la resposta. Si volem entendre completament com funciona Π, almenys ens agradaria saber que Π s'aturarà (i proporcionarà una sortida). El teorema sobre la irresolubilitat del problema de Halting (l'aturada) estableix que, en general, no és possible dir si un programa arbitrari Π s'atura en una entrada arbitrària x. Per descomptat, per a un programa particular Π amb una entrada particular x, pot ser possible demostrar que aquest programa en particular Π s'atura en aquesta entrada particular x. Tant de bo, de fet, el programari que decideix si un ciutadà té dret a bono social o no s'aturarà i podem donar-ne fe.

Un exemple de programari que millor no s'atura mai podria ser un sistema operatiu. Per tant, si Π és ara el sistema operatiu (per exemple, un telèfon intel·ligent) i x és una configuració inicial (per exemple, la configuració que tenia quan es va comprar a la botiga), llavors preferirem que $\Pi(x)$ mai s'aturaria i que el sistema operatiu continua funcionant per sempre (per seguir gestionant el telèfon).

Si volem tenir un cert control sobre el programari i què fa, en particular, hauríem de ser capaços de saber si un programa s'atura en una determinada entrada. El següent teorema ens diu que això en la seva forma sense restriccions és simplement impossible.

Theorem (Insolubilitat del problema de l'aturada)**.** *No pot existir un programa informàtic H que resolgui el problema d'aturada indicant si un programa informàtic arbitrari Π acabarà el seu càlcul quan s'iniciï a l'entrada x.*

Un cop més, insistim que la impossibilitat d'un programa d'aquest tipus H no depèn del nostre desconeixement actual. És un obstacle lògic que simplement no es pot superar. La IA avançada no pot ajudar amb això; ni ara ni mai en el futur. Ara procedirem a donar un esbós de demostració del teorema.

La demostració anirà per l'anomenat *raonament per contradicció*. Per tant, suposem que existís un programari[5] H que per a qualsevol entrada X i Y em diu si el programa X s'aturarà a l'entrada Y. Després d'alguns raonaments arribem a una impossibilitat de manera que arribem a la conclusió que la nostra suposició original (existeix un H d'aquest tipus) ha de ser realment errònia: no pot haver-hi un programa d'aquest tipus H.

[5] H de *Halting*.

Suposem llavors que existís un programari H que donat un altre programa X i donada una entrada Y a X ens indica si el programa X s'atura a l'entrada Y. Sense pèrdua de generalitat assumirem que H donarà el valor 1 en cas que el programa X s'atura a l'entrada Y i que H donarà el valor 0 en cas que el programa X entra en un bucle a l'entrada Y. Així, el programa H pren dos arguments i escrivim $H(X,Y)$ per al resultat de H. En resum, suposem que existís un programa $H(X,Y)$ amb la propietat que

$$H(X,Y) = \begin{cases} 1 & \text{quan } X(Y)\downarrow \\ 0 & \text{quan } X(Y)\uparrow . \end{cases} \tag{2}$$

Destaquem que el programa H té dues entrades, un programa[6] X i una entrada Y per al programa X. Per a la nostra generació actual el fet que un programa també pugui ser una entrada no és realment estrany, ja que estem acostumats als ordinadors. Per exemple, H podria ser un sistema operatiu, X i una aplicació que hem baixat i Y una entrada per a l'aplicació X. El sistema operatiu H carregaria X a la memòria de treball, l'alimentaria amb l'entrada Y i executaria com hauria de funcionar el programa X a l'entrada Y. Així, el programa H té com a entrada un altre programa X. Durant generacions anteriors a nosaltres, això va ser una mica un gir mental. Però de nou, estem tan acostumats que un programa al final només s'emmagatzema com a nombre binari. En particular, estem bé acostumats que el codi d'un programa X (com a nombre binari) podria ser entrada al mateix programa X; aquesta situació s'indicaria amb $X(X)$. És exactament aquest truc al que jugarem ara per obtenir alguna anomenada auto-referència[7].

