ICOT Technical Memorandum: TM-1298

TM-1298

Workshop on Knowledge Representation for Legal Reasoning

by
K. Nitta, H. Tsuda & K. Yokota

May, 1994

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5

Institute for New Generation Computer Technology

Workshop
on
Knowledge Representation for Legal Reasoning

Friday, March 18, 1994
Northeastern University
Boston, U.S.A

Workshop Chairpersons: Associate Professor Carol Hafner
(Northeastern University)
Dr.Katsumi Nitta
(ICOT)

目次

1. はじめに 1
2. 経緯 1
3. 参加者 1
4. プログラム 2
5. 議事内容 4
6. 論文，アブストラクト 12
7. Stephen Wong のコメント 12
8. まとめ 13

［別紙］ 論文，アブストラクト集

``An Order Sorted Logic for Legal Reasoning Systems`` 16
Makoto Haraguchi
``A Knowledge Representation for Ethics Case Comparison and Some Ramifications for Legal Knowledge Representation.`` ... 19
Kavin D. Ashley
``Legal Reasoning on a Deductive Object-Oriented Database and its Extensions`` 20
Kazumasa Yokota
``Legal Knowledge Representation: A View from the Backbench`` 45
Devid B. Skalak
``A Formal Model of Legal Argumentation`` 50
Giovanni Sartor
``Knowledge Representation Language: Quixote`` 76
ICOT 2nd Lab.
``Quixote as a Toll for Natural Language Processing`` 81
Satoshi Tojo and Hiroshi Tsuda
``A Legal Reasoning System on a Deductive Object-Oriented Database`` 86
Chic Takahashi and Kazumasa Yokota
``Representation of viewpoint in New HELIC-II`` 95
Katsumi Nitta
``OWNERSHIP: A Case Study in the Representation of Legal Concepts`` 103
L. Thorne McCarty
``Building up a Legal Ontology from a General Ontology`` 104
Takahira Yamaguchi and Masaki Kurematsu
``Balancing Expressiveness and Tractability: Expectation-Driven Problem Formulation`` 110
L. Karl Branting
``A Legal Reasoning System based on Situation Theory.``
Satoshi Tojo and Stephen Wong 123

``Representation of Legal Knowledge 143
by Logic Flowchart and CPF``

1 はじめに

知識情報処理技術の法律分野への応用は1970年代の初期から試みられ、今まで多くの法的推論システムが開発されてきている。海外では、International Conference on AI and Law (ICAIL) という国際会議が、1987年から隔年で開催されて国際的な交流がなされている。日本では、明治学院大学の吉野一教授が、1984年からLES(Legal Expert System)プロジェクトを主催し、精力的な研究活動を行ってきた。このプロジェクトは、昨年から文部省の重点領域研究に採用され、法律家とコンピュータの研究者、約20名からなる大きなプロジェクトへと発展している。一方、ICOTにおいては、1991年ごろから並列応用システムとしてのHELIC-II, 知識表現言語Quixoteの応用としてのTRIALという2つの法的推論システムが開発されてきている。吉野教授のプロジェクトと、ICOTの2つの研究グループはタスクグループその他の交流を通じて密接な関係にある。特に、今年度は、法律の知識表現に関して熱心な検討をしてきた。しかしながら、国内の研究者だけでは情報が限られているため、検討内容が限定されてしまうという問題点があった。そこで、海外でワークショップを開催してそれぞれの法的推論システムで検討されている知識表現方式を比較し、知識表現言語に必要な機能を検討するための材料とすることにした。

2 経緯

以前から親交のあったKahl Branting教授(Wyoming大学)に、ワークショップの趣旨を伝え、呼びかけるべき研究者のリストを作成していただいた。このリストは、約20名のアメリカの研究者からなっている。このリストに従って、ワークショップへの興味の有無をemailで打診した。また、アメリカへの出張を予定していた津田氏(ICOT2研)東条氏(MRI)に依頼して、アメリカの主要な研究者を訪問して、参加を呼びかけていただいた。その結果、参加希望者が比較的多く集中しているBostonで、開催することを決定し、Northeastern大学のCarol Hafner教授に会場のアレンジを依頼した。

3 参加者

参加者の内訳を以下に示す(敬称略)

(1) 日本

(a) 重点領域研究関係

- 吉野 一(明治学院大学)
- 原口誠(東京工業大学)
- 山口高平(静岡大学)

(b) ICOT Helic-II 関係

- 新田克己(ICOT)

- 大崎宏 (JIPDEC)
- 東条敏 (MRI)
- Stephen Wong(UCSF)

(c) ICOT Quixote 関係

- 横田一正 (ICOT)
- 津田宏 (ICOT)
- 高橋千恵 (JIPDEC)

(d) ICOT 全体とりまとめ

- 兼子利夫 (ICOT)

(2) アメリカ

- Kahl Branting(Wyoming 大学)
- Kavin Ashley(Pittsburg 大学)
- Thorne McCarty(Rutges 大学)
- Lonald Loui(Washington 大学)
- Donald Berman(Northeastern 大学)
- David Skalak(Masachusetts 大学)

(3) イタリア

- Giovanni Sartor(Bologna 大学)

海外の参加者は、いずれも法的推論システムの開発で実績のある人ばかりである。特に Kahl Branting 氏、 Kavin Ashley 氏、 David Skalak 氏は法律家であると同時にコンピュータの専門家でもあり、この分野の第一線で活躍している。

4 プログラム

```
9:00 -  9:15    (small breakfast)
9:15 -  9:20   Opening
 (Chair: Stephen Wong)
```

9:20 - 9:50 Hajime Yoshino

''The Representation of Legal Knowledge in terms of Logical Flow Chart and Compound Predicate Formula''

9:50 - 10:20 Ronald Loui

''Representing modest rationales''

10:20 - 10:50 Makoto Haraguchi

''An Order Sorted Logic for Legal Reasoning Systems''

10:50 - 11:00 (break)

(Chair: Makoto Haraguchi)

11:00 - 11:30 Kavin D.Ashley

''A Knowledge Representation for Ethics Case Comparison and Some Ramifications for Legal Knowledge Representation.''

11:30 - 12:00 Kazumasa Yokota

''Legal Reasoning on a Deductive Object-Oriented Database and its Extensions''

12:00 - 13:30 (lunch)

(Chair: Takahira Yamaguchi)

13:30 - 14:00 Devid B.Skalak

''Legal Knowledge Representation: A View from the Backbench''

14:00 - 14:30 Carol Hafner

''Representing Legal Purposes in Normative Process Schemas''

14:30 - 15:00 Giovanni Sartor

''A Formal Model of Legal Argumentation''

15:00 - 15:15 demonstration of micro Quixote

(Chair: Karl Branting)

15:15 - 15:45 Katsumi Nitta

''Representation of viewpoint in New HELIC-II''

15:45 - 16:15 L.Thorne McCarty

''OWNERSHIP: A Case Study in the Representation of Legal Concepts''

16:15 - 16:45 Takahira Yamaguchi

''Building up a Legal Ontology from a General Ontology''

16:45 - 17:00 (break)

(Chair: Donald Berman)

17:00 - 17:30 L.Karl Branting

''Balancing Expressiveness and Tractability: Expectation-Driven

Problem Formulation''

17:30 - 18:00 Satoshi Tojo and Stephen Wong

''A Legal Reasoning System based on Situation Theory.''

18:30 - (dinner)

5 議事内容

以下にワークショップで発表された論文の概要, 質疑・所見を示す。

(1) The Representation of Legal Knowledge in terms of Logical Flow Chart and Compound Predicate Formula

(a) 発表者

Hajime Yoshino

(b) 概要

論理フローチャートと言語 CPF による法的知識の表現についての説明。CPF の開発動機は法的知識を表現するためであり、法律家によって使いやすく、かつ詳細な記述が可能で、しかも形式化された推論に使えるものでなければならないことによる。CPF は述語を多重に埋め込んだ表記をする。実行するときは全部同じレベルにフラットに展開される。ラムダ記法による変数束縛、クラスの記法、ひとつの述語に ID を与えてそれを他から言及できる機能がある。その他、シンタックスの紹介、応用例、セマンティクスに関する説明があった。

(c) 質疑・所見

Q:(Loui) ID シンボルの与え方について、変数を含むようなものをいつ固定するか?

A: 例えば変数 Z を含むようであれば、Z を固定した式といっしょに表記する。

Q:(Loui) フローチャートの and/or 表示について説明要請。

A: 補足説明あり。

(2) Representing modest rationales

(a) 発表者

Ronald Loui

(b) 概要

Rule は case とともに拡張される。その拡張のしかたについて、5 つの原理 (rationale) を設け、それらの原理によって議論がどのように動くかを解析する。例としては、「車を公園に止めてはいけない」という一般規則を設けている。この例では、救急車は止めてもよいから定義を私用車に限ったり、一般に静かにする必要のある保護地で駐車

禁止とする規則から公園を適用範囲に含めたりと、法の運用に関する諸相を原理という面からまとめ、それによる議論の推移をモデル化する。

(c) 質疑・所見

Q:(Hafner) rule + case → Argument という図式は循環するのが一般であるが？

A: そのような場合を考慮していることを後に示す。

Q:(Skalak) Arguments と rationale の関係？

A: Argument の帰結の変化をシーソーの絵を用いて説明。

Q:(McCarty,Ashley) シーソーの絵の中における rationale の位置は？

A: 絵において説明。

Q:(Berman) 5 つの rationale の価値、独立性、組み合わせについて問題提起。

法的議論のモデル化ということで、形式化にやや距離をおいた話である。その分、法がバックグラウンドである人たちにより興味を引き起こしたように感じた。

(3) An order sorted logic for legal reasoning systems

(a) 発表者

Makoto Haraguchi

(b) 概要

法的推論システムの枠組としてのソート論理による言語の提案。研究の動機としては、高階の推論が可能であり、用語分類の階層を容易に作ることができ、他の法的推論システムに翻訳可能である言語を実装することである。ソートの階層を設け、ソートのアブソープション (より高階の規則への一般化)、および event 型のソート 's' に対する関数 's _ f' による具体的な事象記述の方法について説明があった。

(c) 質疑・所見

特になし。

(4) A Knowledge Representation for Ethics Case Comparison and Some Ramifications for Legal Knowledge Representation

(a) 発表者

Kavin Ashley

(b) 概要

法的論争においては、倫理上の問題点もつきまとう。こうした倫理に関する論争も扱えるように、理由づけに関する階層構造を利用したシステム Truth Teller に関する概説が述べられた。Truth Teller では、従来の HYPO の dimension 概念が、理由や

倫理上の原理 (principle) などへ拡張されており、 dimension に関する similarity と dissimilarity の管理・推論が高次なものとなっている。

(c) 質疑・所見

HYPO における対象は、 trade-secrete 法という客観的な法的要素 (factor) の切り出しが容易である分野であった。一方、 Truth Teller においては倫理という、記号論的切り出しが困難なこともあり、どれほど一般性のあるシステムが構築できるかは疑問である。しかしながら、倫理や倫理性を有する論拠に関する学説等を参考にしてある程度 HYPO の拡張システムを構築することは可能であるだろう。むろん、ここでも法的概念語や倫理的概念語を注意深く切り出す問題をクリヤーしなければならない。これは工学の問題というより、純然として倫理学もしくは法律学の課題であり、理工学の立場にたつ報告者としては、倫理学者と法律学者のより詳細な分析を期待したいところである。

(5) Legal Reasoning on a Deductive Object-Oriented Database and its Extension

(a) 発表者

Kazumasa Yokota

(b) 概要前の小節までの発表が、個体の identity を想定した一階論理であったのに対し、クラス表現に制約をつけた式で、知識表現の対象物を記述する知識表現言語 Quixote の研究の歴史と制限版 Quixote のデモンストレーションがあった。これも 事物の間の *is_a_subclass_of* 関係を表現および推論の基本とする言語であるが、前節のソート概念と異なる点は、ソートに制約を付加した形の拡張項によりクラスとして表現する点にある。処理系は基本的には、拡張項の単一化と制約処理系からなり、制約処理系は経験則で前向き推論を制御する方式を採用している。

さらに、標準的な abductive logic programming の考え方を「拡張項の単一化＋制約処理系」に組み込み、仮説付きの解の生成を行うことにも成功している。すなわち、『Aか？』という質問に対し、『もしもBならばAですよ』なる形式の解を生成する。これは法律の条文に対して十分な解釈知識が知識ベースに与えられていない場合に有効に機能すると考えられる。

また法律では、事実の世界と法的概念の世界を区別して考察したり、あるいは逆に同一視して考察する場合がありえる。たとえば国際統一売買法において、『申入』はある要件を満たせば『申込』となる。ここで、事実の世界では、『申込』と『申入』は同一の行為を指示しているが、法的概念としては明確な違いがある。講演では Quixote が 持つ module 機能によって、こうした概念世界と事実世界を独立した単位として区別することにより、上記のような問題点をうまく扱えるとしている。

(c) 質疑・所見

Q: 一階論理による equality axiom をうまくつかえば、上記の問題点は一階論理によっても扱えるのではないのか?

A: これは申込と申入のインスタンスが”同じ”項になってしまうので、以後、申入と申込の区別を述語により行うことが不可能になってしまう。したがって、一階論理を用いる限り、『申し入れ』のインスタンスが述語『申込』を満たす、との形式で推論を行うことになると考えられる。さらに言えば、『申し入れ』と『申込』は、外延的には同一だが、内包的には異なる点に着目すれば、いわゆる”内包論理”によっても上記の問題は扱えるはずであり、一階の内包論理を現在の論理の拡張として実装すれば、事実の世界と法的概念世界との整合性、同一性、差異性の問題は十分扱えるはずである。

(6) Legal Knowledge Representation: A View from the Backbench

(a) 発表者

Devid Skalak

(b) 概要

今までに開発された３つの法律ＥＳ（HYPO (Asley,1990)，CABARET，BankXX）を通して，法的知識表現の目標について，第１段階は，判例と法令の知識表現を考察する事であり，第２段階は，法的理論，事案の典型性（factual prototypes），法的論議の知識表現を考察する事であると指摘した．法的理論については，法的理論を記述するための factor 群や法的理論間の関連を表現すると共に，法的理論の修正機構の重要性を指摘し，事案の表現については，典型的な factor 群を例示した．また，法的論議の知識表現が今後の重要課題であり，法的論議の構成要素として，論議を支持する最良事例，反対論議を支持する最良事例など１２項目を整理すると共に，法的論議を評価する機構が今後大きな問題になる事を指摘した．

(c) 質疑・所見

質疑応答では，法的論議を評価するレベルの問題が取り上げられ，法令レベルの評価，判例レベルの評価等が考えられる等の議論が交わされた．休憩時間中に Skalak と意見交換をしたが，法的論議評価機構のように，コンテキストにより概念の評価が変わるという問題は，機械学習・知識獲得において大きな課題であるが，まだ決め手になる理論・技術はないのが現状である．彼は，この問題は一般的には concept drift（概念の意味が揺れ動く？）の問題であると指摘し（広く認められた用語ではないが），ＡＩにおける重要な方向を示唆していると感じた．

(7) Representing Legal Purposes in Normative Process Schemas

(a) 発表者

Carol Hafner

(b) 概要

事案に法令ルールが適用されて法的推論が実行されるが，ルールの目的(purpose)を陽に表現して，因果関係を深く捉える法的推論の必要性を指摘した．具体的には，定性推論の一表現形式である，定性プロセス理論によって，法的規範適用プロセスの表現を検討している．定性プロセス理論は，対象の生成・消滅や性質の変化を引き起こす原因をプロセスと考え，プロセスに関連する対象，プロセス成立のための前提条件，プロセスにおける変量間の関係，プロセスが変量に与える影響，によってプロセスを記述する．現在，この定性プロセス理論により，簡単な例題レベルで法的規範の表現を試みている．

(c) 質疑・所見

質疑応答では，プロセス成立のための前提条件は論理表現に変換できる等，定性プロセス理論の表現形式そのものに関心が集まった．定性プロセス理論は，工学分野で広く応用されているものの，それだけで利用可能な範囲は限られており，量的な推論との融合などが議論されている．それに対し，法律分野は，工学分野と比較すれば情報の粒度が粗くて済むため，適用可能性は高いかもしれないと思った．但し，法律問題はより開いた問題であるため，正確なプロセス記述やプロセス収集の困難性が，工学分野以上に大きな課題になると感じた．

(8) RA logic programming model of legal argumentation

(a) 発表者

Giovanni Sartor

(b) 概要　論理プログラミングによる法的論議モデルを考察し，derivability，subargument，opposing argument and counterargument，comparison of arguments，relative strength of arguments，norms as inference rules，applicability and undercutting argumentes，exception，non-monotonic reasoning in law，interpretation arguments，prefernce arguments の観点から，法的論議モデルを構成する様々な概念を整理し，その定義を与えた．

(c) 質疑・所見

ＡＩ全体に通じる課題に関する議論が少なかったが，基本的には，よく整理された発表内容であった．

(9) demonstration of micro Quixote

(a) 発表者

Hiroshi Tsuda, Chie Takahashi

(b) 概要

Macintosh 版の micro-Quixote を用い、知識表現言語 Quixote の説明およびデモンストレーションを行なった。 Quixote には

```
1: object identity
2: subsumption constraint
3: property inheritance
4: module and rule inheritance
5: answer with asumptions and conditional query
```

といった特徴がある。日本の国技である相撲の例題を通して、それぞれについて簡単な説明を行なった。 5 の question と answer については、 micro-Quixote を実際に動かして説明をした。

(c) 質疑・所見

macintosh を利用している人が多いようで、 mac 版の開発環境に関する質問があった。 system7 + Think C Ver.6 である。

(10) Representing personal standpoint in new HELIC-II

(a) 発表者

Katsumi Nitta

(b) 概要

ICOT で研究開発している新 HELIC-II とその言語について、期待される機能を述べた。

(c) 質疑・所見

Q: HELIC-II の意味は?

A: 日本語の発音で HELIC-II は「へりくつ」とよみ、自分の都合のいいように論理を構築するという意味である。

Q:2 つの否定の使い分け方は?

A:HELIC-II では 2 つの否定を扱うことができる。 1 つは classical nagation であり、もう一つは nagation as failure である。

nagation as failure は証明が失敗したことによる否定をあらわし、例外などを表現する時に使い、classical nagation ははっきりした否定の事実がある時に使う。

(11) OWNERSHIP: A Case Study in the Representation of Legal Concepts

(a) 発表者

Thorne McCarty

(b) 概要

computational jurisprudence の 1 つとして ownership の概念に焦点を絞り、分析を行なった。とくに LLD(Language for Discourse) で Hohfeld での概念を形式化し分析した。例えば、Hohfeldism boundle of rights は容易で paticular objects and actions は困難であることを示した。

(c) 質疑・所見

Q: Hohfeld の形式の変数は?

A: メタ変数。

英米法の所有格の概念の変換分析は英語自体の微妙なニュアンスの違い, 法律独特な使用法があり、分かりづらかった

(12) Building up a Hybrid Dictionary as an Ontology for Legal Reasoning

(a) 発表者

Takahira Yamaguchi

(b) 概要

一般のオントロジー (EDR 辞書など) から法律用のオントロジーを構築する方法を述べた。

(c) 質疑・所見

Q: British Columbia のプロジェクトでもオントロジー構築をしていて旨くいっていないが、同じような問題はないのか?

A:British Columbia のプロジェクトは知らないがオントロジー構築の困難さは良く承知しているつもりであり、様々な文献を参考にして構築を進めている。

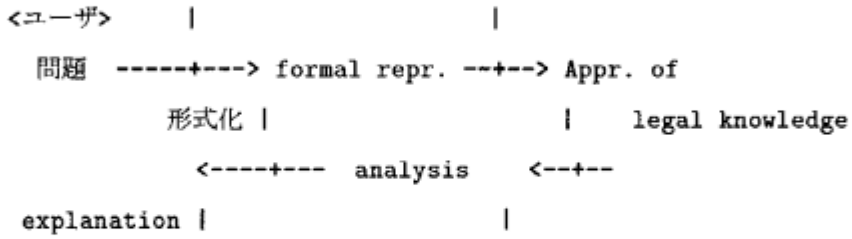
(13) Balancing Expressiveness and Tractability: Expectation-Driven Problem Formulation

(a) 発表者

Karl Branting

(b) 概要

法律エキスパートシステムは以下のような構成で作成される。



ユーザの問題を形式的に表現する枠組が適切であるためには、以下の要件を満たしていなければならない。

extensibility：行為の任意の sequence が記述できる

specificty：法律的に関連するあるあらゆる側面でケースを記述できる

ところがここで

表現力が高い --> case matching

case indexing のコストが高くなる

というジレンマが生じてしまう。

本発表で提唱される EDPF (Expectation-driven problem formulation) というシステムは、表現力と処理の難易度のバランスを取った枠組である。その基本的アイディアは、いくつかの古い case に基づいて、新しい case をモデル化するということである。

(c) 質疑・所見

case の検索は、実際には対話的に少しずつ情報を付け加えることで行なっている。また、検索に当たってはシステムの枠組だけでなく、ユーザの sophistication もある程度必要であろう。

表現と計算 (処理) のバランスは、知識表現を考える上で最も基本的な問題である。言語を作るというのは、そのジレンマに対処し、それなりの答を提示してすることに他ならない。話を法律のケース表現に絞ることで、このジレンマをなるべく明解に分析しようという姿勢は興味深かった。ただ、EDPF がどれだけバランスが取れているのか、もう少し客観的な評価はできないものだろうか。

(14) A Legal Reasoning System based on Situation Theory

(a) 発表者

Satoshi Tojo

(b) 概要

自然言語の意味論から出発した状況理論を法的推論システムに応用した situation-theoretic model (SM) を提唱し、知識表現言語 Quixote によるその実装について述べた。

SM はルールベースの法的推論のモデルであり、状況理論におけるインフォン、状況、およびそれらのサポート関係といった概念を利用している。さらに、新しい判例と古い判例との比較をするために、インフォンおよび状況のマッチの拡張を考えている。つまり、インフォンについては exact, partial, weak マッチを、状況については exact および partial (これはさらに閾値を与えることで拡張される) マッチがある。

知識表現言語 Quixote を利用することで、SM は自然に記述することができる。Quixote におけるモジュール、属性項といった概念は、状況理論の状況、インフォンに対応付けることができる。また、Quixote の仮説生成 (アブダクション) は、状況推論をベースにした法的推論に適している。

(c) 質疑・所見

PCFG (probabilistic context-free grammar) や Hobbs, Stickel による cost-based abduction に見られるように、論理プログラムの枠組に確率・統計情報を入れて拡張するという手法は最近の自然言語処理のテーマの一つである。本発表で述べられた、インフォンおよび状況のマッチングを拡張することで、推論をコントロールするというアイディアはこの観点からも非常に面白い。実装に関しては、Quixote 単体では限界があるため、他の知識処理プログラムと組み合わせて行なうことが考えられる。これは、現在 ICOT で提案されている異種分散協調問題解決系の応用になるかも知れない。

6 論文, アブストラクト

別紙に提出された論文や OHP のコピー及びアブストラクトを示す。提出された論文のうちで、この報告書に掲載することによって、著作権の問題が生じる恐れのあるものについては、アブストラクトのみを掲載することとした。なお、これらの論文の掲載にあたっては、著者の了解を得ている。

7 Stephen Wong のコメント

これまでの報告は日本側からのコメントであったので、海外からの参加者である Stephen Wong 氏のコメントを以下に示す。

All the participants of this workshop are the leading AIL researchers from both Japan and United States, as well as Italy. The workshop was divided into four sessions and operated under a tight schedule. I chaired the first session in which three distinguished researchers, Professor Hajime Yoshino from Meijigakuin University, Professor Ronald Loui from Washington University, and Professor Makoto Haraguchi from Tokyo Institute of Technology spoke about three different

schemes of representing legal knowledge in AI programs. Through the implementations of these schemes differ, they share a common thread - strongly based on the logic programming paradigm.

My observations of this workshop are briefly presented in the following. First, the ICOT researchers presented good computational models and solid implementation tools for legal reasoning. Dr. Nitta discussed the new development of HELIC-II, which includes a three-level representation of legal knowledge and a computational model for interactive legal debate between the computer and the user. Dr. Giovanni Sartor from Bologna University pointed out the possibility of merging one of the negation techniques into the defeasible reasoning method of the new HELIC-II. Mr. Kazumasa Yokota, the chief researcher of ICOT, described one of the legal reasoning systems, TRIAL, developed using the database programming language, Quixote. Miss Chie Takahashi and Mr. Hiroshi Tsuda demonstrated the downsized version of Quixote, MicroQuixote on a MacIntosh computer. They made note that this package is available as a free software. Mr. Satoshi Tojo and myself introduced a situation-theoretic model for legal knowledge representation and discussed how this model can be mapped into the Quixote language for reasoning about a variety of legal cases and situations.

On the other hand, the presentations of most other researchers were on the abstract modeling of legal knowledge. For example, Professor Carol Hafner of Northeastern University presented an approach of presenting legal purposes using a certain cased-based technique. Professor Thorne McCarty talked about the notion of legal ownership that troubles him for twenty years. Professor Karl Branting proposed a novel indexing scheme for formulate cases in AIL programs. Quite interestingly, Dr. Sartor has given over thirty-three rigorous definitions on specifying legal reasoning programs.

The atmosphere of this workshop encouraged steady exchange of ideas. The projects discussed here also showed the innovative use of logic programming tools to attack difficult AIL problems, although many computer programs are yet to be developed. My major complaint is the tight schedule. There were many good questions to be raised but often had to cut short due to the time constraint. It would be interesting to see the progress of many prototypes presented in this workshop in the near future. Finally, I consider that this workshop is a success.

8 まとめ

(1) 全体について

スケジュールがきついため、進行が若干遅れ気味であったが大体順調に進行した。これだけの発表があるなら、２日間の日程にすべきであるという声があった。企画当初はもっと小規模で開催し、夕方は総合討論の予定であったが、アメリカの重要な研究者が予想以上に参加してくれたため、内容は充実した反面スケジュールはきつくなった。法的推論に関するワークショッ

プはアメリカではほとんど開かれていなかったので今回のワークショップはアメリカの研究者同志の交流にも役立ったとのことである。

(2) 内容について

日本とアメリカの研究者の発表には著しい差があった。それは、日本側の発表が比較的整理された知識をどのように (how) 表現するかを中心に発表し、形式的表現を重要視したのに対し、アメリカ側の発表は、何を (what) 表現するかを中心に発表したからである。これは、法的推論システムの開発経験の差によって、知識内容の分析の差によるところが大きい。しかし、それ以外に、日本側は論理に基づいた形式的な記述を非常に重視するのに対して、アメリカ側は形式化はあまり重要していないように見える。アメリカ側から出されたさまざまな知識を、日本側の提案した言語でどのように表現するのかを検討することはその言語の表現能力を評価するのに大変意義があることである。

(3) その他

ワークショップは討議も活発になされ、大成功であった。2年に1度の国際会議の間の年に開かれたというタイミングの良さと、重要な研究者がそろったということが成功の原因である。このワークショップによって知識表現言語に必要な機能をリストアップすることが可能になっただけでなく、日本の研究活動をアメリカにアピールすることができた。また、Quixote のデモを行うことによって、ユーザの拡大の可能性を大いに高めることができた。

[別紙]

論文, アブストラクト集

An Order Sorted Logic for Legal Reasoning Systems

Makoto Haraguchi

Department of Systems Science, Tokyo Institute of Technology

4259 Nagatsuta-cho, Midoriku, Yokohama 227

A large conceptual taxonomic hierarchy is needed to describing legal domains. Under such a taxonomy, legal rules as well as legal theory rules can be described as formal rules accessing the taxonomic hierarchy. According to the representation method of legal knowledge in terms of Horn clauses, taxonomy is represented by a set of rules of the form

$$\forall x.\ \ s_1(x) \rightarrow s_2(x),$$

meaning that s_1 is a subclass of s_2. In other words, so called *isa* relation is also defined as a logical rule. However, some researchers have already pointed that it is unnatural to have *isa* relations in the forms of logical rules. For instance, suppose we have another rule meaning that s_2 is a subclass of s_3. In order to conclude the fact that s_1 is also a subclass of s_3, we apply inference rules two times. However that fact is a direct consequence of the transitive law for partial ordering, provided we have the taxonomic knowledge in the form of partially ordered set. Such an ordered set is sometimes called a sort hierarchy([5],[6]).

Sort Hierarchy: A sort hierarchy is a partially ordered set $(S, \leq)$, where S is a set of sort symbols.

Each non-logical symbols is supposed to have a corresponding sortal specification. In fact, a function symbol f has its domain sorts $s_1, ..., s_n$ and codomain sort s. Similarly, a predicate symbol has its domain sorts $s_1, ..., s_m$. Moreover any variable has its sort s. The instantiation for the variable is restricted to terms of the same sort s.

Given a sort hierarchy $(S, \leq)$, a unification that takes the sortal information into account is called an order-sorted unification. Formally a unifier is called an order-sorted unifier if it maps variables to terms of more specific sorts.

Then the generalization based on the sort hierarchy can be described as a sortal absorption defined as follows:

Sortal Absorption Suppose we have a rule

$\forall x : s_1.\ W(x) \rightarrow A(x)$

and the sortal subsumption $s_1 \leq s_2$. The sortal absorption replaces $x : s_1$ with a variable $y : s_2$ of more general sort s_2. As a result, we have $\forall y : s_2.\ W(y) \rightarrow A(y)$. In order to apply the sortal absorption to our legal knowledge, our representation of legal rules in the form of Horn-clauses should be modified.

Sort of Event Type: We firstly suppose that some sort symbol s has corresponding functor s_f. s is called an sort of event type. For instance, a functor $contract\_f : person, person, object \rightarrow contract$ is associated with a sort *contract* which we declare as a sort of event type. Intuitively speaking, a term $contract\_f(X, Y, Z)$ means a an instance of *contract* whose arguments are X of *person*, Y of *person* and Z of *object*, respectively. In order to describe factual information, we present a special predicate $oc :$ *event*, where we assume that any sort symbol s of event type is a subsort of *event*. For instance, a fact that a person a has contracted with a person b for an object c can be written as $oc(contrac\_f(a, b, c))$. Moreover, in order to get "attributes" from the term of event type *contract* , a predicate *describe* is used. The predicate is assumed to have the following unit clause as a definition clause.

$$describe(contract\_f(X : person, Y : person, Z : object), [X, Y, Z])$$

Canonical Representation of Legal Rules: For any legal rule whose head contains a compound term $s\_f(t_1, ..., t_n)$ of event type s, the term is replaced with a new variable x of s. Moreover, $describe(x, [t_1, ..., t_n])$ is added to the body of rule.

According to the canonical representation, the Clause2 of Article94 is represented as folllows:

$$\begin{aligned}
&oc(Contract1 : contract)\\
&describe(Contract1, [X : person,\\
&\quad Y : person, Obj : object]),\\
&oc(Falsity : falsity,)\\
&describe(Falsity, [Contract1]),\\
&oc(Contract2 : contract),\\
&describe(Contract2, [Y,\\
&\quad Z : person, Obj]),\\
&good_faith(Z, Falsity),\\
&\rightarrow cannot_set_up(X, Y, Contract2)
\end{aligned}$$

Now we illustrate how our sortal absorption as well as order-sorted unification is useful in finding an appropriate generalization. For this puropose, suppose we have an initial goal

$$\leftarrow \quad cannot\_set\_up(a, c, registration\_f(b, c, imm\_X)).$$

Since $registration\_f(b, c, imm\_X)$ is a term of sort *registration* which is not a subsort of *contract*, the head of the rule and the goal cannot be unified by order-sorlted unification. This unification failure, however, presents us a useful information about possible generalizations. In fact, as long as a generalization s of sort *contract* is applied to the specific sort *registration*, s should be a super sort of both *contract* and *registration*. Hence the sortal absorption should be tried for $lawful\_act = lub\{contract, registration\}$. Generally a minimally specific super sort of *contrlact*, *registration* can be a candidate for our generalization.

The unification failure of unsorted terms simply implies that we need a generalization. Unlike unsorted case, the unification failure of sorted terms shows a possible direction to generalize sorts. The operation needed to decide the direction is the algebraic operation to find a minimal upper bound of a set of sorts. Thus our sorted absorption will save the computational resources for a large and complex knowledge base for legal domains.

References

[1] M. Haraguchi *A form of analogy as an abductive inference*, In *Proc. 2nd Workshop on Algorithmic Learning Theory*, pages 266-274, Japanese Society for Artificial Intelligence, 1991.

[2] M. Haraguchi *What kinds of knowledge and inferences are needed to realize legal reasoning?* (in Japanese), *Proc. 6th symposium on knowledge representation and legal reasoning system*, Legal Expert System Association in Japan, 1992.

[3] S. Muggleton. and W. Buntine. *Machine invention of first-order predicates by inverting resolution*, In *Proc. Workshop on Machine Learning*, pages 339–352, 1988.

[4] S. Muggleton. *Inductive logic programming*, in *Proc. 1st Workshop of Algorithmic Learning Theory*, pages 42–66, 1990.

[5] C.Walther: Many sorted unification, JACM 35, 1, 1–17 , 1988.

[6] C.Beierle et.al.: An order-sorted logic for knowledge representation systems, Artif. Intell., 55, 149–191 , 1992.

A Knowledge Representation for Ethics Case Comparison and Some Ramifications for Legal Knowledge Representation

Kevin D. Ashley
University of Pittsburgh
Intelligent Systems Program,
School of Law, and Learning Research and Development Center
Pittsburgh, Pennsylvania 15260
ashley@vms.cis.pitt.edu, (412) 648-1495, 624-7496

Abstract

TRUTH-TELLER, a program for testing a case-based comparative evaluation model in practical ethics, compares cases presenting ethical dilemmas about whether to tell the truth. Its comparisons list ethically relevant similarities and differences (i.e., reasons for telling or not telling the truth which apply to both cases, and reasons which apply more strongly in one case than another or which apply only to one case). The program reasons about reasons in generating context-sensitive comparisons; it infers new reasons, qualifies existing reasons and considers reasons in the aggregate. The reasons may invoke ethical principles or selfish considerations.

We describe a knowledge representation for this practical ethical domain including representations for reasons and principles, truth telling episodes, contextually important scenarios, and comparison rules. In a preliminary evaluation, a professional ethicist scored the program's output for randomly-selected pairs of cases. The empirical evaluation confirmed that TRUTH-TELLER made comparisons robustly and with some degree of context sensitivity.

The work contributes to AI CBR / AI and Law efforts to integrate general principles and context-sensitive information in symbolically assessing case similarity. The work points the way to model symbolic comparisons of problems to paradigmatic cases in order to address conflicts of principles. It attempts to integrate lower level factual dimensions with higher level dimensions and principles through a variety of techniques such as hierarchical representations of some dimensions and hierarchical relationships among dimensions. It also furthers research on cognitive and philosophical models of ethical judgement and decision-making.

---

*This work is supported the The Andrew W. Mellon Foundation. This presentation reports on work performed by Bruce McLaren, a graduate student in the Intelligent Systems Program. We are grateful to Athena Beldecos, a graduate student in the University of Pittsburgh History and Philosophy of Science Department, for her work in researching casuistic models in the philosophical literature and her participation in protocols for case comparison. We thank Vincent Aleven for his helpful comments about evaluating the program. We are also grateful to Ken Schaffner, University Professor of Medical Humanities, George Washington University, for participating in our preliminary evaluation.

Legal Reasoning on a Deductive Object-Oriented Databases and its Extensions

Kazumasa Yokota

Institute for New Generation Computer Technology (ICOT)

e-mail: kyokota@icot.or.jp

Abstract

A legal reasoning system is a large-scale knowledge information processing system into which many technologies such as artificial intelligence, natural language processing, and databases are integrated. From a database point of view, this application has many kinds of data and knowledge, and provides many research topics for next generation databases: features of very large databases and knowledge-bases, their classification, treatment of partial information, query processing containing high-level reasoning, and so on. Further, it suggests ideas for the boundaries between databases and applications. In this paper, first, I explain our experimental legal reasoning system that is based on the deductive object-oriented database system *QUIXOTE*, and, secondly, I propose a framework of heterogeneous, distributed, cooperati· ·lem solvers *Helios* as an extension of legal reasoning systems.

1 Introduction

Recently, legal reasoning has attracted much attention from researchers in the field of artificial intelligence, with great expectations for its *big* application. Legal reasoning systems are very important applications, whose development, like that of theorem provers, dates back to before artificial intelligence was proposed, (for example, see [5]). In fact, laws are related not only to the judicial world but also to all social activities. To support legal interpretation and reasoning in a wide range of situations, many systems have been developed, including those capable of planning tax-saving strategies, negotiation of payment of damages, making contract documents, predicting judgements and supporting legislation. Many works on expert systems for such applications have been published, while powerful legal database systems have not yet been reported.

In the Japanese FGCS (Fifth Generation Computer System) project, legal reasoning systems were considered quite critical and a prototype legal reasoning system TRIAL was developed [8, 7, 12].

For the above systems, we provide database and knowledge-base management facilities: *Quixote*[11] and Kappa-P [12, 4]. *Quixote* is a deductive object-oriented database (DOOD) [6, 3, 2]language and knowledge representation language, used for describing and classifying complex legal data and knowledge, while Kappa-P is a parallel nested relational database management system, used to store large volumes of legal data. Especially, all data and knowledge in TRIAL is written in *Quixote* and some advanced query processing facilities, such as hypothetical reasoning and hypothesis generation (abductive reasoning), are provided for legal reasoning by *Quixote*.

In the experience, we found sevaral features indispensable for future generation database systems. For the purposes, we are developing a system *Helios* [10] for heterogeneous, distributed, cooperative problem solvers, which is also an extension of legal reasoning systems.

In this paper, first, we report on the experimental system, TRIAL, and illustrate the effectiveness of the advanced features of the DOOD system. Secondly, based on our practical experience, we discuss the roles we expect databases to play in legal applications, and the features that should be provided for next generation databases. Then, we propose a framework of *Helios* as new legal reasoning systems. In Section 2, we briefly explain some of the features of *Quixote*, based on an example of legal reasoning and show the effectiveness of a deductive object-oriented database in knowledge information processing applications. In Section 3, we discuss the features demanded for next generation databases, especially derived from knowledge information processing, and describe a framework of heterogeneous, distributed, cooperative problem solvers for more powerfully processing knowledge information. Lastly, we summarize the discussions.

2 Outline of a Deductive Object-Oriented Database Language *Quixote*

2.1 Example

To concretely illustrate the types of problems we are studying, we take the following new case related to "*karōshi*" (death from overwork):

> Mary, a driver employed by company 'S,' died from a heart attack while taking a break between jobs. Can this case be applied to the worker's compensation law?

We will first give a brief summary of the legal reasoning process we adopted. Next, we will show part of the legal knowledge necessary for this example, and the desirable interaction sequence between user and knowledge-base management system to deal with the example. And last, we will present some requirements of knowledge representation languages obtained from this example.

2.1.1 Legal Reasoning Process

We decompose the analytical legal reasoning process into three steps: *fact finding*, *statutory interpretation*, and *statutory application*. Although fact finding is very important, it is beyond the capabilities of current technologies. So, we assume new cases to already be represented in a form appropriate for our system. Statutory interpretation is a particularly interesting theme from an artificial intelligence point of view. TRIAL focuses on both statutory interpretation and statutory application.

Among the many approaches to statutory interpretation, we decide to use the following procedure:

1. *Analogy detection*

   Given a new case, precedents with similarities to the case are retrieved from an existing precedent database.

2. *Rule abstraction*

   Precedents (interpretation rules), extracted by analogy detection, are abstracted until the new case can be applied to them.

3. *Deductive reasoning*

   Apply the new case, in a deductive manner, to the abstract interpretation rules transformed in the previous step. This step may include statutory application because it is used in the same manner.

Among these steps, the analogy detection strategy is essential in legal reasoning for *more efficient* detection of *better* precedents. Analogy detection ultimately determines the quality of the result. As the primary objective of TRIAL is to investigate the possibilities of *Quixote* in this area and develop a prototype system, we limit the scope of our present study. That is, we investigate the extent that interpretation rules should be abstracted for a new case, to get plausible answers. We do not attempt to devise a general abstraction mechanism.

2.1.2 Example Knowledge-Base

In this example, we use a statute and a theory for its application:

- *labor law*: An organization is responsible to employee compensation, if the case judgment is for 'insurance.'
- *theory*: If the case judgment is for both 'job-causality' and 'job-execution', then the case judgment is for 'insurance.'

Assume that there are two precedents related to the law which have already been abstracted as follows:

- *precedent 1 (job-execution)*: If an *employee* has a *relation* of employment, and this relationship causes the *case*, then the judgment considers the case part of 'job-execution.'
- *precedent 2 (job-causality)*: In the *case*, if the *disease* related incident occurred within the *job*'s period, then the judgment considers the case 'job-causal.'

