



UNI 4 JUSTICE

UNIVERSITAS PER LA GIUSTIZIA. PROGRAMMA PER LA QUALITÀ DEL SISTEMA GIUSTIZIA E PER L'EFFETTIVITÀ DEL GIUSTO PROCESSO

AZIONE 1.2


STRUMENTI OPERATIVI DI MONITORAGGIO E CONSULTAZIONE PER IL POTENZIAMENTO DELL'UPP

D1.2.3 DATABASE DELLE SENTENZE E MASSIME IN AKOMA NTOSO DISPONIBILE CON TECNOLOGIE WEB

AZIONE 2.2

MODELLI DI TRASFORMAZIONE DIGITALE AVANZATA E DELLE INTERFACCE HCI

D2.2.2 PROTOTIPO DI EDITOR WEB PER ELABORARE UNA SENTENZA DIGITALE IN FORMATO XML AKOMA NTOSO ANNOTATA E SEMANTICA

	<h2 style="text-align: center;">D1.2.3 e D2.2.2– Linea di Azione 1 e 2</h2> <p style="text-align: center;">D1.2.3 Database delle sentenze e massime in Akoma Ntoso disponibile con tecnologie Web</p> <p style="text-align: center;">D2.2.2 Prototipo di editor web per elaborare una sentenza digitale in formato XML Akoma Ntoso annotata e semantica</p>
---	---

Storia del documento

Versione	Data	Autore	Partner	Descrizione
1.0	10/06/2023	Monica Palmirani	UNIBO	Primo draft
2.0	11/07/2023	Stefano Notari	UNIBO	Secondo draft
3.0	13/08/2023	Monica Palmirani	UNIBO	Terzo draft
4.0	16/09/2023	Monica Palmirani	UNIBO	Finale

Indice

Storia del documento	1
1. Sommario	1
Parte I a cura dell'Università di Bologna	2
2. Convertitore di sentenze PDF in testo	2
3. L'editor LIME-SENTENZE	2
4. La banca dati	5
5. Le Funzionalità del portale web	6
6. Problemi incontrati	7
Parte II a cura dell'Università di Verona - Houdini: una tecnologia per il ragionamento deontico non-monotono.	8

1. Sommario

Questo deliverable riporta quattro risultati:

- i. un convertitore da PDF a testo per la preparazione dei documenti;
- ii. l'editor specializzato dotato di parser per trasformare i PDF e i testi in Akoma Ntoso;
- iii. la banca dati delle sentenze secondo le tecniche semantiche di annotazione apportate con lo standard Akoma Ntoso;
- iv. un tool per l'annotazione delle regole giuridiche secondo un modello deontico non-monotono basato sulla logica defeasible realizzato dall'Università di Verona.

Parte I a cura dell'Università di Bologna

2. Convertitore di sentenze PDF in testo

Si è sviluppato un convertitore da PDF a testo per preservare quanto più possibile delle sentenze che non sono in un formato annotabile. Questo tool è servito soprattutto quando gli UUGG non avevano a disposizione i file word o quando si sono scaricate le sentenze dal portale Pst del Ministero della Giustizia.

<http://u2.cirsfid.unibo.it/sentenze/frontend/>



3. L'editor LIME-SENTENZE

Per marcare in XML Akoma Ntoso si sono costruiti dei parser tali da processare la struttura delle sentenze dividendone le parti principali:

1. Introduzione
2. fatto
3. motivazione
4. decisione
5. firme.

Si sono identificati ove possibile le parti, i giudici, gli avvocati, nonché i riferimenti giurisprudenziali e normativi collegandoli a Normattiva e EUR-LEX in caso di citazione di norme europee.



Figura 1 – Trasformazione in Akoma Ntoso e marcatura degli elementi strutturati incluse le parti.

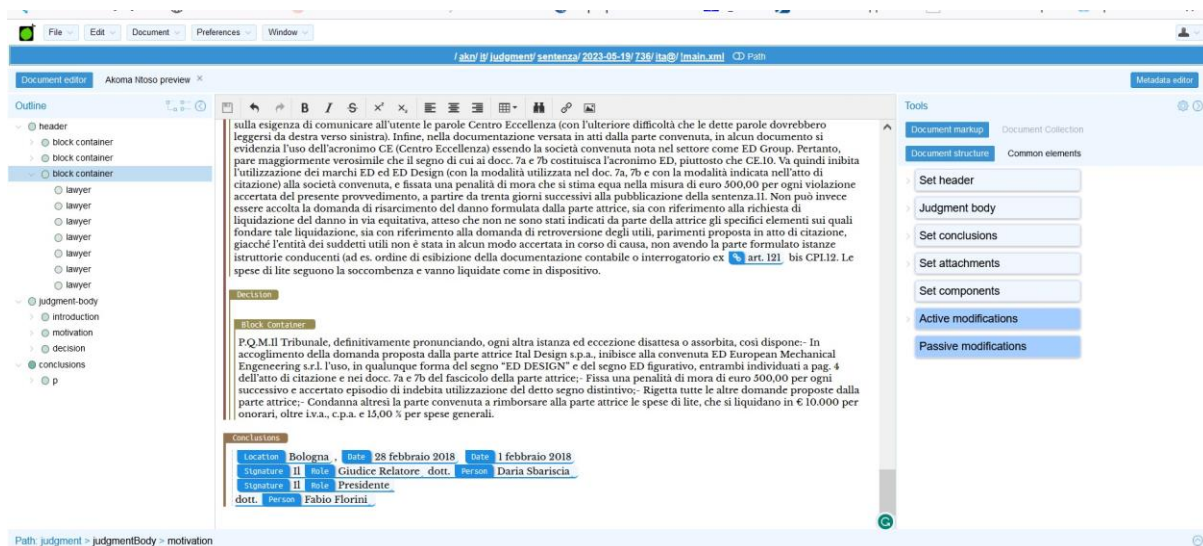


Figura 2 – Trasformazione in Akoma Ntoso e marcatura degli elementi strutturati inclusi i riferimenti normativi.

Si sono anche elaborati degli strumenti per annotare i testi secondo le tassonomie e le ontologie sviluppate nel progetto. Per esempio, in merito al micro-spaccio, il codice rosso, gli sfratti si sono identificati degli indicatori guida che possono agevolare le ricerche ma anche l'elaborazione futura di modelli AI.