Per tant, utilitzant el suposat programa d'aturada H no és difícil veure que podem ajustar H per obtenir un programa $\tilde{H}(X)$ amb només un argument X amb la propietat

$$\tilde{H}(X) = \begin{cases} 0 & \text{quan } X(X)\uparrow \\ \uparrow & \text{quan } X(X)\downarrow . \end{cases}$$

Com faríem el programa $\tilde{H}$ tot fent servir el suposat H? Volem $\tilde{H}(X) = 0$ en el cas que $X(X)\uparrow$. Com podem decidir si $X(X)\uparrow$? Per descomptat, utilitzem el programa H per a aquest fi i simplement executem $H(X,X)$ i programarem $\tilde{H}$ perquè doni un 0 en cas que $H(X,X) = 0$.

Tanmateix, en cas que $H(X,X) = 1$ farem alguna cosa especial: fem que $\tilde{H}(X)$ entri en un bucle tonto. Per exemple, mitjançant l'execució del programa $\tilde{\Pi}$ de la subsecció anterior. L'algorisme descrit així ens dóna el nou programa $\tilde{H}$.

Així, $\tilde{H}$ és un programa determinat i ben definit i el podem alimentar a si mateix com a entrada, és a dir, podem avaluar $\tilde{H}(\tilde{H})$. Però ara, tenim una contradicció ja que en el cas $\tilde{H}(\tilde{H})\downarrow$ això només pot ser en el cas $\tilde{H}(\tilde{H}) = 0$ i llavors, per definició de $\tilde{H}$ aquest és exactament el cas quan $\tilde{H}(\tilde{H})\uparrow$. I viceversa,

[6] El fet que X no sigui un programa o que Y no sigui una entrada vàlida per a X són aspectes tècnics que no són essencials per a l'argument.

[7] El terme tècnic en anglès és Self-Reference.

si $\tilde{H}(\tilde{H})\uparrow$, per la definició de $\tilde{H}$, això només pot ser quan $\tilde{H}(\tilde{H})\downarrow$. Això és clarament una contradicció ja que hem conclòs

$$\tilde{H}(\tilde{H})\downarrow \iff \tilde{H}(\tilde{H})\uparrow .$$

Així, la nostra suposició original que existís un programa d'aquest tipus H amb les propietats indicades a (2) condueix a una contradicció i concloem que no existeix pas aquest algorisme H. Això completa l'esbós de prova de la indecidibilitat del problema de *Halting*[8].

El problema de l'aturada i els problemes relacionats apareixen en diversos problemes de la vida real. Un corol·lari directe és que no hi pot haver un escàner de virus infal·lible; amb o sense IA. De la mateixa manera, el problema de l'aturada fa que la verificació sense restriccions de la correcció del programa (el que tant volem per al programari que afecta els ciutadans) és impossible. De nou, totes aquestes impossibilitats es mantenen amb o sense IA. Simplement, aquests problemes no són computables.

No diem que per un programari determinat no es pot decidir si s'atura amb una certa entrada o no. En particular, si el programa BOSCO està ben dissenyat i programat és molt probable que fàcilment podem demostrar que per a qualsevol entrada admesa, el programa BOSCO s'atura i dona una resposta. El següent pas seria demostrar que la resposta sempre fos correcta. Però, per poder dir *correcta* i per poder demostrar la correcció amb rigor matemàtic, requerim una especificació formal del programari. El punt important per a nosaltres és: per saber com funciona el programari BOSCO, tenir accés al codi font és necessari però no suficient.

3 Sobre el llenguatge per a lleis computacionals

És una pràctica habitual que la llei s'escrigui en llenguatge natural, fins i tot si aquesta llei s'ha de fer complir de manera automatitzada. Es parla molt a la literatura del procés *De la llei al codi* i, en general, encara no està clar quines són les pràctiques òptimes. Alguns projectes opten per llenguatges seminaturals com el projecte Catala [7, 3] però el llenguatge natural segueix sent el mode més acceptat.