Note that these statements are abstracted from certain concrete precedents in the rule abstraction step by abstracting concrete concepts (e.g., a case name, or a person name) into abstract concepts (shown in italics): *employee, relation, case, disease,* and *job*. We will show our implementation of rule abstraction in section 2.4.1.

Finally, for the above knowledge-base, we consider queries and expected answers.

- *query 1*: According to past precedents and theory, what kind of judgment can we predict for the new case?
- *answer 1*: If in the new case, Mary's activities are work related and they are the cause of the new case, the judgment is for 'insurance.'
- *query 2*: According to labor law, what responsibility does Mary's company have?
- *answer 2*: If in the new case, Mary's activities are work related and they are the cause of the new case, company 'S' is responsible for compensation.

2.2 Outline of *QUIXOTE*

From a database point of view, *QUIXOTE* is a DOOD language, while, from a logic programming point of view, it is an extended constraint logic programming language based on subsumption constraints. In this section, we explain some of its features, used in the above example. For details on *QUIXOTE*, see[9, 11, 12].

2.2.1 Object Identity and Subsumption Relation

Simple concepts can be represented as *basic objects* in QUIXOTE. For example, the following are basic objects:

$$mary,\ driver,\ employee,\ male,\ female,\ person.$$

Basic objects are partially ordered by the *subsumption relation* '$\sqsubseteq$'. For example,

$$mary \sqsubseteq driver, \quad driver \sqsubseteq employee, \quad male \sqsubseteq person, \quad female \sqsubseteq person.$$

Relations between concepts such as "Mary is a driver," and "heart attack is a kind of disease" can be represented by this partial ordering.

Complex concepts, such as "a company whose name is 'S'," are represented by *object terms* [1]:

$$company\ [name = \text{"}S\text{"}\ ],$$

where *company* is a basic object, *name* is a *label*, and S is an object term as the value of *name*. Basic objects themselves can also be considered object terms. Generally, an object term is a variable or term of the following form:

$$o[l_1 = t_1, \cdots, l_n = t_n]\ (0 \leq n)$$

where $o, l_1, \cdots, l_n$ are basic objects and $t_1, \cdots, t_n$ are object terms.

Subsumption relations '$\sqsubseteq$' between basic objects are extended to object terms. For example:

$$\begin{aligned} company\ [name = \text{"}S\text{"}\ ] &\sqsubseteq company, \\ company\ [name = \text{"}S\text{"},\ president = \text{"}Johns\text{"}\ ] &\sqsubseteq company\ [name = \text{"}S\text{"}\ ]. \end{aligned}$$

2.2.2 Subsumption Constraints

To represent a property of an object, a *dotted term* is used. For example, *mary.employer* represents 'Mary's employer'.

To represent relations such as "Mary is employed by company 'S' ", we use a *subsumption constraint* between an object term and a dotted term:

$$mary.employer \cong company\ [name = \text{"}S\text{"}\ ],$$

where $A \cong B$ means that $A \sqsubseteq B$ and $B \sqsubseteq A$.

The syntactic construct for representing an object term with subsumption constraints is called an *attribute term*. Let o be an object term and C be a set of subsumption constraints,

[1] Although QUIXOTE can handle object sets as terms, we do not describe this here.

then $o|C$ is an attribute term. *QUIXOTE* also provides some syntax sugars:

$$\begin{aligned} o \mid \{o.l \sqsubseteq t\} &\Leftrightarrow o/[l \rightarrow t], \\ o \mid \{o.l \sqsupseteq t\} &\Leftrightarrow o/[l \leftarrow t], \\ o \mid \{o.l \cong t\} &\Leftrightarrow o/[l = t]. \end{aligned}$$

Using these constructs, the following attribute term represents the new case: "Mary died from a heart attack while taking a break."

$$\textit{new-case}/[\textit{who=mary}, \textit{while=break}, \textit{result=heart-attack}].$$

2.2.3 Rules

A rule is defined as follows:

$$a_0 \Leftarrow a_1, \cdots, a_n \parallel D,$$

where $a_0, a_1, \cdots, a_n$ are attribute terms and D is a set of subsumption constraints. a_0 is called a *head*, $a_1, \cdots, a_n \parallel D$ is called a *body*, and a_i's are called *subgoals*. A rule means that if the body is satisfied, the head is satisfied. If a body is empty, the rule is called a *fact*.

For example, the following is a rule for judgment:

$$\begin{aligned} \textit{judge}\ [\textit{case=X}]/[\textit{judge}\rightarrow\textit{insurance}\,] \\ \Leftarrow \textit{judge}\ [\textit{case=X}]/[\textit{judge}\rightarrow\textit{job-causality}\,], \\ \textit{judge}\ [\textit{case=X}]/[\textit{judge}\rightarrow\textit{job-execution}\,] \\ \parallel \{X \sqsubseteq \textit{case}\}. \end{aligned}$$

It means that if the judgment of a case, *judge* $[case=X]$ where $X \sqsubseteq case$, is 'job-causality' and 'job-execution', then the judgment is for 'insurance'.

2.2.4 Modules

QUIXOTE allows sets of rules to be modularized:

$$m :: \{r_1, \cdots, r_n\},$$

where m is an object term called a *module identifier* (mid) and $r_1, \cdots, r_n$ are rules.

The definition of rules is extended for external reference of objects:

$$a_0 \Leftarrow m_1 : a_1, \cdots, m_n : a_n \parallel D,$$

where $m_1, \cdots, m_n$ are module identifiers. This rule means if a_i and D are satisfied in module m_i for $1 \leq i \leq n$, then a_0 is satisfied. As an attribute term can be separated into an object term and

a set of constraints, the rule can be rewritten as follows:

$$o_0 \mid C_0 \Leftarrow m_1 : o_1, \cdots, m_n : o_n \parallel C,$$

where $a_i = o_i | C_i (0 \leq i \leq n)$ and $C = C_1 \cup \cdots \cup C_n \cup D$. If module m has the rule and a_0 is satisfied, we use the expression: "o_0 *exists* in m and has C_0 in m."

Rules are imported and exported by *rule inheritance*, defined in terms of the binary relation (written as '$\sqsupseteq_S$') between modules, called a *submodule relation*:

$$m_1 \sqsupseteq_S m_2, \quad m_1 :: \{R_1\}, \quad m_2 :: \{R_2\} \implies m_1 :: \{R_1, R_2\}, \quad m_2 :: \{R_2\},$$

where m_1, m_2 are modules and R_1, R_2 are sets of rules.

In a submodule definition, an expression in the right hand side of '$\sqsupseteq_S$' may be a formula consisting of module identifiers and set operators. Operators, '$\cup$' and '$\setminus$', are provided for constructing the union and the difference of two modules.

A *database* or a *program* is defined as the triplet (S, M, R) of a finite set S of subsumption relations, a set M of submodule relations, and a set R of rules.

2.2.5 Query Processing

Query processing in QUIXOTE corresponds to resolution and constraint solving in constraint logic programming.

A *query* is defined as a pair (A, P) (written ?-A;;P) of a set A of attribute terms and a program P, where A is referred to as the *goal* and P the *hypothesis*. Consider a database DB. A query ?-A;;P to DB is equivalent to query ?-A to $DB \cup P$ (If $DB = (S_1, M_1, R_1)$ and $P = (S_2, M_2, R_2)$ then $DB \cup P = (S_1 \cup S_2, M_1 \cup M_2, R_1 \cup R_2)$). That is, P is inserted into DB before A is processed. In other words, P works as a hypothesis for ?-A.

As hypotheses are incrementally inserted into a database, nested transactions are introduced to control such insertions. See [11] for details.

An *answer* is defined as the triplet (D, V, E). D, which is called the assumption part of the answer, is a set of subsumption constraints that cannot be solved during query processing; V, the conclusion part of the answer, is a set of variable constraints that are bound during query processing; and E, the explanation part of the answer, is the corresponding derivation flow. Note that only subsumption constraints of object properties can be in the assumption part.

2.3 Applications

We have developed various knowledge information processing applications such as natural language processing, genomic information procession, and legal reasoning, in *Quixote*. In this section, we explain the TRIAL legal reasoning system which we implemented in *Quixote*. By showing the overall architecture, a design strategy of a knowledge-base, and an implementation of the example knowledge-base on TRIAL, we demonstrate *Quixote*'s utility for constructing knowledge-bases with partial information.

2.3.1 The Implementation of TRIAL

The overall system architecture is shown in Figure 1.

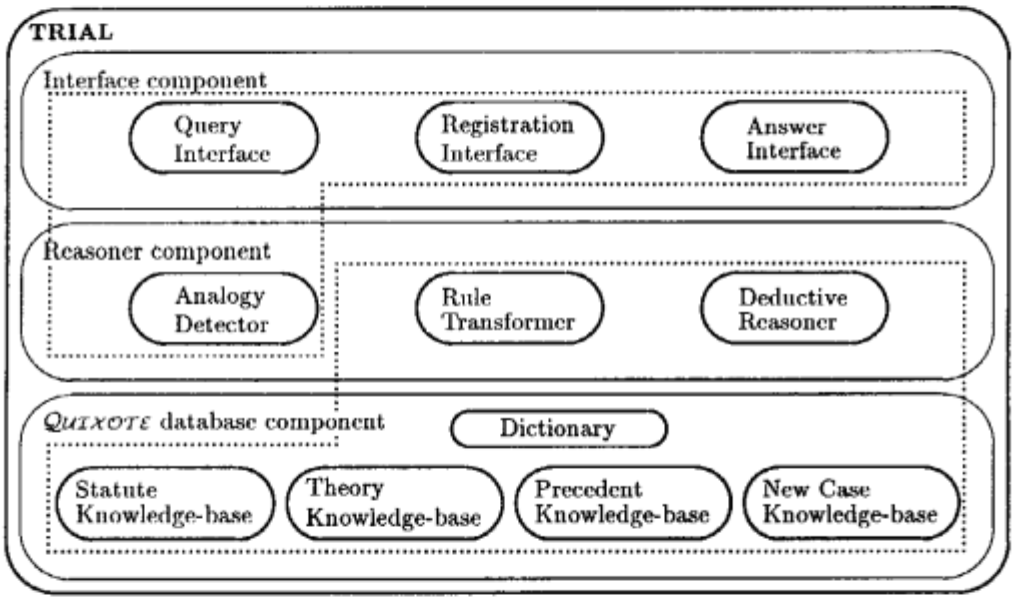


Figure 1: Architecture of TRIAL

Note that the all data and knowledge in the database component is written in *Quixote*.

In this section, we show how our example knowledge-base is implemented in TRIAL (as modules in *Quixote*). We also show related query interaction.

The new case is represented as follows:

$$
\begin{aligned}
\mathit{new\text{-}case} :: \{&\mathit{new\text{-}case}/[\mathit{who}{=}\mathit{mary},\\
&\quad \mathit{while}{=}\mathit{break},\\
&\quad \mathit{result}{=}\mathit{heart\text{-}attack}\,];;\\
&\mathit{relation}[\mathit{state}{=}\mathit{employ}, \mathit{employee}{=}\mathit{mary}\,]\\
&\quad /[\mathit{affiliation}{=}\mathit{organization}[\mathit{name}{=}\text{“}S\text{”}\,],\\
&\quad \mathit{job}{\rightarrow}\mathit{driver}\,]\},
\end{aligned}
$$

where ";;" is a delimiter between rules.

The statute and the theory of its application are as follows:

$$
\begin{aligned}
\mathit{labor\text{-}law} :: \{\ &\mathit{organization}\ [\mathit{name}{=}X]\\
&/[\mathit{responsible}{\rightarrow}\mathit{compensation}\ [\mathit{object}{=}Y,\ \mathit{money}{=}\mathit{salary}\,]]\\
&\Leftarrow \mathit{judge}\ [\mathit{case}{=}C]\ /[\mathit{judge}{\rightarrow}\mathit{insurance}\,],\\
&\quad \mathit{relation}\ [\mathit{state}{=}Z,\ \mathit{employee}{=}Y]\\
&\qquad /[\mathit{affiliation}{=}\mathit{organization}\ [\mathit{name}{=}X]]\\
&\quad \| \{C \sqsubseteq \mathit{case}\}\}.\\
\mathit{theory} :: \{\ &\mathit{judge}\ [\mathit{case}{=}X]/[\mathit{judge}{\rightarrow}\mathit{insurance}\,]\\
&\Leftarrow \mathit{judge}\ [\mathit{case}{=}X]/[\mathit{judge}{\rightarrow}\mathit{job\text{-}causality}\,],\\
&\quad \mathit{judge}\ [\mathit{case}{=}X]/[\mathit{judge}{\rightarrow}\mathit{job\text{-}execution}\,]\\
&\quad \| \{X \sqsubseteq \mathit{case}\}\}.
\end{aligned}
$$

The abstract interpretation precedent rules are abstracted from the original precedent knowledge-base rules by TRIAL and the parameterization method.

$$
\begin{aligned}
\mathit{case}_1 :: \{\ &\mathit{judge}[\mathit{case}{=}X]/[\mathit{judge}{\rightarrow}\mathit{job\text{-}execution}\,]\\
&\Leftarrow \mathit{relation}[\mathit{state}{=}Y, \mathit{employee}{=}Z]/[\mathit{cause}{=}X], X\\
&\quad \| \{X \sqsubseteq \mathtt{parm}.\mathit{case},\ Y \sqsubseteq \mathtt{parm}.\mathit{state},\ Z \sqsubseteq \mathtt{parm}.\mathit{employee}\}\}.\\
\mathit{case}_2 :: \{\ &\mathit{judge}[\mathit{case}{=}X]/[\mathit{judge}{\rightarrow}\mathit{job\text{-}causality}\,]\\
&\Leftarrow X/[\mathit{while}{=}Y,\ \mathit{result}{=}Z],\\
&\quad \| \{X \sqsubseteq \mathtt{parm}.\mathit{case},\ Y \sqsubseteq \mathtt{parm}.\mathit{while},\ Z \sqsubseteq \mathtt{parm}.\mathit{result}\}\}.
\end{aligned}
$$

The object 'parm' represents the abstraction parameters. This object results from abstracting precedents. It is used to control judgment prediction. Its properties are defined as follows:

$$
\begin{aligned}
\mathit{parm} :: \{\ &\mathtt{parm}/[\mathit{case}{=}\mathit{case},\ \mathit{state}{=}\mathit{relation},\ \mathit{while}{=}\mathit{job},\\
&\quad \mathit{result}{=}\mathit{disease},\ \mathit{employee}{=}\mathit{person}\,]\}.
\end{aligned}
$$

To prevent the over-abstraction, these values restrict the range of variables X, Y, and Z in both precedent rules.

To use '**parm**' for case_1 and case_2, we define the following submodule relation:

$$parm \sqsupseteq_S case_1 \cup case_2.$$

This information is dynamically defined in the rule abstraction step, because the choice of precedents is done on an experimental basis.

The knowledge-base has the following subsumption relations:

case	$\sqsupseteq$	*new-case*,	*person*	$\sqsupseteq$	*mary*,
relation	$\sqsupseteq$	*employee*,	*job-causality*	$\sqsupseteq$	*insurance*,
disease	$\sqsupseteq$	*heart-attack*,	*job-execution*	$\sqsupseteq$	*insurance*,
job	$\sqsupseteq$	*break*.			

We can now query the knowledge-base:

1. According to past precedents and theory, what kind of judgment can we predict for the new case?

   ?- *new-case* : *judge* [*case=new-case*]/[*judge=X*];;

   new-case $\sqsupseteq_S$ *parm* $\cup$ *theory*.

   Note that the module *parm* is defined to inherit the abstracted precedent rules. Thus, we get three answers, in which the first is returned unconditionally, and the last two include hypotheses:

   (a) $X \sqsubseteq$ *job-causality*,

   (b) if *new-case* : *judge* [*case=new-case*] has *judge* $\sqsubseteq$ *job-execution*, then $X \sqsubseteq$ *insurance*,

   (c) if *new-case* : *relation* [*state=employ,employee=mary*] has *cause=new-case*, then $X \sqsubseteq$ *insurance*.

2. According to labor law, what responsibility does Mary's company have?

   ?- *new-case* : *organization* [*name="S"*]/[*responsible=X*];;

   new-case $\sqsupseteq_S$ *parm* $\cup$ *labor-law*.

   Note that, before issuing this query, the module *new-case* is already a submodule of module *theory* as an effect of the previous query. Thus, two answers are returned with generated hypotheses:

   (a) if *new-case : judge* [*case=new-case*] has *judge* $\sqsubseteq$ *job-execution*,
   then $X \sqsubseteq$ *compensation* [*obj=mary, money=salary*].

   (b) if *new-case : relation* [*state=employ, employee=mary*] has
   cause=new-case,
   then $X \sqsubseteq$ *compensation* [*obj=mary, money=salary*].

Quixote returns explanations (derivation graphs) with corresponding answers to TRIAL. The TRIAL user interface displays this graphically if requested by the user. Judging an answer by the validity of the generated hypotheses and the corresponding explanation, the user can also update the database or change its abstraction strategy.

2.3.2 Other Useful Features

Some other features of *Quixote* are also useful to TRIAL system.

- Property inheritance mechanism allows to reduce the amount of knowledge descriptions.
- A rule can be designated not to generate any hypothesis. Among the example knowledge-bases of TRIAL, rules in statute and theory knowledge-bases are designated as such.
- Various browsing commands are available. For example, a module before/after saturation of rule inheritance can be displayed.

3 Towards New Generation Database Systems

3.1 Knowledge-Bases Reconsidered

From the experiences of TRIAL, we can list several requirements for next generation database systems, which need not necessarily be only for legal applications:

1. *Processing partial information*

   As data and knowledge are often not given in a perfect form, unlike conventional applications, we must consider ambiguous or erroneous data as well as null values and logically incomplete information such as negation and disjunction. In *Quixote*, we use subsumption constraints to handle ambiguous and partially lacking properties. They are useful not only to knowledge databases but also to scientific databases.

2. *Realizing an environment for thinking experiments*

   Answers are not necessarily given uniquely in knowledge information processing, but are refined by repeating trial-and-error querying with the users. In this sense, the features of hypothetical reasoning and hypothesis generation are very important.

3. *Framework of very large database and knowledge-base*

   Classification mechanisms are very important for storing large databases and knowledge-bases. Subsumption and submodule hierarchies contribute to such classification. Especially, a framework that allows inconsistent data and knowledge to co-exist is needed.

4. *Integration of heterogeneous data and knowledge*

   Even if we consider only one application, we can find various kinds of data and knowledge within it. For example, legal data includes large amounts of text data as the primary data, and abstracted data or knowledge (including rules) as the higher level data. Although we have not integrated such data and knowledge into TRIAL, the integration of such heterogeneous data and knowledge will become very important.

5. *Knowledge discovery in databases*

   To classify large databases and knowledge-bases, two kinds of knowledge discovery will be needed: how to find erroneous and lacking information; how to find new knowledge (abstracted rule in the above example).

6. *Integration of technologies in related areas*

   Database technologies have been spreading: for example, we now have deductive databases, database programming languages, deductive object-oriented databases, and very large knowledge-bases. More technologies in many areas such as artificial intelligence, programming languages, and operating systems, should be embedded into database systems.

3.2 Heterogeneous, Distributed, Cooperative Problem Solvers *Helios*

Among the requirements in the previous subsection, *Quixote* already supports (1) and (2). To support requirements (3), (4), and (6), we have been engaged in the new system, *Helios*, for heterogeneous, distributed, cooperatibe problem solvers. In this subsection, I describe the outline.

3.2.1 Approaches

Very large knowledge-bases (VLKB) have been required from various areas such as expert systems and applications for their infrastractures of knowledge information processing. However, the term, *knowledge-base*, is frequently semantically overloaded. The confusion seems to come from the differences of backgrounds such as databases, expert systems, and applications, and the differences of following approaches.

The first one is related to the design of applications:

- **bottom-up approach:**

  There are many existing applications, each of which has intrinsic databases, knowledge-bases, or problem solvers. New applications may be developed as the combination or

integration of such existing ones as *an* information system. In database areas, although multi-databases and federated databases have been investigated as such an approach, they must be extended to multi-knowledge-bases or a very large knowledge-base as a part of an integrated information system. In distributed artificial intelligence areas, multi-agent systems are considered along this approach.

- **top-down approach:**

  As in most of traditional applications, *an* application, or its problem is divided into multiple sub-problems, for each of which, a module is developed as a component. From a database point of view, it corresponds to a (logically integrated) distributed database or knowledge-base, where each local database or knowledge-base corresponds to a component. In distributed artificial intelligence areas, distributed problem solvers are considered along this approach.

As these approaches are often mixed up for developing an application, it is difficult to classify all applications from such a viewpoint.

The second one is related to background knowledge:

- **database approach:**

  Databases have been providing more advanced features as in deductive databases and deductive object-oriented databases, which support not only a set of data but also a set of rules or knowledge for knowledge information processing, in other words, databases are growing to be knowledge-bases. On the other hand, there are many works on heterogeneous, distributed databases such as federated databases and multi-databases. The integration of such two directions can be considered as an approach to very large knowledge-bases.

- **knowledge-base approach:**

  Most of conventional expert systems have used small *knowledge-bases*. They have showed limitations of the ability of solving more difficult or unexpected problems. One of the reasons is considered due to the *smallness* of such knowledge-bases. To break through such limitations, extensions of knowledge-bases in the sense are are considered as *very large* knowledge-bases.

We take bottom-up and database approaches and expect to provide an integrated framework for very large knowledge-bases, including distributed problem solvers, multi-agent systems, and computer supported cooperative works.

The important points in our approach are to treat a database, a knowledge-base, a constraint solver, or an application program uniformly as a *problem solver* and to integrate multiple heterogeneous problem solvers as an information system. In this context, *heterogeneity* means that each problem solver may be written in a different language and have different problem solving abilities. In this paper, we propose a new framework of very large knowledge-bases consisting of multiple heterogeneous problem solvers. The basic features of our framework are as follows:

1. Each problem solver can be an *agent* by wrapping in a *capsule*, which encapsulate intrinsic properties of each agent.
2. Each agent can communicate and negotiate with others in an *environment*, where the common communication protocol and type system can be user-defined.
3. An environment and its related agents can be defined also as a *new* agent by a capsule, which hides the internal structure.
4. A user can ask queries to *an* agent, where the user is considered as an environment for the agent. Further, a user can be defined as an agent.

Our contributions in the system are to provide a simple framework for integration and cooperation of multiple heterogeneous problem solvers, the granularity of which is arbitrary, to discuss relations between globality and locality, and to define the language based on the discussions with multi-level transformation features.

3.2.2 Model of Agent and Environment

In this section, we describe basic concepts and features of our model.

(1) Agent and Message

An *agent* is an encapsulated autonomous problem solver, which may be a database, a knowledge-base, a constraint solver, or any application program. Any problem solver (called a *substance*) wrapped with a *capsule* can be an agent as in Figure 2. An agent provides a set of *method protocols* as a part of its capsule's definitions, only through which users can send messages to the agent. Given a message (or a query) to an agent, its capsule interprets it according to its related message protocol, convert the types of the arguments, the syntax of the message, the variable environment, and the internal data structure, and send the transformed message to the internal problem solver (substance).

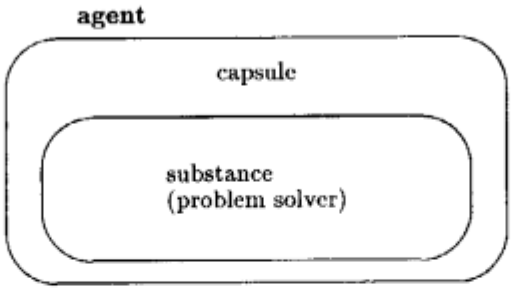


Figure 2: Simple Configuration of an Agent

A *message* to an agent is in the form of a global *communication protocol* consisting of the message identifier, the identifier of a sender agent, the identifier of the receiver agent, and a message. Whether an agent with side effect can process multiple messages or not depends on the ability of the concurrency control of the substance (internal solver). For example, if the substance is an database management system, its capsule send multiple messages because the substance controls concurrency as one of basic functions. If the substance processes a message only sequentially, its capsule serialize message passing.

(2) Agents in an Environment

Given multiple agents for cooperative problem solving, there must be a common *environment*, which knows what agents exist in it, provides common language interface, that is, common type system and data structure, and defines global constraints and communication strategies such as negotiation. As such an environment and a set of agents in it is also a problem solver, they can be defined as *an* agent by a capsule as in Figure 3. An agent consisting of multiple agents is

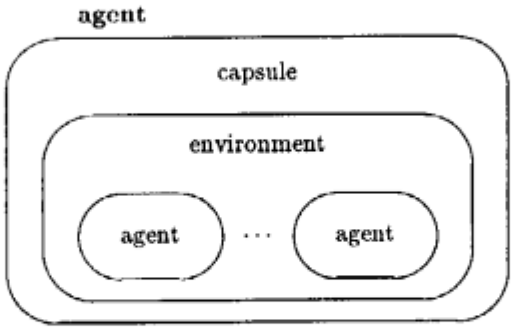


Figure 3: A Compound Agent

called a *compound agent*, while an agent without an environment (Figure 2) is called a *simple agent*.

If a substance in an agent detects unsolvable constraints (or problems) inside during query processing, the agent throw out them to its capsule. The capsule converts and transforms the constraints according to an *export* message protocol into the form of common message in its environment and throw out it to the environment. The environment dissolves the constraints into a set of sub-messages if necessary, generates a processing plan, lists candidates for processing each sub-messages, and throw out them to some of candidates. Each sub-message has information for synchronization of processing the whole message according to the processing plan, which is controlled locally by the message handler of an agent but not by the environment. An environment controls query processing if necessary: listing candidates, which could solve a problem, dispatching according to priority, evaluating answers, and demanding alternative answers. The main features of an environment is shown in Figure 4.

An environment is responsible for initialization and locating agents. If there are some homogeneous agents in an environment, it generates necessary numbers of agents from the corresponding template and allocate them in it. If an agent is not located when it is required, the environment loads and initialize a necessary agent. However, if an agent fails to candidates for solving a query, the query is thrown out to the outer environment.

Note that there are two kinds agents: an *active* and *passive* agents. A passive agent only import messages, that is, it receives a message, process it, and return its answer. every problem solver can be a passive agent. On the other hand, an active agent both import and exports messages, that is, it can detect an unsolvable problem inside, send it as a message to other agents, and receive its corresponding answers.

(3) User as an Environment and an Agent

A user is defined as the outermost environment where there is only one (simple or compound) agent. If an internal agent cannot solve a problem, the problem is thrown out in the outer environment. So, users receive unsolvable problems finally. If users return the answer to the agent, the agent continue to process the suspended message. This model makes prototyping multi-agent programming in our model easier.

For communication between a user and an agent, a user can give his user model, which corresponds to a common type system and data structures defined in an environment. Given a user model to an agent, its capsule transforms all messages between the user and the agent. If necessary, a user can communicate with multiple agents which the common user model is given.

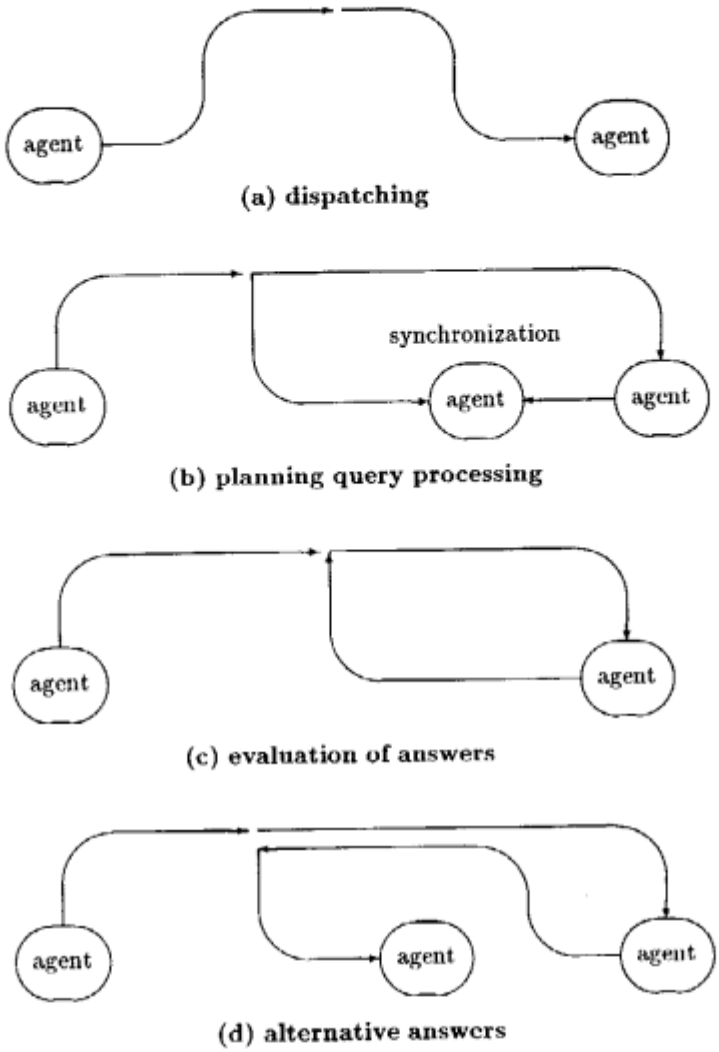


Figure 4: Features of an Environment

Furthermore, a user may be defined also as an agent, that is, a user can process a message sent by its capsule as an agent. This feature helps not only prototyping a system, but also constructing a group-ware environment, if multiple users are defined as agents.

Relations among users and agents are shown in Figure 5.

3.2.3 Features of Language and System

There are three important concepts to homogenize heterogeneous problem solvers: message, environment, and capsule. An environment defines a common space where multiple agents can exists, A capsule is a suit of clothes by which a problem solver can live in an environment. A

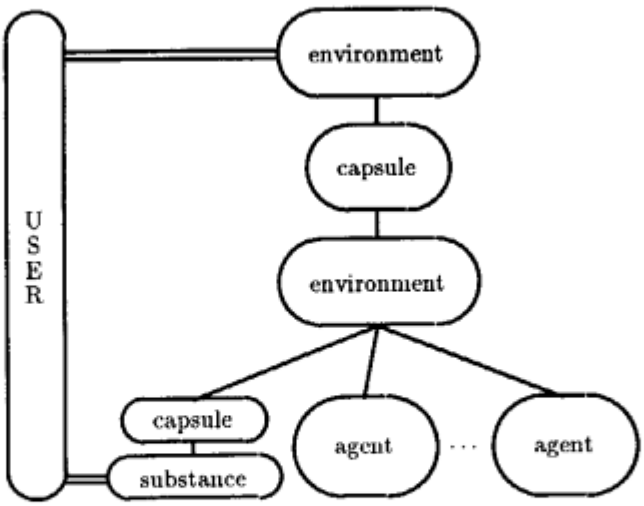


Figure 5: User as an Environment and an Agent

message is a communication tool by which an environment and agents can work cooperatively.

(1) Message

Every information between an environment and an agent is a message in the form of the following communication protocol:

$$(MessageID, FromAgent, ToAgent, Message)$$

$MessageID$ consists of a *message identity* and a *synchronization identity*. As a message identity is decided locally by the sender agent, the identity includes local environment and agent identities to maintain global identity in a distributed environment. When a message is dissolved into sub-messages, synchronization identities are attached to each sub-message and coordinator agents are determined according to the processing plan. For example, assume a message m_0 is dissolved into m_1 and m_2. Although two message may be processed in parallel, they must be merged in some agent. If m_1 plays a role of the coordinator, m_1 has a synchronization identity such that 'wait for m_2' and m_2 has 'post to an agent with m_1'.

$FromAgent$ and $ToAgent$ are paths of this message, along which answers are returned to the agent which asked a query. The paths guarantees not only the validity of message transformation, but also make it possible for a message to return not through an environment except some cases. For example, consider that a query sent by an agent a is dissolved by an agent b into two queries for agents c and d. Answers from c or d cannot be directly sent to a, because both answers must be merged into an answer in b, where variable environments are

retained.

Message consists of a message itself and control information such as transaction management. As the former, there are mainly two kinds of messages: a *query* and an *answer*, which are syntactically differentiated and related by the $Message - ID$. For transaction control, there are three kinds of messages: begin, end, and abort, as usual.

(2) Environment

An environment, written in ENVL (environment description language), consists of the following definitions:

- Agent server, where logical and physical information of agents are defined. Function names, keywords, and method names can be also registered for selection of agents. An agent server is used for searching agents by ambiguous function names and listing agent candidates for processing a message. There are three kinds of directories for the agent server: an *agent directory* for relations between agent names and agent addresses, a *method directory* for relations of method names and agent names, and a *function directory* for relations between function names and agent names.
- The common type system and its corresponding data structures are defined in an environment. In an environment, all data in messages must be strongly typed to guarantee the validity of message transformation. Although the definition of a type system is arbitrary, a poor type system might lose information which agents provide, by data transformation.
- Parametric agents (or agent templates) are defined. When there are multiple similar agents in an environment (for example, n-queen), their common template can be defined as a parametric agent. Parameters should have enough information to generate necessary agents.
- Global constraints in the environment, are defined. Such constraints shared by multiple agents are used for control information of values which agents can get and information passing among agents. For example, you can see the number of columns and rows as in n-queen as unchanged one, and a blackboard as changeable one.
- Strategies of negotiation and conflict resolution, written in ENVL, can be embedded in an environment, although each agent might have intrinsic strategy. For example, the environment can embed the idleness of agents into a procession plan and its answer evaluation.

- The ontology for transformation of vocabularies between agents is defined. As each agent is autonomous not only in message processing but also in vocabularies, the ontology translates them among agents and resolves conflict of their usage.

If a new agent would join the environment, those definitions should be conservatively extended. ENVL does not intend to control inter-agent communication centrally, but shows an ability of definitions. Even if an environment is defined centrally, it can be replicated as an attachment in each agent. Users can select which features should be written in an environment or agents.

Furthermore, the following built-in functions are supported in an environment from a system point of view:

- Management of messages. As an environment is accessed from its internal multiple agents and its external capsule, all messages which passes in the environment are watched to avoid deadlock and loop with a time-out facility.
- Initialization of agents. Agents can be linked to an environment either statically or dynamically. In a case of dynamic link, an environment loads agents when they are required, and generates a necessary number of agents from a parametric agent. An agent directory is generated and maintained by the initialization.
- Dissolving a message and creating a processing plan. There are two cases: a message with multiple messages to reduce communication cost and naturally dividable multiple messages. In both cases, a messages is analyzed and a processing plan as a dependency graph is created by an environment.
- Parallel processing. In the case of an agent without capabilities of concurrency control or parallel processing, an environment manage its copies for parallel processing.
- Control of candidate agents. An environment decides to send a message sequentially or parallelly to candidates listed by the agent server.
- Evaluation of results, control for alternative processing, and composing a solution. A query sometimes accompany a set of constraints for satisfiability check of answers. An environment sends an answer and a set of constraints to the related agent if necessary. Depending on the evaluation results, an environment decides whether alternative message processing is necessary or not.
- Management of ontologies and transformation of vocabularies in messages. A message should be converted and translated at multiple levels, among which an environment is responsible for vocabulary translation among agents.

- Unsolvable query. When all agent does not solve a query, the environment send the query to the outer environment through its capsule.

These features are selected according to the definitions in ENVL.

(3) Capsule

A capsule, written in CAPL (capsule description language), consists of the following definitions:

- Identity and substance. Although a substance might be simple or compound, it is defined as an agent with the identity.
- Method protocol. There are two kinds protocols: *import* and *export*. An import defines a protocol which the agent can accept, while an export defines a protocol which the agent asks processing to external agents. Already mentioned, import protocols are defined in every agent, which export protocols are defined only in active agents. According to such protocols, the syntax and semantics are converted and transformed between an environment and an internal substance. A method directory in a environment is generated from this definitions.
- Transformation of messages. As for transformation of a syntax and types, CAPL provides definition facilities, while, as for transformation of internal data structure (including variable environments), only names of transformation programs to be called are specified for simplicity.
- Self model. Besides a set of methods, what an agent can do is defined as the self model of an agent. A function directory in an environment is generated from this model.
- Lock information. Whether the substance can accept multiple messages or not is specified. If not, messages are serialized by the capsule.

From a system point of view, a capsule has the following features:

- Management of messages. Both import and export messages are managed with a time-out facility for non-activity, long transaction, and external processing.
- Synchronization of dissolved messages. According to a processing plan created by an environment, some agent plays a role of a coordinator, which is responsible for synchronization of sub-messages and merging of their results.
- Conversion and transformation of a message. This process is very complex: for example, consider the differences among Prolog, Lisp, and C. Even if a message belongs to an

untyped language, all the arguments are strongly typed, converted, and transformed into the common form of the environment. This process is bi-directional because its answer must be transformed in the intrinsic form of the substance. As a user must specify how to type, convert, and transform messages, and vice versa in CAPL, he is responsible for the semantical validity and the lossless of information during this process.

(4) Substance (Problem Solver)

There are two kinds of substances (problem solvers) from our model's point of view: *passive* and *active* problem solvers. A passive problem solver receives a query through its capsule from other agents and returns an answer, but does not send a query to other agents, while an active problem solver can send a query and receive its answer through its capsule from other agents. If the message queue in the capsule is considered as a part of an environment, the active problem solver can be said to be *active* from an object-orientation point of view, because the problem solver can watch the message queue autonomously. An agent with an active problem solver is called *active* and an agent with passive problem solver is called *passive*.

Any application program, database, and knowledge-base can be a passive agent as it is by a capsule, while some additional features are required to be an active agent. They are as follows:

- Parsing of unsolvable problems as a part of the syntax of the language.
- Detection of unsolvable problems during query processing.
- Sending unsolvable problems to its capsule and receiving its answers.
- Conversion of answer:
  - Conversion between Boolean values (true/false) and success/failure.
  - Treatment of set values which are returned as answers from other agents.
  - Acceptance of new constraints evaluated by other agents.

For making an agent active, the specific design of the problem solver or the modification of the language processor is required.

3.2.4 Examples of some Applications

We are considering more applications such as a heterogeneous natural language processing system, a heterogeneous distributed file system, a legal reasoning system, a genetic information system, a free agent system in baseball, and a *sumo* system.

Here we show an outline of a *sumo* system, which we are investigating for the prototype system. We intend to solve the following problems:

- How many wins are necessary for a sumo wrestler X to be promoted?

  There may be many expectations.

- Who will win at the next tournament?

- Which is the winner at the match of sumo wrestlers X and Y?

  The history of their matches should be referenced.

- How can the tomorrow's list of matches be planned?

  There are many constraints for matches.

We prepare about 10 agents fot the above problems: a database of sumo wrestlers, a database of records of matches, an arithmetical constraint solver, a tournament calculator, a date calculator, a promotion rule base, a sumo council, a newspaper, and so on. Two databases are stored in Kappa, rules such as promotion and aggregation of records are written in *Quixote*, and algebraic constraints are written in GDCC.

The interesting points in this example are as follows:

- Combination of heterogeneous problem solvers.
- Conversion of SQL to database agents.
- The possibility of a very large database of records.
- Constraint satisfaction problem for preparing matches.
- Inconsistent expectations by the sumo council and newspapers.
- Historical data management of sumo wrestlers and their record.
- Transaction management also in agents without concurrency control.

4 Concluding Remarks

In this paper, I overview two topics:

- TRIAL and *Quixote*

  I explained a prototype system, TRIAL, for legal reasoning system, based on a deductive object-oriented database language, *Quixote*, and pointed out many effective features of *Quixote* for legal reasoning.

- *Helios*

  I gave an outline of a system, *Helios*, for heterogeneous, distributed, cooperative problem solvers. The objective of the system is not an extension of *Quixote* itself, but an integrated framwork of many problem solvers including *Quixote*. As legal reasoning needs various kinds of data and knowledge, *Helios* will show more effectiveness in legal applications than *Quixote*.

Our dicision in these activities was based on such consideration that many applications need not only extensions of a single language, but also an integrated cooperative framework of heterogeneous problem solvers, including databases, knowledge-bases, constraint solvers, and application programs.

Quixote and *μ-Quixote* systems, which run under UNIX, have been released as ICOT free software. As for *Helios*, we already implemented the first version of the prototype system, which supports only basic functions and to describe small applications. Through this experiments, the performance of communication among processes was analyzed. We are now investigating specifications of CAPL and ENVL for the second version and descriptions of conflict resolution and negotiation strategies for the third version. At the same pace, we are doing experiments for describing application including the above sumo problem. This *Helios* will be also released as ICOT free software soon.

Acknowledgments

I would like to thank Chie Takahashi, Toshihiro Nishioka, and Satoshi Tojo for many comments on this early draft, and all members of *Quixote* and *Helios* projects for their valuable discussions.

References

[1] A. Aiba and R. Hasegawa, "Constraint Logic Programming System: CAL, GDCC, and their Constraint Solvers", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[2] S. Ceri, K. Tanaka, and S. Tsur (eds.), *Deductive and Object-Oriented Databases*, (*Proc. the Third Int. Conference on Deductive and Object-Oriented Databases (DOOD'93)*), LNCS 760, Springer, 1993.