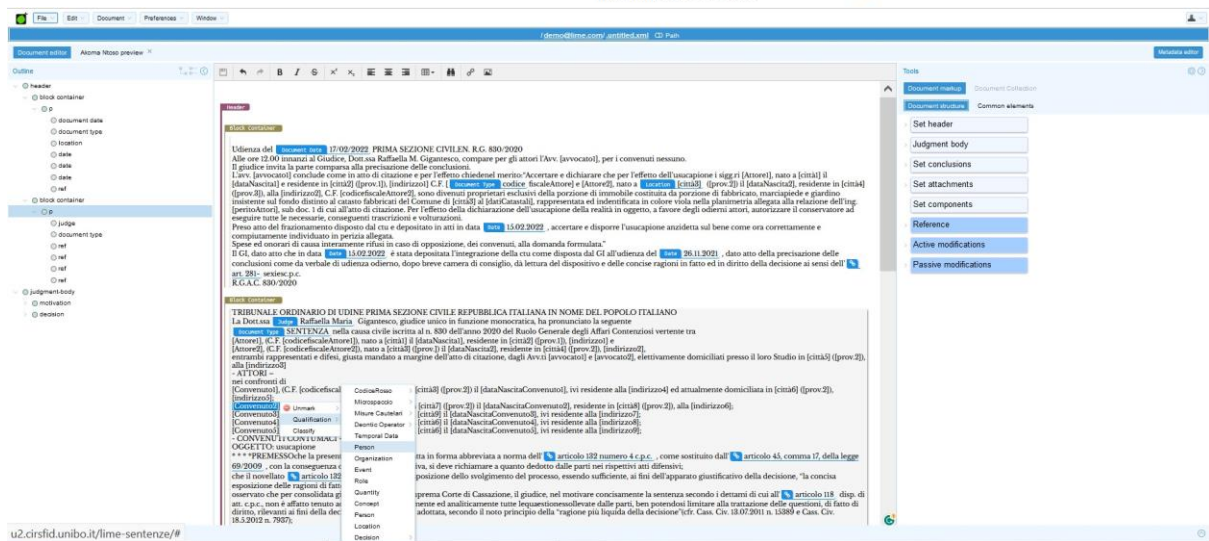


Figura 3 – Annotazione semantica delle sentenze.

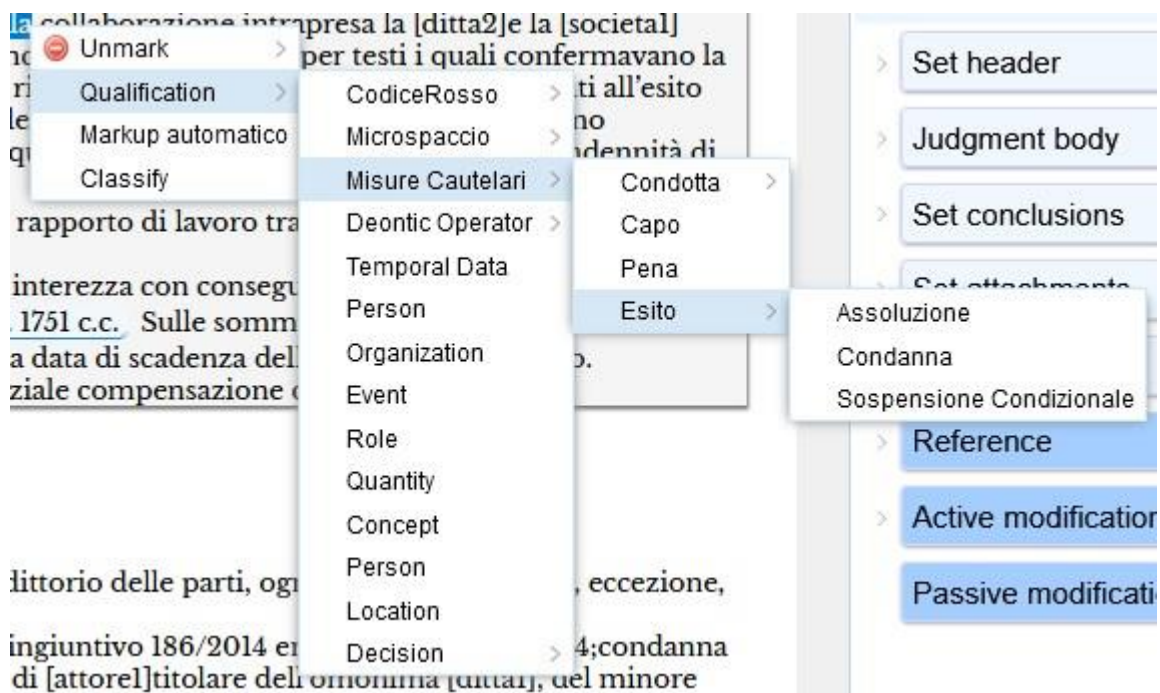


Figura 4 – Menu per annotare parti qualificate della procedura civile o penale.

Ove si è attuata l'anonimizzazione mediante linee guida (e.g., [convenuto]) si è intercettato il parametro per consentire alle statistiche e ai canali di ricerca di poter operare avendo almeno l'elemento semantico (persona – convenuto) a disposizione pur avendo rimosso il valore.



Figura 6 – Portale della banca dati.

E' possibile scaricare tutte le sentenze in XML Akoma Ntoso in open data.