Sobretot si la llei està escrita en llenguatge natural i si aquesta llei tracta d'estipulacions numèriques, aleshores aquesta llei sol ser molt propensa a implicar un comportament matemàtic salvatge o no desitjat d'una altra manera. A la següent secció ho il·lustrarem donant diversos exemples de problemes matemàtics de dret. Hi ha molts altres exemples a la literatura [6, 1] però aquí prendrem principalment exemples de[9] Normes de trànsit europees tal com es descriuen a [8]. Veurem que alguns dels nostres exemples també podrien haver

[8] Insolubilitat del problema de l'aturada

[9] Alguns exemples no es troben a [8] sinó només en una versió completa en línia del mateix `http://www.joostjjoosten.nl//papers/2023ModelCheckingInFoundationsComputableLaws.pdf`.

passat desapercebuts en cas que la llei estigués escrita en (pseudo)codi. Per tant, canviar al pseudocodi no serà un medicament per a totes les matemàtiques.

4 Especificació insuficient

En aquest apartat veurem una causa d'errors molt típica en casos com ara el programa BOSCO. En particular, una font clàssica de problemes amb les lleis computables es pot anomenar *Especificació insuficient*[10]. Un article legal A estipularà determinats requisits per complir amb la llei. Aleshores sovint hi haurà alguna situació S on no està clar si S és legal segons l'article A o no. De vegades, la Especificació insuficient en A és evident, de vegades és més subtil. Començarem amb un exemple senzill de certa subtilesa. Per a això, primer donem[11] l'article corresponent que és l'article 6 del Reglament (CE) núm. 561/2006 del Peuropeu i del Consell, de 15 de març de 2006, relatiu a l'harmonització de determinada legislació social relativa al transport per carretera ([9]), punt 1.

> Article 6.1: El temps de conducció diari no ha de superar les nou hores. No obstant això, el temps de conducció diari es pot ampliar fins a un màxim de 10 hores no més de dues vegades durant la setmana.

A primera vista, l'article sembla poc problemàtic. Detectar la falta d'especificació és una mica subtil. Per veure el problema de l'article 6.1, primer hem de comentar algunes de les paraules de l'article.

Primer, hi ha la paraula *setmana*. El reglament en el seu article 4 (i) és molt explícit en el que entén per setmana[12]:

> Article 4(i): "una setmana" significa el període de temps entre les 00.00 hores del dilluns i les 24.00 hores del diumenge.

Si realment volem ser pedants: l'article 4(i) del reglament no estableix de manera explícita que el diumenge es refereix realment al primer diumenge següent al dilluns considerat. Bé, això no està gens malament i es pot defensar que es pot suposar una mica de coneixement comú. Però tot i així, el coneixement comú no és massa estable i el programador haurà de llegir entre línies si vol escriure un programa que implementi la llei per als lectors de tacògrafs (que clarament s'executarà en un ordinador).

Una altra subtilesa pedant rau en la possibilitat dels anomenats *segons intercalats*. Els segons intercalats ([4, 5]) són segons que es poden afegir a un dia per compensar l'acceleració (positiva o negativa) de la rotació de la terra. No entrem en detalls aquí, però cada mig any el Servei Internacional de Sistemes de Referència i Rotació de la Terra pot decidir afegir o restar un segon intercalat al nostre calendari. Així, un minut natural pot ser de 59 segons (en cas de segon

[10] *Underspecification* en anglès.
[11] La traducció al Català és nostre.
[12] Traducció nostre.

intercalat negatiu), o de 60 segons (per a un minut normal) o de 61 segons (en cas que s'afegeix un segon intercalat positiu a aquest minut en concret). Per convenció, poden haver-hi com a màxim dos segons intercalats a l'any per incloure'ls al final de l'últim dia del mes, preferiblement al juny o al desembre.