[3] C. Delobel, M. Kifer, and Y. Masunaga (eds.), *Deductive and Object-Oriented Databases, (Proc. the Second Int. Conference on Deductive and Object-Oriented Databases (DOOD'91)*, LNCS 566, Springer, 1991.

[4] M. Kawamura, H. Sato, K. Naganuma, and K. Yokota, "Parallel Database Management System: *Kappa-P*", *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, June 1-5, 1992.

[5] L.O. Kelso, "Does the Law Need a Technological Revolution?," *Rocky Mt. Law Rev.*, vol.18, pp.378-392, 1946.

[6] W. Kim, J.-M. Nicolas, and S. Nishio (eds.), *Deductive and Object-Oriented Databases, (Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (DOOD89)*, North-Holland, 1990.

[7] C. Takahashi, K. Yokota, "A Legal Reasoning System on a Deductive Object-Oriented Database," *Proc. 5th Int. Hong Kong Computer Society Database Workshop*, Kow Loon, Hong Kong, February, 1994.

[8] N. Yamamoto, "TRIAL: a Legal Reasoning System (Extended Abstract)," *Joint French-Japanese Workshop on Logic Programming*, Rennc, France, July, 1991.

[9] H. Yasukawa, H. Tsuda, and K. Yokota, "Objects, Properties, and Modules in *Quixote*," *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

[10] K. Yokota and A. Aiba, "A New Framework of Very Large Knowledge Bases", *Proc. Int. Conf. on Building and Sharing of Vary Large-Scale Knowledge Bases'93 (KB&KS'93)*, Tokyo, Dec. 1-4, 1993.

[11] K. Yokota, H. Tsuda, and Y. Morita, "Specific Features of a Deductive Object-Oriented Database Language *Quixote*," *Workshop on Combining Declarative and Object-Oriented Databases, (ACM SIGMOD'93 Workshop)*, Washington DC, May 29, 1993.

[12] K. Yokota and H. Yasukawa, "Towards an Integrated Knowledge-Base Management System — Overview of R&D on Databases and Knowledge-Bases in the FGCS Project," *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

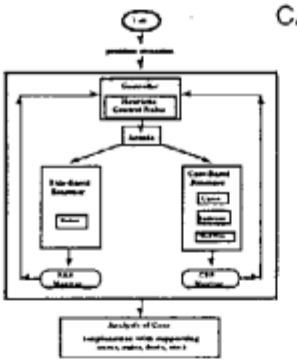
Legal Knowledge Representation: A View from the Backbench

David B. Skalak
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
U.S.A.
skalak@cs.umass.edu

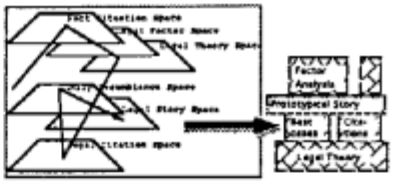
Three Systems

- HYPO (Ashley, 1990)
- CABARET
- BankXX

CABARET



BankXX: Heuristic Search Supplies Argument Building Blocks



Some First Representation Steps

- Cases
- Statutes

Second Steps

- Legal theories
- Recurring situations (factual prototypes)
- Legal arguments
- ....

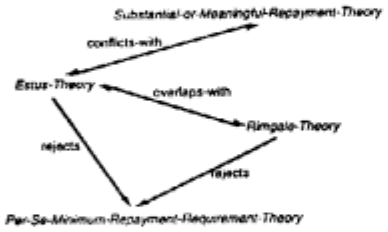
Legal Theories

- In terms of factors
- Ways to combine the factors
- Theory modification methods

Legal Theory Example

name: ESTUS-THEORY
factors: necessary-expenses-minus-plan-payments-factor,
employment-history-factor, earnings-potential-factor,
plan-duration-factor, plan-accuracy-factor,
preferential-creditor-treatment-factor,
secured-claims-modified-factor, debt-type-factor,
nondischarge-7-factor, special-circumstances-factor,
frequency-relief-sought factor, motivation-sincerity-factor,
trustee-burden-factor
view: majority
cases-promulgating: Estus
cases-applying: Estus, Flygare, Makarchuk
cases-rejecting: nil
courts-adopting : 8th circuit
description: "Theory applied in the leading case In re Estus, 695 F.2d 311 (8th Cir. 1982), which states (at 317), ..."

Legal Theory Space



Legal Theory Modifications (Methods)

- limit-theory-by-increasing-prerequisite-factors
- shift-burden-of-persuasion
- reweight-factors

Factual Prototypes

- Trade secrets misappropriation (HYPO, Ashley)
- Income tax: home office deduction
- Bankruptcy

Examples: Factual Prototypes

- student loan
- civil judgment lien
- family farm
- dishonest debtor
- medical calamity
- credit card junkie
- automobile debt
- honest debtor
- bankruptcy repeater
- interrupted income
- divorce
- entrepreneur
- irresponsible debtor
- homeowner
- slimy middle class manipulator

Canons of Interpretation

- General language in a statute, although broad enough to include a particular matter, will not be held to apply to it if specifically dealt with in another part of the same statute.
- To include one thing is to exclude another
- Language of a provision should be taken in the context of a statute as a whole

Computational Legal Representation: Fronts for progress

- Procedural knowledge
  - Legal (re)search control knowledge
- Diffuse knowledge
  - Information need for argument

Argument Representation

- Stereotypical forms of argument
- Information needs for argument

Argument Pieces

- best cases
- leading cases
- supporting cases
- cases sharing a large proportion of domain factors
- contrary best cases for the opposing viewpoint
- contrary cases supporting the opposing viewpoint
- dimensional analysis of the current problem
- supporting citations
- family resemblance prototype
- the factual prototype story category of the case
- applicable supporting legal theories
- nearly applicable supporting legal theories

Legal Research and Search Control Knowledge

- Learning search control knowledge
- Evaluation function
  - What is the correct feature level?

A Problem?

- Evaluation of arguments
  - AI and the superhuman fallacy

The Flip Side of Representation

- Evaluation

A Formal Model of Legal Argumentation

GIOVANNI SARTOR

Abstract. The paper gives a formal reconstruction of some fundamental patterns of legal reasoning, intended to reconcile symbolic logic and argumentation theory. Legal norms are represented as unidirectional inference rules which can be combined into arguments. The value of each argument (its qualification as justified, defensible, or defeated) is determined by the importance of the rules it contains. Applicability arguments, intended to contest or support the applicability of legal norms, preference arguments, purporting to establish preference relations among norms,, and interpretative arguments are also formalised. All those argument types are connected in a unitary model, which relates legal reasoning to the indeterminacy of the legal system, intended as the possibility to derive incompatible defensible arguments. The model is applied to permissive norms and normative hierarchies, and is implemented in a Prolog program.

The notion of a *formal legal argumentation* seems to be self-contradictory: For a long time the two "logics" of legal reasoning — formal symbolic logic and argumentation theory — have been developing separately, in a reciprocal incomprehension, if not in an open clash.

Scholars following a formal approach have privileged the search for correctness, controllability, and certainty, and have therefore stressed the lack of rigour and the indeterminacy of the theories of argumentation. Argumentation has been reduced to logical deduction or non-deductive argumentation has been confined in a "heuristic" moment intended to prepare or control the premises of logical deduction.

The theorists of argumentation have instead emphasized the conflict of opinions, the evaluation of alternatives, and the performance of reasonable choices. They have therefore condemned symbolic logic for its incapacity to capture these fundamental aspects of moral and legal reasoning.

Some authors have attempted to avoid the conflict between logic and argumentation by distinguishing the internal justification of the legal decision, mainly intended as its deduction from consistent legal axioms, and the external justification of those axioms, in which informal argumentative procedures play a major role. This last conception seems *prima facie* to preserve the rights of both logic and informal argumentation, distinguishing their competencies, but may on the contrary motivate a minimisation of the role of formal methods: since deductive procedures (and therefore the "internal justification" as intended by those authors) find a very limited application in legal reasoning, the most significant moments of legal reasoning are pushed into the indeterminate domain of external justification.

The alternative between logic and argumentation must instead be overcome by extending formal methods outside the domain of deduction, to the moments of dialectical conflict — and therefore of choice and evaluation — which characterise legal and moral reasoning. For this purpose we need a model of legal reasoning satisfying both requirements of formality and isomorphism, which seem respectively to express fundamental motivations of symbolic logic and argumentation theory:

- *Formality.* The model must be expressed in a formal language, defined by a compositional syntax, and over which formal inference procedures (and, possibly, formal semantics) are established.
- *Isomorphism.* The model must not be too far from the intuitive, natural language based, legal argumentation. We can distinguish two aspects of isomorphism: (*a*) *structural isomorphism*, meaning that the formal language must preserve the typical linguistic structures of legal language (Bench-Capon and Cohen 1992), and (*b*) *procedural isomorphism*, meaning that the formal inferential procedures must be easily mappable into typical forms of legal argumentation.

Both formality and isomorphism are necessary in a *legal logic* intended, *sensu largo*, as a formalism to analyse and evaluate legal arguments (and to support them by means of computing procedures).

Formality favours analysis: It grants rigour and precision in representing legal arguments and allows analytical results to be extended and checked by syntactical computation. Formality is equally important in a normative prospective: It facilitates the exact evaluation, according to the proposed model, of real argumentation patterns.

Isomorphism also facilitates analysis: A model clarifies the structure of legal reasoning so far as it mirrors this structure. Isomorphism is equally relevant for normative models, which can influence legal reasoning only insofar as they embody the typical patterns of the latter: Effective rationality patterns for legal argumentation cannot be too distant from legal praxis, and from the ideals that the participants associate to that praxis.

1. A formal characterisation of legal argumentation

We will now specify a formalism for legal reasoning intended to satisfy both requirements of formality and isomorphism, and so define the core of a model of legal argumentation. This formalism is the first step of an ongoing work, but it already offers some valuable results: It intuitively represents complex legal contexts, it can follow the dynamics of legal disputes, and it simply conceptualizes some controversial problems of legal theory.

1.1. Reasons and argumentation frameworks

If legal argumentation is a discourse in which — as Alexy (1992) recalls, citing the German Constitutional Court — "reasons are put forward, other reasons are opposed, and finally the better reason should determine the decision", its atomic components must be *reasons*, generally intended as any statements put forward in course of an argumentation. We represent reasons as *inference rules*, which have the form A **if** B, and ground the derivation of the conclusion A whenever the condition B is satisfied. The formal property of inference rules — a concept subsuming notions independently developed in different domains, such as the warrants of Toulmin (1957) and the defaults of Reiter (1980) — is mono-directionality. They can be used only forward (*modo ponente*) and not backward (*modo tollente*): The consequent A can be derived whenever the antecedent B is satisfied, but the negation of B cannot be derived when A is assumed to be false. Therefore the **if** connective in inference rules must be distinguished from logical conditional.

We also admit categorical (or degenerate) inference rules, which allow the unconditioned derivation of any instance of their conclusion A. Categorical inference rules express different types of ungrounded assertions: (*a*) statements of undisputed empirical evidence (facts); (*b*) basic and very general normative postulates; (*c*) tentatively advanced propositions for which no ground is currently available[1].

A simple type of inference rule is here used: The consequent of each rule is a literal and the antecedent is a conjunction of literals[2].

Def. 1. *Inference rules.* An *inference rule* is a formula of the type $<r: p_0$ **if** p_1 **and** ... **and** $p_n>$, where r is the rule name, each p_i is a literal, **if** is the connective relating

[1] Some formalisations of argumentation, such as Simari and Loui (1992) and Prakken (1992), distinguish inference rules and other types of information, such as necessary and contingent knowledge. To simplify our model we will not go into these distinctions.

[2] A literal is an atomic formula or the negation of an atomic formula. More exactly, a positive literal has the form $p(x)$, where p is a predicate symbol and x is a list of terms, and a negative literal has the form **not** $p(x)$, where **not** is logical negation. The *complement* $\overline{q}$ of a literal q denotes the literal opposed to q: if q is a positive literal p, then $\overline{q}$ represents the negative literal **not** p: if q is the negative literal **not** p, then $\overline{q}$ represents the positive literal p. In the paper, to make logical formulae more readable, predicates are represented as sequences of words, and terms appear inside the corresponding predicates, in italics.

antecedent and consequent in inference rules, **and** expresses logical conjunction. Categorical inference rules have simply the form $<r: p>$, where p is a literal.

Rule names are required to express meta-norms (meta-reasons). While rule names are expressed in natural language with such expressions as "art. 2043 of the Italian Civil Code", a rule name r has in our formalism the form $<f(x_1, \ldots, x_n)>$, where f is a new function symbol and $x_1, \ldots, x_n$ are the free variables in the rule. Substituting $x_1, \ldots, x_n$ with appropriate terms $t_1, \ldots, t_n$ we obtain names $f(t_1, \ldots, t_n)$ for all instances of the rules. For example, the name *PrivacyProtection*(*PhotoByJohn, Mary*) denotes the application of the abstract rule *PrivacyProtection*(*x, y*) to the individuals (the photo and Mary) whose names have been substituted for the variables occurring in the name of the abstract rule.

Inference rules containing variables must be interpreted as general assumptions, in which quantifiers are implicit (universal closures), or better, as shorthands for all ground instances of those rules (all instances obtained by substituting individual names for variables). Here are a general norm and a corresponding ground instance:

PrivacyProtection(*x, y*): it is **not** permitted to publish *x* **if** *x* concerns the privacy of *y*.
PrivacyProtection(*PhotoByJohn, Mary*):
it is **not** permitted to publish *PhotoByJohn* **if** *PhotoByJohn* concerns the privacy of *Mary*.

The categorical rule F_1 expresses a fact:

F_1: *PhotoByJohn* concerns the privacy of *Mary*.

The significance of a reason (an inference rule) in an argumentation, is determined by its role in the set of the reasons so far put forward, *i.e.*, in the available *argumentation framework*. Argumentation frameworks do not need to be consistent, and will not be consistent, in all those cases in which a real discussion, involving reasons and counter-reasons, has taken place. We can for example imagine that Mary asserted — in the debate concerning the publication of a photo taken by John without her consent — that her personal image concerns her privacy and is therefore not to be published, while John proposed two objections: (*a*) being Mary a well-known person, her photo is of public interest, and therefore can be published, (*b*) in any case the photo was taken is an open public place, and therefore does not concern the privacy of Mary. The set of the proposed reasons can be formalised in the following argumentation framework Φ_0:

Φ_0={*PublicationLiberty*(*x*): it is permitted to publish *x* **if** *x* is of public interest;
PrivacyProtection(*x, y*): it is **not** permitted to publish *x* **if** *x* concerns the privacy of *y*;
ImagePrivacy(*x, y*): *x* concerns the privacy of *y* **if** *y*'s image occurs in *x* **and** *y* is recognisable in *x*;
PublicPlace(*x, y*): *x* does **not** concern the privacy of *y* **if** *x* concerns *y*'s behaviour in an open public place;
F_1: *PhotoByJohn* is of public interest;
F_2: *Mary*'s image appears in *PhotoByJohn*;
F_3: *Mary* is recognisable in *PhotoByJohn*;
F_4: *PhotoByJohn* concerns *Mary*'s behaviour in an open public place}.

1.2. Derivability

Our limited language offers a simple notion of *derivability*, centred on the notion of inference rule: The consequent of (any ground instance of) an inference rule r in a rule set Σ can be derived from Σ when all conjuncts in the antecedent of r have already been derived from Σ. Obviously, the content of any categorical inference rules can be unconditionally derived.

Def. 2: *Derivation.* A *derivation* of p_0 from a rule set Σ is a finite sequence such that p_0 is the last element of the sequence, and every preceding element is the consequent q_0 of a rule (instance) $<r: q_0$ **if** p_1 **and** ...**and** $p_n>$ in Σ such that $p_1, \ldots, p_n$ precede q_0 in the sequence.

Def. 3: *Derivability*. A statement p_0 is *derivable* from a rule set Σ iff there exists a derivation of p_0 from Σ. In such a case we also say that Σ *argues for* p.

Def. 4: *Argument*: An *argument for* p in argumentation framework Σ is a consistent minimal subset A_1 of (the ground instances of the rules in) Σ such that A_1 argues for p.

Arguments can contain any specifications (ground instances) of general rules. For example, by combining an instance of rule *PublicationLiberty*, and fact F_2, we obtain argument B_1 for the permission to publish Mary's photo:

B_1={*PublicationLiberty*(*PhotoByJohn*): it is permitted to publish *PhotoByJohn* if *PhotoByJohn* is of public interest;
F_1: *PhotoByJohn* is of public interest}.

B_1 allows the derivation D_1:

D_1={*PhotoByJohn* is of public interest; it is permitted to publish *PhotoByJohn*}.

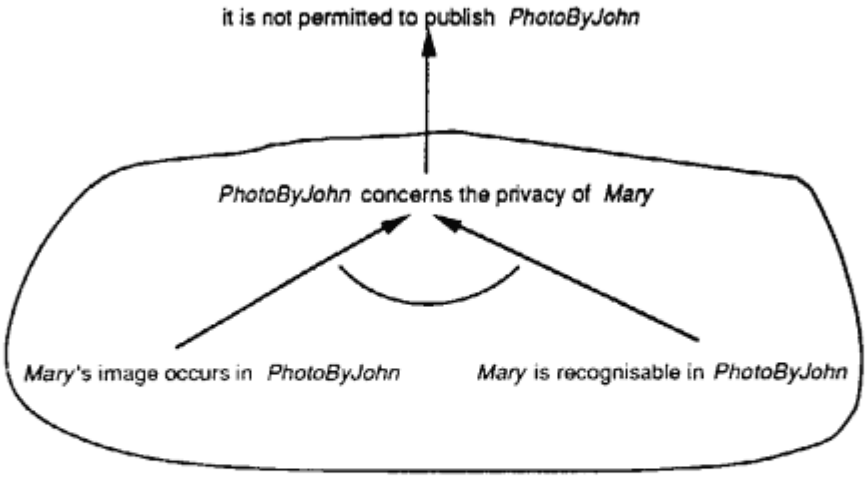
Let us now consider the two requirements of arguments, minimality and consistency:

- The minimality requirement eliminates redundant information, allowing only the rules necessary to derive the conclusion of the argument. Minimality is no real restriction, since any set of reasons for a conclusion can be reduced to a minimal set, simply by discarding redundant assumptions.
- The consistency requirement corresponds to the fundamental rationality criterion according to which a reasoner should not contradict himself (Alexy [1978] 1991, 235f). This requirement is applied to single arguments, and therefore only to specific (ground) rule-instances.

Derivations can be represented as trees, to emphasize the inferential process for reaching the conclusion of the argument. The top node of the tree denotes the conclusion of the argument; each intermediate node is an intermediate result necessary to complete the derivation. The combination of an upper node and the set of the lower nodes related to it by arrows denotes an inference step, that is the elementary inference consisting in deriving a rule consequent (the upper node) when all conditions in the rule antecedent (the lower nodes) are met. The argumentation framework Φ_0, besides the argument B_1 for the permission to publish *PhotoByJohn*, includes the following argument B_2 for the prohibition to publish that photo, which grounds the derivation of Figure 1.

B_2={*PublicationLiberty*(*PhotoByJohn*, *Mary*):
it is not *permitted to publish PhotoByJohn* if *PhotoByJohn* concerns the privacy of *Mary*;
PrivacyProtection(*PhotoByJohn*, *Mary*): *PhotoByJohn* concerns the privacy of *Mary* if
Mary appears in *PhotoByJohn* and *Mary* is recognisable in *PhotoByJohn*;
F_2: *Mary* appears in *PhotoByJohn*;
F_3: *Mary* is recognisable in *PhotoByJohn*}.

Figure 1. *A derivation and a sub-derivation from argument B_2*



1.3. Subargument

To reach the conclusion of an argument intermediate conclusions may have to be drawn. Each one of those conclusions is the final point of a minor argument, that we call *subargument.*

Def. 5: *Subargument.* A_1 is a *subargument* of A_2 iff A_1 is included in A_2 ($A_1 \subseteq A_2$).

Def. 6: *Strict subargument.* A_1 is a *strict subargument* of A_2 iff A_1 is strictly included in A_2 ($A_1 \subset A_2$), *i.e.*, iff A_1 is included in, but not equal to, A_2.

For example, to reach the conclusion of argument B_2 — the prohibition to publish the photo — we need first to establish that the photo concerns Mary's privacy. This conclusion is inferred by the following strict subargument B_3 (the corresponding sub-derivation is circled in figure 1).

B3={*ImagePrivacy(PhotoByJohn, Mary)*: *PhotoByJohn* concerns the privacy of *Mary* if
Mary's image occurs in *PhotoByJohn* and *Mary* is recognisable in *PhotoByJohn*;
F2: *Mary*'s image occurs in *PhotoByJohn*;
F3: *Mary* is recognisable in *PhotoByJohn*}.

1.4. Opposing argument and counterargument

In our model, consistency is required in single arguments, not in the whole argumentation framework. Argumentation is in fact required when a choice in needed, since there is a contrast of interests, theses, and points of view: Conflicting reasons are usually put forward in any argumentation framework. To conceptualise the notion of conflict precisely, we need to consider that an argument can be questioned not only by contesting its conclusion, but also by contesting any previous inference step. We need therefore to distinguish two concepts: *opposing argument* and *counterargument*. To oppose an argument is to contradict its conclusion.

Def. 7: *Opposing argument.* A_1 *opposes* A_2 iff A_1 concludes for p and A_2 concludes for $\overline{p}$. In such a case, we also say that A_1 and A_2 are *opposed.*

A counterargument, instead, does not need to oppose the ultimate conclusion of the attacked argument: It may also oppose any intermediate inference step (any subargument) in that argument.

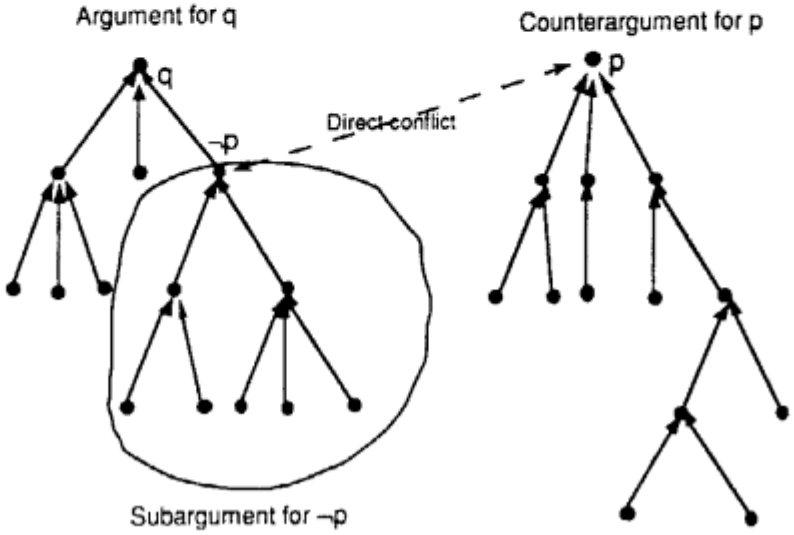
Def. 8: *Counterargument.* A_1 *counterargues* A_2 iff A_1 *opposes* any subargument of A_2.

B_1 and B_2 above are, *e.g.*, opposed arguments, while B_4

B_4=[*PublicationLiberty(PhotoByJohn, Mary)*:
PhotoByJohn does not concern the privacy of *Mary* if
PhotoByJohn concerns *Mary*'s behaviour in an open public place]

opposes the subargument B_3 of B_2 and therefore counterargues B_2, although not opposing B_2. Non-opposing counterarguments express preliminary or "prejudicial" objections. In Figure 2 we can see the relation between an argument and its preliminary counterargument.

Figure 2. *Derivations for an argument and a preliminary counterargument*



1.5. The comparison of arguments

Opposed arguments determine perplexity: They push the reasoners towards incompatible conclusions. To exit perplexity a choice is required, which may be greatly simplified by the following assumptions:

- Only opposed arguments must be compared. The relevance of a prejudicial counterargument must be assessed comparing it to the opposed strict subargument.
- The comparison of the opposed arguments can be reduced to the comparison of their top rules.

Both assumptions correspond to the intuition that derivations are developed step by step: When a derivation step has been performed, and it is able to sustain all possible critiques, it cannot be further questioned because of consequences derived by applying other inference rules to its conclusion[3].

To solve argument conflicts we need to express preference relations between rules. For the moment we state only categorical and consistent preference assertions, represented in the form $<p: r_i$ is preferred to $r_j>$, where p is the name of the preference assertion, and r_i and r_j are the compared rules. To specify the relation between opposed argument, we first need the notion of top rule, intended as the rule establishing the conclusion of the argument. An argument for p cannot contain more than one rule instance establishing p, since arguments are minimal.

[3] On the intuition behind this assumption, cf. Sartor (1993, 193) and Prakken (1992). A different approach would consist in requiring that the succumbing the argument contains a rule inferior to all rules in the opposed one. This approach would come near to the models of Alchourrón (1986) and Alchourrón and Makinson (1981).

Def. 9: *Top rule.* The *top rule* of an argument A_1 for p is the only rule r such that r is contained in A_1 and r has the consequent p.

The conflict between opposed arguments is to be decided according to the comparative strength of their top rules.

Def. 10: *Prevailing argument.* A_1 *prevails over* A_2 iff the top rule of A_1 is preferred to the top rule of A_2.

Def. 11: *Succumbing argument.* A_1 *succumbs under* A_2 iff A_2 prevails over A_1.

For example, if we add to the argumentation framework above the following preference:

P_1[*PrivacyProtection*(x, y), *PublicationLiberty*(x, y)]: *PrivacyProtection*(x, y) is preferred to *PublicationLiberty*(x, y),

we can than easily establish that B_1 and B_2 are opposed (B_1 argues for "it is permitted to publish *PhotoByJohn*" while B_2 argues for "it is not permitted to publish *PhotoByJohn*") and that B_1 prevails over B_2 (the top rule of B_1, *PrivacyProtection*(*PhotoByJohn*), is preferred to the top rule of B_2, *PublicationLiberty*(*PhotoByJohn*, *Mary*).

1.6. The relative strength of arguments

Argument frameworks can be abstracted from their human and social environments — *contra*, cf. Perelman (1979, 6) — although their content depends on the initiative of the involved parties: The parties introduce their reasons in the existing argument framework and can always provide further reasons, but in any state of the argumentation only the so far proposed reasons establish the merit of a claim. To determine the current value of an argument — i.e., its value in the current argumentation framework — we need concepts taking into account every possible interference among all arguments supported by that argumentation framework.

First we identify those arguments able to express relevant critiques: directly defeating and directly questioning arguments.

A *directly defeating argument* is able to invalidate the opposed argument (that we call *directly defeated*). It must not only prevail over the attacked arguments, but it must also be justified (overcome all critiques) in the whole argumentation framework.

Def. 12: *Directly defeating argument.* A_1 *directly defeats* A_2 iff: (*i*) A_1 opposes A_2, (*ii*) A_1 prevails over A_2, and (*iii*) A_1 is justified.

Def. 13: *Defeating argument.* A_1 *defeats* A_2 iff A_1 defeats any subargument of A_2.

Def. 14: *Defeated argument.* A_1 is *defeated* iff there is an A_2 such that A_2 defeats A_1.

A directly questioning argument is able to put into doubt, to make unsure the opposed argument (that we call *directly questioned*), without necessarily invalidating it. A directly questioning argument must not only sustain the comparison with the opposed argument (it must not succumb under the latter), but must also be defensible (capable to stand all critiques) in the whole argumentation framework.

Def. 15: *Directly questioning argument.* A_1 *directly questions* A_2 if: (*i*) A_1 opposes A_2, (*ii*) A_1 does not succumb under A_2, and (*iii*) A_1 is defensible.

Def. 16: *Questioning argument.* A_1 *questions* A_2 iff A_1 directly questions a subargument of A_2.

Def. 17: *Questioned argument.* A_1 *is questioned* iff there exists an A_2 such that A_2 questions A_1.

The capability to stand possible critiques distinguished arguments into justified, defensible, and defeated ones.

A justified argument indicates a sure or justified conclusion, since it overcome any counterargument, *i.e.*, it is not questioned.

Def. 18: *Justified argument.* An argument is *justified* iff it is not questioned.

A defensible argument is able to indicate a possible or defensible conclusion, since it withstands all counterarguments (although it does not necessarily overcome all of them), *i.e.*, it is not defeated.

Def. 19: *Defensible argument.* An argument is *defensible* iff it is not defeated.

The notions of justified and defensible arguments specify the concept of merely defensible argument:

Def. 20: *Merely defensible argument.* An argument is *merely defensible* iff it is defensible but not justified.

A merely defensible argument A_1 clashes against a counterargument A_2 such that A_2 questions A_1, but does not defeat it. This may happen because the top rule of A_2 does not prevail over, nor succumbs under, the top rule of the opposed subargument of A_1, or because a subargument of A_2 is also questioned.

The notions so far introduced allow us to distinguish different classes of legal conclusions:

Def. 21: *Logical conclusion.* A literal p is a *logical conclusion* from an argumentation framework Π iff p is a logical consequence of Π. The notion of logical conclusion is of very little use, since, as we have seen, argument frameworks are normally inconsistent and from inconsistent premises any arbitrary logical consequence can be deduced. Every literal is therefore a logical conclusion from any inconsistent Π.

Def. 22: *Supported conclusion.* A literal p is a *supported conclusion* from an argumentation framework Π iff Π includes an argument for p. All supported conclusions are therefore sustained by some argument, which, nevertheless, may be overridden by counterarguments.

Def. 23: *Defensible conclusion.* A literal p is a *defensible conclusion* from Π iff Π includes a defensible argument for p. Defensible conclusions are thus supported by arguments that are not worse (but not necessarily better) then the corresponding counterarguments. They include all supported conclusions, except those attacked by justified counterarguments.

Def. 24: *Justified conclusion.* A literal p is a *justified conclusion* from Π iff Π includes a justified argument for p. Justified conclusions are supported by arguments that are better that any counterargument. They include all defensible conclusions, except those attacked by defensible counterarguments.

Let us now apply to the argumentation framework Φ_0 above the notions just introduced. If we do not introduce any preference assertion in Φ_0, we have merely defensible arguments for both the permission and the prohibition to publish the photo, to wit respectively B_1 and B_2. Those arguments are merely defensible, since they question one another: They are opposed and there is no preference relation between their top rules. B_2 is also questioned by the prejudicial counterargument B_4, which also is merely defensible, being unable to prevail over the opposed subargument B_3. Let us now also assume the following preference:

P_2[*PublicPlace*(x, y), *ImagePrivacy*(x, y)]: *PublicPlace*(x, y) is preferred to *ImagePrivacy*(x, y)

stating that publicity of the behaviour in open public places prevails over privacy concerning personal image. This preference relation makes B_4 prevail over B_3, and thus defeat B_3 (since B_4 has no other counterargument). Therefore also B_2, including B_3, is defeated, and B_1, freed from its only counterargument, becomes justified. We can therefore derive the justified conclusion that the photo can be published.

1.7. Norms as inference rules

In the model here proposed, legal norms are not always able to produce their effect when the provided conditions are met. Their antecedent only normally or defeasibly justifies their consequent, where "normally" or "defeasibly" means "unless a relevant (defensible or justified) counterargument can be raised".

Defeasibility is a precondition of argumentation: Only if reasons are defeasible can they be discursively opposed and balanced. Defeasibility is also required in monological legal (and moral) reasoning, which is also centred on adjudication between competing reasons: Concrete situations can have many legally (or morally) relevant aspects, expressing conflicting interests protected by the law, so that the overall judgement resulting from the comparison and balancing of all those interests cannot be required from each single norm — that separately considers a single aspect and suggests a corresponding legal solution — but can instead be achieved only by the comparative evaluation of the competing defeasible norms and arguments[4].

Against the formalisation of norms as defeasible inference rules, it may be objected that the antecedent of each norm can be extended with the negations of all possible circumstances preventing the effect of the norm. In this way a perfect conditional norm can always be obtained, the consequent of which is "legally true" whenever its antecedent is satisfied. Against this approach, a number of critiques may be raised:

- The *structural isomorphism* objection. Statements included in legal texts are not perfect conditional norms. Even when there is no doubt on the legal effect established for each specific situation, legal language prefers to use separate norms: a defeasible rule stating the general discipline, and exceptions opposing that discipline under special conditions.
- The *combinatorial explosion* objection. Each perfect conditional norm requires an absolute casuistic precision: Its antecedent must be extended with all the possible (coexisting or alternative) situations susceptible to exclude the realisation of its effect. This style of norm formulation leads to an enormous number of norms, each with an endless antecedent.
- The *deep representation* objection. Legal norms are intended to regulate and balance conflicting interests. The distinction between the separate provisions concerning each of those interests on the one hand, and the criteria for their balancing on the other, emphasizes the substantial motivation behind every legal norm, that would be obscured when one single normative statement expressed the result of all comparative evaluations.
- The *indeterminacy* objection. Perfect conditional norms assume that all normative conflicts have been definitely solved: Only in this case — by conveniently extending the condition of the weaker norms — can we obtain a consistent set of logical conditional. As we shall see in the following, it is true that legal decision makers are obliged to solve normative conflicts, but this task should not be transformed into a false postulate (the consistency of legal systems).
- The *defeasible inference* objection. The conclusion of a defeasible inference rule whose condition is satisfied, can be blocked only by a counterargument, which must be included in the available argumentation framework. If the argumentation framework does not include such a counterargument, we are authorised to derive the conclusion of the defeasible rule. This pattern corresponds to a general feature of legal reasoning (*a*) a legal conclusion must be drawn whenever the facts grounding it have been proved, even if the facts impeding it have not been disproved, but (*b*) if those last facts (the so-called exceptions, or impedient facts) are proved, the legal conclusion must be retracted. Instead, the conclusion of a perfect conditional norm can be derived only when the proof of all elements in the (conjunctive) antecedent of the norm is given, included the proof of the negations (*i.e.*, the disproof) of all impeding facts[5].

The acknowledgement of the defeasibility of all norms tends to eliminate the distinction between "rules" *stricto sensu* and principles, or at least to overcome its conceptualization as a structural, or logical, difference[6]. In our model all norms are amenable to the logical treatment

[4] On partial, and therefore defeasible, moral evaluations, cf. W.D. Ross (1930; 1939).

[5] On the relation between exceptions and perfect conditional norms, cf. Sartor (1991).

[6] In this sense, cf., among others, Dworkin (1977, 24ff) and Alexy (1985, 71ff). Analyses of the historical evolution of the concepts of rule and principle — such as Pattaro (1988) — suggest much less clear-cut distinctions.

usually reserved to principles: They are reasons that may be defeated by prevailing considerations to the contrary, but are not irrelevant, since their conclusions must be accepted so far as these considerations are not proposed and supported in the current argumentation framework. We can therefore understand the relative certainty of legal conclusions, and reject both the illusion of an absolute security and the fall on a compete scepticism.

It is true that rules (but also principles, however they are defined) can be strengthened by extending their antecedents with the negations of their exceptions, but this extension just expresses a comparative evaluation in which the not yet extended rule has been defeated. This is confirmed by the logical properties of exceptions (just remarked in the defeasible inference objection above): The disproval of all exceptions is not necessary to derive the consequent of the norm, but the proof of an exception is sufficient to block this derivation. When we say "*p* **if** *q* **unless** *s*", where *s* is an exceptional situation, we do not mean "*p* **if** *q* **and not** *s*" (where **not** *s* means that *s* is false) but we rather synthesize the rule *p* **if** *q*, and the meta-assertion that this last rule is not to be applied under the condition *s*, in which some prevailing norm establishes a conclusion incompatible with *p* (or with the derivation of *p* by means of the involved rule). Until *s* is not proved (derived), we are not able to argue for that incompatible conclusion, and therefore we still can defeasibly derive *p*.

1.8. Applicability and undercutting arguments

According to Toulmin (1958, 93ff 103ff), in the structure of every argument we can distinguish two components: the *warrant* (the inference rule itself), and its *backing* (the substantial ground supporting that rule). In our formalism, in which rules are given names, backings can be driven back to arguments: A backing is simply an argument pleading for the application of a certain inference rule[7]. Besides arguments for applying an inference rule, there may also be arguments against its application. Both types of arguments are here called *applicability arguments*, and are characterised by culminating into *applicability rules*, that is in meta-rules asserting that other rules are not (or are) applicable.

Def. 25: *Applicability rule*. An *applicability rule* is an inference rule, the consequent of which has the form <*r* is applicable> or <*r* is not applicable>, where "*r*" is a rule name.

Here comes the question whether every rule must previously be shown to be applicable, or whether it may be used, insofar as no argument for its disapplication emerges. The latter solution seems preferable, since otherwise a *regressum ad infinitum* would take place, in search of backings, backing for backings, and so on.

Inapplicability rules require that the *oppose* relation is generalized so that it includes not only rebutting arguments (those contradicting the conclusion of the opposed argument), but also undercutting arguments (those pleading for the inapplicability of the top rule of the opposed argument)[8]:

Def. 26: *Rebutting argument*. A_1 *rebuts* A_2 iff A_1 concludes for p and A_2 concludes for $\overline{p}$.

Def. 27: *Undercutting argument*. A_1 *undercuts* A_2 iff A_1 concludes for "*r* is not applicable" where r is the top rule of A_2.

Def. 7bis: *Opposing argument (new definition)*. A_1 *opposes* A_2 iff: (*i*) A_1 rebuts A_2 or (*ii*) A_1 undercuts A_2.

To illustrate the expressiveness of the linguistic structures just introduced, let us consider an example from Italian tort law. Let us assume that Mary wants compensation from John, saying that he crashed her fence with his car. Here are the rules to be applied (corresponding to art. 2043, 2048, 2046 of the Italian Civil Code):

Ψ_0= {*FaultLiability*(*x*, *d*): *x* is liable for *d* if *x* has accomplished damage *d* by fault;

[7] The notion of backing is also considered in the formal argumentation model of Gordon (1993).

[8] For this terminology, cf. Pollock (1987).

ParentLiability(x, y, d): x is liable for y **if** x is a parent of y **and** y is liable for d;
Incapability(x, d): x **is not** liable for d **if** x was incapable while accomplishing d;
ImpossiblePrevention(x, y, d):
ParentLiability(x, y, d) **is not** applicable **if** x could **not** prevent his son y from accomplishing d;
CulpableIncapabily(x, d): *I*
Incapability(x, d) **is not** applicable **if** x was culpably incapable while accomplishing d;
P_1(x, d): *Incapability*(x, d) is preferred to *FaultLiability*(x, d);
P_2(x, y, d): *ImpossiblePrevention*(x, y, d) is preferred to *ParentLiability*(x, y, d);
P_3(x, d): *CulpableIncapability*(x, d) is preferred to *Incapability*(x, d)}.

1.9. Exceptions

Three types of legal statements can be found in the argumentation framework Ψ_0:

- *Basic norms*, such as *FaultLiability* and *ParentLiability*, which state substantial legal effect.
- *Rebutting exceptions*, such as *Incapability*, which establish that a certain legal effect does not hold under certain conditions, and rebut (all) norms establishing the negated effect;
- *Undercutting exceptions*, such as *ImpossiblePrevention* and *CulpableIncapability*, which indicate that a certain norm is not to be applied under certain conditions. Those exceptions only concern the undercut norm, and do not prevent its effect from being established by means of other rules.

Usually, exceptions are not "taken seriously" in legal theory: Even those authors going beyond imperativistic models — such as Larenz (1992, 145 ff) — consider exceptions as parts or fragments of norms, to be integrated into the norms to which they refer. Against this reduction we must again recall the objections indicated above, and especially the defeasible inference objection: The conclusion of the rule derogated by an exception, must also be derived when there is no information about the exceptions (Sartor 1993). All exceptions are here separate inference rules, which prevail over the opposed (rebutted or undercut) norms. Arguments culminating in exceptions defeat arguments culminating in the opposed norms — when the exception arguments are not themselves questioned or defeated by their own counterarguments — and so block the derivation of the consequents of those norms.

1.10. Non-monotonic reasoning in law

The combination of rules and exceptions contributes to a fundamental characteristic of legal reasoning, *non-monotonicity*. A reasoning is called *monotonic* when it satisfies the following condition: if it derives a statement p from a set of premises Π_1, then it also derives p from every super-set Π_2 of Π_1, obtained by adding further premises to Π_1. Logical deduction is typically monotonic: No logical consequence gets lost by extending the premises from which it has been deduced. In *non-monotonic* reasoning, instead, a statement p derivable from Π_1 may not be derivable from the super-set Π_2. Non-monotonicity allows us to understand the dynamics of legal debates, in which new information may invalidate conclusions provisionally derived[9]. The reasoning model here proposed is non-monotonic, since new information may render new arguments possible, which may defeat previously valid ones. For example, let us extend Ψ_0 with the following assertions:

F_1: *John* has accomplished the damage *FenceBreak* by fault;
F_2: *Mark* is a parent of *John*.