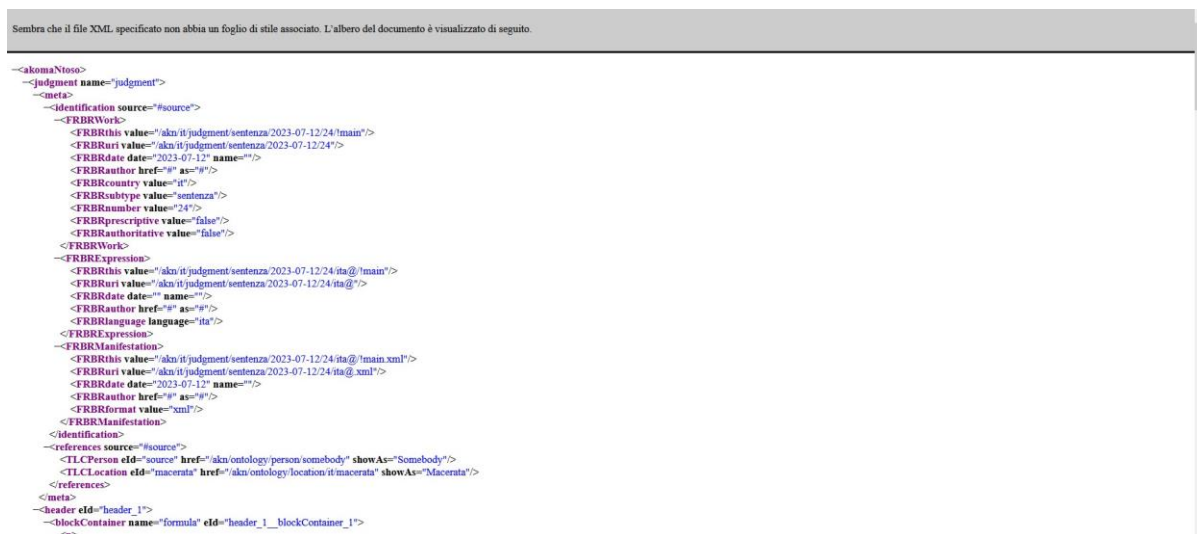


Figura 7 – Formato XML-AKN scaricabile in Open Data.

5. Le Funzionalità del portale web

Le funzionalità che si intende implementare sono le seguenti:

1. Ricerche per parole chiave
2. Ricerche per testo libero
3. Ricerche secondo i campi più comuni (e.g., data, numero sentenza)
4. Ricerche per la ratio decidenti
5. Navigazione di tutta la catena di gradi di giudizio se esistente
6. Navigazione mediante grafici dei riferimenti giurisprudenziali e legislativi nazionali ed europei
7. Navigazione mediante link ipertestuali dei riferimenti giurisprudenziali e legislativi nazionali ed europei
8. Segnalazione delle sentenze che citano norme modificate o abrogate
9. Segnalazione delle sentenze che citano il documento in oggetto
10. Visualizzazione della network analysis dei riferimenti giurisprudenziali e legislativi
11. Statistiche generali e grafici
12. Traduzione delle sentenze in formato XML Akoma Ntoso
13. Navigazione point-in-time dei riferimenti legislativi a Normattiva e EUR-LEX
14. Analisi di similitudine delle sentenze

6. Problemi incontrati

I maggiori problemi incontrati risiedono nel formato dei documenti che spesso sono disponibili in PDF, con firme e sigilli che rendono la digitalizzazione semantica difficoltosa, frammentando con elementi grafici e presentazionali il flusso del contenuto.

Il secondo elemento che ha creato non pochi problemi è l'assenza di una intestazione uniforme che possa servire come metodo di riconoscimento automatico degli elementi introduttivi della sentenza quali il numero, la data, la corte, le parti e gli avvocati.

Il terzo elemento di ostacolo è determinato dalla difformità di organizzazione del testo. Alcuni magistrati dividono in modo ordinato fatto e diritto, nonché anche gli elementi di introduzione quali i riassunti di memorie conclusive delle parti. Altri adottano un flusso continuo di argomentazione mista agli elementi fattuali che difficilmente aiutano a definire la semantica del testo.

Infine, spesso le sentenze sono state scaricate da banche dati che non riportano interamente il testo o gli estremi della sentenza, rendendo così difficoltosa la ricostruzione anche degli elementi base per definire in modo univoco l'URI.

Si deve poi sottolineare il fatto che ad oggi non esiste uno standard ECLI disponibile per le sentenze di primo e secondo grado, ma esistono le specifiche ECLI solo per le alte corti. Questo rende difficile l'assegnazione di un URI univoco, condizione essenziale nel Semantic Web per poter definire asserzioni e annotazioni semantiche.

**Parte II a cura dell'Università di Verona - Houdini: una tecnologia per il
ragionamento deontico non-monotono.**

Houdini: una tecnologia per il ragionamento deontico non-monotono. Verso un apparato di supporto alle decisioni in ambito giuridico, con spiegazione esplicita

Matteo Cristani · Guido Governatori · Francesco
Olivieri · Luca Pasetto · Francesco Tubini ·
Celeste Veronese · Alessandro Villa · Edoardo
Zorzi

Abstract Dagli anni Novanta del secolo scorso, i ricercatori nel campo delle logiche non monotone hanno lavorato su algoritmi per il ragionamento con conoscenze incomplete e inaccurate, che rappresentano l'essenza stessa della non monotonicità.

Sono state sviluppate diverse tecnologie, ma al giorno d'oggi manca ancora un ragionatore efficace che sia disponibile online. In uno sforzo durato più di un anno abbiamo sviluppato una tecnologia in grado di ragionare con la defeasible deontic logic (DDL), in presenza di variabili proposizionali, costanti, valori numerici, stringhe, liste e aspetti deontici con catene di riparazione.

Keywords Defeasible Logic, Deontic Logic, Automated Reasoning

1 Introduzione

La logica defeasible, in particolare nelle sue versione proposizionale deontica, è centrale rispetto a tecnologie già esistenti come *SPINdle* (Lam and Governatori 2009) e altre che menzioneremo in dettaglio nella Sezione 6. Tuttavia al momento non esiste un'architettura moderna che sia disponibile open source e interrogabile da un'interfaccia web. Chiaramente è necessaria una soluzione del genere nella comunità che si occupa di ragionamento non monotono, in particolare con l'obiettivo di automatizzare porzioni di ragionamento legale.

Il sistema descritto in questo articolo è chiamato *Houdini* (Cristani et al. 2022). Si tratta di un sistema di ragionamento per la defeasible deontic logic che è disponibile online e permette di gestire variabili numeriche e stringhe. Qui discutiamo l'implementazione della sua versione attuale (2.2) che estende le versioni precedenti includendo: (i) logica proposizionale, (ii) logica deontica, (iii) catene di riparazione e (iv) gestione di variabili

M. Cristani · L. Pasetto · C. Veronese · A. Villa · E. Zorzi

Department of Computer Science, University of Verona

Strada le Grazie 15, Verona, 37134, VR, Italy

E-mail: {matteo.cristani; luca.pasetto; celeste.veronese; alessandro.villa; edoardo.zorzi}@univr.it

G. Governatori

F. Olivieri

Brisbane, 4000 QLD, Australia

E-mail: guido@governatori.net francesco.olivieriphd@gmail.com

numeriche e di stringhe (caratteristiche che sono solo parzialmente implementate dalle altre tecnologie). La soluzione è presentata da un punto di vista funzionale, e discutiamo anche la soluzione algoritmica sottostante la tecnologia stessa, che è una leggera variante dell'algoritmo di Maher (Maher 2001). Questa variante è effettivamente corretta e completa, ma qui omettiamo la dimostrazione.