Per tant, pot passar que diumenge acabi a les 23:59 o a les 24:01. En el primer cas, què ha de fer el programador? A causa del segon intercalat negatiu, diumenge acabarà a les 23:59 i el moment de les 24:00 de diumenge no existeix? Si el programador ha de seguir la lletra de la llei, la setmana acabarà el proper diumenge a les 24:00? Pensem que, segons l'esperit de la llei, queda clar com actuar. Hauríem d'entendre que l'article 4 (i) realment vol dir: "Una setmana és el període de temps comprès entre l'inici d'un dilluns i el final del diumenge següent després". Innocent, però el fet és que el programador hauria de prendre una decisió legal aquí. Un cop més, aquest és un exemple força innocent, però empitjorarà. Per veure com empitjora l'ambigüitat present a l'article 6.1 haurem de dir què entén el reglament per *temps de conducció diari*. A l'article 4 (k) ens assabentem que[13]

> Article 4, lletra k): "temps de conducció diari": el temps total de conducció acumulat entre el final d'un període de descans diari i l'inici del període de descans diari següent o entre un període de descans diari i un període de descans setmanal.

Continuem en el mode pedant i observem algunes qüestions menors sense importància. Primer observem que *diari* no fa referència al que la majoria de la gent entén per dia. En termes llingüístics d'adjectius, podríem dir que en la frase "temps de conducció diàri" treballem amb un modificador privatiu més que no pas subsectiu. Això vol dir que "temps de conducció diàri" agafa un significat com a conjunt i no és pas el cas que agafem la frase "temps de conducció" i li aplicarem l'adjectiu diària. L'ús de modificadors privatius en la comunicació entre juristes i experts en informàtica sembla una font segur de problemes.

Però, aquí hi ha una altra falta d'especificació: què passa amb el temps de conducció entre dos períodes de descans setmanal? Imagineu que un conductor té un període de descans setmanal i després només treballa un dia abans de prendre un altre període de descans setmanal. Sembla ser una situació no estàndard i una situació que l'empresari voldria que no es produís. No obstant això, la situació es pot donar perfectament i, segons la lletra del reglament, és legal[14] tenir un descans setmanal seguit de quatre hores i mitja de conducció,

[13] Com abans, la traducció és nostra.

[14] La naturalesa del nostre exemple està configurada per complir amb la resta de requisits imposats per l'article 7 i l'article 8.2. Aquí l'article 7 diu "Després d'un període de conducció de quatre hores i mitja, el conductor ha de fer una pausa ininterrompuda no inferior a 45 minuts, tret que faci un període de descans". I l'article 8.2 diu "Dins de cada període de 24 hores després de la finalització del període de descans diari o període de descans setmanal anterior, el conductor haurà d'haver pres un nou període de descans diari". A més, l'article 8.3 diu "Un període de descans diari es pot ampliar per fer un període de descans setmanal regular o un període de descans setmanal reduït". No està força clar si en el cas d'estendre un període de descans diari per convertir-se en un període de descans setmanal si aquesta franja de temps és simultàniament un període de descans diari i setmanal. Perquè el nostre exemple

seguit per 45 minuts de descans, seguit per quatre hores i mitja de conducció, seguit de 45 minuts de descans, seguit un altre cop per quatre hores i mitja de conducció, i després un període de descans setmanal següent.

A primera vista, com que vam esprémer 13,5 hores de conducció en un període curt, sembla que tenim una infracció de l'article 6.1 tal com s'ha citat anteriorment.

No obstant això, com que la nostra activitat de conducció està immediatament delimitada per dos períodes de descans setmanal, l'activitat de conducció entre mig no defineix, en sentit estricte, un temps de conducció diari. És evident que això no és d'acord amb l'*esperit de la llei*.

Abans de passar a exposar la falta d'especificació més sorprenent de l'article 6.1, esmentem una altra qüestió pedant de l'article 6.1 en combinació amb l'article 4(k). Hi ha un cas límit degenerat que és problemàtic per a l'article 4 (k). És a dir, quan un conductor és nou a la seva feina. Quan un conductor és nou vingut i comenci a conduir, la targeta de conductor corresponent no tindrà cap activitat registrada i, en particular, encara no tindrà un període de descans (diari) per la qual cosa tampoc no hi haurà temps de conducció diària. Per descomptat, hi ha una manera fàcil i natural de fer front a aquesta anomalia una mica acadèmica. Però de nou, aquest és un exemple d'una decisió (no problemàtic i senzill en aquest cas) deixada al programador/modelista.