[9] Non-monotonic logics (Ginsberg 1987; Brewka 1991) offer precise formalisations of defeasibility, a concept fundamental in legal reasoning, but so far only occasionally and informally considered in legal theory (Hart 1949; Baker 1977; Raz 1978), although some fundamental legal-theoretical problems, such as the distinction between rules and principles, seem reducible to it. Nonmonotonic logics also solves some puzzling logical problems discussed by legal theorists, such as Weinberger (1975), who contested the application of the *falsum* rule, and Philipps (1966), who tried to deal with rules and exceptions using intuitionistic logic. On nonmonotonic reasoning in law, cfr. Gordon (1988, 1993), Jones and Pörn (1991), Hage (1993), Prakken (1992, 1993), Sartor (1991, 1992, 1993).

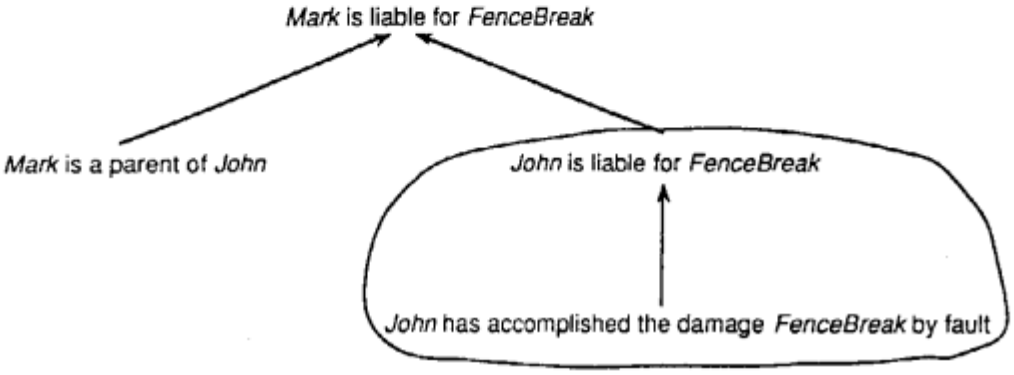
The resulting argumentation framework Ψ_1 ($\Psi_1 = \Psi_0 \cup \{F_1; F_2\}$) offers justified arguments for the liability of both John and Mark. The argument C_1 for John's liability is the following:

C_1 = {*FaultLiability*(*John*, *FenceBreak*):
John is liable for *FenceBreak* if *John* has accomplished the damage *FenceBreak* by fault;
F_1: *John* has accomplished the damage *FenceBreak* by fault}.

The argument C_2 for Mark's liability (see in Figure 3 the corresponding derivation) is obtained by adding to C_1 (whose sub-derivation is circled in Figure 3) the fact F_2 and the rule instance:

ParentLiability(*Mark*, *John*, *FenceBreak*):
Mark is liable for *FenceBreak* if *Mark* is a parent of *John* and *John* is liable for *FenceBreak*.

Figure 3. *The derivation of parent liability by argument* C_2, *including subargument* C_1.



Mark can free himself from liability by satisfying the antecedent of an exception, *e.g.*, by proving that John was incapable at the moment of the accident:

F_3: *John* was incapable while accomplishing *FenceBreak*.

In the argument framework $\Psi_2 = \Psi_1 \cup \{F_3\}$ it is possible to develop the following argument C_3, counterarguing both C_1 and C_2:

C_3 = {*Incapability*(*John*, *FenceBreak*):
John is **not** liable for *FenceBreak* if *John* was incapable while accomplishing *FenceBreak*;
F_4: *John* was incapable while accomplishing *FenceBreak* }.

The conclusion of C_3, that is <*John* is **not** liable for *FenceBreak*> is derived by means of the exception *Incapability*, and this conclusion contradicts the assertion of John's liability, in which C_1 culminated, thanks to the rule *FaultLiability*. *Incapability* prevails over *FaultLiability* (thanks to the preference assertion P_1) and consequently C_1 is directly defeated. Also C_2, which includes C_1 (C_1 is a subargument of C_2), is defeated: C_3 prevents the derivation of Johns liability and so prevents a necessary precondition of the liability of John's father.

Let us now assume that Mary asserts that John's liability was due to his fault (*e.g.*, his incapability derived from drunkenness):

F_4: *John* was culpably incapable while accomplishing *FenceBreak*.

In the argument framework $\Psi_3 = \Psi_2 \cup \{F_4\}$, C_3 is undercut by:

C_4 = {*CulpableIncapably(John, FenceBreak)*:
Incapability(John, FenceBreak) is not applicable if
John was culpably incapable while accomplishing *FenceBreak;*
F_4: *John* was culpably incapable while accomplishing *FenceBreak*}.

C_4 defeats C_3(by the preference relation P_2) and consequently C_3 can no longer defeat C_1 and C_2: These last arguments recover their validity and again establish the liability of John and Mark as justified conclusions. Let us now assume that Mark affirms he could do nothing to prevent John's action: He surely did his best to bring up his son. More exactly, he affirms the following rule and fact:

GoodUpbringing(x, y, d): *x* could not prevent his son *y* from accomplishing *d* if *x* brought *y* up well;
F_5: *Mark* brought *John* up well.

In the normative context Ψ_4 so obtained ($\Psi_4 = \Psi_3 \cup \{GoodUpbringing, F_5\}$) Mark's liability fails, while John's liability remains. C_1 (concerning John) is no longer attacked by any valid counterargument, while C_2 (concerning Mark) is defeated by C_5:

C_5={*ImpossiblePrevention(Mark, John, FenceBreak)*:
Incapability(Mark, John, FenceBreak) is not applicable if
Mark could not prevent his son *John* from accomplishing *FenceBreak;*
GoodUpbringing(Mark, John, FenceBreak):
Mark could not prevent his son *John* from accomplishing *FenceBreak* if *Mark* brought *John* up well;
F_5: *Mark* brought *John* up well}.

Mary does not question that Mark has given a good upbringing to his son, but she denies the relevance of this fact: She affirms that only a specific hindrance to control John justifies the statement that Mark could not prevent the fact, and denies that such a hindrance exists in the concrete case.

PossibleControl(x, y, d): *x* could prevent his son *y* from accomplishing *d* if *x* was able to control *y;*
F_7: *Mark* was able to control *John.*

These additional premises transform Ψ_3 into Ψ_4 ($\Psi_4 = \Psi_3 \cup \{PossibleControl, F_7\}$), which offers a counterargument C_6, questioning C_5:

C_6={*PossibleControl(Mark, John, FenceBreak)*:
Mark could prevent his son *John* from accomplishing *FenceBreak* if *Mark* was able to control *John;*
F_7: *Mark* was able to control *John*}.

Let us assume that no element emerges allowing *PossibleControl* to prevail over *GoodUpbringing* or *vice versa*. We must then conclude that C_5 and C_6 — which contradict each other and have the same "strength" — are both merely defensible. Mark's liability, being contested by a questioned argument (C_5), is itself merely defensible.

1.11. Interpretation arguments

Mary could adopt a different strategy to contest the last argument of John (C_5): She could contest the *GoodUpbringing* rule affirming the inapplicability of that norm:

InapplicabilityGoodUpbringing(x, y, d): *GoodUpbringing(x, y, d)* is not applicable.

This move is sufficient to block — in the argument framework $\Psi_4 = \Psi_3 \cup$ {*InapplicabilityGoodUpbringing*} — the application of the *GoodUpbringing* rule, since John has

given no backing (no argument) for the applicability of that rule[10]. If John wants to maintain the rule, he must state, *e.g.*, the general principle that correct statutory interpretations constitute applicable rules and the interpretative assertion that *GoodUpbringing* constitutes an exact interpretation of art. 2048 of the Italian civil code, which is a statutory provision:

StatuteAuthority(x, y): x is applicable **if** x is the correct interpretation of y **and** y is a statutory provision;
InterpretativeThesis$_1(x, y, d)$: *GoodUpbringing*(x, y, d) is the correct interpretation of *2048ItalianCivilCode;*
StatutoryProvision$_1$: *2048ItalianCivilCode* is a statutory provision.

In the argumentation framework $\Psi_5 = \Psi_4 \cup \{StatuteAuthority, InterpretativeThesis_1, StatutoryProvision_1\}$ the applicability of the *GoodUpbringing* rule in the case at hand (involving John, Mark and a fence break) becomes the conclusion of the following argument:

C_7=[*StatuteAuthority*(*GoodUpbringing*(*John, Mark, FenceBreak*), *2048ItalianCivilCode*)):
GoodUpbringing(*John, Mark, FenceBreak*) is applicable if
GoodUpbringing(*John, Mark, FenceBreak*) is the correct interpretation of *2048ItalianCivilCode* and
2048ItalianCivilCode is a statutory provision;
InterpretativeThesis$_1$(*John, Mark, FenceBreak*):
GoodUpbringing(*John, Mark, FenceBreak*) is the correct interpretation of *2048ItalianCivilCode;*
StatutoryProvision$_1$: *2048ItalianCivilCode* is a statutory provision.

Argument C_7 may be assumed to prevail (by specificity) over the unconditioned assertion of the inapplicability of the *GoodUpbringing* rule, and therefore to defeat it[11].

InterpretativeThesis$_1$ also is an unconditioned assertion. It can be rebutted by an opposed interpretative argument C_8 based on the interpretative canon that only literal interpretations are (usually) correct, and the interpretative thesis that *ImpossiblePrevention* is not a literal interpretation:

LiteralInterpretationCanon(x, y): x is not the correct interpretation of y if x is not a literal interpretation of y.
InterpretativeThesis$_2(x, y, d)$:
GoodUpbringing(x, y, d) is not a literal interpretation of *2048ItalianCivilCode*.

C_8=[*LiteralInterpretationCanon*(*GoodUpbringing*(*Mark, John, FenceBreak*), *2048ItalianCivilCode*)):
GoodUpbringing(*Mark, John, FenceBreak*) is not the correct interpretation of *2048ItalianCivilCode* if
GoodUpbringing(*Mark, John, FenceBreak*) is not a literal interpretation of *2048ItalianCivilCode;*
InterpretativeThesis$_2$(*Mark, John, FenceBreak*):
GoodUpbringing(*Mark, John, FenceBreak*) is not a literal interpretation of *2048ItalianCivilCode*].

More generally, any statutory interpretation canon can be expressed as an interpretation rule.

Def. 28: *Interpretation rule.* An *interpretation rule* is a rule whose consequent has the form <r is the correct interpretation or y> or <x is not the correct interpretation of y>, where x is a norm and y is a legal text.

Def. 29: *Interpretation argument.* An *interpretation argument* is an argument whose top rule is the instance of an interpretation rule.

We cannot continue the discussion concerning interpretation arguments — on which, cf., among others, Wróblewski ([1969] 1983) and Gianformaggio (1987). Here, it is sufficient to remark that the model here proposed allows interpretative arguments to be combined with substantial ones, in the sense that arguments pleading for substantial assertions (such as John's

[10] We assume that undercutting rules normally prevail over the norms to which they refer. In next paragraph, after having introduced the notion of preference rule, we will formally state this principle.
[11] We will go into the complex problems concerning specificity comparison, which is here assumed as a primitive notion, but is analysed in a number of technical contributions to nonmonotonic reasoning, such as, for example Poole (1985), Nute (1988), Simari and Loui (1992), Geffner and Pearl (1992). Formal notions of specificity are applied to legal reasoning by Prakken (1992), Ryu and Lee (1992), and Loui et alii (1993).

liability) may be defeated, questioned, of supported through applicability arguments, depending upon interpretative arguments.

1.12. Preference arguments

In the previous example we have considered the dialectic of legal reasoning in a framework of established preference relations, where the dynamics of legal argumentation was determined by new facts and interpretative theses. In hard legal problems, nevertheless, comparative evaluations are the central issue: Preference assessments also have to be argumentatively justified. We will therefore extend our argumentation model to preference assessments, by defining the notion of *preference argument*, which will be characterised as the argument making use of a *preference rule*[12].

Def. 30: *Preference rule.* A *preference rule* is an inference rule whose consequent has the form $<r_1$ is preferred to $r_2>$ or $<r_1$ is **not** preferred to $r_2>$ where r_1 and r_2 are the compared rules.

Def. 31: *Preference argument.* A is a *preference argument* iff the top rule of A is a preference rule.

Conflicting preference-arguments — one asserting that r_1 is preferred to r_2, the other, that r_2 is preferred to r_1 — must again be solved by a meta-preference argument establishing a preference relation between the involved preference rules.

To take into account preference arguments, we need to modify the notion of prevailing argument slightly:

Def.10bis: *Prevailing argument.* A_1 *prevails over* A_2 iff A_1 *opposes* A_2 and *it is a justified conclusion* that the top rule of A_1 is preferred to the top rule of A_2.

Note that this definition requires that preference relations are established as justified conclusions, *i.e.*, by means of justified preference arguments. This allows arbitrary levels of meta-argumentation to be developed: The justification of a substantial argument A_1 may require a justified preference argument PA_1, establishing that the top rule of A_1 is preferred to the top rule of an opposed argument A_2; the justification of PA_1 may require a justified meta-preference argument PPA_1, establishing that the top rule of PA_1 is preferred to the top rule of an opposed preference argument PA_2; the justification of PPA_1 may require a justified meta-meta-preference argument $PPPA_1$, and so on.

Preference arguments allow us to argue about comparative evaluations. To show the power of this extension of our formalism, we will go back to a judgement of the German Constitutional Court, the "Lebach Urteil", illustrated in Alexy (1980). This decision concerned a television documentary about a serious crime, in which the names of the participants in the crime were mentioned and their photos shown. One of those participants (who had a minor role in the fact) affirmed that this programme violated his privacy and compromised the chances of his social rehabilitation, and therefore violated his personality right (right to the free development of the personality) protected by the German Constitution. The German Constitutional Court, as Alexy observes, grounds his decision in three steps:

a. The Court admits that in the present case a comparative evaluation is necessary since two conflicting constitutional provisions are applicable: the right to privacy, excluding the publication of private information, and the right to communication, granting the liberty of propagating information. Moreover, it observes that none of these rights is to be unconditionally preferred to the other: Only the particular circumstances of the single case allow a choice to be made.

[12] Here a simplified formalisation of the notion of preference is proposed, which is nevertheless sufficient for our purposes. The extension of preference assertions into preference rules was suggested to the author of the present contribution by F. Lachmayer.

b. It affirms that the interest of the public to be informed by television about crimes usually overcomes the serious violation of privacy regularly determined by television programmes concerning criminal facts. This predominance, nevertheless, is not to be recognised when those fact are no longer actual.

c. It concludes that, under this last circumstance (as in the Lebach case), showing a documentary concerning a criminal fact is not admissible if it can determine a new or additional prejudice to the author of the fact.

Our language allows a straightforward formalisation of the argument of the German Court.

a. Incompatible prima facie conclusions. In the first step the Court observes that *prima facie* two alternative conclusions are derivable from the constitutional rules. The corresponding argumentation framework Θ_0 can be represented as follows:

Θ_0={*PrivacyProtection*(*x*): *x* is not permitted if *x* violates privacy;
CommunicationLiberty(*x*): *x* is permitted if *x* is a form of communication;
PrivacyViolation(*x*): *x* violates privacy if
x is the television programme about a criminal fact mentioning the authors of that fact;
CommunicationForm(*x*): *x* is a form of communication if
x is a television programme about a criminal fact mentioning the authors of that fact;
F_1: *LebachProgramme* is a television programme about a criminal fact mentioning the authors of that fact}.

The rules *PrivacyProtection* and *CommunicationLiberty* express the constitutional evaluation of privacy and communication; the rules *PrivacyViolation* and *CommunicationForm* specify the constitutional rules in relation to the facts of the case: The television programme about a criminal fact, which indicates the authors of this fact, constitutes both a communication and a violation of privacy; F_1 is the basic fact of the case. Without any preference relation we are able to derive, as the Court did, two merely defensible conclusions: The programme is forbidden by the argument

E_1={*PrivacyProtection*(*LebachProgramme*): *LebachProgramme* is not permitted if
LebachProgramme violates privacy;
PrivacyViolation(*LebachProgramme*): *LebachProgramme* violates privacy if
LebachProgramme is a television programme about a criminal fact mentioning the authors of that fact;
F_1: *LebachProgramme* is a television programme about a criminal fact mentioning the authors of that fact}.

while it is permitted by the argument

E_2={*CommunicationLiberty*(*LebachProgramme*): *LebachProgramme* is permitted if
LebachProgramme is a form of communication;
CommunicationForm(*LebachProgramme*): *LebachProgramme* is a form of communication if
LebachProgramme is a television programme about a criminal fact mentioning the authors of that fact;
F_1: *LebachProgramme* is a television programme about a criminal fact mentioning the authors of that fact}.

b. First comparative evaluation. The instances of *CommunicationLiberty* that concern a television programme of the indicated type are usually preferred to the corresponding instances of *PrivacyProtection:*

CommunicationPreference(*x*): *CommunicationLiberty*(*x*) is preferred to *PrivacyProtection*(*x*) if
x is the television programme about a criminal fact mentioning the authors of that fact.

The preference rule *CommunicationPreference* strengthens argument E_2. More exactly the argumentation framework $\Theta_1 = \{\Theta_0 \cup CommunicationPreference\}$ offers a preference argument P_1 — including the rule instance *CommunicationPreference*(*LebachProgramme*) and fact F_1 — for <*CommunicationLiberty*(*LebachProgramme*) preferred to *PrivacyProtection*(*LebachProgramme*)>. This argument makes E_2 defeat E_1, and therefore

makes the conclusion of E_2 (the permission of the programme) justified. Nevertheless, the judges also affirmed that this preference only exists for programmes on present facts, *i.e.*, that *PrivacyProtection* is not applicable to programmes shown a long time after the concerned facts, as the Lebach documentary:

SubsequentFact(x):
CommunicationPreference(x) is **not** applicable if *x* was shown a long time after the concerned facts,
F_2: *LebachTransmission* was shown a long time after the concerned facts,

We also assume that undercutting rules are generally preferred to the rules to which they refer:

UndercuttingPreference(r_1, r_2): r_1 is preferred to r_2 if r_1 has the consequent "r_2 is **not** applicable".

$\Theta_2 = \{\Theta_1 \cup SubsequentFact,\ F_2,\ UndercuttingPreference\}$ includes a counterargument — containing the rule instance *SubsequentFact*(*LebachProgramme*) and the fact F_2 — that undercuts the preference assertion *CommunicationPreference*. Therefore, the conflict between *PrivacyProtection* and *CommunicationLiberty* remains undecided: Both contradictory legal qualifications of the Lebach programme remain merely defensible.

c. Second comparative evaluation. To reach a justified conclusion, a second comparative evaluation is needed, establishing that privacy prevails under the conditions of the case:

PrivacyPreference(x): *PrivacyProtection(x)* is preferred to *CommunicationLiberty(x)* **if**
x is a television programme about a criminal fact mentioning the authors of that fact **and**
x was shown a long time after the concerned facts **and**
x causes a new violation of privacy.

Let us add the last fact:

F_3: *LebachProgramme* causes a new violation of privacy.

In the argumentation framework $\Theta_3 = \{\Theta_2 \cup PrivacyPreference,\ F_3\}$, the preference argument P_2 — including the rule instance *PrivacyPreference*(*LebachProgramme*) and fact F_3 — argues for <*PrivacyProtection*(*LebachProgramme*) is preferred to *CommunicationLiberty*(*LebachProgramme*)>. P_2 allows E_1 to defeat E_2, and so justifies E_1's conclusion, to wit the non permissibility to show the Lebach documentary. Note that our formalisation immediately reflects the structure of the argumentation of the German Court, in a precise formalism.

2. The legal system as an argumentation framework

The model here proposed emphasizes the richness and the variety of the logical structures of legal language: all statements (obligations, permissions, authorisations, applicability rules, interpretation rules, preference rules, definitions, etc.) susceptible to being used in justifying legal conclusions are fully entitled legal norms. All those statements, although different in meaning, function, logical status (some of them, in particular, are meta-statements) can be used in a uniform way in deriving legal conclusion — *i.e.*, as inference rules — and interact reciprocally in the game of arguments and counterarguments.

Moreover, if we accept that the legal system contains general rules and exceptions, conflicting norms, principles expressing incompatible legal interest, and we take those aspects seriously, we must reject the traditional postulate of the consistency of the legal system, and consequently the image of the legal system as an axiomatic base, all whose logical implications should be accepted as (justified) legal conclusions. We must, instead, come to consider the legal system as an argumentation framework, that is as a repertory of material to be used (in combination with the ascertained facts) in the struggle of competing arguments and meta-arguments.

The vision of the legal system as a heterogeneous, stratified, and inconsistent argumentation framework does not exclude the importance of coherency standards: On the contrary, we need those standard to permanently rationalize incoherent legal systems. The most elementary of those standards, which could advantageously substitute the false postulate of consistency, is the ideal of *determinacy*. We can distinguish a *relative determinacy*, concerning single factual cases, and *absolute determinacy*, concerning all possible cases.

Def. 32: *Relative determinacy*. A legal system Σ is *determinate* relatively to a case K_0 iff the set $\Sigma \cup K_0$ has no merely defensible consequences.

Def. 33: *Absolute determinacy*. A legal system Σ is *absolutely determinate* iff, in every possible case K_i, the set $\Sigma \cup K_i$ has no merely defensible consequences.

A (relatively or absolutely) determinate legal system Σ is "certain", in the sense that every conclusion derivable from it is either justified of defeated: Σ can be applied without doubts (in the case K at hand, or in all possible cases). An indeterminate legal system, instead, is uncertain, in the sense that it offers defensible arguments *pro* and *contra* certain legal conclusions. Every indeterminate legal system originates inconsistent legal argumentation frameworks, when extended with the appropriate factual assertions, but the converse is not always true: Preference relations may establish determinacy in inconsistent argument frameworks.

Determinacy is a fundamental ideal for both legislation and legal argumentation. It cannot instead be considered an objective quality of every legal system, so hiding the creative nature of the activities intended to rationalize the legal system in order to approximate determinacy. To understand those activities, two attitudes towards the legal system must be distinguished:

a. An external (or realistic) point of view, that regards law as the set of criteria *de facto* used by legal decision makers. The external view is adopted by the "observer" who intends the describe the normative contexts in which legal decisions take place, or to anticipate those decisions.

b. An internal (prescriptivistic) point of view that considers law as the set of criteria that should be used (according to some normative model of legal reasoning) in legal decision-making. This view is adopted by the "participant", who intends to take, justify, or suggest a legal decision.

As far as the (*a*) aspect is concerned, an indeterminacy arises when alternative legal premises — alternative legal ideologies (A. Ross 1958) — are effective in society, or may anyway seem susceptible to being accepted by legal decision-makers. In such a situation, certainty of law is compromised, since the citizen is not able to anticipate the legal decisions concerning his behaviour. Legislators, judges, and legal scientists should try to correct this situation (issuing new legal texts, stating new decisions, proposing new interpretative arguments), but external indeterminacy cannot certainly be eliminated simply by "postulating" the consistency or the coherence of the legal system.

As far as the (*b*) aspect is concerned, instead, an indeterminacy arises when a person involved in a legal evaluation (typically a judge, but also a legal scientist, a civil servant, or a simple citizen) is perplexed, since his legal assumptions include conflicting reasons, grounding incompatible conclusions, upon which meta-reasons do not establish a precise ranking. Every internal (prescriptivistic) reconstruction of the legal system inevitably presents a certain degree of absolute indeterminacy. Nevertheless, this indeterminacy must be overcome when it produces a relative indeterminacy in the case at hand: The perplexed legal decision-maker cannot suspend his judgement nor adopt arbitrarily one of the alternative (merely) defensible solutions. This last solution would violate the universalisability principle (Hare 1962; Alexy 1978] 1991, 250ff), requiring that equal cases are solved in the same way: Different cases, equal in all relevant aspects, could be treated differently by choosing alternative defeasible solutions. Therefore, relative determinacy of the internal point of view must be pursued, by adding new reasons to the argumentation framework, until no merely defensible conclusion can be derived, as far as the case at hand is concerned (this aspect is well represented in the Lebach example).

This does not mean that there is always just one right legal solution (that the legal system can be postulated to be absolutely determinate, or that there is just one way to obtain determinacy), but rather that the legal decision-maker should try to reach relative determinacy by providing his best reasons — in the framework of the socio-legal-political ideologies he endorses, these ideologies also being susceptible of extensions and adaptations. There is a complex relationship between relative and absolute determinacy: The additional reasons, added to a legal system Σ to obtain relative determinacy in a case K_1, being applicable to all possible cases (according to the mentioned universalisability principle), may render Σ determinate also in future cases K_2, ..., K_i, and therefore contribute to approximating the ideal of absolute determinacy, but may also produce new undecided normative conflicts and so make Σ indeterminate in other cases K_j, ..., K_n, so furthering that same ideal.

The additional reasons required to reach relative determinacy, like any substantial evaluation, cannot be offered by formal methods: It is up to the lawyer to assess new legally relevant reasons applicable in the concrete situation. Formal legal reasoning is not required to eliminate ideas, or intuitions, but to make a coherent and universalisable use of many conflicting intuitions. Legal logic should therefore be able to dynamically follow the conclusions deriving from the changing context of choices, evaluations, and assumptions upon which legal argumentation is based. The approach here illustrated, modelling the evolution of those conclusions while the referred premises change, allows fundamental aspects of legal reasoning — the dialectic contrast of theses and points of view — to be represented while preserving formal rigour.

The conceptualization of legal systems as argumentation frameworks also offers new solutions to some problems of legal theory. Here two highly controversial issues, permissive norms and normative hierarchies, are considered.

2.1. Permissive norms

In legal theory there is a lively discussion concerning the role of permissive norms: (*a*) are permissions simply the negation of prohibitions (as in standard deontic logic, where "permitted" means "not forbidden": $Pp = \neg O \neg p$), and in this case (*b*) what new content can a permissive norm bring into a legal system?

In fact, if p is not forbidden in the system (it does not hold that $O\neg p$) then the permissive norm Pp does not seem to give any significant normative indication, while if p is forbidden (it holds already that $O\neg p$), than Pp creates an inconsistency, so violating the postulate of consistency. Some authors have also affirmed that permissions simply abrogate (eliminate from the system) pre-existing forbidding norms. Nevertheless, according to this last opinion Pp would become redundant immediately after eliminating $O\neg p$, and would have no further effect on the dynamics of the legal system.

Both (*a*) and (*b*) questions above can get a positive answer, if the legal system is considered an argumentation framework: Permissions are simply the negation of probibitions, and they contribute positively to the content of the legal system. The positive contribution of permissive norms consists in their use in permission arguments rebutting the opposed prohibition arguments and so preventing the derivation of prohibitions (established by inferior forbidding norms). For example, constitutional permissions (the so called liberty rights) rebut subsequent (but inferior) statutory forbidding norms[13].

2.2. Legal hierarchies

As everybody admits, legal systems are hierarchical: They include criteria (meta-rules) which establish preference relations between norms. When we abandon the postulate of consistency, and develop a logical model for reasoning with inconsistent information, a new understanding of

[13] The here proposed notion of permissive norm comes near to those representations of permissions as exceptions, or better as prescriptions intended to block forbidding norms — cf., among others, Alchourrón and Bulygin (1984), Hernández Marín (1990) — but formal models of argumentation seem to allow a more intuitive formalisation of this intriguing phenomenon.

those criteria is possible. They do not contribute to the creation of consistency — unless in the special cases, defined by each legal system, in which a norm is to be considered as tacitly abrogated, or as invalid, and therefore cancelled from the legal system or not admitted in it. Their purpose is, instead, to adjudicate the conflicts between lower level rules, assigning relative priorities to them.

This representation has the advantage of explaining how weaker norms — although blocked in some cases by stronger incompatible prescriptions — may nonetheless determine justified legal conclusions in those cases in which the conflict does not arise, and automatically expand themselves when those stronger prescriptions are cancelled or defeated[14]. In this prospect, the traditional principles to "solve" antinomies (the principle of hierarchy, speciality, and posteriority) can be represented as preference rules:

Ξ_0={*Posteriority*(x, y): x is preferred to y if x is posterior to y;
Speciality(x, y): x is preferred to y if x is more special than y[15];
Hierarchy(x, y): x is preferred to y if x is superior to y}.

The meta-ordering among those criteria can correspondingly be defined by means of categorical meta-preference rules, such as the following (stating that hierarchy prevails over speciality and posteriority, speciality prevails over posteriority):

Ξ_1={*HierarchySpeciality*(*Hierarchy*(x, y), *Speciality*(y, x)): *Hierarchy*(x, y) is preferred to *Speciality*(y, x);
HierarchyPosteriority(*Hierarchy*(x, y), *Posteriority*(y, x)): *Hierarchy*(x, y) is preferred to *Posteriority*(y, x);
SpecialityPosteriority(*Speciality*(x, y), *Posteriority*(y, x)): *Speciality*(x, y) is preferred to *Posteriority*(y, x)}.

The *HierarchyPosteriority* rule, *e.g.*, establishes that the meta-norm *Hierarchy*(x, y), asserting the predominance of norm x over norm y, is preferred to the meta-norm *Posteriority*(y, x), stating the predominance of y over x.

The preference rules in Ξ_0 can be used in meta-arguments establishing that substantial arguments culminating in posterior, more special, or superior norms normally prevail respectively over arguments culminating in antecedent, more general, or inferior norms. In case of conflict between those preference rules, the meta-preference rules in Ξ_1 can be used in meta-meta-arguments establishing the defeating meta-argument, which determines the best basic (substantial) argument.

Let us consider a simple example. The Italian Constitution establishes the right to strike, that we represent simply as:

StrikeRight(x): it is permitted that x abstains from work if x is on strike.

A recent Italian statutory provision establishes that every civil servant has to work every working day (at least seven hours per day). We simplify this rule as follows:

WorkingObligation(x): it is not permitted that x abstains from work if x is a civil servant.

Let us assume that Mark is a civil servant on strike:

F_1: *Mark* is a civil servant;
F_2: *Mark* is on strike.

---

[14] For a discussion with examples, cf. Sartor (1992). On the formal models of legal hierarchies the works of Alchourrón and Makinson (1981), Alchourrón (1986), and Gärdenfors (1992) should be recalled, where the theory of belief revision is adopted, an approach to the dynamics of knowledge developed in number of technical contributions by the same authors — such as Alchourrón, Gärdenfors, and Makinson (1985) and Gärdenfors (1988). The relation between argumentation and belief revision should be further studied, and possibly the two approaches could be integrated in a comprehensive model of the dynamic of normative system (Sartor 1992).

[15] Here the speciality or specificity principle is represented as a concerning the comparison of rules. Probably, its direct application to the comparison of arguments (that we cannot discuss here) would be more consistent with the argumentation framework so far presented (Poole 1985; Prakken 1992; Loui et alii 1992).

Let us also state that *StrikeRight* (which is a constitutional provision) is superior to *WorkingObligation*, while *WorkingObligation* is posterior to *StrikeRight*:

F_3: *StrikeRight*(*x*) is superior to *WorkingObligation*(*x*);
F_4: *WorkingObligation*(*x*) is posterior to *StrikeRight*(*x*).

In the argumentation framework Ξ_2, containing all rules so far introduced ($\Xi_2 = \Xi_0 \cup \Xi_1 \cup \{StrikeRight, WorkingObligation, F_1, F_2, F_3, F_4\}$), we have two arguments, the first for <permitted(*Mark* abstains from work)>, the latter for <not permitted(*Mark* abstains from work)>:

G_1={*StrikeRight*(*Mark*): it is permitted that *Mark* abstains from work if *Mark* is on strike;
F_2: *Mark* is in strike};
G_2={*WorkingObligation*(*Mark*): it is not permitted that *Mark* abstains from work if *Mark* is a civil servant;
F_1: *Mark* is a civil servant}.

The two arguments are opposed, so that, to establish which one of them defeats the other, we need to compare their top rules. Unfortunately we have two opposed preference arguments:

G_3={*Hierarchy*(*StrikeRight*(*Mark*), *WorkingObligation*(*Mark*)):
StrikeRight(*Mark*) is preferred to *WorkingObligation*(*Mark*) if
StrikeRight(*Mark*) is superior to *WorkingObligation*(*Mark*);
F_3: *StrikeRight*(*Mark*) is superior to *WorkingObligation*(*Mark*)};
G_4={*Posteriority*(*WorkingObligation*(*Mark*), *StrikeRight*(*Mark*)):
WorkingObligation(*Mark*) is preferred to *StrikeRight*(*Mark*) if
WorkingObligation(*Mark*) is posterior to *StrikeRight*(*Mark*);
F_4: *WorkingObligation*(*Mark*) is posterior to *StrikeRight*(*Mark*)}.

To solve the conflict between G_3 and G_4, we use the meta-meta-rule establishing that hierarchy prevails over posteriority (the preference relation is antisymmetric, so that for no pair of rules r_1 and r_2 is it possible that both r_1 is preferred to r_2 and r_2 is preferred to r_1).

G_5={*HierarchySpeciality*(*Hierarchy*(*StrikeRight*(*Mark*),*WorkingObligation*(*Mark*)),
Speciality(*WorkingObligation*(*Mark*), *StrikeRight*(*Mark*)):
Hierarchy(*StrikeRight*(*Mark*),*WorkingObligation*(Mark)) is preferred to
Speciality(*WorkingObligation*(*Mark*), *StrikeRight*(*Mark*))}.

On the basis of argument G_5, we can justifiably say that — as far as *StrikeRight* and *WorkingObligation* are concerned — hierarchy prevails over speciality. This means that G_3 defeats G_4, so that we can justifiably derive that *StrikeRight*(*Mark*) is preferred to *WorkingObligation*(*Mark*). This finally allows G_1 to defeat G_2, so that we can justifiably assert the conclusion of G_1: *Mark* is permitted to abstain from his work.

2.3. Computability and the limits of the model

The model of legal reasoning just proposed represents a computable specification: It can be easily transferred into a logic program (to be found in the appendix), simply by expressing each one of the definitions above as a program clause. It is not surprising that this model, whose central concern is isomorphism with legal reasoning, also allows a simple and efficient implementation in a computer program: Simplicity in knowledge representation and effectiveness in inference procedures are fundamental requirements of both human and automatic knowledge processing.

Nevertheless, simplicity and computational efficiency are also achieved at the price of some rigid restrictions concerning the language here proposed, its inference procedures, the modeled aspects of legal reasoning:

a. Argumentation frameworks contain only inference rules, and every inference rule has the same simple syntactical structure: The consequent is a literal and the antecedent a conjunction of literals.
b. The conflict between couples of opposed arguments is decided considering just the top rules of those arguments.
c. Speciality is a primitive notion not formally analysed.
d. Some typical logical inferences have not been accounted for, such as suppositional reasoning (a proposition is assumed, a conclusion is drawn from this supposition, and then the supposition is discharged to obtain a conclusion no more dependant on the supposition), *reductio ad absurdum* (we suppose *p*, infer **not** *p* from this supposition, and conclude for **not** *p* independently of the suppositions) and *dilemma* (when all alternatives lead to the conclusion *p*, then *p* can be derived from the disjunction of those alternatives).
e. The traditional non-deductive legal inferences, such as *argumentum a simili*, *a fortiori*, and *a contrario* have not been considered.
f. The relations between the parties of the argumentation have not been considered, nor have the limits in the available resources.
h. No semantics has been given for the "logic" here defined, but just an inferential machinery.

Many extensions of the proposed model can be correspondingly considered:

a. Linguistic structures not reducible to inference rules, or to the type of inference rules here considered should be expressible to deal with some legal contexts (in particular disjunctive factual assertions should be representable).
b. Alternative strategies to treat conflicts should possibly be considered. For example, arguments leading to the same conclusions could be merged, so that they their strengths are added when facing opposed arguments.
c. The notion of specificity should be formally defined on the basis of the content of the regulations involved.
d. A larger portion of deductive reasoning should be included in the model[16].
e. Formal specification should be possibly proposed of some aspects of the most relevant patterns of non deductive legal reasoning.
f. A formalisation of the positions of the parties involved (including their being bound to their assertions, their faculty of accepting or contesting the assertions of other parties, etc.) should be provided to explain real legal dialogues, the limitations of the available resource (especially time) should be made explicit, as should the remedies to those limitations and other typically procedural aspects of legal argumentation[17].
g. It remains an open question whether argumentation systems — like legal logic in general — really need semantics, but advances in nonmonotonic logics and belief revision will possibly allow convincing semantical models to be developed.

Notwithstanding the shortcomings just indicated, the simple model here proposed should not be underestimated: It is already able to (partially) model most legal argumentations, in an intuitive and computable way. Its extensions — which would hopefully allow a stricter connection to be established with more comprehensive, but informal, theories of legal argumentation, such as those of Alexy ([1978] 1991), Peczenick (1983), and Aarnio (1987) — should try to preserve the simplicity and effectiveness so far maintained.

---

[16] As in the (much more complex) argumentation system which try to combine full first order logic with inference rules, as Simari and Loui (1992), Prakken (1992), Pollock (1987, 1992), Gordon (1993), Vreeswijk (1993). In comparison to those formalisms, nevertheless, here meta-argumentation has been especially developed.
[17] For a formal model of dialectics, cfr., for all, Rescher (1977) and especially Gordon (1993), who uses a non monotonic logic (Geffner and Pearl 1992) to define and implement into a computer program the "pleading game", a formal model of civil process inspired by Alexy ([1978] 1991), which also includes an explicit regulation of the interaction between the involved parties.

3. Appendix. A computer program for legal argumentation

As promised, we will now introduce a computable specification of the notions above defined. The program is able to determine — producing the corresponding arguments — the supported, defensible, and justified conclusions derivable from any given argumentation framework. It may therefore offer a limited but valuable support in legal argumentation: It is up to the parties to have the right ideas, that is to suggest the contents to be included in the argumentation framework; it is up to the program to determine the implications of all those contents. Here it is not possible to explain in detail the functioning of the program. The interested reader (with some knowledge of logic programming) can easily map the program clauses into the definitions in the paper. The program was developed in LPA Prolog (Macintosh version), but, possibly with minor changes, should run under any Prolog interpreter accepting the Edinburgh syntax.

```
:- op(120, xfy, and).
:- op(130, xfy, if).
:- op(100, fy, non).
:- op(110, xfx, preferred_to).
:- op(140, xfx, :).

argument_for(A, L):-
      argues_for(A, L),
      consistent(A).
argues_for([Name: L if Body|A1], L):-
         Name: L if Body,
         argument_for(A1, Body).
argues_for(A, L1 and L2):-
         not var(L1),
         argues_for(A1, L1),
         argues_for(A2, L2),
         concat(A1, A2, A3).
argues_for([Name: L], L):-
      Name: L.
argues_for([], L):-
      call(L).

justified_argument_for(A,L):-
   argument_for(A,L),
   justified(A).
justified(A):-
   not (questioned(A)).

defensible_argument_for(A,L):-
   argument_for(A,L),
   defensible(A).
defensible(A):-
   not (defeated(A)).

defeats(A1, A2):-
   directly_defeats(A1, A2).
defeats(A1, [FirstRule|RestA2]):-
   defeats(A1, RestA2).

directly_defeats(A1,A2):-
   opposes(A1,A2),
   prevails_over(A1, A2),
   justified(A1).
opposes(A1,A2)
   rebuts(A1, A2);
   undercuts(A1,A2).

defeated(A1):-
   defeats(A2, A1).

questions(A1, A2):-
   directly_questions(A1, A2).
questions(A1, [FirstRule|RestA2]):-
   questions(A1, RestA2).

directly_questions(A1, A2):-
   rebuts(A1,A2),
   not succumbs_under(A1, A2),
   defensible(A1).
directly_questions(A1, A2):-
   undercuts(A1,A2),
   defensible(A1).

questioned(A1):-
   questions(A2, A1).

top_rule(First, [First|Rest]).

rebuts(A1, A2) :-
   top_rule(R2: L2 if T2, A2),
   complement(L2, L1),
   argument_for(A1, L1).

undercuts(A1, A2) :-
   top_rule(R2: L2 if T2, A2),
   argument_for(A1, non applicable(R2)).

counterargues(A1, A2):-
   rebuts(A1, A2);
   undercuts(A1, A2).
counterargues(A1, [Fist|Rest]):-
   counterargues(A1, Rest).
```

```
prevails_over(A1, A2):-
   top_rule(R1: H1 if T1, A1),
   top_rule(R2: H2 if T2, A2),
   justified_argument_for(A3, preferred_to(R1, R2)).

succumbs_under(A1, A2):-
   prevails_over(A2, A1).

non X preferred_to Y :-
   Y preferred_to X.

consistent([Rule|RestArg]):-
   consistent_with(Rule, RestArg),
   consistent(RestArg).
consistent([]).

consistent_with(L, Arg) :-
   complement(L, ComplL),
   not member(_: ComplL if _, Arg).

complement(non L, L).
complement(L, non L):-not(L = non(_)).

member(X, [X|_]).
member(X, [_|Ys]) :- member(X,Ys).
```

References

Aarnio, A. 1987. *The Rational as Reasonable*. Dordrecht: Reidel.