Il resto del report è organizzato come segue. La Sezione 2 introduce il quadro logico di base, la Sezione 3 discute l'architettura della tecnologia, mentre la Sezione 4 fornisce un'analisi delle strutture dati impiegate e delle ottimizzazioni implementate. La Sezione 6 si conclude con la revisione dei lavori correlati e la discussione delle prossime linee di ricerca.

2 Defeasible deontic logic

La Logica Defeasible (Nute 1994) è un formalismo non monotono basato su regole che è flessibile ed efficiente per i nostri scopi. La sua forza risiede in due caratteristiche principali. In primo luogo, la sua proof theory costruttiva ci permette di trarre conclusioni significative da basi di conoscenza (potenzialmente) conflittuali e incomplete. Nei sistemi non monotoni, conclusioni più accurate possono essere ottenute quando diventano disponibili più informazioni. In secondo luogo, sono state proposte molte varianti del formalismo per modellare diverse aree applicative, in particolare il comportamento di agenti (Governatori et al. 2016), ragionamento legale (Cristani et al. 2017; Governatori et al. 2021), e flussi di lavoro da una prospettiva di controllo di conformità dei processi aziendali (Olivieri et al. 2013; Governatori 2015; Cristani et al. 2016).

La definizione del linguaggio per la defeasible deontic logic con catene di riparazione può essere trovata in (Governatori et al. 2013, 2019), e qui la omettiamo per motivi di spazio.

3 Algoritmi per la Defeasible Deontic Logic

Houdini è un'applicazione web basata su Spring e scritta in Java. L'architettura del sistema è suddivisa in tre componenti principali: interfaccia utente, parser (il quale funge anche da modulo di validazione dell'input) e reasoner. L'architettura così suddivisa è rappresentata in figura 1.

L'interfaccia utente interattiva consente di fornire in input al sistema teorie espresse in defeasible deontic logic sia tramite l'upload di un file JSON, sia compilando il web-form generato dinamicamente dall'applicazione. In tal caso la teoria verrà comunque convertita dall'applicazione in una struttura in formato JSON, in modo tale da fornire al modulo di validazione un formato di file univoco e che rispetti le stesse regole sintattiche.

Le teorie fornite in input possono contenere tre tipi di informazioni: *fatti*, *regole* e *relazioni di superiorità*. Un fatto è sostanzialmente un singolo letterale, esprimibile mediante qualsiasi stringa alfanumerica (con l'aggiunta del carattere '_') ed eventualmente preceduto dal simbolo '~ per indicarne la **negazione**. Le regole sono costituite da un corpo opzionale (singoli letterali o sequenze separate da virgola), una freccia indicante la natura della regola stessa ed un singolo letterale (non opzionale) che funge da testa. Le relazioni di superiorità, infine, sono espresse indicando gli identificativi¹ delle regole coinvolte separati dal carattere '>', volto ad indicare la superiorità della regola di sinistra su quella di destra.

¹ Le regole sono automaticamente etichettate con numeri progressivi, seguendo l'ordine di inserimento: r_1, r_2, \dots

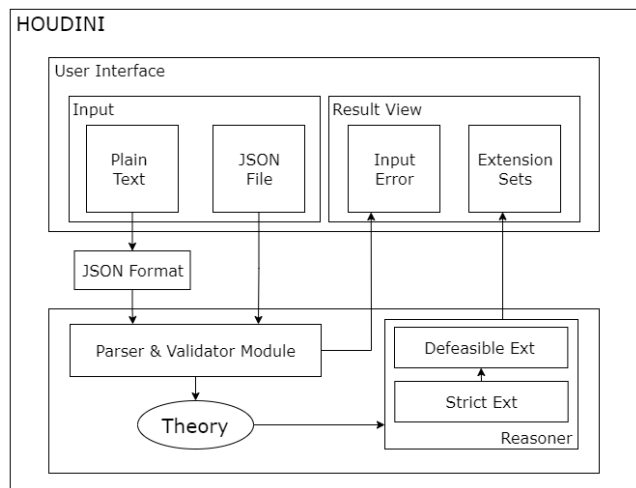


Fig. 1: Architettura e flusso dei dati in Houdini

Il modulo di parsing riceve in input la struttura JSON rappresentante l'input dell'utente e, per prima cosa, utilizza un parser CFG (implementato in Antlr) per verificarne la validità sintattica. Si occupa quindi di costruire la rappresentazione ad oggetti della teoria, la quale sarà quindi fornita al reasoner che procederà al calcolo dell'estensione. Il reasoner opera per mezzo di due moduli interni, uno per il calcolo dell'estensione stretta della teoria (*Strict Reasoner*) ed uno per il calcolo dell'estensione defeasibile (*Defeasible Reasoner*). Il compito dello *Strict Reasoner* è il calcolo di $+\Delta$ e $-\Delta$, che vengono costruiti incrementalmente identificando i letterali facenti parte degli insiemi mediante una strategia di *iniezione*: i fatti, facenti parte di $+\Delta$ per definizione, vengono rimossi da tutti i copri delle regole di cui fanno parte, causando quindi la potenziale attivazione di nuove regole (il cui corpo risulta vuoto), le teste delle quali saranno aggiunte a $+\Delta$ e innescheranno la medesima procedura di iniezione utilizzata per i fatti. Il procedimento viene ripetuto iterativamente fintanto che nuove regole risultano attive, terminando quindi a punto fisso. A questo punto, l'insieme $+\Delta$ è stato interamente popolato, e la computazione passa ad identificare quei letterali che, non essendo nè fatti nè tantomeno teste di regole strette, rappresentano i primi elementi da inserire in $-\Delta$. La stessa procedura di iniezione viene quindi applicata a partire da questi elementi per costruire interamente $-\Delta$.