A hores d'ara hauria de quedar clar que una activitat diària de conducció pot caure repartit entre dos dies. Considerem ara la situació en què un conductor comença a conduir un diumenge al vespre i acaba un dilluns al matí. Fins aquí tot bé. Tanmateix, si el temps de conducció diari acumulat en aquest període sumava 10 hores, veiem una altra falta d'especificació. Segons la normativa es permet tenir un període de conducció de 10 hores però no més de dues vegades per setmana. Atès que la setmana es defineix com a que comença el dilluns a les 00:00 i acaba el diumenge a les 24:00, l'article 6.1 del reglament no té clar a quina setmana s'ha d'atribuir el temps diari de conducció ampliat: a la setmana que contingués el diumenge on va començar la conducció, o a la setmana que contingués el dilluns on va acabar la conducció. El conductor és lliure d'escollir? Un cop més, la regulació té falta d'especificació aquí. En la desambiguació presentada a [8], el període de descans diari ampliat s'assigna sempre a la setmana que comença el dilluns on va acabar la conducció. Diversos lectors de tacògrafs semblen prendre decisions diferents i, per tant, implementen lleis diferents. Alguns programaris de tacògraf tenen una opció per fixar una determinada elecció o per triar la distribució per minimitzar la multa. És evident que tota aquesta incertesa s'hauria pogut evitar tenint articles inequívocs al reglament. Sense l'ús de mètodes formals, és molt probable que la falta d'especificació encara estigui present en algun lloc possiblement en un lloc ocult o inesperat.

funcioni, hem entès que si un període de descans diari s'amplia per convertir-se en un període de descans setmanal, llavors deixa de ser un període de descans diari i es converteix només en un període de descans setmanal.

References

[1] G. Errezil Alberdi. *Industrial Software Homologation: Theory and case study. Industrial Software Homologation: Theory and case study Analysis of the European tachograph technology with EU transport Regulations 3821/85, 799/2016, and 561/06 and their consequences for Europeans citizens*. Technical Report. Formal Vindications S.L., 2019.

[2] Richard K. Guy. "E16: The 3x+1 problem". In: *Unsolved Problems in Number Theory*. 3rd ed. Springer New York, 2013, pp. 330–336. ISBN: 978-0-387-26677-0.

[3] Liane Huttner and Denis Merigoux. "Catala: Moving Towards the Future of Legal Expert Systems". In: *Artificial Intelligence and Law* (Aug. 2022). DOI: 10.1007/s10506-022-09328-5. URL: https://inria.hal.science/hal-02936606.

[4] ITU Radiocommunication Assembly. "Recommendation ITU-R TF.460-6: Standard-frequency and time-signal emissions". In: *International Telecommunication Union* (2002). URL: https://www.itu.int/dms_pubrec/itu-r/rec/tf/R-REC-TF.460-6-200202-I!!PDF-E.pdf.

[5] *leap-seconds.list*. 2024. URL: https://data.iana.org/time-zones/data/leap-seconds.list.

[6] Denis Merigoux, Marie Alauzen, and Lilya Slimani. "Rules, Computation and Politics: Scrutinizing Unnoticed Programming Choices in French Housing Benefits". In: *Journal of Cross-disciplinary Research in Computational Law* 1.3 (2023). (forthcoming). URL: https://hal.inria.fr/hal-03712130.

[7] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. "Catala: A Programming Language for the Law". In: *Proceedings of the ACM on Programming Languages* 5 (2021). DOI: 10.1145/3473582. URL: https://doi.org/10.1145/3473582.

[8] Moritz Müller and Joost J. Joosten. "Model-checking in the Foundations of Algorithmic Law and the Case of Regulation 561". In: *ArXiv:2307.05658 [cs.LO]* (2023).

[9] European Parliament and Council of the European Union. "Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport". In: *Official Journal of the European Union* (2006).