Alchourrón, C.E. 1986. Conditionality and the Representation of Legal Norms. In *Automated Analysis of Legal Texts*. Ed. A.A. Martino e F. Socci, 175-186. Amsterdam: North Holland.

Alchourrón, C.E., and E. Bulygin. 1984. Permission and Permissive Norms. In *Theorie der Normen. Festgabe für Ota Weinberger zum 65. Geburtstag*. Ed. W. Krawietz, H. Schelski, G. Winkler, and A. Schramm, 349-371. Berlin: Duncker und Humblot.

Alchourrón, C.E., and D. Makinson. 1981. Hierarchies of Regulations and Their Logic. In *New Studies on Deontic Logic*. Ed. R. Hilpinen, 123-148. Dordrecht: Reidel.

Alchourrón, C.E., P. Gärdenfors, and D. Makinson. 1985. On the Logic of Theory Change: Partial Meet Functions for Contractions and Revisions. *Journal of Symbolic Logic* 50: 510-530.

Alexy, R. [1978] 1991. *Theorie der juristischen Argumentation*. Frankfurt: Suhrkamp.

Alexy, R., 1980. Die logische Analyse juristischer Entscheidungen. In *Argumentation und Recht.*. Ed. W. Hassemer, A. Kaufmann, and U. Neumann, 181-212. ARSP, Beiheft 14. Wiesbaden: Steiner.

Alexy, R. 1985. *Theorie der Grundrechte*. Frankfurt: Suhrkamp.

Alexy, R. 1992. *Legal Argumentation as Rational Discourse. La crisis del derecho y sus alternativas*. Consejo general del poder judicial, Madrid, 30 November-4 December.

Baker, G.P. 1977. Defeasibility and Meaning. In *Law, Morality, and Society: Essays in Honour of H.L.A. Hart*. Ed. P.M.S. Hacker and J. Raz, 26-57. Oxford: Clarendon.

Bench-Capon, T.J.M., and F.P. Coenen. 1992. Isomorphism and Legal Knowledge Based Systems. *Artificial Intelligence and Law* 1: 65-86.

Brewka, G. 1991. *Nonmonotonic Reasoning. Logical Foundations of Commonsense*. Cambridge: Cambridge University Press.

Dworkin, R.M. 1977. *Taking Rights Seriously*. London: Duckworth.

Gärdenfors, P. 1988. *Knowledge in Flux*. Cambridge, Mass.: MIT.

Gärdenfors, P. 1992. The Dynamics of Normative Systems. In *Expert Systems in Law*. Ed. A.A. Martino, 195-200. Amsterdam: North Holland.

Geffner, H., and G. Pearl. 1992. Conditional Entailment: Bridging Two Approaches to Default Reasoning. *Artificial Intelligence* 53: 209-244.

Gianformaggio, L. 1987. Logica e argomentazione nell'interpretazione giuridica ovvero i giuristi presi sul serio. *Studi senesi* 36 (3): 461-489.

Ginsberg, M.L. (editor). 1987. *Readings in Nonmonotonic Reasoning*. Los Altos, Cal.: Morgan Kaufmann.

Gordon, T.F. 1988. *The Importance of Nonmontonicity for Legal Reasoning. In Expert Systems in Law: Impacts on Legal Theory and Computer Law*. Ed. H. Fiedler, F. Haft, and R. Traunmüller, 111-126. Tübingen: Attempto.

Gordon, T.F. 1993. *The Pleadings Game. An Artificial Intelligence Model of Procedural Justice.* Phd. Thesis. Darmstadt.

Hage, J. 1993. Monological Reason Based Logic: A Low Level Integration of Rule-Based Reasoning and Case-Based Reasoning. In *The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference*, 30-39. New York, N.Y.: ACM.

Hare, R.M. 1962. *Freedom and Reason.* Oxford: Clarendon.

Hart, H.L.A. 1948-9. The Ascription of Responsibility and Rights. *Proceedings of the Aristotelian Society* 49: 171-194.

Hernández Marín, R. 1991. Practical Logic and the Analysis of Legal Language. *Ratio Juris* (4): 322-335.

Jones, A.J., and I. Porn. 1991. *DEON'91: First International Workshop on Deontic Logic in Computer Science, Amsterdam, The Netherlands, December 11-13, 1991, Proceedings,* ed. J.-J.C. Meyer and R.J. Weiringa 232-247. Amsterdam: Vrjie Universiteit.

Larenz, K. 1992. *Methodenlehre der Rechtswissenschaft.* Berlin: Springer. 2. Auflage. Verkützte Studienausgabe von 1991. Methodenlehre der Rechtswissenschaft. 6. Auflage.

Loui, R.P, J. Norman, J. Olson, and A. Merrill. 1993. A Design for Reasoning with Policies, Precedent, and Rationales. In *The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference*, 202-211. New York: ACM.

Loui, R., J. Norman, K. Stiefvater, A. Merill, A. Costello, and J. Olson. 1992. *Computing Specificity.* Technical Report WUCS-92-46. St Louis, Mo.: Department of Computer Science, Washington University.

Nute, D. 1988. Defeasible Reasoning: A Philosophical Analysis in Prolog. In *Aspects of Artificial Intelligence*, ed. J.H. Fetzer, 251-288. Dordrecht: Kluwer.

Pattaro, E. 1988. Models of Reason, Types of Principles and Reasoning. Historical Comments and Theoretical Outlines. *Ratio Juris* 2: 109-122.

Peczenick, A. 1983. *The Basis of Legal Justification.* Lund: Peczenick.

Perelman, C. 1979. *Logique juridique. Nouvelle rhetorique.* 2nd ed. Paris: Dalloz. Seconda edizione.

Perelman, C., and L. Olbrechts-Tyteca. 1958. *La nouvelle réthorique. Traité de l'argumentation.* Paris: Presses Universitaires de France.

Philipps, L. 1966. Sinn und Struktur der Normenlogik. *ARSP* 50: 317-329.

Pollock, J.L. 1987. Defeasible Reasoning. *Cognitive Science* 11: 481-518.

Pollock, J.L. 1992. How to Reason Defeasibly. *Artificial Intelligence*: 1-42.

Poole, D.L. 1985. On the Comparison of Theories: Preferring the Most Specific Explanation. In *Proceedings IJCAI*: 144-147.

Prakken, H. 1992. *Logical Tools for Modelling Legal Arguments.* Amsterdam: Prakken.

Prakken, H. 1993. A Logical Framework for Modelling Legal Argument. In *The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference*, 1-9. New York, N.Y.: ACM.

Raz, J. 1978. Reasons for Action, Decisions and Norms. In *Practical Reasoning.* Ed. J. Raz, 128-143. Oxford: Oxford University Press.

Reiter, R. [1980] 1987. A Logic for Default Reasoning. In *Readings in Nonmonotonic Reasoning*, ed. M.L. Ginsberg, 68-93. Los Altos (California): Morgan Kaufmann. First published in Artificial Intelligence 13: 81-132.

Rescher, N. 1977. *Dialectics.* Albany, N.Y.: State University of New York.

Ross, A. 1958. *On Law and Justice.* London: Steven and Sons.

Ross, W.D. 1930. *The Right and the Good.* Oxford: Clarendon.

Ross, W.D. 1939. *Foundations of Ethics.* Oxford: Clarendon.

Ryu, Y.U., and R.M. Lee. 1991. Defeasible Deontic Reasoning: A Logic Programming Model. In *Deon'91. First International Workshop on Deontic Logic in Computer Science. Amsterdam, The Netherlands, December 11-13. Proceedings.* Ed. J.-J.Ch Meyer and R.J. Wieringa, 347-363. Amsterdam: Vrjc Universiteit Amsterdam.

Sartor, G. 1991. The Structure of Legal Norms and Nonmonotonic Reasoning in Law. In *The Third International Conference on Artificial Intelligence and Law. Proceedings of the Conference*, 155-164. New York, N.Y.: ACM Press.

Sartor, G. 1992. Normative Conflicts in Legal Reasoning. *Artificial Intelligence and Law* 1: 209-235.

Sartor, G. 1993. A Simple Computational Model for Nonmonotonic and Adversarial Legal Reasoning. In *The Fourth International Conference on Artificial Intelligence and Law. Proceedings of the Conference*, 192-201. New York, N.Y.: ACM Press.

Simari, G.R., and R.P. Loui. 1992. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence* 53: 125-157.

Toulmin, S. 1958. *The Uses of Argument*. Cambridge: Cambridge University Press.

Vreeswijk, G. 1993. *Studies in Defeasible Argumentation*. Dissertation. Amsterdam: Vreeswijk.

Weinberger, O. 1975. Ex Falso Quodlibet in der Preskriptiver Sprache. *Rechtstheorie*: 17-32.

Wroblewski, J. [1969] 1983. Legal Reasoning in Legal Interpretation. In *Meaning and Truth in Judicial Decision*, 71-103. Helsinski: A-Thieto Oy. (First published in *Logique et analyse* 13: 3 ss.)

IDG-CNR, Via Panciatichi 56/16, 50127 Florence, Italy
CIRFID, University of Bologna, Via Galliera 3, 40121 Bologna, Italy

Knowledge Representation Language

Quixote

ICOT
2nd Lab.

Quixote

Motivation

- Deductive Object-Oriented Database (DOOD) Language
- Extended Constraint Logic Programming (CLP) Language
- As Knowledge Representation Language:
  - How to represent concepts?
  - How to relate concepts?
  - How to classify concepts?
  - How to treat assumption-based reasoning?

Quixote

A Knowledge Representation Language *Quixote*

What is *Quixote* ?

Core Language for KBMS of FGCS
Providing Facilities for Integrated KB

- big *Quixote*
  KL1/PIMOS + C,GNU-Emacs,X/UNIX
- micro-*Quixote*
  C/UNIX,DOS,Mac

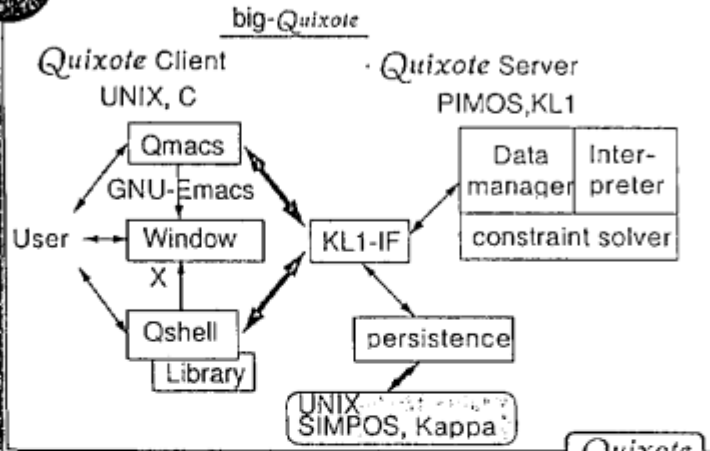
Quixote

Key Features of *Quixote*

(1) Object Identity → representation
(2) Subsumption Constraint → relation
(3) Property Inheritance → classification
(4) Hierarchical Module
(5) Conditional Query & Answer with Assumption → assumption-based reasoning
(6) Call external constraint solver. → heterogeneous problem solving

Quixote

System Configuration



Quixote

Example: SUMO

"Will Wakanohana be Yokozuna?"

object

-Wakanohana → Hanada Masaru

relation

-Wakanohana ISA sumo_wrestler

rule

-IF he wins in this season,
he will be Yokozuna next season.

abduction inference

-Answer? "if ..... then yes."

Quixote

big-Quixote vs. micro-Quixote

	big-Quixote	micro-Quixote
OS/machine	PIMOS/PIM+UNIX	UNIX,DOS,Mac
Language	KL1, C, Lisp	C
Code Size	60,000 lines	199Kbyte
Portability	depend on KLIC	good
subsumption inheritance	○	○
module	○	△
solution merge	○	×
assumptionQA	○	○
DBfeature	○	×

Quixote

(1) Object Identity

Object Terms

Object Identifier (OID)

- Basic Object Term

apple, cider, 花田勝

- Complex Object Term

intrinsic attributes

apple[color=green]

Quixote

Objects and Their Properties

Specification of an object and its properties (extrinsic attributes)

花田勝
(Masaru Hanada)

- born in Jan.20, 1971
- ring name is Wakanohana
- current rank is Ozeki

attribute term

hanada_masaru/[birth="1/20/1971",
ring_name=wakanohana, rank=ozeki]

Quixote

Properties and Subsumption Constraints

Extrinsic property as subsumption constraint

color=red
taste -> sour
area+ <- {nagano,aomori}

Dotted Term

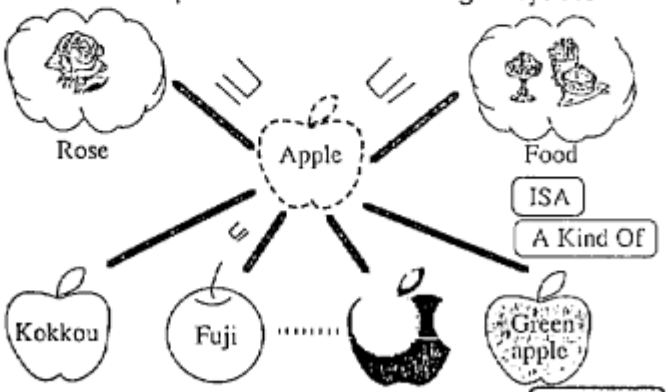
object . property

apple.color = red
apple.taste =< sour
apple.area* >+ {nagano, aomori}

Quixote

(2) Subsumption Constraint

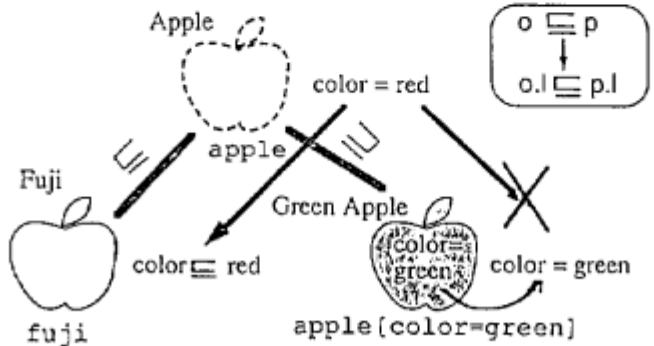
Subsumpion Relation among Objects



Quixote

(3) Property Inheritance

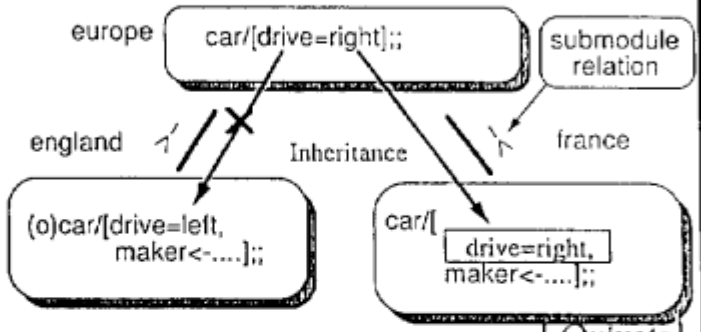
Property Inheritance by ⊑ ("=<")
Intrinsic Attributes as Exception



Quixote

(4) Hierarchical Modules and Rule Inheritance

"In Europe, cars usually drive on the right.
But in England, cars drive on the left."



Quixote

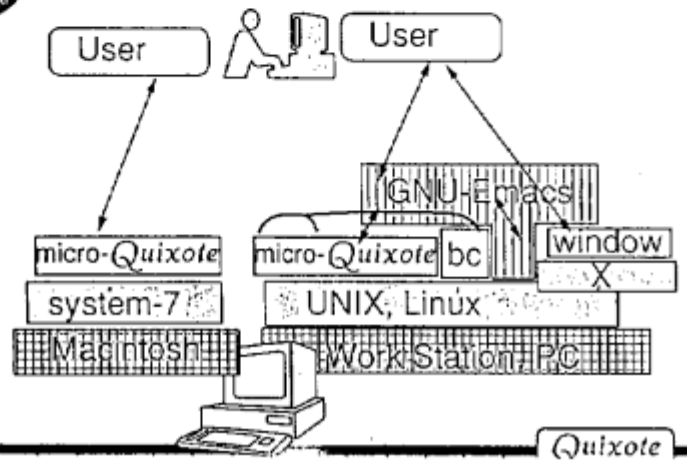
(5) Conditional query & answer with assumption

SUMO program

```
&program;;
&subsumption;;
   wrestler >= {koji_hanada, masaru_hanada,...};;
   .....
&rule;;
   ozeki/[upper=yokozuna, lower=sekiwake];;
   .....
   next[name=X, rank=Y] <= .....;;
   chado/[ring_name=akebono, rank=yokozuna,
          age=24, height=204, weight=224, ...];;
   .....
&end.
```

Quixote

Demo System Configuration



Quixote

Question

"Who are single?"

```
?-X/[family="single", ring_name=Y]
     || {X=<wrestler}.
```

"Who are over 30?"

external constraint

```
?- X/[age=A, ring_name=Y]
     || {X=<wreslter, A #># 30}.
```

"Who's speciality is NAGE?"

```
?-X/[speciality=A, ring_name=Y]
     || {X=<wreslter, A #regexp# "Nage"}.
```

Quixote

Answer with assumption

Rule: knowledge about Wakanohana

```
masaru/[rank=ozeki, birth=1971,....];;
nickname[name=waka, real=masaru];;
```

"If he wins 15, his grade will be up."

```
next[name=X, rank=Z]<=
nickname[name=X,real=RN], RN/[rank=Y,
win=15], Y/[upper=Z] || {X=<wrestler,Y=<rank};;
```

Query : "What is Waka's next rank?"

```
{?-next[name=waka, rank=N];....}
```

Answer: abduction

{N=yokozuna if {masaru.win==15}}

Quixote

Conclusion

- Object Identity
- Subsumption Constraint
- Property Inheritance
- Hierarchical Module
- Conditional Query & Answer with Assumtion
- Call external constraint solver

Applications:

legal reasoning

natural language processing

(feature structure, situated inference)

biological DB

Quixote

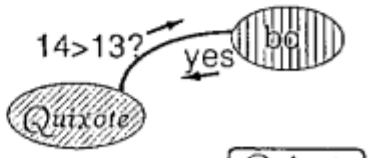
Conditional query (6) Call external cstr. solver

Query with Additional Information:
"If Waka wins 14matches,
what is his next rank?"

```
{?- next[name=masaru, rank=X]
  &program;;&rule;;
      masaru/[win=14];;
  &end.
```

additional information

Answer:
[yes.]
(no assumption)



Quixote

To use *Quixote*:

Both *Quixote* systems are registered as ICOT Free Softwares (IFS)
Anonymous FTP from
`ftp.icot.or.jp`

Please enjoy!

Quixote

ICTAI (Intl. Conf. on Tools with AI), 199. Boston

QUIXOTE as a Tool for Natural Language Processing

Satoshi Tojo, Hiroshi Tsuda
Hideki Yasukawa, Kazumasa Yokota, and Yukihiro Morita
Institute for New Generation Computer Technology (ICOT)
4-28 Mita 1, Minato-ku, Tokyo 108, JAPAN

Abstract

We developed a language *QUIXOTE* as a tool to deal with various information of natural language processing (NLP). *QUIXOTE* is a hybrid language of deductive object-oriented database (DOOD) and constraint logic programming (CLP) language. The new mechanism of *QUIXOTE* is a combination of an object-orientation concept such as *object identity* and the concept of *module* that classifies a large knowledge base. In addition, its logical inference system is extended to be able to make restricted abduction. We first apply *QUIXOTE* to the sorted feature structure of constraint-based grammar formalisms. Next, we show that *QUIXOTE* can contribute to the description of situation-based semantics. We implemented a system to make abductive reasoning to clarify hidden information. Also we resolve the problem of noun phrase reference.

1 Basic Structure of *QUIXOTE*

This section outlines *QUIXOTE*[13, 14] as a tool for NLP. *QUIXOTE* is classified as both a DOOD and CLP language. It has the following features that are also useful in NLP:

- OO features, such as object identity,
- attribute-value data structure with subsumption constraints,
- efficient information description with modules and inheritance hierarchy, and
- question and answer with assumptions.

QUIXOTE system is implemented with the KL1 language, a parallel logic programming language for the parallel inference machine, PIM, in ICOT.

Generally, *QUIXOTE* programs consist of subsumption relations among basic object terms (1.1), submodule relations among modules (1.3), and rules.

QUIXOTE rules have the following syntax.

$$\overbrace{m_0 :: H}^{\text{head}} \quad \overbrace{\mid HC}^{\text{head\_constraint}} \quad \Leftarrow \overbrace{m_1 : B_1, \cdots}^{\text{body}} \quad \overbrace{\parallel BC}^{\text{body\_constraint}} \quad ;;$$

Here, H and B_i are *object/attribute terms*. HC and BC are *constraints*. m_i, called a *module identifier*, specifies the module in which terms or rules resides. The body, constraint, and module identifier can be omitted.

The above rule resides in module m_0. It implies that, if every B_i holds in a module m_i under constraints BC, then H and constraints HC hold in m_0.

1.1 Object Term

Let Obj,$Bobj$.$Cobj$, and Var be a collection of *object terms*, *basic object terms*, *complex object terms*, and *variables*, respectively. Then, $Obj = Bobj \cup Cobj \cup Var$.

Basic object terms are atomic symbols such as `cat` and `apple`.

Subsumption relation $\sqsubseteq$, is a partial order relation between basic object terms. `a` $\sqsubseteq$ `b` means that `a` is more specific than `b`, or a ISA `b`, for example, `cat` $\sqsubseteq$ `animal` or `animal` $\sqsubseteq$ `creature`. Special basic object terms $\bot$ and $\top$ satisfy $\forall x \in Bobj, \bot \sqsubseteq x, x \sqsubseteq \top$. *QUIXOTE* comprises a complete lattice from $< Bobj, \sqsubseteq >$.

A *complex object term* is a term having the form $o[l_1 = v_1, l_2 = v_2, \cdots]$, where $o \in Bobj$ and $\forall i, l_i \in Bobj, v_i \in Obj$. Each label l_i is called an *intrinsic attribute*. The order of labels is not significant, so `o[l=a,m=b]` and `o[m=b,l=a]`, for example, are identical.

The subsumption relation is extended to treat complex object terms. For example, when `cat` $\sqsubseteq$ `animal`,

`cat[age = 2, sex = male]` $\sqsubseteq$ `cat[age = 2]`
`cat[color = white]` $\sqsubseteq$ `animal`.

1.2 Attribute Term, Constraints, and Property Inheritance

An *attribute term* is an object term with property specifications represented as:

$$head/[l_1 op_1 v_1, l_2 op_2 v_2 \ldots],$$

where $head \in Obj$ and $\forall i. l_i \in Bobj. op_i \in \{=, \leftarrow, \rightarrow\}, v_i \in Obj$. Each label l_i is called an *extrinsic attribute*.

An attribute term can be transformed into an object term with a set of constraints.

$$\begin{aligned} o/[l = X] &\Leftrightarrow o \mid \{o.l \cong X\} \\ o/[m \rightarrow U] &\Leftrightarrow o \mid \{o.m \sqsubseteq U\} \\ o/[n \leftarrow V] &\Leftrightarrow o \mid \{V \sqsubseteq o.n\} \end{aligned}$$

$O \mid C$ means an object term O with constraint C. `o.l` is called a *dotted term* and specifies the value of the `l` attribute of an object term `o`. *Constraints* is a set of formulas having the form $< term >< op >< term >$, where $< term >$ is an object/dotted term. and $< op > \in \{\cong, \sqsubseteq\}$.

By default, the attributes of an object are inherited by its related objects in terms of $\sqsubseteq$, as follows. This is called the *property inheritance*.

$$o \sqsubseteq p \Rightarrow \forall l, o.l \sqsubseteq p.l$$

For example, when `macintosh` $\sqsubseteq$ `apple` and `apple.color` $\cong$ `red`, then `macintosh.color` $\sqsubseteq$ `red` holds.

For complex object terms, however, the values of intrinsic attributes override those of extrinsic ones. For example, even if `apple.color` $\cong$ `red` holds. `apple[color=green].color` remains `green` and is not subsumed by `red`.

1.3 Module and Object Identity

A *QUIXOTE* program can be divided into several modules. Each module is identified by an object term called a *module identifier* and consists of a set of rules.

Like many programming languages, *QUIXOTE* has an inheritance mechanism between modules called *rule inheritance*. Submodule relation $\sqsupseteq_S$ between module identifiers specifies the rule inheritance. For example, when `john` $\sqsupseteq_S$ `common_knowledge`, all rules in the module `common_knowledge` are inherited by the module `john`. Here, `john` is called a *submodule*, while `common_knowledge` is a *supermodule*.

To realize rule inheritance exceptions, each rule can have an inheritance mode `o.l` or `ol`. A rule with `o` *overrides* inherited rules which have the same head. A rule with `l` is a *local* rule, hence is not inherited by the submodules. Inheritance mode `ol` is a combination of `o` and `l`.

The following example describes the knowledge "In Europe, cars usually drive on the right. But cars drive on the left in England."

```
england ⊒S europe, france ⊒S europe
europe :: car/[drive = right];;
england :: (o)car/[drive = left];;
```

Two object terms are *identical* if they currently bind to a literally equal object term except for the order of labels. For example, if `X` binds to a. `o[l=X,m=b]` and `o[m=b,l=a]` are identical. In this sense, ground object terms work as object identifiers.

Within a module, the values of the identical attributes of identical objects must be equal. In different modules that are not in the submodule relations, however, identical objects can have distinct values.

For example, the following program becomes inconsistent because `john` has a different `age` value in the module `sit_1993`.

```
sit_1993 :: john/[age=20];;
sit_1993 :: john/[age=30];;
```

However, the following is not inconsistent, when the submodule relation does not hold between `sit_1983` and `sit_1993`.

```
sit_1983 :: john/[age=20];;
sit_1993 :: john/[age=30];;
```

1.4 Answer with Assumptions

A query sentences have the following forms.

$$?- m_1 : G_1, \cdots \parallel C.$$
$$?- m_1 : G_1, \cdots \parallel C;; PG.$$

Here, m_i is a module identifier, G_i an object/attribute term, C a constraint, and PG a program. In response to a query, *QUIXOTE* returns answer substitutions with a set of constraints among dotted terms called *assumptions*.

Except for constraints among dotted terms, *QUIXOTE* works like a conventional CLP language [4]. However, dotted term constraints in the body constraints are accumulated as assumptions if they are not satisfied by the head constraints. Assumptions can be seen as lacking information in the DB. Deriving assumptions is a kind of abduction.

The second type of query, which is often used in hypothetical reasoning applications, adds the additional

The interesting point of sentences (4) and (5) is that the conclusions are different despite application of the same conditional clauses. This is because there is hidden partial knowledge in the two people and this is not mentioned explicitly. The objective of the program introduced below is to determine the nationalities of Bizet and Verdi when they are asked, with inferences on implicitly mentioned information.

The natural language expressions that correspond to (4) and (5) are directly translated into *QUIXOTE* as follows:

```
hypothesis_a ::
    bizet/[nationality = italy] <=
    compatriots[per1=bizet,per2=verdi] ;;
hypothesis_b ::
    verdi/[nationality = france] <=
    compatriots[per1=bizet,per2=verdi] ;;
```

The first rule says that *being compatriots of two persons named Bizet and Verdi* implies that *Bizet's nationality being Italian* in a *hypothesis of person A*. The second rule can be interpreted in a similar way. We need other rules to terminate inference chaining, viz. the definition of `compatriots`, `bizet`, and `verdi`.

```
world :: compatriots[per1=X,per2=Y] <=
  X/[nationality=N1], Y/[nationality=N2]
  || {N1=<nation,N2=<nation,N1==N2} ;;
world :: bizet;;
world :: verdi;;
```

These definitions are valid in every module, viz. in the most general module `world`. We can write the module hierarchy as follows: [2]

```
hypothesis_a >- world ;;
hypothesis_b >- world ;;
```

We need to define the subsumption relation beforehand as follows:

```
nation >= italy ;;
nation >= france ;;
```

The above are all of the rules. Please note that what person *A* knows and what person *B* knows is not mentioned anywhere.

Now, we would like to introduce how *QUIXOTE* works. First, let us ask the nationality of Bizet:

```
?-hypothesis_a:bizet/[nationality=N].
```

The result of inference is as follows:

```
** Answer 1 **
 IF hypothesis_a:verdi.nationality
    == bizet.nationality
    hypothesis_a:verdi.nationality
    =< nation
 THEN
    N == italy
** Answer 2 **
    N == Unbound
```

The system returned two answers. The latter is easy, for it is similar to conventional Prolog: we can infer nothing of the nationality of Bizet because there is no direct reference to it in the chaining of inference rules. The feature of *QUIXOTE* is the ability to find the former answer. In the subsumption mapping of objects, the system can find a minimal model to satisfy the query, and the system returns the model with conditions, that is from `if` to `then`.

2.4 Treatment of Noun Phrase Reference

In this section, we would like to focus upon the problem of noun phrase reference. This problem appears in various forms: opaqueness (*de re* and *de dicto*), anaphora scope, metaphor and metonymy, confusion of roles and attributes-values, and so on. Here, we will analyze the following sentences.

Hitchcock saw himself in that movie. (6)

Hitchcock saw a unicorn in that movie. (7)

(6) has several possible interpretations: one is that a person named Hitchcock saw a movie and he was playing the role of someone. The other is that Hitchcock saw someone was playing the role of Hitchcock (in a biographical movie, for example).[3] (7) refers to an object that does not exist in this world.

Let us consider the following program:

```
real::see[agt=hitchcock,obj=X] <=
  movie:cast[name=hitchcock,
             title=hitchcock_life,act=X];;
real::hitchcock;;
movie::cast[name=X,title=M,act=X] <= M:X;;
movie::cast[name=X,title=M,act=Y]
  <= M:Y/[actor=X],real:X;;
hitchcock_life::
  man[place=bus_stop]/[actor=hitchcock];;
hitchcock_life::
  hitchcock/[actor=orson_welles];;
```

[2] In *QUIXOTE* syntax, $\sqsupseteq_S$ is >-.

[3] There is a possibility that Hitchcock is playing the role of Hitchcock himself.

programs before the inference. For details of the procedural semantics of QUIXOTE, see [8].

Consider the following example. It indicates that there is a book, and that the shipping fee is 500 yen if it is hard-cover, or 300 yen if soft-cover.

```
sit::book;;
sit::ship[fee=500] <= book/[cover->hard];;
sit::ship[fee=300] <= book/[cover->soft];;
```

The first clause tells us only of the existence of an object, book, and nothing about its properties. The second clause means that if book exists in sit and the cover property is subsumed by hard, ship[fee=500] holds in sit. How about asking a query ?-sit:ship[fee=Yen], which asks the shipping fee of the book, to this program? QUIXOTE returns the following two independent answers. [1]

```
Yen=500  if  sit:book.cover=<hard
Yen=300  if  sit:book.cover=<soft
```

Each answer makes an assumption about the cover property of book which comes from the body of the second or third clauses. Neither constraints are satisfied by the head constraint, which is empty in this example, so they are accumulated as assumptions.

2 NLP in QUIXOTE

2.1 Representation of Feature Structure

Consider the sorted (typed) feature structure[3], upon which the latest framework of HPSG[9] is constructed. It is the feature structure whose nodes are labeled with sort symbols. The inheritance mechanism between supersorts and subsorts enables the efficient representation of lexicon or grammar.

For example (1) is a simple HPSG-like sorted feature structure representing the word "run." word, vp, run, phrase, or np specifies the sort of each structure.

$$\begin{bmatrix} \texttt{word} \\ \texttt{CAT} : [\texttt{vp}] \\ \texttt{PH} : [\texttt{run}] \\ \texttt{SUBCAT} : \begin{bmatrix} \texttt{phrase} \\ \texttt{CAT} : [\texttt{np}] \end{bmatrix} \end{bmatrix} \quad (1)$$

Attribute terms in QUIXOTE are capable of partially describing information. The basic object term and subsumption relation $< Bobj, \sqsubseteq >$ in QUIXOTE naturally comprises types and their inheritance hierarchies. To treat the inheritance and information partiality, it is natural to describe a sorted feature structure with an attribute term whose head is a basic object term representing the sort. The property inheritance mechanism (1.2) corresponds to the inheritance between sorted feature structures. (1) is represented by the following subsumption relations and two attribute terms.

```
Y=<word, Z=<phrase
Y/[cat->vp,ph->run,subcat=Z]
Z/[cat->np]
```

In comparison with related KR languages, PST (Partially Specified Term) in CIL[6] and ψ-terms in LOGIN[1] are closely related to the feature structures. However, attribute terms in QUIXOTE are more powerful because CIL does not have an inheritance mechanism and LOGIN cannot handle constraints.

2.2 QUIXOTE and Situated Inference

We first define a *situated inference* rule as follows:

$$s_0 \models \sigma_0 \Leftarrow s_1 \models \sigma_1, s_2 \models \sigma_2, \cdots, s_n \models \sigma_n \| C. \quad (2)$$

This sample rule can be interpreted as follows: if s_1 supports σ_1, s_2 supports σ_2, and so on, thus we can infer that s_0 supports σ_0, under constraint C. In QUIXOTE, situated inference rules are rephrased as follows[11]:

situation theory		QUIXOTE
situation	$\Leftrightarrow$	module
infon	$\Leftrightarrow$	object term
role	$\Leftrightarrow$	label
supporting ($\models$)	$\Leftrightarrow$	membership (:)

The most important correspondence between them is the description of a rule, viz. (2), and a QUIXOTE rule (3).

$$m :: \sigma <= m_1 : \tau_1, m_2 : \tau_2, \ldots, m_n : \tau_n \| C. \quad (3)$$

2.3 Abductive Reasoning in QUIXOTE

Let us consider the following famous sentences ([10] p. 15; also in [2] p. 105), that are utterances made by two people, A and B, as an example of conditional reasoning:

A: "If Bizet and Verdi are compatriots, they are Italian." (4)

B: "If Bizet and Verdi are compatriots, they are French." (5)

[1] In the syntax of QUIXOTE system, $\sqsubseteq$ is =<.

The first three lines of the program above say that some agent called `hitchcock` is watching some actor `X`, in the movie called `hitchcock_life` (life of Hitchcock). The fourth line represents that, in the `real` world, there is a person called `hitchcock`. The fifth to seventh lines claim that in the situation of some movie called `M`, it is probable that:

- the name `X` can refer to the very role in the movie, or
- the name `X` can refer to the role `Y` that is played by `X` in the movie.

Finally in the eighth to eleventh lines, two roles, that of a man at a bus stop and that of `hitchcock`, are required in the movie `hitchcock_life`.

Here, let us ask what Hitchcock saw in his real life. We acquire the following answers.

```
?-real:see[obj=X,agt=hitchcock].
** Answer 1 **
    X == man[place=bus_stop]
** Answer 2 **
    X == hitchcock
```

In the case of (7), there is no object that corresponds to the unicorn in the real world, so only one interpretation can be derived by a similar program. This means that the unicorn necessarily exists only in a hypothetical world.

3 Conclusion

We introduced the *QUIXOTE* language as a tool to deal with complicated natural language phenomena. Compared with related works such as F-logic [5] (as a DOOD language), CLP languages [4, 12], KR languages such as LOGIN[1] and CIL[6], situated inference system PROSIT [7], the new mechanism of *QUIXOTE* is summarized as follows. First, an object-orientation concept such as *object identity* is introduced into the logic programming as the fundamental philosophy. Secondly, the concept of *module* enables us local definition in a large knowledge-base. Thirdly, its logical inference system is extended to be able to restricted abduction.

We first applied the sorted feature structure of *QUIXOTE* to constraint-based grammar formalisms, and then we showed that *object identity* and *module* in *QUIXOTE* could contribute to the description of situation-based semantics.

References

[1] H. Aït-Kaci and R. Nasr. LOGIN: A Logic Programming Language with Built-In Inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[2] J. Barwise. *The Situation in Logic*. CSLI Lecture Notes 17, 1989.

[3] B. Carpenter. *The Logic of Typed Feature Structure*. Cambridge University Press, 1992.

[4] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proceedings of the 14th ACM POPL*, pages 111–119, Munich, 1987.

[5] M. Kifer. Logical Foundation of Object-Oriented and Frame-Based Language. *Technical Report 90/14*, SUNY at Stony Brook, June 1990.

[6] K. Mukai and H. Yasukawa. Complex Indeterminates in Prolog and its Application to Discourse Models. *New Generation Computing*, 3(4):441–466, 1985.

[7] H. Nakashima, S. Peters, and H. Schutze. Communication and Inference through Situations. In *Proc. of IJCAI '91*, pages 76–81, 1991.

[8] T. Nishioka, R. Ojima, H. Tsuda, and K. Yokota. Procedural Semantics of a DOOD Programming Language *QUIXOTE*. In *SIG-DBS No.94* of Inf. Proc. Soc. Japan., pages 1–10, 1993. (in Japanese).

[9] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, 1993. (to appear).

[10] W. V. Quine. *Methods of Logic (revised edition)*. New York: Holt, Rinehart, and Winston. 1959.

[11] S. Tojo and H. Yasukawa. Situated Inference of Temporal Information. In *Proc. of FGCS92*, pages 395–404, 1992.

[12] H. Tsuda, K. Hasida, and H. Sirai. JPSG Parser on Constraint Logic Programming. In *Proc. of 4th ACL European Chapter*, pages 95–102, 1989.

[13] H. Yasukawa, H. Tsuda, and K. Yokota. Objects, Properties, and Modules in *QUIXOTE*. In *Proc. of FGCS '92*, pages 257–268, 1992.

[14] K. Yokota and H. Yasukawa. Towards an Integrated Knowledge-Base Management System. In *Proc. of FGCS '92*, pages 89–112, 1992.

A Legal Reasoning System on a Deductive Object-Oriented Database

Chie Takahashi
JIPDEC *

Kazumasa Yokota
ICOT †

Abstract

A legal reasoning system is a large-scale knowledge information processing system into which many technologies such as artificial intelligence, natural language processing, and databases are integrated. From a database point of view, this application features many kinds of data and knowledge, and provides many research topics for next generation databases: features of very large databases and knowledge-bases, their classification, treatment of partial information, query processing containing high-level reasoning, and so on. Further, it suggests ideas for the boundaries between databases and applications. In this paper, we explain our experimental legal reasoning system that is based on the deductive object-oriented database system QUIXOTE, and show how the effectiveness of the extended features for next generation databases.

1 Introduction

Recently, legal reasoning has attracted much attention from researchers in the field of artificial intelligence, with great expectations for its *big* application. Legal reasoning systems are very important applications, whose development, like that of theorem provers, dates back to before artificial intelligence was proposed, (for example, see [3]). In fact, laws are related not only to the judicial world but also to all social activities. To support legal interpretation and reasoning in a wide range of situations, many systems have been developed, including those capable of planning tax-saving strategies, negotiation of payment of damages, making contract documents, predicting judgements and supporting legislation. Many works on expert systems for such applications have been published, while powerful legal database systems have not yet been reported.

In the Japanese FGCS (Fifth Generation Computer System) project, legal reasoning systems were considered quite critical and two prototype legal reasoning systems were developed: HELIC-II [5] and TRIAL [6, 11, 9].

For the above systems, we provide database and knowledge-base management facilities: QUIXOTE[10] and Kappa-P [9]. QUIXOTE is a deductive object-oriented database (DOOD) language and knowledge representation language, used for describing and classifying complex legal data and knowledge, while Kappa-P is a parallel nested relational database management system, used to store large volumes of legal data. Especially, all data and knowledge in TRIAL is written in QUIXOTE and some advanced query processing facilities, such as hypothetical reasoning and hypothesis generation (abductive reasoning), are provided for legal reasoning by QUIXOTE.

In this paper, we report on the experimental system, TRIAL, and illustrate the effectiveness of the advanced features of the DOOD system. Based on our practical experience, we discuss the roles we expect databases to play in legal applications, and the features that should be provided for next generation databases. This paper presents new applications of DOOD languages to knowledge information processing by presenting an overview of the TRIAL system in QUIXOTE as an example and discussing the features that will be required by next generation database systems. In Section 2, we briefly explain the kinds of features needed for legal reasoning. By way of example, we consider the case of worker's compensation law. In Section 3, we describe some of the features of QUIXOTE, based on the above example. In Section 4, we consider the entire example database used in TRIAL. Lastly, we discuss the features demanded of next generation databases, especially those derived from legal applications.

2 Legal Reasoning

2.1 Basic Model

The analytical legal reasoning process is considered as consisting of three steps: *fact finding*, *statutory interpretation*, and *statutory application*. Although fact finding is very important as a starting point, it is beyond the

*Japan Information Processing Development Center (JIPDEC), 3-5-8, Shibakoen, Minato-ku, Tokyo 108, JAPAN, e-mail: j-takaha@icot.or.jp.