Il calcolo di $+\partial$ e $-\partial$, invece, utilizza come punto di partenza i letterali che non sono ancora stati assegnati ad alcun insieme. Per ognuno di essi, il *Defeasible Reasoner* verifica le condizioni di appartenenza ai due insiemi, applicando quindi la procedura di iniezione/disattivazione delle regole con i letterali identificati come idonei. A differenza del caso precedente, però, l'inserimento dei letterali candidati nei rispettivi insiemi di appartenenza non avviene fintanto che l'algoritmo, che itera sull'insieme dei candidati, non raggiunge il punto fisso (ciò avviene quando nessun letterale è aggiunto all'insieme). A questo punto, la procedura si conclude e l'estensione della teoria viene resa disponibile per la visualizzazione.

Fig. 2: Houdini's home page

Fig. 3: Houdini - calcolo dell'estensione

4 Houdini: descrizione della tecnologia

4.1 Specifiche tecniche

Allo stato attuale, Houdini è un'applicazione web a singolo endpoint e dotata di un'interfaccia utente intuitiva e dinamica. La scelta del linguaggio di programmazione Java per l'implementazione è dovuta sia all'alta manutenibilità e portabilità che questo assicura, sia all'esistenza di framework web consolidati adatti a sviluppare applicazioni di facile comprensione per i non professionisti del settore informatico, quali studenti e ricercatori. In particolare, l'implementazione backend di Houdini ha previsto l'utilizzo di Java 11 e Spring Boot 2.7.1, mentre l'interfaccia utente si basa interamente su Javascript, risultando quindi leggera e portabile. Houdini è interamente open source e disponibile su richiesta.

4.2 Dati e interfaccia utente

L'interfaccia utente di Houdini comprende due pagine differenti: la pagina principale (Figura 2), che consente l'interimento delle teorie espresse in defeasible deontic logic manualmente o tramite file JSON, e la pagina di visualizzazione dei risultati (Figura 3), che mostra l'estensione della teoria ricevuta in input.

L'utente è supportato nel corretto inserimento delle teorie (evitando, ad esempio, errori di formattazione e caratteri non consentiti) tramite messaggi di errore generati in risposta ad ogni carattere inserito e categorizzati tramite l'uso di colori differenti. I campi di testo per

l'inserimento degli elementi della teoria vengono prodotti dinamicamente e possono essere eliminati dall'utente, in modo da mantenere una visualizzazione pulita e prima vi campi inutilizzati. Oltre ai tre campi di testo inizialmente forniti, è presente un pulsante di upload che può essere utilizzato dall'utente per il caricamento del file JSON contenente l'intera teoria già formalizzata, la cui sintassi, molto simile a quella utilizzata per l'inserimento manuale, risulta semplice ed intuitiva.

La seguente regola, ad esempio, è sintatticamente corretta ed accettata dal sistema: syntactically correct and is allowed by the system: “a, [O]b, $x > 3 \Rightarrow [P] \sim c$ ”. Il corpo della regola, “a, [O]b, $x > 3$ ”, è costituito da tre elementi: il letterale “a”, una proposizione deontica marcata come obbligo (O) con variabile “b” e parametro “x”, ed un vincolo logico “ $x > 3$ ”.

Le relazioni di superiorità consistono semplicemente in due identificativi di regole separati dal carattere “>”: il lato sinistro della relazione identifica la regola considerata superiore, mentre il destro quella inferiore. Gli identificativi delle regole sono generati concatenando un numero incrementale al carattere “r”, un esempio di relazione di superiorità valida per il sistema è quindi “r5 > r1”. Quando l'utente ha completato l'inserimento della teoria, Houdini processa e valida l'input ricevuto e procede al calcolo delle estensioni logiche $\pm\#_m$, $\# \in \{\Delta, \delta\}$, $m \in \{C, O, P\}$, mostrando gli insiemi ottenuti nella pagina dedicata.

4.3 Parsing

Per calcolare l'estensione logica di una teoria DDL, è essenziale avere una rappresentazione chiara e precisa della stessa. Di conseguenza, Houdini richiede che gli utenti forniscano dati conformi a una sintassi precisa che non possa generare alcuna ambiguità. In particolare, le teorie vengono validate per mezzo di una specifica grammatica context-free, che viene quindi utilizzata dal parser di Houdini (generato utilizzando i parse tree listener di ANTLR 4) per convalidare la teoria lato server (si veda Figura [II](#)). A controllare la validità dell'input contribuiscono in primo luogo i controlli lato client, volti sia ad evitare computazione non necessaria da parte del parser, sia a fornire all'utente un feedback immediato riguardo eventuali errori sintattici. Il modulo di parsing non solo valida l'input ricevuto, restituendo se necessario un messaggio di errore all'utente, ma costruisce inoltre le strutture dati necessarie al reasoner per il calcolo dell'estensione logica. Sia in questa fase di costruzione (che avviene incrementalmente durante la validazione dell'input), sia durante la verifica di correttezza vera e propria, il sistema adotta numerose procedure volte ad ottimizzare la computazione, quali il raggruppamento e la prioritizzazione delle espressioni più comuni (scelte in modo empirico). Inoltre, le espressioni matematiche (che possono comparire all'interno di una proposizione di testa della regola) e le espressioni logiche (che possono comparire all'interno di un vincolo logico del corpo della regola) vengono trasformate nelle rispettive versioni in Reverse Polish Notation (RPN), per velocizzarne le successive valutazioni con argomenti reali; le espressioni RPN garantiscono inoltre requisiti minimi di memoria.

4.4 Reasoner

Il modulo di reasoning implementa le funzionalità chiave di Houdini. Agendo immediatamente dopo a fase di parsing, calcola l'estensione logica completa della teoria ricevuta in input per mezzo di due sottomoduli distinti: Strict Reasoner e Defeasible Reasoner, il cui funzionamento è già stato approfondito in letteratura.

4.5 Strutture dati e ottimizzazione

L'algoritmo di reasoning agisce su istanze Literal (una per ogni letterale della teoria) e Rule (una per ogni regola) che sono state inizializzate dal parser e che contengono, insieme a un'istanza generale Theory che rappresenta la teoria, tutte le informazioni necessarie per l'esecuzione del reasoner. Le istanze Literal contengono, ad esempio, i riferimenti ai loro opposti, a tutte le regole in cui compaiono come teste e come elementi dei loro corpi, gli insiemi a cui sono stati assegnati (se lo sono stati), e la loro modalità deontica. Le istanze Rule, invece, tengono traccia ad esempio di tutti i letterali che compaiono nei loro corpi e nelle loro teste, della modalità deontica della testa, dei parametri, delle espressioni logiche e dell'ordine in cui compaiono nelle espressioni matematiche delle teste, ecc. Inoltre, tutte queste classi implementano metodi specifici che il reasoner utilizza per il calcolo degli insiemi risultato. La scelta di un così stretto legame tra dati e metodi che possono agire all'interno degli oggetti è dovuta alla necessità di utilizzare funzioni simili ma non identiche a seconda del tipo di letterale (normale o proposizione), della sua modalità, della presenza di espressioni matematiche nella testa della regola, ecc.

5 Risultati sperimentali

Il calcolo dell'estensione di una teoria defeasible proposizionale può essere fatto in tempo lineare (Maher 2001). Dopo diversi test sulla correttezza di Houdini, fatti controllando le sue conclusioni su un benchmark contenente diverse teorie, comprendenti molte sfaccettature di ragionamento e casi limite, testiamo le sue prestazioni confrontandole con quelle di SPINdle. SPINdle è un'implementazione allo stato dell'arte della logica defeasible che implementa diverse varianti di questo tipo di logica. Esperimenti recenti hanno dimostrato che SPINdle supera in termini di performance altre implementazioni di ragionamento defeasible (Batsakis et al. 2018; Hecham et al. 2018) ed è comparabile ad altri sistemi altamente ottimizzati di ragionamento non monotono (Robaldo et al. 2022, 2023). In più, SPINdle è stato utilizzato con successo in applicazioni basate sulla logica defeasible (in ambito legale) (Islam and Governatori 2018; Governatori 2015).

Abbiamo eseguito due esperimenti² su teorie create a mano e comparato i due ragionatori rispetto due statistiche temporali: il tempo totale in secondi, cioè l'intervallo di tempo che va dall'inizio del caricamento e la fase di parsing fino alla fine della fase di ragionamento (nel momento stesso in cui le conclusioni sono presentate all'utente) e il tempo di ragionamento in secondi, che include solo il secondo elemento. Qui non riportiamo dettagli sugli esperimenti per teorie defeasible proposizionali, che sono stati descritte in dettaglio in Cristani et al. (2022). In particolare, riportiamo qui solamente il tempo totale di computazione per il primo esperimento.

Compariamo i due ragionatori su teorie di diverse dimensioni (sia in termini di regole sia in termini di letterali); nel secondo esperimento fissiamo la dimensione delle teorie ma cambiamo, a caso, i letterali considerati fatti. Questi test sono stati fatti per individuare in modo isolato la performance di Houdini con insiemi di regole fisse, con l'obiettivo successivo di sviluppare un sistema deontico dove *regole* rappresentano il *background normativo* mentre i *fatti* gli eventi sui cui la legge legifera.

² Tutti i test sono stati eseguiti localmente su un computer con 8GB di RAM, un Intel i5 da 3.40 Ghz, e Ubuntu 18.04 LTS. I test su Houdini sono stati fatti girare con Java 11 mentre quelli di SPINdle (versione 2.2.4) sono stati fatti girare con Java 8. Abbiamo bypassato l'interfaccia utente e inserito direttamente le teorie (nel formato corretto) nei due ragionatori, salvando poi, in un file esterno, le conclusioni e i tempi.

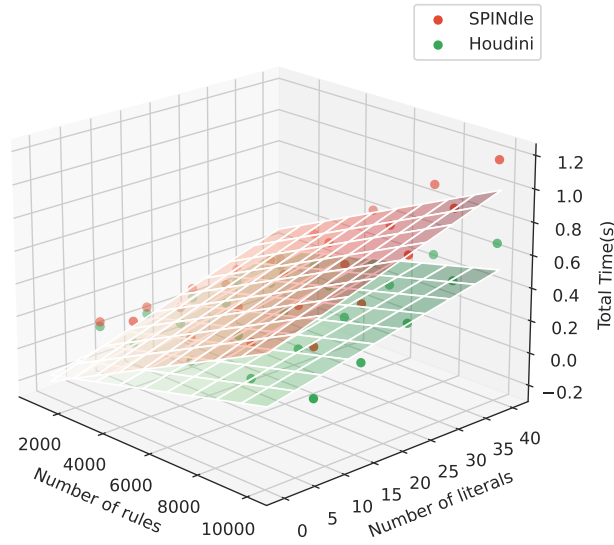


Fig. 4: Scatterplot dei tempi medi in secondi - tempo totale di SPINdle e Houdini con 25 iperparametri. I piani sono stati computati tramite classica regressione lineare.

Questa procedura genera una teoria stocastica che non è totalmente diversa da teorie che potrebbero derivare da situazioni realistiche. In particolare, le regole iniziali hanno più probabilità di essere attivate rispetto a quelle successive, poiché dipendono dallo stato di regole che le ‘precedono’ nell’attivazione, e sono meno.

La differenza principale tra teorie generate in questo modo risiede nei letterali X non derivabili che rompono ‘catene di derivazione’ che, altrimenti, comprenderebbero tutte le regole, dalla prima all’ultima, portando sempre alle stesse conclusioni.

Fissando $N = \{1000, 2500, 5000, 7500, 10000\}$, $W = \{8, 16, 24, 32, 40\}$, $pc = ps = 0.0025$ e $c = 200$, generiamo 20 teorie casuali per ogni parametro $(n, w) \in N \times W$; quindi, in totale, testiamo i due ragionatori su 500 teorie, 20 teorie per combinazione degli iperparametri (25).

Per quanto riguarda pc e ps , i loro valori sono (non troppo) piccoli, per evitare teorie con troppe o troppo poche conclusioni; in particolare vogliamo evitare il caso dove $+d$ è costituito solo dai primi w letterali b_0, \dots, b_{w-1} e il caso dove $+d$ include tutto, a parte X . Il primo caso si avrebbe con valori troppo alti, il secondo con valori troppo bassi.

I risultati sono indicati in Figura 4. Una tabella con tutti i dettagli è presente in (Cristani et al. 2022). Houdini supera in performance SPINdle in entrambi i casi.