†Institute for New Generation Computer Technology (ICOT), 21F., Mita-Kokusai Bldg., 1-4-28, Mita, Minato-ku, Tokyo 108, JAPAN, e-mail: kyokota@icot.or.jp.

capabilities of current technologies. So, we assume new cases to already be represented in an appropriate form for our system. Statutory interpretation is a particularly interesting theme from an artificial intelligence point of view. Our legal reasoning system, TRIAL, focuses on statutory interpretation as well as statutory application.

Although there are many approaches to statutory interpretation, we follow the procedure below:

- *analogy detection*
  Given a new case, similar precedents to that case are retrieved from an existing precedent database.
- *rule transformation*
  Precedents (interpretation rules), extracted by analogy detection, are abstracted until the new case can be applied to them.
- *deductive reasoning*
  Apply the new case, in a deductive manner, to abstract interpretation rules transformed by rule transformation. This step may include statutory application because it is used in the same manner.

Among these steps, a strategy enabling analogy detection is essential to legal reasoning for *more efficient* detection of *better* precedents, which ultimately determines the quality of the results of legal reasoning. As the primary objective of TRIAL is to investigate the possibilities of *Quixote* in this area and develop a prototype system, we focus only on a small target. That is, to what extent should interpretation rules be abstracted for a new case, to get an answer with a plausible explanation, but not for a general abstraction mechanism.

2.2 Example

In this paper, we consider a simplified example related to "*karōshi*" (death from overwork) to discuss the applicability of *Quixote* to legal reasoning. A new case, *new-case*, is as follows:

> *Mary*, a *driver*, employed by a company, "*S*", died from a *heart-attack* while taking an *intermission* between jobs. Can this case be applied to the *worker's compensation law*?

3 Features of *Quixote*

Quixote is based on several concepts: object identity, subsumption relation, subsumption constraint, property inheritance, module, submodule relation, and rule inheritance. From a database point of view, the language is a DOOD language while, from a logic programming point of view, it is thought of as an extended constraint logic programming language based on subsumption constraints. There are some differences from the new F-logic[4, 2]: the representation of object identity, the introduction of subsumption constraints, update semantics, and query processing. In this section, we explain some of its features, used in the above example. See the details of *Quixote* in [9, 8, 10].

3.1 Object Identity and Subsumption Relation

Objects in *Quixote* are identified by extended terms called *object terms*, that correspond to object identifiers (oids). An object term consisting of an atomic symbol is classified as *basic*, while an object term in the form of a tuple is referred to as *complex*. In the above example, *mary*, *driver*, *heart-attack* and *intermission* are basic, while *org*[*name* ="*S*"] is complex. Generally, an object term is a variable or a term having the following form:

$$o[l_1 = t_1, \cdots, l_n = t_n] \ (0 \leq n)$$

where $o, l_1, \cdots, l_n$ are basic and $t_1, \cdots, t_n$ are object terms. $l_1, \cdots, l_n$ are called *labels*.

Object terms are related to each other by a *subsumption relation* (a kind of is_a relation). Given partial order between the basic object terms, it is extended between complex object terms as usual:

$$\begin{aligned} org[name = \text{"}S\text{"}] &\sqsubseteq org. \\ org[name = X, president = Y] &\sqsubseteq org[name = X, president = X] \end{aligned}$$

As the construction of a lattice from a partially ordered set, like that in [1], is well-known, we can assume that a set of object terms with *top* and *bot* constitutes a lattice, without losing generality. The meet and join operations of o_1 and o_2 are denoted by $o_1 \downarrow o_2$ and $o_1 \uparrow o_2$, respectively.

3.2 Subsumption Constraints and Property Inheritance

The property of an object is represented as a subsumption constraint. The pair constituted by an object term o and a label l, denoted $o.l$, is called a *dotted term*, which plays the role of a variable ranging over the domain of the object terms. Furthermore, a pair constituted by a dotted term and a label is, itself, also a dotted term. If t_1, t_2 is an object term or dotted term, then a *subsumption constraint* is defined as follows:

$$t_1 \sqsubseteq t_2.$$

That is, a *property* of an object o is a subsumption constraint with a dotted term starting with o. In other words, it is defined as a triple constituted by a label, a subsumption relation, and a value. For example, the *result* of the *new-case* is represented by the following subsumption constraint: $new\text{-}case.result \cong heart\text{-}attack$.

The syntactic construct for representing an object term with subsumption constraints is called an *attibute term*. Let o be an object term, C a set of subsumption constraints, then $o|C$ is an attribute term. For example, the following attribute term represents that the *new-case* is that *Mary* died from a *heart-attack* while taking an *intermission*:

$$\begin{aligned} new\text{-}case|\{&new\text{-}case.who \cong mary,\\ &new\text{-}case.while \cong intermission,\\ &new\text{-}case.result \cong heart\text{-}attack\} \end{aligned}$$

There are some syntax sugars in *Quixote*:

$$o|\{o.l \sqsubseteq t\} \Leftrightarrow o/[l \rightarrow t] \quad o|\{o.l \sqsupseteq t\} \Leftrightarrow o/[l \leftarrow t] \quad o|\{o.l \cong t\} \Leftrightarrow o/[l = t]$$

Thus, the above attribute term can be represented as follows:

$$new\text{-}case/[who = mary, while = intermission, result = heart\text{-}attack]$$

Notice that there are two kinds of properties: those that appear in object terms and those as subsumption constraints. The former are *intrinsic* for an object, while the latter are *extrinsic* for an object. Regarding the extrinsic properties, the following constraint solver is applied to a set of subsumption constraints:

$$\begin{array}{rcl@{\qquad}rcl} o = o & \Longrightarrow & \text{eliminated} & & & \\ o_1 \sqsupseteq o_2 & \Longrightarrow & o_2 \sqsubseteq o_1 & o_1 \sqsubseteq o_2, o_2 \sqsubseteq o_3 & \Longrightarrow & o_1 \sqsubseteq o_3 \\ o_1 \sqsubseteq o_2, o_1 \sqsubseteq o_3 & \Longrightarrow & o_1 \sqsubseteq o_2 \downarrow o_3 & o_1 \sqsubseteq o_3, o_2 \sqsubseteq o_3 & \Longrightarrow & o_1 \uparrow o_2 \sqsubseteq o_3 \end{array}$$

Any set of subsumption constraints will produce a unique solution by applying the above rules[7].

It is natural to assume that extrinsic properties are inherited by object terms with respect to $\sqsubseteq$-ordering. Consider the following example:

$$\begin{aligned} &myocardial\text{-}infarction \sqsubseteq heart\text{-}attack\\ &heart\text{-}attack/[arteriosclerosis \rightarrow yes] \end{aligned}$$

Since *myocardial-infarction* is a kind of *heart-attack* and *heart-attack* has a property $[arteriosclerosis \rightarrow yes]$, *myocardial-infarction* has the same property by inheritance from *heart-attack*.

Property inheritance between objects is defined by the following rule:

$$\begin{array}{rcl} o_1 \sqsubseteq o_2 & \Longrightarrow & o_1.l \sqsubseteq o_2.l \\ o_1[l = t] & \Longrightarrow & o_1.l = t \end{array}$$

According to the rules, we obtain downward and upward inheritance, multiple inheritance, and exception as follows:

$$\begin{array}{rcl} o_1 \sqsubseteq o_2, o_2/[l \rightarrow t] & \Longrightarrow & o_1/[l \rightarrow t] \\ o_1 \sqsubseteq o_2, o_1/[l \leftarrow t] & \Longrightarrow & o_2/[l \leftarrow t] \\ o_1 \sqsubseteq o_2, o_2 \sqsubseteq o_3, o_2/[l = t] & \Longrightarrow & o_1/[l \rightarrow t], o_3/[l \leftarrow t] \\ o_1 \sqsubseteq o_2, o_1 \sqsubseteq o_3, o_2/[l \rightarrow t_2], o_3/[l \rightarrow t_3] & \Longrightarrow & o_1/[l \rightarrow t_1 \downarrow t_2] \\ o_1 \sqsupseteq o_2, o_1 \sqsupseteq o_3, o_2/[l \leftarrow t_1], o_3/[l \leftarrow t_2] & \Longrightarrow & o_1/[l \leftarrow t_1 \uparrow t_2] \\ o_1/[l \rightarrow t_2] & \Longrightarrow & o_1[l = t_1]/[l = t_1] \end{array}$$

3.3 Rule and Module

A rule is defined as in constraint logic programming, as follows:

$$a_0 \Leftarrow a_1, \cdots, a_n || D$$

where $a_0, a_1, \cdots, a_n$ are attribute terms and D is a set of subsumption constraints. a_0 is called a *head*, $a_1, \cdots, a_n, D$ is called a *body*, and a_i is called a *subgoal*. A rule means that if the body is satisfied then the head is satisfied. If a body is empty, then the rule is called a *fact*.

For example, the following is a rule for judgement.

$$\begin{aligned} judge[case = X]/[judge \rightarrow insurance] \\ \Leftarrow & judge[case = X]/[judge \rightarrow job\text{-}causality], \\ & judge[case = X]/[judge \rightarrow job\text{-}execution] \\ & || \{X \sqsubseteq case\}. \end{aligned}$$

It means that if the judgement of some case, $judge[case = X]$ where $X \sqsubseteq case$, is *job-causality* and *job-execution*, then the judgement is *insurance*.

A module corresponds to a part of the world (situation) or a local database. The module concepts play an important role in classifying knowledge, modularizing a program or a database, assumption-based reasoning, and dealing with any inconsistency in a database in QUIXOTE.

A *module* is defined as a set of rules as follows:

$$m :: \{r_1, \cdots, r_n\}$$

where m is an object term called a *module identifer* (mid) and $r_1, \cdots, r_n$ are rules. m is sometimes used instead of a module itself, if there is no confusion.

The definition of rules is extended for external reference of objects:

$$m_0 :: \{a_0 \Leftarrow m_1 : a_1, \cdots, m_n : a_n || D\}$$

where $m_0, m_1, \cdots, m_n$ are mids. It means that the module m_0 has a rule such that if a_i and D are satisfied in the module m_i for all $1 \leq i \leq n$, then a_0 is satisfied in the module m_0. As an attribute term can be separated into an object term and a set of constraints, the rule can be rewritten as follows:

$$m_0 :: \{o_0 | C_0 \Leftarrow m_1 : o_1, \cdots, m_n : o_n || C\}$$

where $a_i = o_i | C_i (0 \leq i \leq n)$ and $C = C_1 \cup \cdots \cup C_n \cup D$.

Importing and exporting rules are done by *rule inheritance*, defined in terms of the binary relation (written $\sqsupseteq_S$) between modules, called a *submodule relation* as follows:

$$m_1 \sqsupseteq_S m_2, \ \ m_1 :: R_1. \ \ m_2 :: R_2 \iff m_1 :: R_1 \cup R_2, \ \ m_2 :: R_2$$

where m_1, m_2 are modules and R_1, R_2 are sets of rules. The right hand side of $\sqsupseteq_S$ in a submodule definition may be a formula of mids with set operations. For example, if we have

$$\begin{array}{llll} m_1 :: \{r_{11}, r_{12}, r_{13}\} & m_4 :: \{r_{41}, r_{42}\} & m_2 :: \{r_{21}, r_{22}\} & m_4 \sqsupseteq_S m_1 \cup m_3 \\ \{m_2, m_3\} :: r_{31} & m_5 \sqsupseteq_S m_2 \backslash m_3 & & \end{array}$$

then m_4 has $\{r_{11}, r_{12}, r_{13}, r_{31}, r_{41}, r_{42}\}$ and m_5 has $\{r_{21}, r_{22}\}$.

It is possible for inconsistent knowledge to co-exist by making use of the module mechanism. For example, consider that it cannot be said for certain whether *mary* has *arteriosclerosis*. The following shows how such a problem is handled:

$$\begin{aligned} new\text{-}case_1 :: mary/[arteriosclerosis \rightarrow yes]. \\ new\text{-}case_2 :: mary/[arteriosclerosis \rightarrow no]. \end{aligned}$$

where $new\text{-}case_1$ and $new\text{-}case_2$ are not related by submodule relation.

A *database* or a *program* is defined as the triple (S, M, R) of a finite set of subsumption relations S, a set of submodule relations M, and a set of rules R.

3.4 Query Processing

Query processing basically corresponds to resolution and constraint solving in constraint logic programming. One of the main features of data and knowledge in knowledge information processing, such as legal reasoning, is that the information is partial, that is to say, sufficient information need not necessarily be given. For example, a new case might lack some important facts. So, query and an answer is extended for treating partial information.

A *query* is defined as the pair (A, P) (written ?-A;;P) of a set of attribute terms A and a program P, where A is referred to as the *goal* and P as a *hypothesis*.

Consider a database DB. A query ?-A;;P to DB is equivalent to a query ?-A to $DB \cup P$ (If $DB = (S_1, M_1, R_1)$ and $P = (S_2, M_2, R_2)$ then $DB \cup P = (S_1 \cup S_2, M_1 \cup M_2, R_1 \cup R_2)$). That is, P is inserted into DB before A is processed. In other words, P works as a hypothesis for ?-A. As hypotheses are incrementally inserted into a database, nested transactions are introduced to control such insertions. See the details in [10].

An *answer* is defined as the triple (D, V, E) of a set of subsumption constraints D that cannot be solved during query processing, a set of variable constraints V that are bounded during query processing, and the corresponding derivation flow E. D lacks information in the database, obtained by abduction, V is an answer in the sense of constraint logic porgramming and E is the explanaiton.

3.5 *Quixote* System

A *Quixote* system consists of a *client* as a user interface in C on UNIX^{TM} and a *server* as a knowledge-base engine in a parallel logic programming, KL1, which was designed and developed by ICOT to run under UNIX. A server and clients are connected by the TCP/IP protocol. One of the user interfaces is *Qmacs* using *GNU-Emacs*, while others are windows that are implemented using *X-Window* and which display some figures graphically. The overall architecture is shown in Figure 1, while Figure 2 shows an example of a graphical figure.

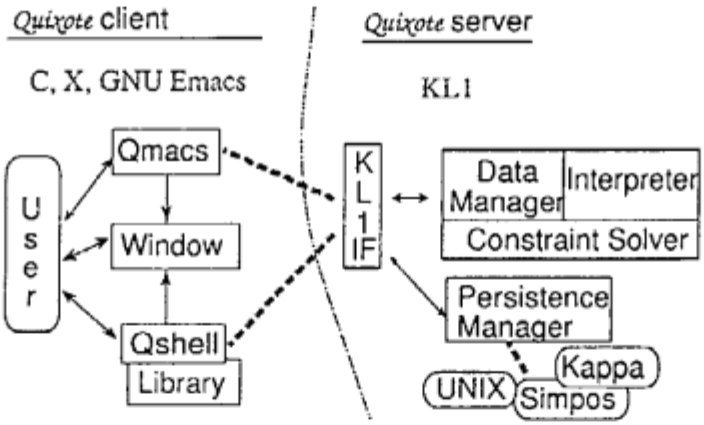


Figure 1: Architecture of *Quixote* system

4 Legal Reasoning on *Quixote*

In this section, we explain our legal reasoning system, TRIAL, written as a *Quixote* system. The overall architecture of the system, written in KL1, is shown in Figure 3. *Quixote* supports the functions of rule transformation and deductive reasoning as native functions besides the database component, while TRIAL supports analogy detection besides the interface component. All data and knowledge in the database component is written in *Quixote*.

A new case, *new-case*, (in the New Case Database), is represented as the module *new-case* in *Quixote* as follows:

$$
\begin{aligned}
new\text{-}case :: \{&new\text{-}case/[who = mary,\\
&\quad while = intermission,\\
&\quad result = heart\text{-}attack];;\\
&relation[state = employ, employee = mary]\\
&\quad /[affiliation = organization[name = \text{"}S\text{"}],\\
&\quad job \rightarrow driver]\}
\end{aligned}
$$

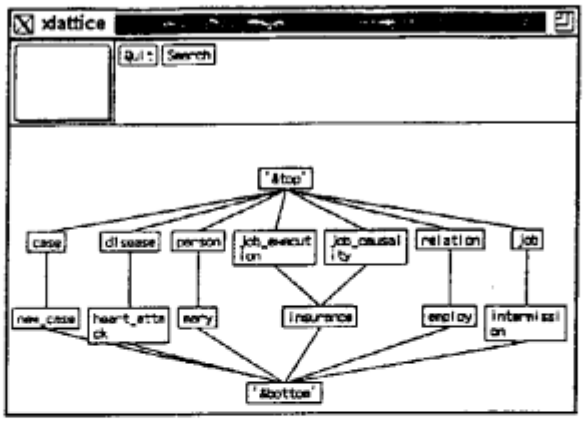


Figure 2: A window display of a lattice

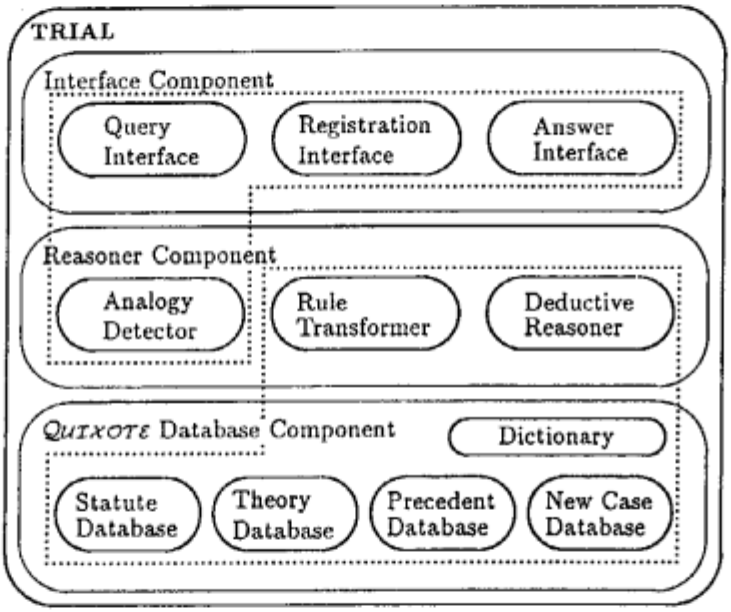


Figure 3: Architecture of TRIAL

where ";;" is a delimiter between rules. Assume that there are two *abstract* precedents [1] of *job-causality* and *job-execution*:

$$
\begin{aligned}
case_1 :: judge[case = X]/[judge \rightarrow job\text{-}execution] \\
&\Leftarrow relation[state = Y, employee = Z]/[cause = X], X \\
&\quad ||\{X \subseteq \mathtt{parm}.case, Y \subseteq \mathtt{parm}.state, Z \subseteq \mathtt{parm}.employee\};; \\
case_2 :: judge[case = X]/[judge \rightarrow job\text{-}causality] \\
&\Leftarrow X/[while = Y, result = Z], \\
&\quad ||\{X \subseteq \mathtt{parm}.case, Y \subseteq \mathtt{parm}.while, Z \subseteq \mathtt{parm}.result\}.
\end{aligned}
$$

Note that variables X, Y, and Z in both rules are restricted by the properties of an object parm. That is, they are already abstracted by parm and their abstract level (the range of variables) is controlled by parm's properties. Such precedents are retrieved from the precedent database by analogy detection and are abstracted by rule transformation.

We must consider the *labor-law* (in the statute database) and a *theory* (in the theory database) as follows:

[1] In this paper, we omit the rule transformation step and assume that abstract interpretation rules are given.

$$
\begin{aligned}
\mathit{labor\text{-}law} :: \mathit{organization}[\mathit{name}=X] \\
&/[\mathit{responsible} \rightarrow \mathit{compensation}[\mathit{object}=Y, \mathit{money}=\mathit{salary}]] \\
&\quad \Leftarrow \mathit{judge}[\mathit{case}=C]\ /[\mathit{judge} \rightarrow \mathit{insurance}], \\
&\quad\ \ \mathit{relation}[\mathit{state}=Z, \mathit{employee}=Y] \\
&\quad\quad /[\mathit{affiliation}=\mathit{organization}[\mathit{name}=X]] \\
&\quad\ \ \|\{C \sqsubseteq \mathit{case}\}.
\end{aligned}
$$

$$
\begin{aligned}
\mathit{theory} :: \mathit{judge}[\mathit{case}=X]/[\mathit{judge} \rightarrow \mathit{insurance}] \\
&\quad \Leftarrow \mathit{judge}[\mathit{case}=X]/[\mathit{judge} \rightarrow \mathit{job\text{-}causality}], \\
&\quad\ \ \mathit{judge}[\mathit{case}=X]/[\mathit{judge} \rightarrow \mathit{job\text{-}execution}] \\
&\quad\ \ \|\{X \sqsubseteq \mathit{case}\}.
\end{aligned}
$$

Furthermore, we must define the parm object as follows:

$$
\begin{aligned}
\mathit{parm} :: \mathrm{parm}/[&\mathit{case} = \mathit{case}, \mathit{state} = \mathit{relation}, \mathit{while} = \mathit{job}, \\
&\mathit{result} = \mathit{disease}, \mathit{employee} = \mathit{person}].
\end{aligned}
$$

This object results from abstracting precedents and is used for the control of predicting judgements.

To use parm for case_1 and case_2, we define the following submodule relation:

$$\mathit{parm} \sqsupseteq_S \mathit{case}_1 \cup \mathit{case}_2.$$

This information is dynamically defined during rule transformation, because the choice of precedents is experimental.

Furthermore, we must define the subsumption relations:

case	$\sqsupseteq$	*new-case*	*person*	$\sqsupseteq$	*mary*
relation	$\sqsupseteq$	*employee*	*job-causality*	$\sqsupseteq$	*insurance*
disease	$\sqsupseteq$	*heart-attack*	*job-execution*	$\sqsupseteq$	*insurance*
job	$\sqsupseteq$	*intermission*			

Then, we can ask some questions with a hypothesis to the above database:

1) If *new-case* inherits *parm* and *theory*, then what kind of *judgment* do we obtain?

$$
\begin{aligned}
&?\text{-}\ \mathit{new\text{-}case} : \mathit{judge}[\mathit{case}=\mathit{new\text{-}case}]/[\mathit{judge}=X];; \\
&\mathit{new\text{-}case} \sqsupseteq_S \mathit{parm} \cup \mathit{theory}.
\end{aligned}
$$

We get three answers, in which the first is returned unconditionally, while the latter two are answers with assumptions:

- $X \sqsubseteq \mathit{job\text{-}causality}$
- if $\mathit{new\text{-}case}:\mathit{judge}[\mathit{case}=\mathit{new\text{-}case}]$ has $\mathit{judge} \sqsubseteq \mathit{job\text{-}execution}$, then $X \sqsubseteq \mathit{insurance}$
- if $\mathit{new\text{-}case}:\mathit{relation}[\mathit{state}=\mathit{employ}, \mathit{employee}=\mathit{mary}]$ has $\mathit{cause}=\mathit{new\text{-}case}$, then $X \sqsubseteq \mathit{insurance}$

2) If *new-case* inherits *labor-law* and *parm*, then what kind of responsibility should the organization to which Mary is affiliated have?

$$
\begin{aligned}
&?\text{-}\ \mathit{new\text{-}case} : \mathit{organization}[\mathit{name}=\text{“}S\text{”}]/[\mathit{responsible}=X];; \\
&\mathit{new\text{-}case} \sqsupseteq_S \mathit{parm} \cup \mathit{labor\text{-}law}.
\end{aligned}
$$

We get two answers with assumptions:

- if $\mathit{new\text{-}case}:\mathit{judge}[\mathit{case}=\mathit{new\text{-}case}]$ has $\mathit{judge} \sqsubseteq \mathit{job\text{-}execution}$,
  then $X \sqsubseteq \mathit{compensation}[\mathit{obj}=\mathit{mary}, \mathit{money}=\mathit{salary}]$
- if $\mathit{new\text{-}case}:\mathit{relation}[\mathit{state}=\mathit{employ}, \mathit{employee}=\mathit{mary}]$ has
  $\mathit{cause}=\mathit{new\text{-}case}$,
  then $X \sqsubseteq \mathit{compensation}[\mathit{obj}=\mathit{mary}, \mathit{money}=\mathit{salary}]$

For analogy detection, the parm object plays an essential role in determining how to abstract rules, as in case_1 and case_2, which properties or objects are to be abstracted using the properties of parm, and which values are to be given for the properties of parm. In this experimental system, which adds to the basic functions of QUIXOTE, we have experimented with not only hypothetical reasoning and abduction, but also such abstraction, that is, analogy detection.

For TRIAL's user interface, QUIXOTE returns explanations (derivation graphs) with corresponding answers, if necessary. The TRIAL interface displays this graphically according to a user's request. By judging an answer on the basis of the validity of the assumptions and the corresponding explanation, the user can update the database or change the abstraction strategy.

The TRIAL system was implemented for four months by two persons. This highlights the productivity of QUIXOTE for such knowledge information processing systems.

5 Concluding Remarks

Knowledge information processing applications will play an important role in future information processing systems, including future generation databases. Legal reasoning systems, one such application, show typical requirements for future database systems.

From the experiences of TRIAL, we can list several requirements for next generation database systems, which need not necessarily be only for legal applications:

- *Processing partial information*
  As data and knowledge are often not given in a perfect form, unlike conventional applications, we must consider ambiguous or erroneous data as well as null values and logically incomplete information such as negation and disjunction. In *Quixote*, we use subsumption constraints to handle ambiguous and partially lacking properties. They are useful not only to knowledge databases but also to scientific databases.

- *Realizing an environment for thinking experiments*
  Answers are not necessarily given uniquely in knowledge information processing, but are refined by repeating trial-and-error querying with the users. In this sense, the features of hypothetical reasoning and hypothesis generation are very important.

- *Framework of very large database and knowledge-base*
  Classification mechanisms are very important for storing large databases and knowledge-bases. Subsumption and submodule hierarchies contribute to such classification. Especially, a framework that allows inconsistent data and knowledge to co-exist is needed.

- *Integration of heterogeneous data and knowledge*
  Even if we consider only one application, we can find various kinds of data and knowledge within it. For example, legal data includes large amounts of text data as the primary data, and abstracted data or knowledge (including rules) as the higher level data. Although we have not integrated such data and knowledge into TRIAL, the integration of such heterogeneous data and knowledge will become very important.

- *Knowledge discovery in databases*
  To classify large databases and knowledge-bases, two kinds of knowledge discovery will be needed: how to find erroneous and lacking information; how to find new knowledge (abstracted rule in the above example).

- *Integration of technologies in related areas*
  Database technologies have been spreading: for example, we now have deductive databases, database programming languages, deductive object-oriented databases, and very large knowledge-bases. More technologies in many areas such as artificial intelligence, programming languages, and operating systems, should be embedded into database systems.

Quixote and *μ-Quixote* systems, which run under UNIX, have been released as ICOT free software. We have been extending the features of the systems as next generation database systems and a framework of heterogeneous, distributed, cooperative problem solvers to enable their application to a wider range of knowledge information processing applications such as natural language processing and genetic information processing systems.

Acknowledgments

The authors wish to thank Nobuichiro Yamamoto (Hitachi, Ltd.) for designing and implementing the TRIAL system, and all the members of the *Quixote* project for their valuable advice.

References

[1] H.Aït-Kaci. "An Algebraic Semantics Approach to the Effective Resolution of Type Equations", *Theoretical Computer Science*, no.45, 1986.

[2] A.J. Bonner and M. Kifer, "Transaction Logic Programming", *Proc. Int'l Logic Programming*, 1993.

[3] L.O. Kelso, "Does the Law Need a Technological Revolution?", *Rocky Mt. Law Rev.*, vol.18, pp.378-392, 1946.

[4] M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages", *SUNY TR 93/06*, June, 1993.

[5] K. Nitta, Y. Ono, T. Chino, T. Ukita, and S. Amano, "HELIC-II: A Legal Reasoning System on the Parallel Inference Machine", *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

[6] N. Yamamoto, "TRIAL: a Legal Reasoning System (Extended Abstract)", *Joint French-Japanese Workshop on Logic Programming*, Renne, France, July, 1991.

[7] H. Yasukawa and K. Yokota, "Labeled Graph as Semantics of Objects", *Proc. Joint Workshop of SIGDBS and SIGAI of IPSJ*, Nov., 1990.

[8] H. Yasukawa, H. Tsuda, and K. Yokota. "Objects, Properties, and Modules in QUIXOTE", *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

[9] K. Yokota and H. Yasukawa, "Towards an Integrated Knowledge-Base Management System — Overview of R&D on Databases and Knowledge-Bases in the FGCS Project", *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, June 1-5, 1992.

[10] K. Yokota, H. Tsuda, and Y. Morita, "Specific Features of a Deductive Object-Oriented Database Language QUIXOTE", *Workshop on Combining Declarative and Object-Oriented Databases, (ACM SIGMOD'93 Workshop)*, Washington DC, May 29, 1993.

[11] K. Yokota and M. Shibasaki, "Can Databases Prodict Legal Judgements?", *Joint Workshop of IPSJ SIGDSS and IEICE SIGDE (EDWIN)*, Nagasaki, July 21-23, 1993. (in Japanese)

Representation of viewpoint in New HELIC-II

Katsumi Nitta, Masato Shibasaki, Tsuyoshi Sakata, Takahiro Yamaji
Hiroshi Ohsaki, Satoshi Tojo, Takayuki Suzuki

Institute for New Generation Computer Technology (ICOT)

1 Introduction

Since 1991, we have been developing a legal reasoning system, HELIC-II, on the parallel inference machine PIM [Nitta 92], developed by ICOT. Although HELIC-II is a powerful reasoning system, its inference mechanisms is not sufficient as a general legal reasoning model. Based on our experience with HELIC-II, we commenced our work in a new version of HELIC-II last year. Our goal for the new HELIC-II is to develop a model of general legal reasoning, and to realize it as a software tool in KLIC [Chikayama 89].

Especially, we focused our attention on the reasoning process of argument and on the role of standpoint and viewpoint during argument, because we feel that the cognitive aspect is vital to creating a viable legal reasoning model.

In this paper, we introduce the functions of the revised HELIC-II, its knowledge representation language, and an example of a debate.

2 Overview of the new HELIC-II

2.1 Functions of the new HELIC-II

The new version of HELIC-II has two main functions. The first creates an argument for pursuing a given goal. This function consists of devising arguments to achieve a goal, listing up counter arguments, and to finding strong, favorable arguments. Therefore, the function is a kind of theorem prover, corresponding to the first world undertaken by the prosecutor in court.

The second function creates a counter argument to defeat the other party's argument. This is realized by changing the relative priorities of rules or by changing facts. This corresponds to the work of the prosecutor attorney as they debate.

2.2 HELIC-II three layer reasoning model

Our model consists of three layers (Fig.1).

1. Proof layer (Theorem Proving):
   The role of this layer is to generate an argument for persuing a given goal in the form of an inference tree. This layer consists of two inference engines, namely a rule-based engine and a case-based engine,and three knowledge sources, these being a conceptual dictionary, a rule base of legal rules and a case base of case rules. Both inference engines generate proof trees to reach a single by means of backward reasoning. The case-based engine applies case rules by similarity-based matching [Branting 89], [Nitta 92].

2. Defeasible reasoning layer (Reasoning based on viewpoint):
   This layer contains a module that handles defeasible reasoning and viewpoints. Our defeasible reasoning is based on the concept of priority between rules [Sartor, G. 93], [Prakken, H. 93]. If two defeasible rules result in a conflict, a rule that has priority over another rule is employed. A viewpoint consists of the relative priorities of standpoints, while a standpoint consists of the relative priorities of legal rules and case rules.

3. Debate layer (Argumentation):
   This layer contains a module that finds a counter argument to defeat the other party's argument by reinforcing the current viewpoint. Therefore, the role of this layer is to provide advice to one of the parties during debate.

3 A knowledge representation language for the new HELIC-II

3.1 Overview of *NL*

We designed a knowledge representation language to develop a legal reasoning system based on our previous analysis.

Temporarily, we are referring to our language as *NL*, standing "New HELIC-II Language".

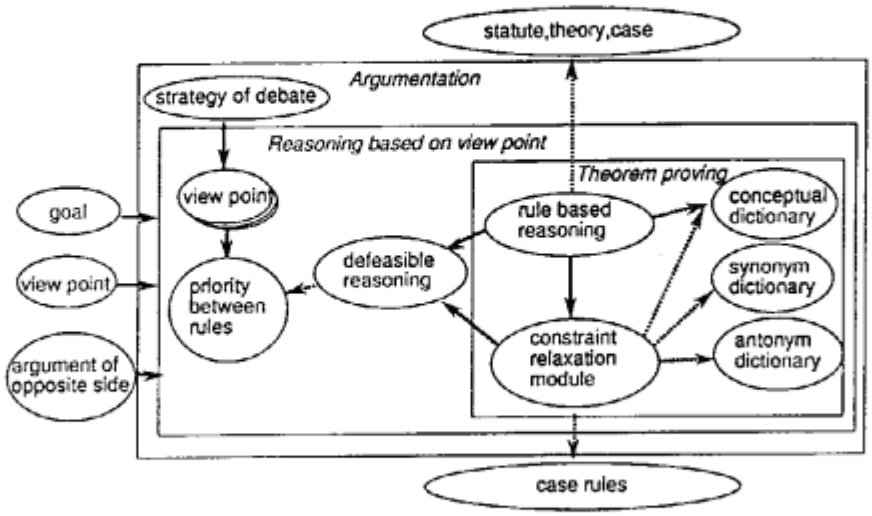


Fig1: Three layers of legal reasoning

NL is an extension of the "LOGIN" [Ait-Kathy 86][Smolka 89] logic programming language with inheritance, by implicit case, negation as failure, higher order representation, defeasible reasoning and analytical reasoning.

(1) Type

Type is a set of objects, and the partial order relation of two types, $A <_T B$, means that A is a subset of B. *Type definition* consists of a "feature definition" and a "subset relation" between types.

$t_i[l_1 : X_1/f_1, ..., l_n : X_n/f_n]$.
$t_i <_T t_j$ (t_i is a type symbol, l_i is a label, X_i is a variable, f_i is a feature definition, and $<_T$ is a subset relation)

For example, the following is the type definitions of "Japanese is a person whose nationality is Japan."

person[age:integer, parent:person, nationality:country].
japanese[nationality:japan].
japanese $<_T$ person.
japan $<_T$ country.

(2) Terms

Types are classified into *verb-type* and *noun-type*. Using these types, we define different kinds of terms - H-term and ψ-term.

H-term takes the following forms.

$$v(c_1 = q_1, c_2 = q_2, ..., c_n = q_n)$$
$$\neg\, v(c_1 = q_1, c_2 = q_2, ..., c_n = q_n)$$

(v is a ψ-term whose top level type is a verb-type, c_i is a case symbol and q_i is an H-term or ψ-term.)

ψ-term takes the following form.

$$X/t$$
$$X/t[l_1 \Rightarrow t_1, l_2 \Rightarrow t_2, ..., l_n \Rightarrow t_n]$$

(X is a type-variable, t is a type, l_i is a label, t_i is a ψ-term. We can omit "$X/$", if it is not necessary.)

Let the *H-term_type* be defined as follows.

$$H\text{-}term\_type[verb : verb\_type, c_1 : X_1, ..., c_n : X_n]$$

Then, the H-term

$$v(c_1 = q_1, c_2 = q_2, ..., c_n = q_n)$$

is considered as a syntax sugar of the following ψ-term, if none of cases are omitted.

$$H\text{-}term\_type[verb \Rightarrow v, c_1 \Rightarrow q_1, ..., c_n \Rightarrow q_n]$$

(3) Variables

NL uses two kinds of variables - a type-variable and a H-term-variable.

A *type-variable* is a symbol whose first character is a capital letter, that appears immediately a type or in place of a type. Identical variables are bound to each other. The following example means "person wishes his child to love him."

wish(agent=Y/person, object=love(agent=person[parent⇒Y], object=Y))

A *H-term-variable* is a symbol whose first character is "@". It appears just after H-term and used as H-term identifier. In H-term denoted by H-term-variable, omitted cases are treated as they have universal quantified variables.

For example, if "place" and "time" are implicit cases of "hit", then the following rule

crime_of_violence(a-object= @act1) ←
hit(agent=X/person, object=Y/person) | @act1.

is interpreted as follows.

crime_of_violence(a-object=hit(agent=X/person, object=Y/person,
place=W, time=Z)) ←
hit(agent=X, object=Y, place=W, time=Z).

In addition to *H-term-variable*, *NL* uses *H-term-constant* as H-term identifier. A *H-term-constant* is a symbol whose first character is "#." It also appears just after H-term. In H-term denoted by H-term-constant, omitted cases are treated as they have skolem constants.

(4) Legal Rules

Legal rules take the following form.

$u :: m_0 : h_0 \leftarrow m_1 : h_1, m_2 : h_2, ..., m_n : h_n.$
(u is a unit name, m_i is a module name, h_i can be either *H-term* or *not H-term* whose cases can have *extended H-term*.)

Here, *unit name* is a rule name, and *not H-term* is an H-term to which "not" is attatched.

Facts are represented by legal rules for which the right hand side is "true." We can thus omit the right-hand side to a represent fact.

(5) Case Rules

A Case rule takes the following form.

$u :: m_0 : h_0 \leftarrow m_1 : h_1, m_2 : h_2, ..., m_n : h_n \mid CRC.$
(u is a unit name, m_i is a module name, h_i is H-term or not H-term, CRC is a constraint relaxation condition)

NL supports the control of unification of specific terms by describing the following conditions in CRC.

- "limit" specifies the upper limit of type generalization.
- "exact" indicates important types or terms. Important types or terms should always be satisfied, while other types or terms may not have.
- Function for measuring the rate of satisfied conditions on the right-hand side of a case rule.

Following is an example of case rule.

u:: liable(agent=tom, object=@injury) ←
read_a_book(agent=jim, place=house1)|@reading,
S/surprise(agent=tom, object=jim, place=house1)|@surprise,
fall(agent=jim, source=sofa, place=house1)|@fall,
T/causality(cause=@surprise, effect=@fall),
injured(agent=jim, place=house1) |@injury,
causality(cause= @fall, place= @injury),
| { limit(surprise, surprise), limit(tom, person), limit(jim, person),
limit(causality, causality), limit(injured, injured),
exact(S), exact(T), exact(@injury), sim > 0.7 }.

The meaning of this rule is "while Jim was reading a book, Tom surprised him. Jim fell from the sofa, and was injured. In this case, Tom is liable for Jim's injury."

(6) Unit, standpoint, viewpoint

Even though a *unit* is defined as an identifier of a legal rule or a case rule, we can extend the definition of a unit to also represent a set of legal rules or case rules.

For example, we can define the unit *decision_of_supreme_court* whose members are $u_i \quad (i > 0)$.

$$decision\_of\_supreme\_court \ :: \ \{u_1, u_2, ..., u_n\}$$

A *standpoint* is a set of priority relations between two units. For example, the standpoint "lex_superior" is that *decision_of_supreme_court* has priority over *decision_of_high_court*.

$$lex\_superior :: \{decision\_of\_high\_court <_S decision\_of\_supreme\_court\}$$

A *viewpoint* is a set of priority relations between standpoints. For example, the viewpoint "view1" is that *lex_superior* has priority over *lex_posterior*.

$$view1 :: \{lex\_posterior <_V lex\_superior\}.$$

(7) Defeasible reasoning

Defeasible reasoning, based on the priorities of rules is one type of non monotonic reasoning.

First, we will define some basic concepts. An *argument* for some goal p is the minimum set of instances of rules and facts needed to draw p. An argument B is the *counter argument* of argument A, provied if A has a sub-argument A' for q and B is the argument for $\neg q$. B *directly defeats* A, if B is a counter argument of A, if the top of the defeasible rule of B takes priority over the top of the defeasible rule of A, and if none of the sub-arguments of B is defeated by the others. B *defeats* A, if B directly defeats the sub-argument of A.

Arguments are classified into *defeated arguments, merely plausible arguments* and *justified argumets*. A defeated argument is one which is defeated by some other arguments. A merely plausible argument is one that is not defined by any counter argument. A justified argument defeats all counter arguments. A *plausible argument* is a merely plausible argument or a justified argument.

Given a goal, facts, a viewpoint and some control parameter, NL will generate plausible arguments.

```
?- solve(Goal, Facts, Viewpoint, Mode, SolutionTable).
```

If we specify "normal mode", NL generates all plausible arguments by retracting unnecessary arguments. On the contrary, if we specify "analysis mode", NL generates all arguments, all counter argunments for each argument, and all counter arguments for each counter argument, and so on. The resultant arguments are stored in the solution table.

A solution table contains the arguments that will produce the given goal, possible counter arguments, and category for each sub-argument. Therefore, we can draw viewpoints which makes an argument to justified one.

4 Conclusion

We presented overview of the new HELIC-II. On this model, we have been trying to simulate the dabate using actual cases.

The mechanisms of interpretation of legal rules and flexible analogical reasoning are next subjects of the research of the new HELIC-II.