Presentiamo qui i risultati del secondo esperimento, basato sul benchmark discusso in (Robaldo et al. 2023). Compariamo le performances di Houdini e SPINdle su 15 diverse teorie prese dal loro lavoro, e generate basandoci sulla combinazione di due iperparametri

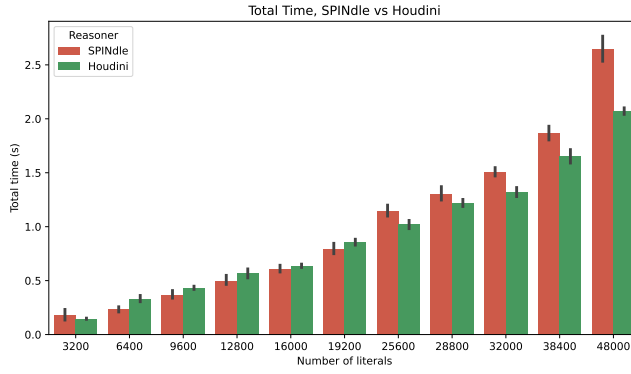


Fig. 5: Confronto tra SPINdle e Houdini. Tempo totale in base al numero di letterali della teoria. Le linee nere indicano gli intervalli di confidenza al 95%.

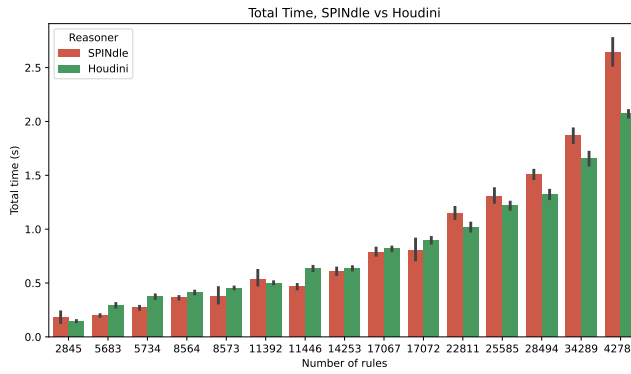


Fig. 6: Confronto tra SPINdle e Houdini. Tempo totale in base al numero di regole della teoria. Le linee nere indicano gli intervalli di confidenza al 95%.

(tre ‘use cases’ e cinque ‘sizes’). Per ogni teoria, entrambi i ragionatori vengono fatti girare cinque volte più due run poi scartate, per evitare il problema di una partenza ‘a freddo’ e outliers estremi. I risultati sono poi aggregati basandoci sul numero di regole e il numero di letterali della teoria Figure 5 e 6 riportano i risultati di questo esperimento. Usiamo, come misura di performance, il *tempo totale* in secondi: il tempo di caricamento (della teoria) più il tempo di ragionamento (che per SPINdle include anche il tempo di trasformazione della teoria). Tutti i test sono stati fatti girare localmente su un computer con 16GB di RAM, con un Intel i7 (12esima generazione) e Ubuntu 22.04.

I datasets creati da Robaldo et al. (2023) hanno lo scopo di rappresentare una teoria defeasible che tratta di un certo numero di casi, riguardanti individui e eventi. Data la natura proposizionale delle implementazioni di logica defeasible deontica, le regole sono istanziate basandosi su casi specifici, e tutti i casi (determinati dalla combinazione degli ‘use cases’ e la ‘size’) sono combinati in un dataset. Alternativamente (in versione *batch-mode*, trattata in Islam and Governatori (2018)), si può separare ogni singolo caso in un insieme di fatti e poi computare l’estensione di un caso basandosi sull’insieme dei fatti specifico di quel caso

Use Cases	Size	Reasoner		
		SPINdle	Houdini	Houdini (Batch)
10	10	0.184	0.146	0.053
20	30	0.795	0.855	0.274
30	50	2.646	2.071	0.706

Table 1: Tempo totale medio, in secondi, in base agli ‘use cases’ e alle ‘sizes’ (tre combinazioni del benchmark in [Robaldo et al. \(2023\)](#)), per SPINdle, Houdini e Houdini in batch-mode.

e l’insieme di regole originale. Per ovviare al problema di partenze ‘a freddo’ una teoria è caricata in memoria una sola volta, e poi clonata e riusata per ogni singolo caso.

Houdini supera SPINdle in termini di performances quando si considera il tempo totale in base al numero di letterali. Arriva ad ottimi risultati su grandi teorie, sebbene la differenza sia meno rilevante per teorie più piccole. Si ha un risultato simile quando si considerano statistiche di tempo in base al numero di regole. In più, gli esperimenti mostrano che l’uso della batch-mode risulta in un grande miglioramento delle performances.

6 Lavori simili

Negli ultimi trent’anni sono state eseguite molte indagini circa algoritmi per la logica defeasible proposizionale, iniziando con il lavoro seminale di Nute ([1994](#)), passando per indagini tecniche circa metodi algoritmici [Antoniou et al. \(2000\)](#), fino ad estensioni logiche del framework logico per includere operatori deontici da parte di Nute ([1998](#)). Come abbiamo indicato prima, SPINdle ha avuto molto successo e ha superato implementazioni precedenti. Dopo lo sviluppo di SPINdle, diverse alternative sono state proposte, con un focus su ragionamenti a larga scale con istanze. [Tachmazidis et al. \(2012\)](#) ha proposto una implementazione dell’algoritmo di SPINdle basata su un map-reduce parallelizzato, mentre [Maher et al. \(2020\)](#) propone una versione semplificata della logica per renderla più adatta alla parallelizzazione. [Rohaninezhad et al. \(2015\)](#) indaga circa un ‘grounder’ per SPINdle. Comunque, non è chiaro se questi approcci effettivamente ottengono un miglioramento di performances rispetto a SPINdle quando uniti a database relazioni e tecnologie di query ([Islam and Governatori 2018](#); [Liu et al. 2021](#)).

7 Conclusioni e lavori futuri

Abbiamo presentato un sistema che computa estensioni logiche strette/defeasible positive/negative di una teoria defeasible deontica. Questa tecnologia viene comparata contro SPINdle, un sistema già esistente. Le performances sono migliori, grazie ad un’accurata analisi delle limitazioni di SPINdle, incluse in Houdini. Abbiamo dimostrato che questo sistema supera versioni correnti e precedenti di altri sistemi per il reasoning defeasible deontico.