References

[Ait-Kathy 86] Ait-Kathy, H. and Nasr,R. *"LOGIN:A Logic Programing Language with Builit-in Inheritance."* in *"J. Logic Prog."*,1986, vol.3, pp. 185-215

[Branting 89] L.K.Branting : *"Representing and Reusing Explanations of Legal Precedents"* in *"International Conference on Artificial Intelligence and Law "*, 1989.

[Chen 89] Chen,W.,Kifer,M.and Warren,D.S. *"HiLog as a Platform for Database Language"* in *"2nd Intl. Workshop on Database Programming Languages"*, Oregon Coast,OR,June 1989.

[Chikayama 89] Chikayama,T,ICOT. *"A Portable and Reasonably Efficient Implementation of KL1."* in *"International Conference on Logic Programming"*, The MIT Press,1993.

[Gelfond 90] Gelfond,M.,and V.Lifschitz. *"Logical Programs with Classical Negation."* in *"Proceedings of the Seventh International Logic Programming Conference"*, New York:MIT Press, 1990, pp.579-597.

[Nitta 92] Nitta,K. *"HELIC-II: a legal reasoning system on the parallel inference machine."* in *"International Conference on FGCS92"*, 1992, pp.1115-1124.

[Smolka 89] Smolka,G.and Ait-Kaci.H. *"Inheritance Hierarchies: Semantics and Unification"* in *"J. Symbolic Computation"*,1989, pp.343-379

[Sartor, G. 93] Sartor,G. *"A Simple Computational Model for Nonmonotonic and Adversarial Legal Reasoning"* in *"International Conference on Artificial Intelligence and Law"*,1993,pp.192-201

[Prakken, H. 93] Prakken,H. *"A logical framework for modeling legal argument"* in *"International Conference on Artificial Intelligence and Law"*,1993,pp.1-9

OWNERSHIP:
A Case Study in the Representation of Legal Concepts
(Preliminary Draft)

L. Thorne McCarty
Computer Science Department
and
Faculty of Law
Rutgers University
New Brunswick, New Jersey 08903

Abstract

This paper is an exercise in computational jurisprudence. It seems clear that the field of AI and Law should draw upon the insights of legal philosophers, whenever possible. But can the computational perspective offer anything in return? We explore this question by focusing on the concept of OWNERSHIP, which has been debated in the jurisprudential literature for centuries. Although the intellectual currents here flow mostly in one direction — from legal philosophy to AI — we show that there are also some insights to be gained from a computational analysis of the OWNERSHIP relation.

In particular, the paper suggests a computational explanation for the emergence of abstract property rights, divorced from concrete material objects.

Presented at:
Conference in Celebration of the 25th Anniversary of the IDG
Florence, Italy
December 1–3, 1993

Building up a Legal Ontology from a General Ontology

Takahira Yamaguchi and Masaki Kurematsu
Faculty of Engineering Shizuoka University
3-5-1 Johoku Hamamatsu Shizuoka, 432 JAPAN

ABSTRACT

In order to develop legal expert systems, we must build up several legal knowledge bases: a rule base for provisions for laws, a case base for judicial precedents, a knowledge base for legal common sense or theory. If such legal knowledge bases would be developed independently, the cost would become enormous. So as to avoid the troublesome, in the field of knowledge engineering, the research on knowledge sharing and reuse is coming up just now, such as the ARPA Knowledge Sharing Initiative in USA. Although there are several issues to achieve the research, it is one key issue to build up a useful domain-specific ontology efficiently and correctly. In this case, the ontology just means the following: concept primitives that compose legal knowledge bases and a concept tree. Because the domain-specific ontology is also a knowledge base, it needs much effort to build it up in no support of a knowledge base building environment. So this paper comes into a new area on the environment to build up a legal ontology from a general ontology that has been already developed. The general ontology comes from a concept dictionary that has been developed at Japan Electronic Dictionary Research Institute. The development process for a legal ontology has three steps: (1) a user (a legal expert) inputs an initial legal ontology, (2) The environment tries to match the initial legal ontology with a general ontology, and (3) The environment tries to identify something wrong with the initial legal ontology using knowledge acquisition strategies, and then has the interaction with the user. The matching mechanism has two level: symbol-level matching and knowledge-level matching. The general concepts matched with a legal concept by the former are lower boundaries in a matched space in a general ontology, and ones by the latter are upper boundaries. The knowledge acquisition strategies has the following: (1) the difference between depths in two ontology; (2) the difference between the number of child nodes in them; and (3) the difference between paths in them. An experiment has been done to the domain of United Nations Convention on Contracts for the International Sale of Goods. It is shown that the matching mechanism and strategies have been available to acquire the definition of legal concepts of act, declaration of intention, offer, reply, acceptance and so on.

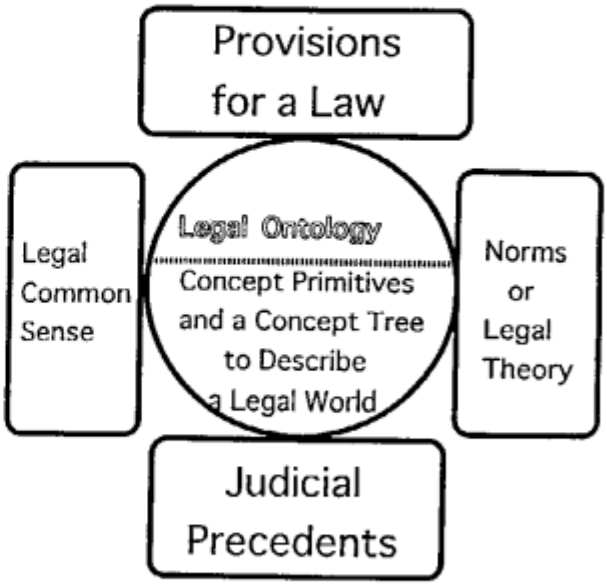
Workshop on Knowledge Representation for Legal Reasoning
March 18, Northeastern Univ. at Boston, Sponsored by ICOT

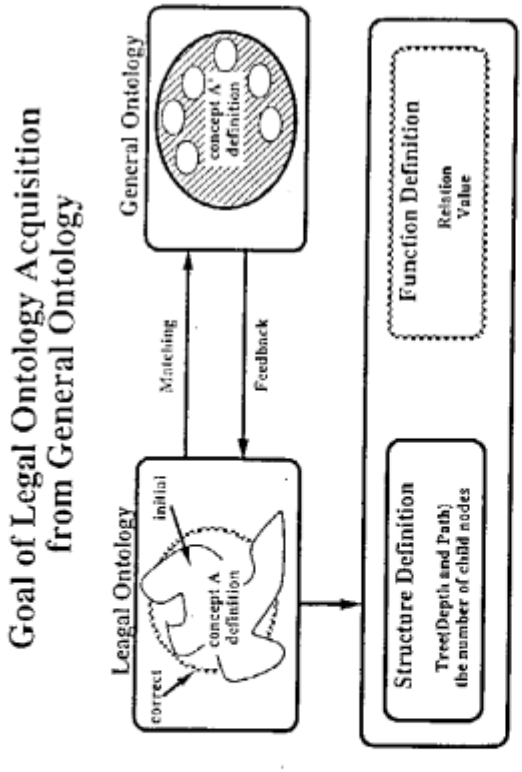
A Legal Ontology Based Environment to Build up a Legal Knowledge Base

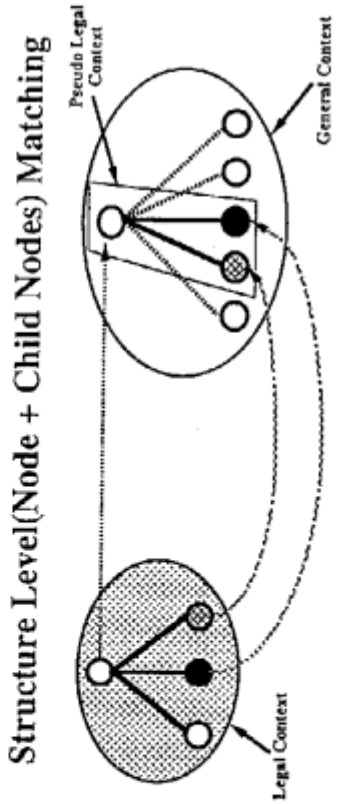
Building up a Legal Ontology from a General Ontology

T. Yamaguchi and M. Kurematsu

Faculty of Engineering
Shizuoka University







Development Processes for A Legal Ontology

STEP1:

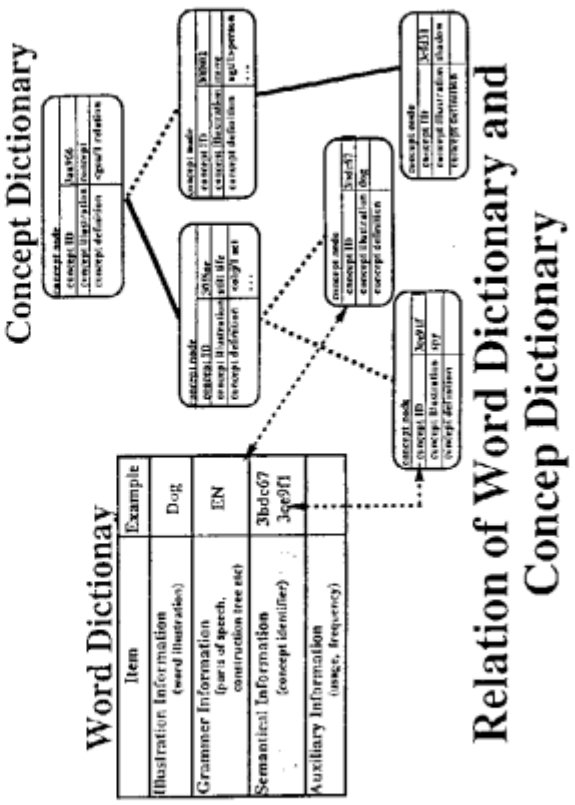
A user (a domain expert) inputs an initial legal ontology (concept descriptions and a concept tree).

STEP2:

The system matches the legal ontology with a general ontology that has a word dic and a concept dic.

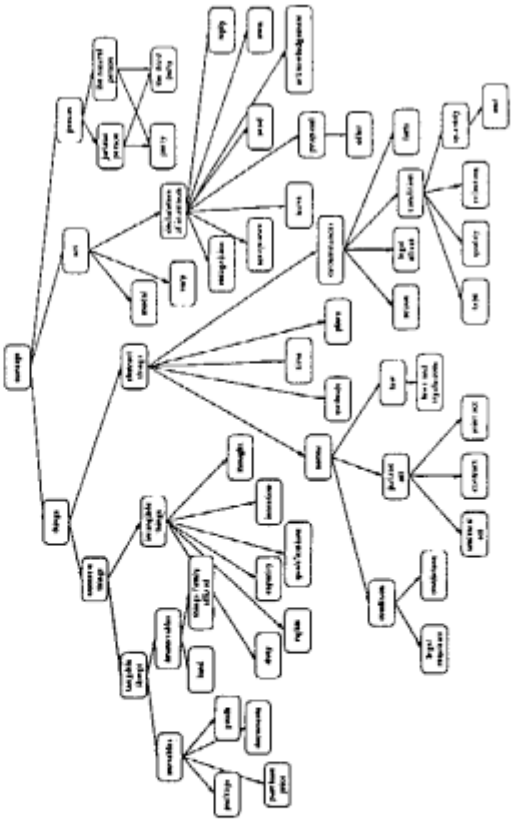
STEP3:

The system asks the user something is wrong with the legal ontology, applying knowledge acquisition strategies to the matched result.



Concept Relation Label in EDR

agent, a-object, object, cause, implement, material, source, goal, place, scene, manner, time, time-from, time-to, quantity, number, condition, cooccurrence, purpose, sequence, basis, and, or, modifier, possesor, beneficiary, from-to, unit



Concept Hierarchy Tree(Legal Dictionary before Refinement)

Concept definition for "declaration of intention" in Legal Dictionay before Refinement

goal/1-> a person or an agent that behaves like a person.

object/1-> intention

Matching a Legal Ontology with a General Ontology

1. Matching a legal term in a legal ontology with a word illustration in a general ontology

→ The matched word illustrations are lower boundaries in a matched space in a general ontology.

2. Matching a concept description in a legal ontology with one in a general ontology.

→ The matched concept descriptions are upper boundaries in a matched space in a general ontology.

Knowledge Acquisition (Interview) Strategies

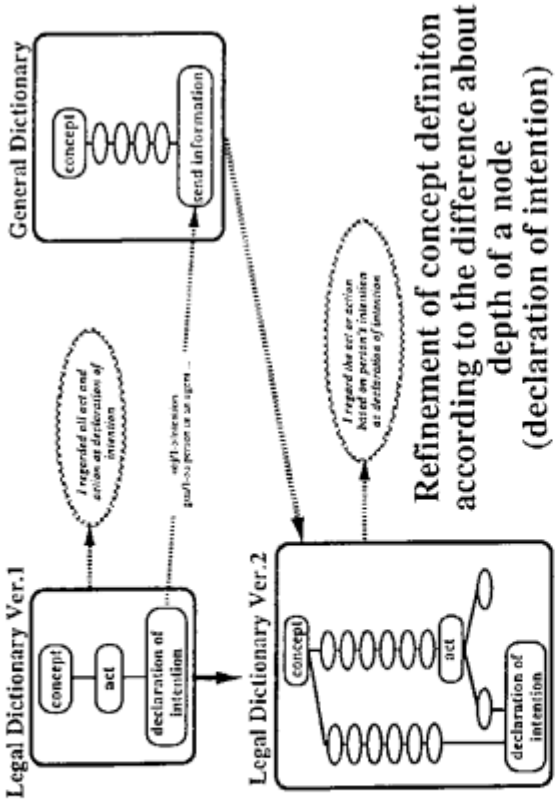
- ○ the difference between depths in two ontologies
- ○ the difference between the number of child nodes in two ontologies
- ○ the difference between ' paths between two concepts' in two ontologies

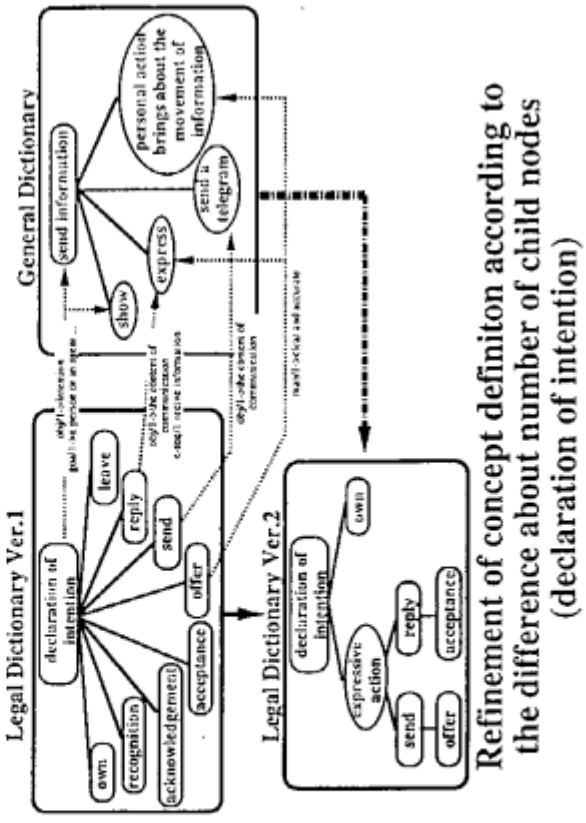
Depth of Legal Dictionary and General Dictionary

node name	Legal Dictionary	General Dictionary
person	1	1
things	1	5
purchase price	5	4
methods	3	2
time	3	2
place	3	2
norms	3	4
condition	4	11(5/6/7)
restriction	5	7
action	4	5
quantity	5	2
unit	6	7(4)
act	1	8
vary	2	4
declaration of intention	2	5
send	3	5
offer	3	8(6/7),9(8)
reply	3	8(7)
acceptance	3	8
acknowledgement	3	8(7)
own	3	5
leave	3	5
recognition	3	5

Number of Child Nodes of Legal Dictionary and General Dictionary

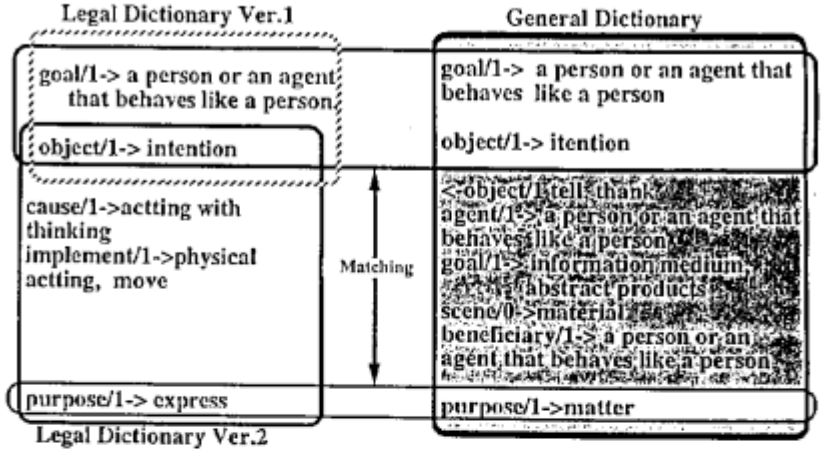
node name	Legal Dictionary	General Dictionary
person	2	2
things	2	2
purchase price	0	1
methods	0	1
time	0	4
place	0	2
norms	3	2
condition	2	0
restriction	0	0
action	0	0
quantity	1	0
unit	0	1
act	3	0
vary	0	4
declaration of intention	8	1
send	0	1
offer	0	1, 0
reply	0	1
acceptance	0	1
acknowl-edgement	0	1
own	0	1
leave	0	1
recognition	0	1





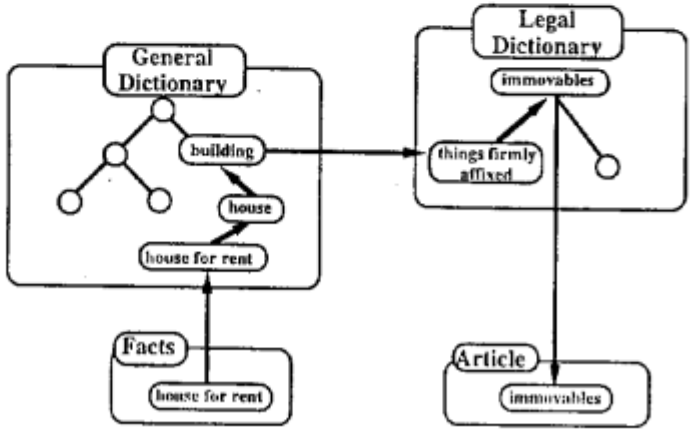
Concept Definition for "declaration of intention" in Legal Dictionay after Refinement

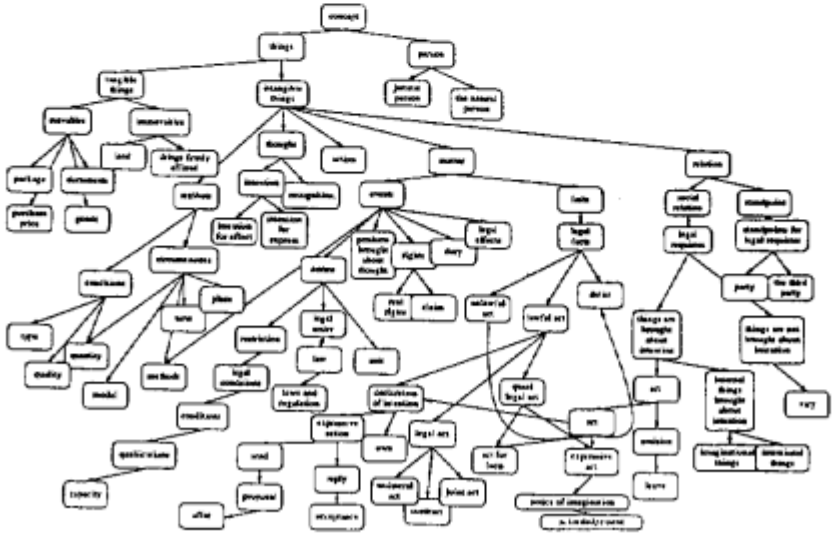
object/1->	intention
cause/1->	actting with thinking
implement/1->	physical actting, move
purpose/1->	express



Refinement of concept definiton according to the difference about concept description (declaration of intention)

Hybrid Dictionary for Legal Resoning





Concept Hierarchy Tree(Legal Dictionary after Refinement)

Balancing Expressiveness and Tractability: Expectation-Driven Problem Formulation

L. Karl Branting
Department of Computer Science
University of Wyoming
Laramie, Wyoming 82071-3682
karl@eolus.uwyo.edu

Abstract

This paper distinguishes two criteria for representational adequacy and identifies three tasks whose complexity is exacerbated by highly expressive representation formalisms: case matching; case retrieval; and problem formulation. This paper proposes an approach to reducing these costs, termed *expectation-driven problem formulation* (EDPF), which uses previous cases as models to guide and constrain the formulation of new cases. Under this approach, previous cases are represented as a sets of exemplars at various levels of abstraction. Each exemplar can be used as a model for a meaningful portion of a new case. By constraining the representation of new cases to conform to the representational conventions of previous cases, EDPF speeds case representation and reduces the danger of representational inconsistency. Moreover, the similarity between a new case and previous cases is determined through the process of formulating the new case, thus obviating separate indexing and matching stages.

1 Introduction

A fundamental issue in the design of any AI system is the tradeoff between representational adequacy and tractability. The most expressive representations, such as first-order predicate calculus, suffer from severe tractability problems (Cook, 1971). Conversely, the most tractable representations, such as vectors of numeric or symbol features, are severely limited in expressiveness. The most appropriate representation for a given problem is one that optimizes this tradeoff, maximizing the ability to represent the factors relevant to problem solving in the domain without sacrificing efficiency.

Two key criteria for representational adequacy can be distinguished. The first is *extensibility*, the ability to express arbitrary sequences of actions. The facts of a legal case constitute a narrative or story, and a fundamental characteristic of true stories is that their details can never be precisely anticipated. A case representation consisting of a template (or, equivalently, a frame, feature vector, or script) to be instantiated by the facts of a particular case is incapable of representing configurations of causal, temporal, or intentional relations that weren't anticipated in the design of the template. In view of the unpredictable variety of possible narratives than can give rise to legal consequences, it is unrealistic to suppose that a template capable of accommodating all such narratives can be devised for most areas of law. Accordingly, an adequate representation language must be extensible.

The second criterion of representational adequacy is *specificity*, the ability to represent case facts in sufficient detail to express all legally relevant aspects of the case. The most effective legal arguments are those that are grounded in the particular facts of the given cases. Such arguments can be produced only to the extent that the specific case facts on which they are based can be represented. Thus, a system whose representation language is limited in the aspects of a case that it can express will be able to generate only a correspondingly limited set of distinct explanations.

Relational representations, such as first-order predicate calculus, McCarty's Language for Legal Discourse (McCarty, 1989), and the semantic network representations of BRAMBLE (Bellairs, 1989) and GREBE (Branting, 1991a), have a high degree of extensibility and specificity. However, there are three distinct costs associated with such highly expressive case representation formalisms.

The first is the cost of case matching. The cost of determining the degree of match between two cases represented as feature vectors is proportional to the number of features in each case. Similarly, determining the "most-on-point-cases" under the dimensional approach requires only counting the number of shared dimensions. If cases are represented using a relational formalism, by contrast, determining the degree of match requires determining the size of the largest common subgraph. This problem is, in general, exponential in the size of the graphs.[1]

The second cost is in difficulty of retrieval. Retrieval is relatively straightforward when cases are represented as feature vectors because individual case features are meaningful indices to such cases. Techniques for accurate retrieval of cases represented as feature vectors have been explored by a number of researchers (Porter et al., 1990;

[1] K subgraph-subgraph isomorphism, the problem of determining whether two graphs share isomorphic subgraphs containing k or more edges, is $\mathcal{NP}$-complete (Garey and Johnson, 1979). However, the problem of determining the largest k for which there is a k subgraph-subgraph isomorphism would be in $\mathcal{NP}$ only if the complement of k subgraph-subgraph isomorphism were in $\mathcal{NP}$, which it appears not to be. Thus, determining the largest common subgraph of two graphs is exponential in the size of the graphs.

Kolodner, 1984; Ashley, 1990). By contrast, no general retrieval mechanism for cases represented using a relational language has been identified that is guaranteed both to be accurate and significantly faster than exhaustive matching (Branting, 1992b).[2]

The third, and probably most important, cost associated with highly expressive representations is the cost of *problem formulation*, the expression of a problem in a form amenable to manipulation by a computer. In systems that use featural representations of cases, problem formulation is typically quite straightforward, requiring only assignment of values to each case slot. Problem formulation in a relational representation, by contrast, is typically complex, time-consuming, and prone to error and inconsistency. For example, representing even a simple story in first-order predicate calculus is a challenging undertaking even for an experienced logician.[3]

Inconsistencies can arise in two different ways in the formulation of relational problem representations. First, the very expressiveness of relational representations makes multiple representations for a single given proposition possible (*e.g.*, "Mary drove John to work" can be represented as a `driving` action with Mary as the `driver` and John as the `passenger`, or as a `transportation` act with John and Mary as `agents` and a car as the `instrumentality`). Second, a single event can be described at various levels of abstraction (*e.g.*, an action an be described as a statement, a telegraph message, or a written contractual offer).

These difficulties are illustrated by the author's experience with GREBE, which used a semantic network representation for cases. Representations of GREBE's test cases consisted on average of 89 tuples, each of which had to be entered by hand. Entering cases of this size was a lengthy process—typically several orders of magnitude longer than GREBE's run-time—and the resulting representation often required considerable debugging. Moreover, different knowledge enterers often chose to represent identical facts differently, creating the danger of inconsistent analyses of equivalent cases. Limitations in problem formulation, rather than problem solving, prevented GREBE from being usable in any practical setting (Branting, 1991a).

Devising a suitable representation for legal applications therefore requires charting a course between the Scylla of representational inadequacy and the Charybdis of intractability. The approach proposed by this paper is based on the observation that meaningful cases are not arbitrary collections of facts, but share a high degree of semantic and structural similarity to one another. As a result, previous cases can be used as models to guide and constrain the formulation of new cases. This process, termed *expectation-driven problem formulation* (EDPF), can significantly speed the formulation of new cases and significantly reduce the danger of inconsistent repre-

[2]However, retrieval techniques satisfying these criteria have been devised for specialized types of graphs (Cook, 1989).

[3]See (Davis, 1990) for an extensive exploration of these complexities.

sentation of equivalent facts. Moreover, representing new cases in terms of previous cases has the benefit that the degree of similarity between the new and old cases can be determined as a side effect of formulating the new case.

The next section describes how cases can be represented in a manner that permits portions of multiple precedents to be used in the representation of new cases. Section three then sets forth an algorithm for expectation-driven problem formulation and sketches an example of how the algorithm could be used in the domain of contracts.

2 The Constituent Structure of Cases

The primary impediment to using past cases as models for the formulation of new cases is that a new case may resemble aspects of multiple past cases. The solution to this problem rests on the observation that useful knowledge seldom consists of isolated facts, but instead tends to consist of collections of related facts. A simple example is a frame. The object/slot/value triples constituting a frame can be viewed as a collection of propositions that are related because they all concern the same object.

In general, it is natural to manipulate a collection of facts as a unit if it can be viewed as an instance of some more abstract concept. There is abundant psychological evidence that most human concepts are characterized by have multiple exemplars of varying prototypicality (Murphy and Medin, 1985; Lakoff, 1987). A concept at one level of abstraction may therefore have multiple instances, each of which can be described by a set of relations applying to concepts at a lower level of abstraction. See Figure 1. A set of facts (*i.e.*, relations applied to objects) constituting an exemplar of a concept is termed the *elaboration* of the exemplar. Conversely, a concept of which a given set of facts is an elaboration is a *composition* of the facts. A precedent case can therefore be represented as sets of domain relations among objects at the same level of abstraction and sets of elaboration relations among objects at different levels of abstraction. The benefit of this representation from the standpoint of problem formulation is that each exemplar of a concept occurring in a precedent can provide a model for representing an instance of the concept in a new case.

If precedents are represented as sets of exemplars, then when a user adds a concept to a new case under construction the system can fetch the elaboration of the most prototypical exemplar of the concept and present it to the user as a set of suggested additional facts. The user can accept, reject, or modify these suggested facts. The rejection or modification of suggested facts may make some other exemplar of the concept match the given facts better, so the system may change its suggestions based on the user's reaction. An added concept may itself occur in the elaboration of some more abstract concept. The most prototypical instance of the more abstract concept

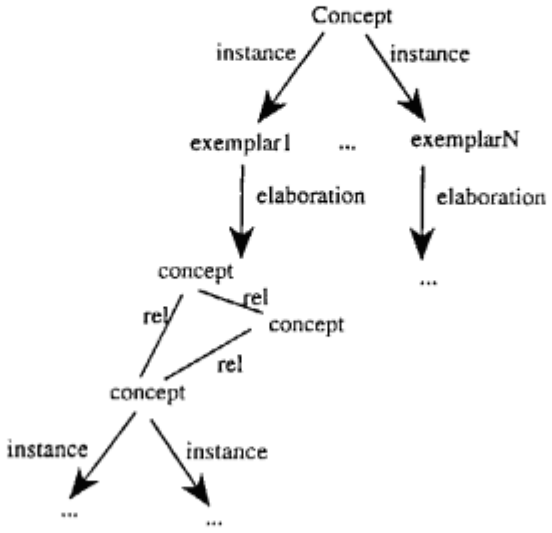


Figure 1: A hierarchy in which concepts have multiple exemplars, each consisting of a set of relations among concepts at a lower level of abstraction.

that best matches the case under construction can be fetched and used to suggest possible additional facts at the same level of abstraction as the fact just added.

Thus, viewing precedent cases as collections of exemplars permits the construction of new cases to be guided and constrained by the structure of the precedents. When this process of new case representation is completed, the relation between the new case and the precedents that it most closely matches is known, perforce, as a side-effect of the process of case formulation. The next section sets forth an algorithm for EDPF and illustrates the algorithm with a simplified example from the domain of contracts.

3 The Process of Expectation-Driven Problem Formulation

The basic requirement for expectation-driven problem formulation is that precedent cases should be parsed into their constituent structure, *i.e.*, into exemplars, when added to the system.

Given a set of precedents, each represented as a set of exemplars, the algorithm for EDPF can be described as follows:

Expectation-Driven Problem Formulation

CURRENT-CASE ← {}
UNTIL all relevant facts have been represented DO

- The user selects an object from a menu and adds it to the current case (or specifies a new object). Objects are presented in order of their prototypicality. If CURRENT-CASE ≠ {}, the new object should be the slot value of, or have as its slot value, some object in CURRENT-CASE.
- (Downward remindings) IF the object just added represents a concept with exemplars, THEN DO
  - Find the most prototypical exemplar of the concept consistent with the rest of CURRENT-CASE and present the slot values of the object to the user as "defaults", *i.e.,* suggested additions to the facts.
  - Provide the user with the option of inspecting alternative instances of the concept to see if the defaults suggested by these alternatives better match the user's intentions.
  - Permit the user to modify or reject any default. If this causes the just-added object to match a different concept instance more closely, refine the match.
  - Permit the user to supply new names for the old case names occurring in default objects, propagating name substitutions throughout the default facts in which they appear.
- (Upward remindings) IF the just-added object O occurs in the elaboration of instances of more abstract objects {O'}, THEN DO
  - Determine the object $O'_i \in \{O'\}$ whose elaboration best matches CURRENT-CASE. If multiple instances match the current case equally, choose the most prototypical, providing the user with the option of inspecting alternatives. IF O'_i is not already a part of CURRENT-CASE, add it, replacing any inconsistent objects.
  - Present any objects in the elaboration of O'_i that are not already present in CURRENT-CASE as defaults.
  - Permit the user to modify or reject any default. If this causes current case to match some other composition better than O'_i, refine the match.
  - When the user accepts a default object, follow the same steps as when a object is added, *i.e.,* present slot values as defaults, and prompt user to accept, reject or modify them.

Techniques for identifying the elaboration that best matches a given set of facts, making a consistent substitution of new case objects for old case objects, presenting the substituted facts as defaults, permitting the user to cycle through alternative elaborations, and refining a match based on a user's rejection or modification of default facts are set forth in (Branting, 1992a).

The EDPF algorithm can be illustrated with a highly simplified example from the contracts domain. Suppose that a system contains three precedents whose legal consequences can be summarized as follows:

- **P1.** There was an offer, an acceptance, performance by the offeror, and non-performance by the offeree. It was held that there was a contract, and offeree was in breach of it.
- **P2.** There was an offer, an acceptance under substantially modified terms, and performance by the original offeree. The acceptance under substantially modified terms was held to be a counteroffer, which was accepted by the offeree's performance.
- **P3.** There was an offer, a withdrawal of the offer, and a purported acceptance of the offer. The acceptance was held to be ineffective.

A fragment of a representation of these cases is set forth in Figure 2. While innumerable different frame-based or first-order predicate calculus representations of these cases are possible, the representation shown in Figure 2 assumes that the *basic level* (Smith and Medin, 1981) for description of contracts cases consists of illocutionary actions, such as proposing, agreeing, and refusing, and physical actions, such as transferring money or goods and performing services. Instances of contractual concepts, such as offering or accepting, are represented in terms of these basic level concepts. The basic level concepts themselves have instances with elaborations, *e.g.*, there are many different ways to communicate a proposal. Thus, basic level concepts are at an intermediate level of abstraction.

In Figure 2, `contract` and `no-contract` are instances of the concept `contract-case`. `Contract` has P1 and P2 as instances, and `no-contract` has P3 as an instance. The elaboration of P1 is a series of legal actions: an `offer`, an `acceptance`, a `performance`, and a `breach`. The elaboration of `offer1` consists of a statement by Buyer to Seller whose illocutionary force is a proposal, whose content is a transaction in which Buyer transfers money to Seller and in exchange Seller transfers goods to Buyer, and whose instrumentality is a telephone.

Suppose that a user of the system desires to enter a new case consisting of an offer, a counteroffer, a withdrawal of the counteroffer, and a purported acceptance of the counteroffer by the original offeror. Suppose that the user begins by entering the concept `telephone`, intending to enter the fact that the offeror telephoned a

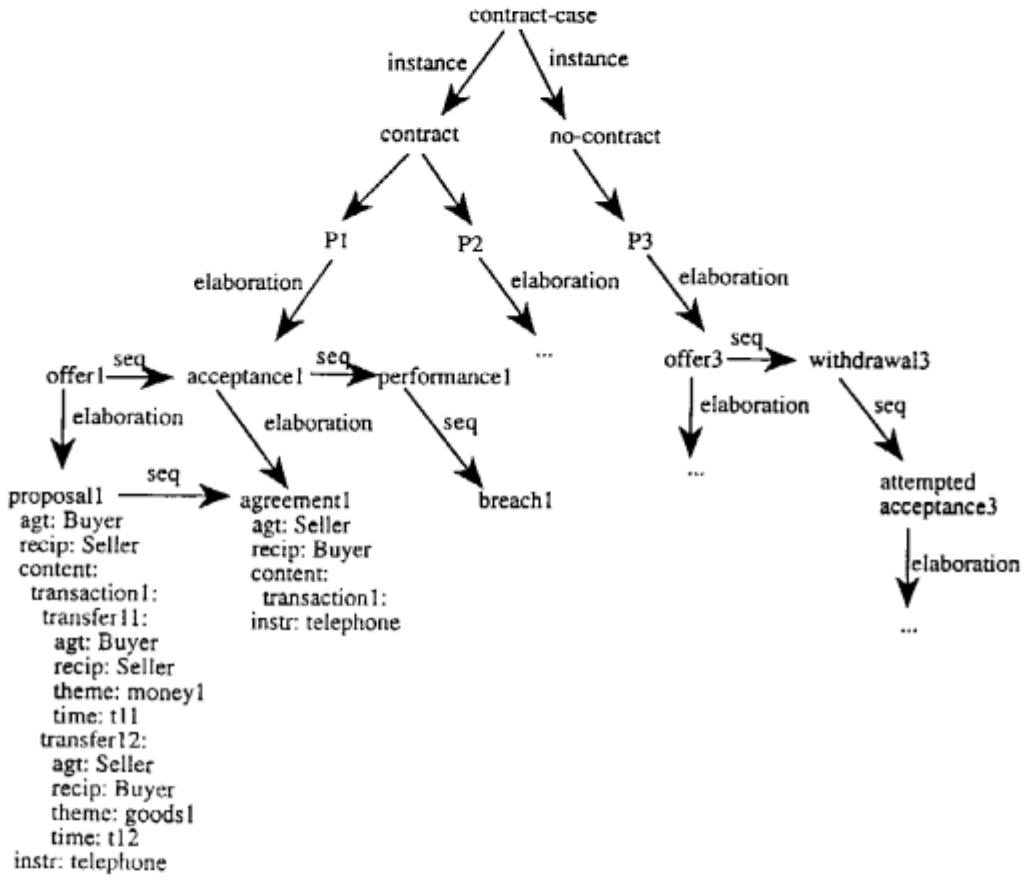


Figure 2: A fragment of a knowledge base containing three contracts precedents.

proposal to the offeree. Describing a case in terms of primitives such as `telephone` creates the danger of inconsistent representation, because `telephone` is at a lower level of abstraction than the illocutionary acts in the basic level of the existing case library. However, in the portion of the knowledge base shown in Figure 2, telephone occurs only in the instrumentality slot of an instance of the concept `proposal`. The sole reminding is therefore to this instance, `proposal1`. The system therefore suggests the most prototypical instance of `proposal` consistent with CURRENT-CASE, `proposal1`, as a default.

If the user accepts the default proposal object with the instrumentality slot filled by `telephone`, the remaining slots in `proposal1` are then presented to the user as potential additional facts. If the user supplies a specific name (*e.g.*, "John") to

replace "Buyer" as the value of the agt slot in the proposal action. the name will be propagated through the remaining slots, replacing "Buyer" as the agt in the first transfer and recipient in the second transfer in the proposed transaction.

proposal1 itself constitutes the elaboration of an instance of a more abstract concept in case P1: offer. The system therefore finds the most prototypical instance of offer, offer1, and presents it as a default. offer1 is, in turn, part of the elaboration of P1, so the system suggests the next object in the elaboration of P1, acceptance1, and the elaboration of acceptance1, agreement1, as defaults. Figure 3, which shows this partial representation of the new case, follows the convention that objects occurring only once in any case are numbered by the case in which they first appear, *e.g.,* offer1 is the instance of offer that occurs in precedent P1. Objects that occur more than once in a case are numbered the the case in which they first appear and the instance of the object within the case, *e.g.* transfer12 is the second instance of transfer that occurs in case 1.

Let us suppose that, as will generally be the case, the user has no idea whether the requirements for offer or acceptance are satisfied and therefore does not know whether to accept or reject these defaults. However, the user notes that, unlike agreement1, the transaction in the agreement action of the new case is not identical to the transaction in the proposal action. The user therefore inspects alternative instances of agreement until the user finds one in which the content is the proposed transaction together with some modifications. When the user accepts this instance of agreement, agreement2, the current case now matches a different instance of offer, offer2, in which a proposal was followed by an agreement to a modification of the proposed transaction. The proposal and agreement actions of the new case are now matched to offer2. At the level of contractual actions, the new case now consists of a single offer. This offer still matches P1, so an acceptance action is again proposed as a default, and the elaboration of the most prototypical acceptance, an agree action, is against suggested by the system as a default. However, the new case that the user wishes to describe consists of an agreement to a modification of the original offer followed by a retraction of the agreement (*i.e.,* in legal effect, a withdrawal of the counteroffer). The user therefore inspects alternative instances of contract-case until the user finds one, P3, in which the action following the offer, a withdrawal, has a retraction action as its elaboration. The user then accepts the retraction action as a default. A portion of the representation of the new case is shown in Figure 4.

To summarize this process, the system's reminding process steers the user towards the basic level of abstraction at which the current knowledge base is represented—the level of illocutionary and physical actions—and away from the lower level of abstraction that the user initially attempts. P1 provides a proposal action, proposal1, that serves as a model for the representation of the proposal in the new case. At a

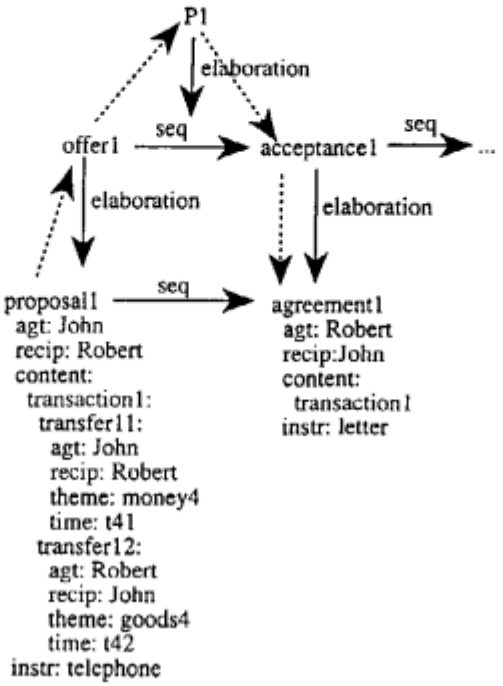


Figure 3: Accepting `proposal1` has triggered a series of remindings, shown as dotted lines, through `offer1` to precedent P1 and down through `acceptance1` to `agreement1`.

higher level of abstraction, P2 provides a model of an `offer`, `offer2`, that consists of a `proposal` followed by an `agreement` to a modified transaction. Finally, P3 provides a model of an `offer` followed by a `withdrawal`. The new case is therefore represented using components of all three precedents. When this process is completed, no further indexing or matching is necessary, since the similarity between the new case and the precedents is known as a consequence of the process of representation.