Questo lavoro presenta lo sviluppo di Houdini corrente, su cui stiamo lavorando per aggiungere altre funzionalità. Abbiamo menzionato alcune possibilità. Allo stato attuale, Houdini tratta regole proposizionali, regole deontiche (con catene di riparazione), batch-mode, uso tramite API, linguaggi di input formale, variabili numeriche e stringhe. Stiamo completando una versione online; nel frattempo, la versione corrente è disponibile per il download (con il codice sorgente) presso <https://github.com/edo-vi/Houdini>.

References

- Antoniou G, Billington D, Governatori G, Maher MJ, Rock A (2000) A family of defeasible reasoning logics and its implementation. In: ECAI 2000, pp 459–463
- Batsakis S, Baryannis G, Governatori G, Ilias T, Antoniou G (2018) Legal representation and reasoning in practice: A critical comparison. In: Palmirani M (ed) Jurix 2019, IOS Press, pp 31–40
- Cristani M, Olivieri F, Tomazzoli C (2016) Automatic synthesis of best practices for energy consumptions. In: IMIS, IEEE Computer Society, pp 154–161, URL <https://doi.org/10.1109/IMIS.2016.79>
- Cristani M, Olivieri F, Rotolo A (2017) Changes to temporary norms. In: ICAIL, ACM, pp 39–48, URL <https://doi.org/10.1145/3086512.3086517>
- Cristani M, Governatori G, Olivieri F, Pasetto L, Tubini F, Veronese C, Villa A, Zorzi E (2022) Houdini (unchained): An effective reasoner for defeasible logic. In: AI³@AI*IA, CEUR-WS.org, CEUR Workshop Proceedings, vol 3354
- Governatori G (2015) The Regorous approach to process compliance. In: 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop, IEEE Press, pp 33–40
- Governatori G, Olivieri F, Rotolo A, Scannapieco S (2013) Computing strong and weak permissions in defeasible logic. *J Philosophical Logic* 42(6):799–829, URL <http://dx.doi.org/10.1007/s10992-013-9295-1>
- Governatori G, Olivieri F, Scannapieco S, Rotolo A, Cristani M (2016) The rationale behind the concept of goal. *Theory Pract Log Program* 16(3):296–324, DOI 10.1017/S1471068416000053
- Governatori G, Olivieri F, Cristani M, Scannapieco S (2019) Revision of defeasible preferences. *International Journal of Approximate Reasoning* 104:205–230
- Governatori G, Rotolo A, Sartor G (2021) Logic and the law: Philosophical foundations, deontics, and defeasible reasoning. In: Gabbay DM, Horty J, Parent X, van der Meyden R, van der Torre L (eds) *Handbook of Deontic Logic and Normative Reasoning*, vol 2, College Publications, London, chap 9, pp 655–760
- Hecham A, Croitoru M, Bisquert P (2018) A first order logic benchmark for defeasible reasoning tool profiling. In: Benzmüller C, Ricca F, Parent X, Roman D (eds) *Rules and Reasoning, RuleML+RR*, Springer, LNCS 11092, pp 81–97
- Islam MB, Governatori G (2018) RuleRS: A rule-based architecture for decision support systems. *AI and Law* 26(4):315–344, URL <doi.org/10.1007/s10506-018-9218-0>
- Lam HP, Governatori G (2009) The making of SPINdle. In: Governatori G, Hall J, Paschke A (eds) *Rule Representation, Interchange and Reasoning on the Web*, Springer, no. 5858 in LNCS, pp 315–322
- Liu Q, Islam MB, Governatori G (2021) Towards an efficient rule-based framework for legal reasoning. *Knowledge Base Systems* 224(19):107082
- Maher MJ (2001) Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming* 1(6):691–711
- Maher MJ, Tachmazidis I, Antoniou G, Wade S, Cheng L (2020) Rethinking defeasible reasoning: A scalable approach. *Theory and Practice of Logic Programming* 20(4):552–586, URL [10.1017/S1471068420000010](https://doi.org/10.1017/S1471068420000010)
- Nute D (1994) Defeasible logic. In: *Handbook of Logic in Artificial Intelligence and Logic Programming*
- Nute D (1998) Norms, priorities, and defeasible logic. In: McNamara P, Prakken H (eds) *Norms, Logics and Information Systems*, IOS Press, Amsterdam, pp 201–218
- Olivieri F, Governatori G, Scannapieco S, Cristani M (2013) Compliant business process design by declarative specifications. In: Boella G, Elkind E, Savarimuthu BTR, Dignum F, Purvis MK (eds) *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, Springer, LNCS, vol 8291, pp 213–228, DOI 10.1007/978-3-642-44927-7, URL <http://dx.doi.org/10.1007/978-3-642-44927-7>
- Robaldo L, Batsakis S, Callegari R, Calimeri F, Fujita M, Governatori G, Morelli MC, Pisano G, Satoh K, Tachmazidis I (2022) Taking stock of available technologies for compliance checking on first-order knowledge. In: Callegari R, Ciatto G, Omicini A (eds) *CILC 2022: Italian Conference on Computational Logic*, CEUR 3204
- Robaldo L, Batsakis S, Callegari R, Calimeri F, Fujita M, Governatori G, Morelli MC, Pacenza F, Pisano G, Satoh K, Tachmazidis I, Zangari J (2023) Compliance checking on first-order knowledge with conflicting and compensatory norms: a comparison among currently available technologies. *Artificial Intelligence and Law* DOI 10.1007/s10506-023-09360-z
- Rohaninezhad M, Arif SM, Azman Mohd Noah S (2015) A grounder for spindle defeasible logic reasoner. *Expert Systems with Applications* 42(20):7098–7109, DOI 10.1016/j.eswa.2015.04.065
- Tachmazidis I, Antoniou G, Flouris G, Kotoulas S, McCluskey L (2012) Large-scale parallel stratified defeasible reasoning. In: Raedt LD, Bessière C, Dubois D, Doherty P, Frasconi P, Heintz F, Lucas PJF (eds) *ECAI 2012*, IOS Press, vol 242, pp 738–743, URL [10.3233/978-1-61499-098-7-738](https://doi.org/10.3233/978-1-61499-098-7-738)