4 Conclusion

Expectation-driven problem formulation represents an approach to balancing expressiveness with tractability under which the representation of new cases is biased towards the conventions used in representing precedents. Neither extensibility nor specificity are compromised under this approach because the user can, optionally,

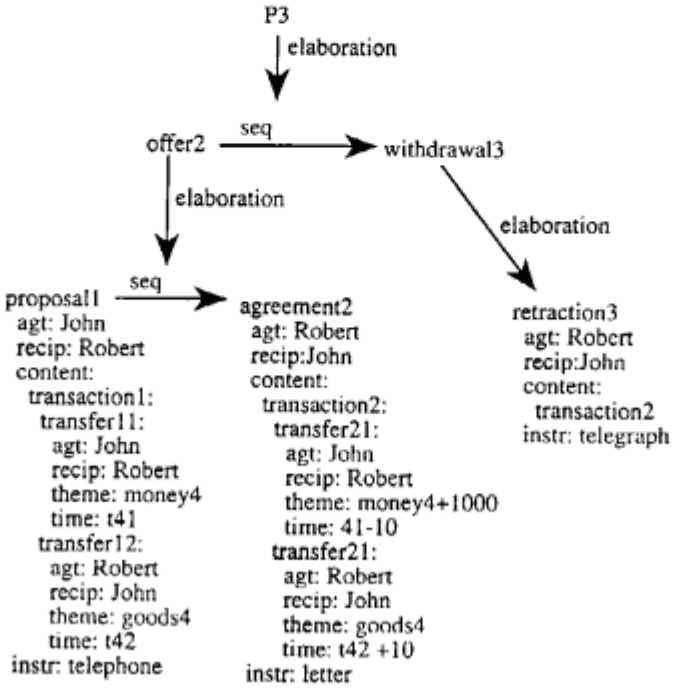


Figure 4: The new case is now represented in terms of exemplars drawn from three different precedents.

introduce entirely new terms or structures. Indeed, extensibility is facilitated because existing structures can be arbitrarily nested or concatenated. However, EDPF makes it far easier to reuse old structures than to create new ones, and the ability to generate remindings at multiple levels of abstraction insures that the user will be made aware of past uses of structures that resemble meaningful portions of the case under construction.

There are several difficult issues in fully implementing EDPF. First, the effectiveness of EDPF rests on hypothesis that legal cases are decomposable into non-interacting exemplars. While this hypothesis is consistent with the results of a number of research projects that have explored decomposition of cases into constituent structures, *e.g.*, (Branting, 1991b; Veloso, 1992; Redmond, 1990; Sycara and Navinchandra, 1991), it has not been rigorously established in the domain of law. Second, the effectiveness of EDPF depends on the quality of the computer-human interface. Technically naive users are likely to find the process of graphically manipulating objects representing actions and events quite unfamiliar. The choice of an appropriate

basic level vocabulary is therefore extremely important to the effectiveness of an EDPF system. Presenting the natural language equivalent of each fact in a separate window, and permitting at least some manipulations (such as accepting or rejecting facts) to be performed on the natural language equivalents may make an EDPF system much more comprehensible.

The task of problem formulation is of critical importance if legal systems are to provide useful advice about complex problems. EDPF represents an effort to bring the full knowledge of a an automated legal reasoning system to bear on this critical first step in legal problem solving.

References

Ashley, K. (1990). *Modelling Legal Argument: Reasoning with Cases and Hypotheticals.* MIT Press, Cambridge, Massachusetts.

Bellairs, K. (1989). *Contextual Relevance in Analogical Reasoning: A Model of Legal Argument.* PhD thesis, University of Minnesota.

Branting, L. K. (1991a). *Integrating Rules and Precedents for Classification and Explanation: Automating Legal Analysis.* PhD thesis, University of Texas at Austin.

Branting, L. K. (1991b). Reasoning with portions of precedents. In *Proceedings of the Third International Conference on Artificial Intelligence and Law*, pages 145–154, Oxford, England.

Branting, L. K. (1992a). A case-based approach to problem formulation. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, Indiana.

Branting, L. K. (1992b). A model of the role of expertise in analog retrieval. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, Indiana.

Cook, D. (1989). ANAGRAM: An analogical planning system. Technical report, Department of Computer Science, University of Illinois.

Cook, S. A. (1971). The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium onf the Theory of Computing*, pages 151–158.

Davis, E. (1990). *Representations of Commonsense Knowledge.* Morgan Kaufmann Publishers, Inc., Palo Alto.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability.* Freman, New York.

Kolodner, J. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: a Computer Model.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things.* University of Chicago Press, Chicago and London.

McCarty, L. T. (1989). A language for legal discourse 1. Basic features. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, Vancouver, B.C.

Murphy, G. L. and Medin, D. L. (1985). The role of theories in conceptual coherence. *Psychological Review*, 92:289–316,299.

Porter, B. W., Bareiss, E. R., and Holte, R. C. (1990). Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(1–2).

Redmond, M. (1990). Distributed cases for case-based reasoning; facilitating use of multiple cases. In *Proceedings of AAAI-90*, Boston. American Association for Artificial Intelligence.

Smith, E. E. and Medin, D. L. (1981). *Categories and Concepts.* Harvard University Press.

Sycara, K. and Navinchandra, D. (1991). Influences: a thematic abstraction for creative use of multiple cases. In *Proceedings of the Third DARPA Case-Based Reasoning Workshop*, pages 133–144. Morgan Kaufmann.

Veloso, M. (1992). *Learning by Analogical Reasoning in General Problem Solving.* PhD thesis, Carnegie Mellon University.

A Legal Reasoning System based on Situation Theory

Satoshi Tojo
Mitsubishi Research Inst., Inc.
tojo@mri.co.jp

Stephen Wong
UCSF
swong@lri.library.ucsf.edu

Abstract

Legal reasoning systems research is a new field attracting both AI researchers and legal practitioners. The purpose of this paper is to introduce a formal model of legal reasoning, based on situation theory. On that abstract model, we show an example of reasoning system implemented in a knowledge-base management language *QUIXOTE*, regarding the language as a situated inference system.

1 Introduction

Legal reasoning systems research is a new field which has attracted researchers from both the legal and AI domains. Most legal reasoning systems draw arguments by interpreting judicial precedents (old cases) or statutes (legal rules), while more sophisticated systems include both kinds of knowledge. Surveys of the leading projects can be found in [4, 5].

Thus far, those legal reasoning systems seem to have had weak foundation in formalization, and they have been *ad hoc* combination of various forms of logical inference. Our prime purpose of this paper is to give a sound foundation to legal reasoning system in terms of situation theory [1]. And, in addition, we implement this model into a computational form in Knowledge Base Management System (KBMS) *QUIXOTE* [7]. Regarding the concept of *module* of *QUIXOTE* as situation, we show that the language can work as a situated inference system. The set of knowledge bases includes a dictionary of legal ontologies, a database of old cases, and a database of statutes.

The organization of this paper is as follows. Section 2 describes the formulation of legal knowledge at the abstraction level using the theory of situations. Section 3 illustrates the realization of this formulation at the KBMS level using *QUIXOTE*, and its situated inference mechanisms. The last section concludes this paper.

2 Situation Theory for Legal Reasoning

This section introduces a formal model for legal reasoning, especially, penal code, at the abstraction level. The formulation is based on *situation theory*, so we call it a situation-theoretic model ($\mathcal{SM}$).

2.1 General Terms

The ontologies of $\mathcal{SM}$ include objects, parameters, relations, infons, and situations. An object designates an individuated part of the real world: a constant or an individual in the sense of classical logic. A parameter refers to an arbitrary object of a given type. An n-placed relation is a property of an n-tuple of argument roles, $r_1, \cdots, r_n$, or slots into which appropriate objects of a certain type can be anchored or substituted. For example, we can define 'eat' as a four-place relation of *Action* type as:

$$< \text{eat:}Action \mid eater{:}ANIMAL,\ thing\text{-}eaten{:}EDIBLE\text{-}THING,\ location{:}LOC,\ time{:}TIM >$$

where *eater, thing-eaten, location,* and *time* are roles and the associated types, *ANIMAL* denotes the type of all animals, *EDIBLE-THING* denotes the type of all edible substances, and *LOC* and *TIM* are types of spatial and temporal location.

An infon σ is written as $\ll Rel, a_1, ..., a_n, i \gg$, where *Rel* is a relation, each argument term a_k is a constant object or a parameter, and i is a polarity indicating 1 or 0 (true or false). If an infon contains an n-place relation and m argument terms such that $m < n$, we say that the infon is *unsaturated*; if m = n, it is *saturated*. Any object assigned to fill an argument role of the relation of that infon must be of the appropriate type or must be a parameter that can only anchor to objects of that type. An infon that has no free parameters is called a *parameter-free* infon; otherwise, it is a *parametric* infon. If σ is an infon and f is an *anchor* for some or all of the parameters that occur freely in σ, we denote, by $\sigma[f]$, the infon that results by replacing each v in the domain of f that occurs freely in σ by its value (object constant) $f(v)$. If I is a set of parametric infons and f is an anchor for some or all of the parameters that occur freely in I, then $I[f] = \{\sigma[f] \mid \sigma \in I\}$.

$\mathcal{SM}$ is a triplet $\langle \mathcal{P}, \mathcal{C}, \models \rangle$, where $\mathcal{P}$ is a collection of abstract situations including judicial precedents, a new case, c_n, and a world, w, that is a unique maximal situation of which every other situation is a part. $\mathcal{C}$ is a concept lattice in which objects are introduced and combined with the *subsumption* relation ('=<'), that is an *is-a* relation intuitively, each other. 'An object of a type' is interpreted as 'an object is subsumed by another object corresponds to that type'. '$\models$' is the support relation, and our interpretation is:

Definition 2.1 (Supports Relation) *For any $s \in \mathcal{P}$, and any atomic infon σ, $s \models \sigma$ if and only if (iff) $\sigma \in s$.* □

2.2 Situated Inference Rules

Reasoning in law is a rule-based decision-making endeavor. A legal reasoning process can be modeled as an inference tree of four layers. The bottom layer consists of a set of basic facts and hypotheses, the second layer involves case rules of individual precedents, the third layer involves case rules which are induced from several precedents or which are generated from certain legal theories, and the top layer concerns legal rules from statutes. An individual or local case rule is used by an agent in an old case to derive plausible legal concepts and propositions. These rules vary from case to case, and their interpretation depends on the particular views and situational perspectives of the agents. An induced case rule has a broader scope and is generalized ¿from a set of precedents. Legal rules are general provisions and definitions of crimes. They are supposed to be universally valid in the country where they are imposed, and neutral. That is, the applicability of these rules is independent from the view of either side (plaintiff or defendant) and every item of information (infon) included is of equal relevance. Though it rarely happens, it may be possible for an agent to skip one or two case rule layers in attaining a legal goal.

In such a rule-oriented legal domain, *situated inference* has the following general form:

Rule 1 (General Rule) $s_0 \models \sigma_0 \Leftarrow s_1 \models \sigma_1, s_2 \models \sigma_2, ..., s_n \models \sigma_n / B$,
where $\sigma_0, \sigma_1, ..., \sigma_n$ *are infons, and* $s_0, s_1, ..., s_n$ *are situations.* □

This rule can be read as: "if s_1 supports σ_1, s_2 supports σ_2, and so on, then we can infer that s_0 supports σ_0 under the background conditions or constraints B." $s_0 \models \sigma_0$ is called the head of the rule while the remainder is called the body of the rule. The background conditions, B, are required to be coherent and satisfied before execution of the rule. Note that $c \models I/B$ implies that $c \cup B \models I$, where $c \models I$ as a shorthand for $c \models \sigma_1, c \models \sigma_2, ..., c \models \sigma_n$.

We are particularly interested in three rule instances: local case rules, induced case rules, and legal rules. A local rule is as follows:

Rule 2 (Local Rule) *For* $c \in \mathcal{P}$, $cr: c \models \sigma \Leftarrow c \models I/B_{cr}$. □

where I is called the antecedent of the rule, σ is the consequent, and cr is the label of the rule, which is not part of the rule but which serves to identify the rule. Sometimes, we simply write cr: $c \models \sigma \Leftarrow I/B_{cr}$. Both σ and I are parameter-free. One unique feature of rules in the legal domain is that the consequent is not disjunctive and often a single predicate. The reliability and the scope of application of a local rule will be subject to a set of *background conditions*, B_{cr}. The conditions include information such as an agent's goal and hypotheses; these are crucial in debate to establish the degree of certainty and the scope of applicability of that rule. Usually, it becomes necessary to take background conditions into account and investigate what they are. Many case rules exist in one case and often yield incompatible

conclusions. But, the background conditions clarify their hypotheses and perspective. When there is no danger of conclusion, we can write such a rule without stating its background conditions.

Another form of case rule is generalized or induced from several precedents. Owing to its generic nature, an induced case rule is represented as a constraint between two parametric infons, rather than parameter-free ones. Denote I' and σ' as a set of parametric infons and a parametric infon, respectively, such that all parameters that occur in the latter also appear in the former. An induced rule is written as:

Rule 3 (Induced Rule) *For any* $c_1, ..., c_k \in \mathcal{P}$, $c = c_1 \cup c_2, \cup ... \cup c_k$, $ir: c \models \sigma' \Leftarrow I'/B_{ir}$. □

where c is coherent and ir is the rule label. Similarly, a legal rule is:

Rule 4 (Legal Rule) $lr: w \models \sigma' \Leftarrow I'/B_{lr}$. □

where lr is the rule label and B_{lr} states the background legal theory, such as the aim of punishment or the aim of crime prevention, but not both. Such information is crucial in interpreting the antecedent infons.

2.3 Substitution and Anchoring

When a situation of a new case, c_n, supports a similar antecedent of a local rule of c_o, one can draw a conclusion about the new case that is similar to the consequent of that rule.

Definition 2.2 (Local Rule Substitution) *For* $c_n, c_o \in \mathcal{P}$, $cr^s: c_n \models \sigma\theta$ *if* $cr: c_o \models \sigma \Leftarrow I/B_{cr}$ *and* $c_n \models I'/\{B_{cr}\theta \cup B_n\}$ *such that* $I' \simeq_s I$. □

where cr^s is the label of the new rule, B_n is the original background of I' of the new case, and the combined condition after the substitution, $B = B_{cr}\theta \cup B_n$, is coherent. The notation $\simeq_s$ denotes the matching relation between two situations. Section 3 discusses how such a matching is implemented in *Quixote*. The function θ forms a *link* that connects c_n with c_o. This function replaces all terms (objects and relations) in σ and B_{cr} that also occur in I with their matched counterparts in I'. Normally, the background conditions are not included.

To combine the conclusions supported by different situations, the background conditions of both conclusions must be compatible. That is why the background conditions of Rule 1 must be coherent. Rather than substitution, a consequent is derived from a legal rule or an induced rule via anchoring.

Definition 2.3 (Induced Rule Anchoring) *For* $c_n, c_1, ..., c_k \in \mathcal{P}$ *such that* $c = c_1 \cup c_2 \cup ... \cup c_k$, $ir^a: c_n \models \sigma[f]$ *if* $ir: c \models \sigma \Leftarrow I/B_{ir}$ *and* $c_n \models I[f]/\{B_{ir}[f] \cup B_n\}$. □

Definition 2.4 (Legal Rule Anchoring) *For* $c_n \in \mathcal{P}$, $lr^a: c_n \models \sigma[f]$ *if* $lr: w \models \sigma \Leftarrow I/B_{lr}$ *and* $c_n \models I[f]/\{B_{lr}[f] \cup B_n\}$. □

2.4 Matching of Infons and Situations

In order to compare the similarity of a new case with precedent cases, we formalize the infon matching and the situation matching. Suppose that a concept lattice is given, where the subsumption relation ('=<') is defined between concepts. $R(\sigma)$ is a function that extracts '*rel*' from an infon σ.

Definition 2.5 (Infon Matching) *For any two infons σ_1 and σ_2,*

1. *If there is a $R(\sigma_3)$ such that $R(\sigma_1) =< R(\sigma_3)$, and $R(\sigma_2) =< R(\sigma_3)$ in a given concept lattice, then σ_1 and σ_2 are interpreted as* weakly matched infons.
2. *If $R(\sigma_1) = R(\sigma_2)$, then then σ_1 and σ_2 are interpreted as* partially matched.
3. *If all the objects that constitute two infons are identical, then the infons are* exactly matched. □

We give concepts of situation matching below. Note that the concept of situation matching is independent of that of infon matching.

Definition 2.6 (Situation Matching) *For situations s_1 and s_2,*

1. *If, for every infon in s_1, there is an infon that can match it in s_2, and vice versa, then the two situations are interpreted as* exactly matched *situations.*
2. *For any σ_1 in s_1, there is an infon σ_2 in s_2 that can match σ_1, situation s_1 can be* partially matched *with situation s_2. (Note that this partially matching relation is one way; even though s_1 can be partially matched with s_2, s_2 may not be partially matched with s_1.)*
3. *For any σ_1 in s_1 whose* relevance *value is larger than a given threshold level, there is an infon σ_2 in s_2, that can be matched with σ_1, s_1 can be* partially matched *with s_2* w.r.t. relevance value. □

Among several matching definitions, we will adopt *weakly matching* for infons and *partially matching w.r.t. relevance value* for situations, in implementation of the following section, for practical reasons. Let us consider the following pair of descriptions:

$$
\begin{aligned}
s_{new} \models\ & \{\ll abandon, mary^{agent} \gg, \\
& \ll leave, mary^{agent}, june^{object} \gg\} \\
s_{old} \models\ & \{\ll abandon, jim^{agent}, tom^{object}, 3^{relevance} \gg, \\
& \ll leave, jim^{agent}, tom^{object}, 2^{relevance} \gg, \\
& \ll poor, jim^{agent}, 1^{relevance} \gg\}
\end{aligned}
$$

If the threshold value is 2, then s_{new} can be *partially matched* with s_{old} *w.r.t. the value 2.* On the other hand, if 1 is the threshold instead, s_{new} cannot be *partially matched* with s_{old} *w.r.t. the value 1.*

In fig. 1, the combination of facts, case rules, induced rules, and legal rules is depicted.

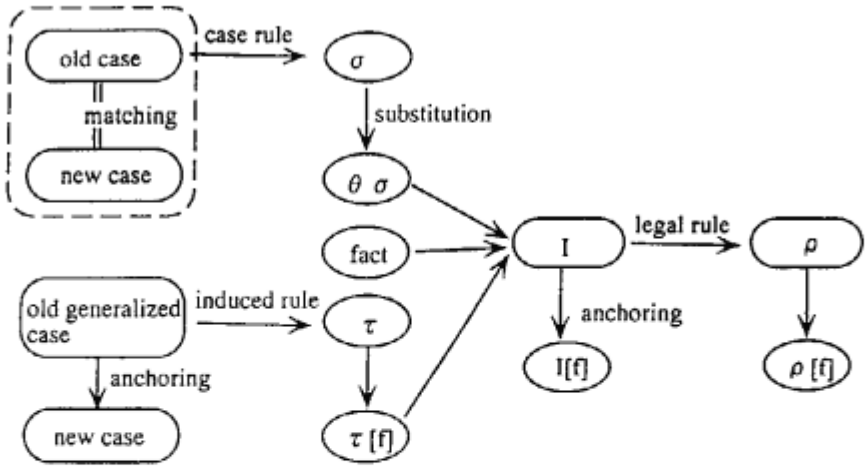


Figure 1: The combination of rules

3 Modeling of Legal Knowledge in *QUIXOTE*

This section introduces the language of *QUIXOTE* [7] and shows how this language can represent the *SM* concepts in computable form. A typical *QUIXOTE* database includes the following data structures: (i) the subsumption relations among basic objects, (ii) the submodule relations among modules, and (iii) rules. Our legal reasoning system consists of three databases: a legal dictionary, cases, and statutes. Accordingly, we first introduce the *objects* and *modules* of *QUIXOTE* and explain the data structure of the legal dictionary, then describe the use of *QUIXOTE rules* to represent case-based rules and statutes.

In *QUIXOTE*, the concepts of *SM* are rephrased as follows:

SM	*QUIXOTE*
situation	module
infon	attribute term
relation name	basic object
type	subsumption
role	label
supporting relation ($\models$)	membership in module (:)

3.1 Description of Case and Rule

A *QUIXOTE* rule has the following form (compare with Rule 1):

$$\overbrace{m_0 :: H}^{head} \mid \overbrace{HC}^{head\_constraints} \Leftarrow \overbrace{m_1 : B_1, \ldots, m_n : B_n}^{body} \parallel \overbrace{BC}^{body\_constraints} \;;;$$

where H or B_i are objects, and HC and BC are sets of formulas (constraints) using subsumption relations. Intuitively, this means that if every B_i holds in a module m_i under the constraints BC, then H and constraints HC hold in m_0. The head constraints and module identifiers can be omitted, and the body constraints, BC, of a rule then constitute the background conditions for that rule.

This study regards a case as being a situation, that is, a set of anchored sentences. Below, we describe a case which is a simplified description of an actual precedent [3].

Mary's Case

On a cold winter's day, Mary abandoned her son Tom on the street because she was very poor. Tom was just 4 months old. Jim found Tom crying on the street and started to drive Tom by car to the police station. However, Jim caused an accident on the way to the police station. Tom was injured. Jim thought that Tom had died in the accident and left Tom on the street. Tom froze to death.

This aforementioned case contains some human objects and several events with different relevancy. The order of values of the relevance attribute is represented by a subsumption relation, (l1 =< l2 =< l3).

```
mary_case :: {mary, tom, jim, accident, cold},
 poor/[agent=mary, relevance=l1],
 abandon/[agent=mary,
          coagent=tom/[mother=mary, age=4months], relevance=l2],
 find/[agent=jim, object=tom/[state=crying], relevance=l1],
 make/[agent=jim, object=accident, relevance=l2],
 injure/[agent=jim, coagent=tom, by=accident, relevance=l2],
 leave/[agent=jim, coagent=tom, relevance=l3],
 death/[agent=tom, cause=cold, relevance=l3]};;
```

The attorneys on both sides interpreted Mary's case according to individual perspectives: one is the responsibility of Mary's actions and the other is that of Jim's. For instance, one attorney reasoned that: "If Mary hadn't abandoned Tom, Tom wouldn't have died. In addition, the cause of Tom's death is not injury but freezing. Therefore there exists a causality between Tom's death and Mary's abandoning."

Another lawyer, however, argued differently: "A crime was committed by Jim, namely, his abandoning Tom. And in addition, Tom's death was indirectly caused by Jim's abandoning Tom. Therefore, there exists a causality between Tom's death and Jim's abandoning."

For a legal precedent, these contradictory claims are documented together with the final verdict from the judge overseeing that precedent. *Quixote* models these arguments with two *case rules* of different interpretations of causality.

```
cr1 :: responsible/[agent=mary,for=death]
        <=
       abandon/[agent=mary,coagent=tom],
       death/[agent=tom, cause=abandon/[agent=mary,coagent=tom]];;

cr2 :: responsible/[agent=jim,for=death/[agent=tom]]
        <=
       leave/[agent=jim, coagent=tom],
       death/[agent=tom, cause=leave];;
```

The idea of an *induced rule* is to abstract some of ground terms in local case rules. As an example, when there are several similar accident cases, the attorneys may make the following generalization:

```
ir1 :: responsible/[agent=X, to=Y, for=Inj]
        <=
       Acc/[agent=X],
       Inj/[agent=Y, cause=Acc]
       || {Acc =< accident, Inj=<physical_damage,
       X =< person, Y =< person};;
```

In `ir1`, traffic accident and injury are abstracted to variable `Acc` and `Inj` and subsumed by their super concepts in the legal dictionary.

Legal rules, or statutes, are formal sentences of codes. We provide a penal code in linguistic form (Japanese penal code, article 199): "In case an intentional action of person A causes the death of person B and the action is not presumed to be legal, A is responsible for the crime of homicide."

The *Quixote* representation of this code is:

```
lr1 :: responsible/[agent=A, to=B, for=homicide]
        <=
       Action/[agent=A],
       illegal/[act->Action],
       death/[agent=B, cause->Action],
       || {Action =< intend, A =< person, B =< person};;
```

In the description above, `illegal[agent=A, action = Action]` claims that the action `Action` done by `A`, such as self-defense, is not legal. The statute for the legality of self-defense is described as follows (Japanese penal code, article 38):

```
lr2 :: illegal/[act = Action]
        <=
       Action,
       || {Action =< intend};;
```

The concept of *anchoring* of $\mathcal{SM}$, mentioned in Section 2.3, is realized in *Quixote* by invoking appropriate rules within a case or statute description.

3.2 Query Processing

Let us consider Mary's case, where *Quixote* draws several conclusions by making different assumptions. In response to the query:

```
?-responsible/[agent=jim, to=tom, for=homicide].
```

that means "Is Jim responsible to Tom for the crime of homicide?", *Quixote* returns the following:

```
** 2 answers exist **
** Answer 1 **
 IF mary_case:death.cause =< leave THEN
 YES
** Answer 2 **
 IF mary_case:death.cause =< traffic_accident THEN
 YES
```

The first answer is one interpretation of the causality in Mary's case: if the cause of Tom's death is some event under Jim's leaving Tom, then Jim is responsible for the homicide. The latter answer says that Jim is responsible if Tom had been killed by Jim's traffic accident. It happens, however, that the latter does not hold, so that the inquiring agent starts a new query which adds information about the cause of Tom's death.

```
?-mary_case:responsible || {mary_case:death.cause==leave}.
```

In response to this second query, the *Quixote* system replies as follows.

```
** 1 answer exists **
** Answer 1 **
 IF mary_case:death.cause == leave THEN YES
```

Thus, we have shown the implementation of our situated inference model in *QUIXOTE*.

4 Conclusion

In this paper, we formalized legal reasoning in terms of $\mathcal{SM}$, where precedent cases and new accidents were regarded as situations, and various kinds of rules as situated inference rules. We also showed that the abstract model was implemented in *QUIXOTE* for prototyping. *QUIXOTE* could represent context-dependent knowledge and situated inference for knowledge base applications. The ability of *QUIXOTE* to model abstract concepts of situation theory in a database environment may pave the way for the knowledge-base (KB) community to tackle concrete, demanding problems, such as building a large scale KB for general linguistic concepts.

References

[1] J. Barwise, *The situation in logic*, CSLI Lecture Notes 17, Stanford, CA, 1988.

[2] K. Devlin *Logic and information*, Cambridge University Press, 1991.

[3] K. Nitta, Y. Ohtake, S. Maeda, M. Ono, H. Ohsaki, and K. Sakane, "HELIC-II: A legal reasoning system on the parallel inference machine," *Proc. Int. Conf. of Fifth Generation Computer Systems*, ICOT, Tokyo, June, 1992, pp. 1115-1124.

[4] E.L. Rissland, (ed.), Special issue: AI and Legal Reasoning, Part 1, *International Journal of Man-Machine Studies*, Vol. 34 No. 6, June 1991.

[5] E.L. Rissland, (ed.), Special issue: AI and Legal Reasoning, Part 2, *International Journal of Man-Machine Studies*, Vol. 35 No. 1, July 1991,

[6] S. Wong, "A situation-theoretic model for trial reasoning," *Proc. of the 6th Int. Symp. on legal knowledge and legal reasoning systems*, Tokyo, Oct., 1992, pp. 32-54.

[7] K. Yokota, H. Tsuda, Y. Morita, S. Tojo, H. Yasukawa, "Specific features of a deductive object-oriented database language *QUIXOTE*," *Proc. of the workshop on combining declarative and object-oriented databases*, ACM SIGMOD, Washington, D.C., May 29, 1993.

A Legal Reasoning System based on Situation Theory

Satoshi Tojo

Mitsubishi Research Inst., Inc.

(tojo@mri.co.jp)

Stephen Wong

University of California
at San Francisco
(swong@lri.library.ucsf.edu)

1 General Terms

- infon $\ll rel, a_1, a_2, \cdots, a_n; i \gg$.
  - i : polarity (positive/ negative)
  - parametric/ parameter-free
  - saturated/ unsaturated
- anchor f

  $\sigma[f]$: parameter-free

  for a set of infons I, $I[f] = \{\sigma[f] | \sigma \in I\}$
- situation

  '*abstract*' situation for state of affairs (*soa*)

  ex. case, accident, world, spatio-temporal location, knowledge and belief, modality, ..., and so on.
- supporting

  $s \models \sigma$ iff $s \ni \sigma$.
- coherent

  No two same infons, each with different polarity.
- compatible

  Two situations are coherent.

$\forall s, u \subset s, s \models (\exists \dot{x} \in u)\sigma$ iff $\exists f, s \models \sigma[f]$

$\forall s, u \subset s, s \models (\forall \dot{x} \in u)\sigma$ iff $\forall f, s \models \sigma[f]$

$s \models I$ iff $\forall \sigma \in I, s \models \sigma$.

2 Situated Inference Rules

2.1 General Rule

Rule 1 (Situated inference rule) *For infons:* $\sigma_0, \sigma_1, ..., \sigma_n$,
$s_0 \models \sigma_0 \Leftarrow s_1 \models \sigma_1, s_2 \models \sigma_2, ..., s_n \models \sigma_n / B$. □

2.2 Rules for Legal Reasoning

Suppose $\mathcal{P}$ as a set of cases,

Rule 2 (Local Rule) *For* $c \in \mathcal{P}$, $c \models \sigma \Leftarrow c \models I / B_{cr}$. □

Rule 3 (Induced Rule) *For any* $c_1, ..., c_k \in \mathcal{P}$, $c = c_1 \cup c_2, \cup ... \cup c_k$, $c \models \sigma' \Leftarrow I' / B_{ir}$.
□

Rule 4 (Legal Rule) $w \models \sigma' \Leftarrow I' / B_{lr}$. □

3 Substitution and Anchoring

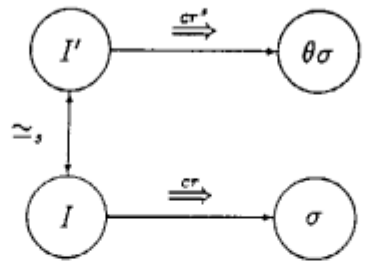


Figure 1: Substitution of Local Rules

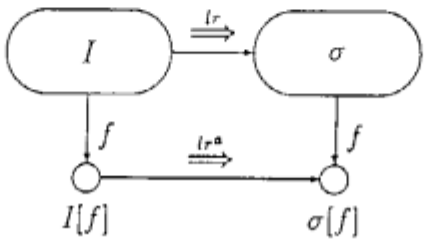


Figure 2: Anchoring for Legal Rules

4 Dictionary

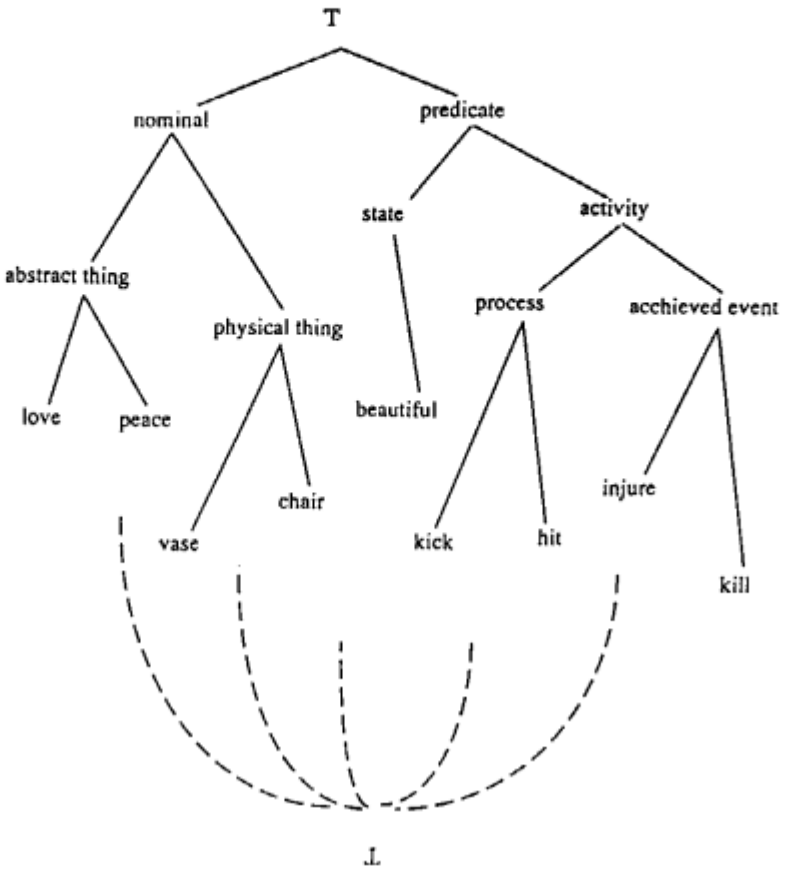


Figure 3: The Structure of Lexicon

5 Matching of Infons and Situations

5.1 Infon Matching

$\sigma_1 = \ll R_1, a_1, b_1, c_1, \cdots; i_1 \gg$
$\sigma_2 = \ll R_2, a_2, b_2, c_2, \cdots; i_2 \gg$

exact matching $R_1 = R_2, a_1 = a_2, \cdots, i_1 = i_2$.

partial matching $R_1 = R_2$.

weak matching R_1 and R_2 have a common superconcept.

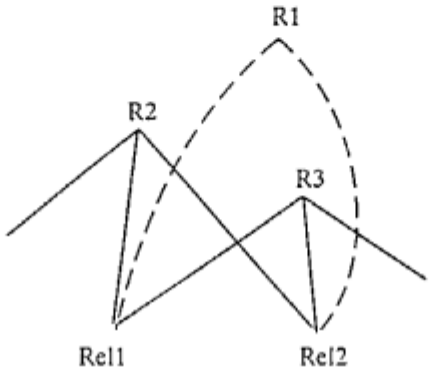


Figure 4: Common Superconcept in Lattice

5.2 Situation Matching

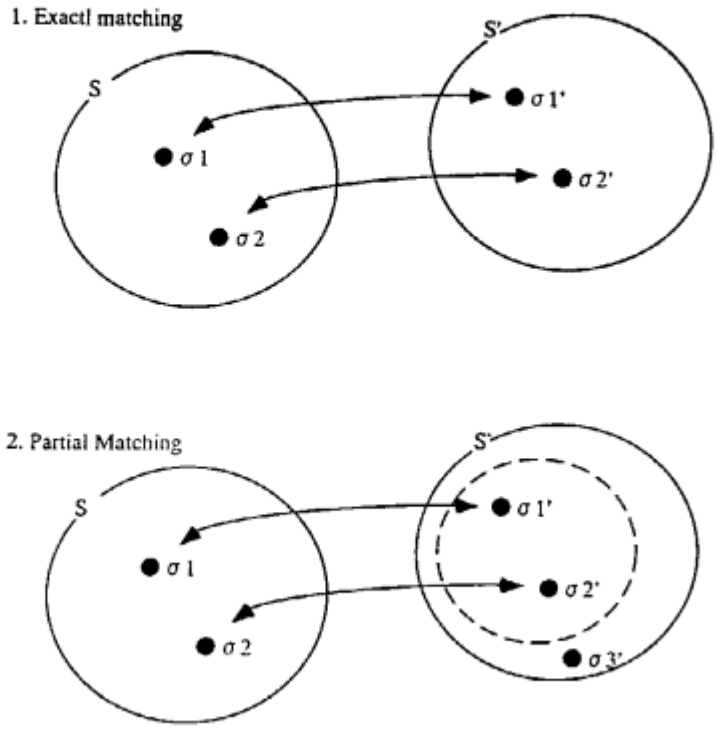


Figure 5: Situation Matching (1)

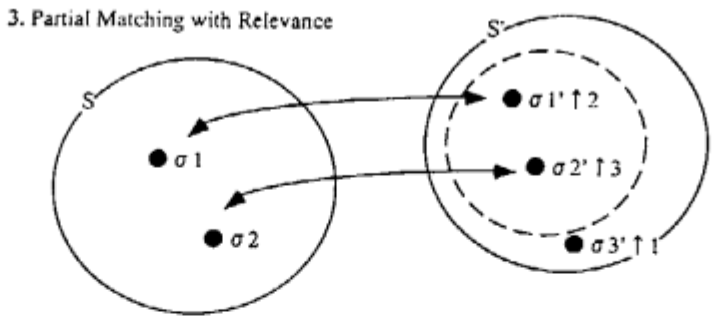


Figure 6: Situation Matching (2)

Example.

$$s_n \models \{\ll \text{abandon}, \text{mary}^{agent} \gg, \ll \text{leave}, \text{mary}^{agent}, \text{june}^{object} \gg\}$$

$$s_o \models \{\ll \text{abandon}, \text{jim}^{agent}, \text{tom}^{object}, 3^{relevance} \gg, \ll \text{leave}, \text{jim}^{agnet}, \text{tom}^{object}, 2^{relevance} \gg, \ll \text{poor}, \text{jim}^{agent}, 1^{relevance} \gg\}$$

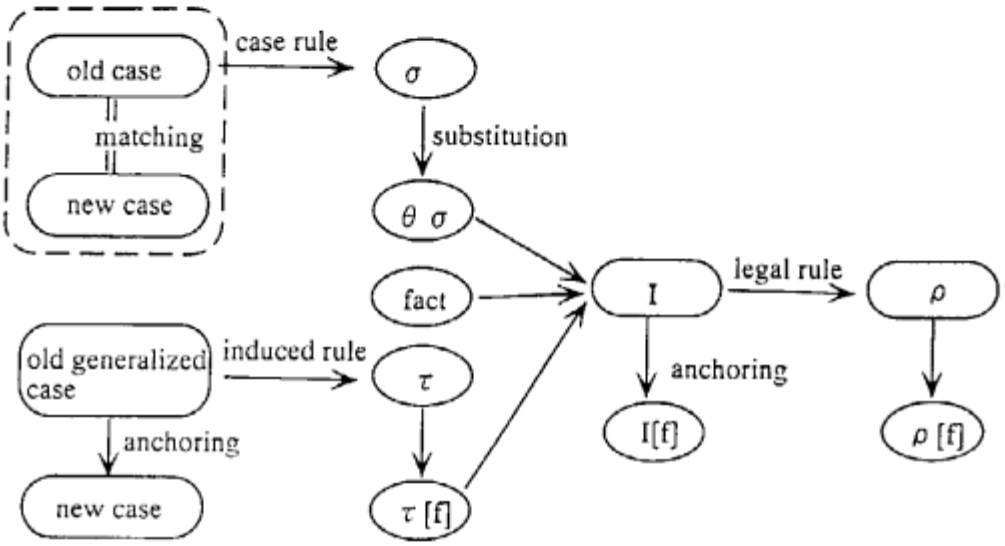


Figure 7: The combination of rules

6 Implementation in *QUIXOTE*

SM	*QUIXOTE*
situation	module
infon	attribute term
relation name	basic object
type	subsumption
role	label
supporting relation ($\models$)	membership in module (:)

```
mary_case :: {mary, tom, jim, accident, cold},
 poor/[agent=mary, relevance=11],
 abandon/[agent=mary,
   coagent=tom/[mother=mary, age=4months], relevance=12],
 find/[agent=jim, object=tom/[state=crying], relevance=11],
 make/[agent=jim, object=accident, relevance=12],
 injure/[agent=jim, coagent=tom, by=accident, relevance=12],
 leave/[agent=jim, coagent=tom, relevance=13],
 death/[agent=tom, cause=cold, relevance=13]};;

c >- cr1;;
cr1 :: responsible/[agent=mary,for=death]
 <=
abandon/[agent=mary,coagent=tom],
death/[agent=tom, cause=abandon/[agent=mary,coagent=tom]];;

c >- cr2;;
cr2 :: responsible/[agent=jim,for=death/[agent=tom]]
 <=
leave/[agent=jim, coagent=tom],
death/[agent=tom, cause=leave];;

ir1 ::responsible/[agent=X, to=Y, for=Inj]
<=
```

```
        Acc/[agent=X],
        Inj/[agent=Y, cause=Acc]
        || {Acc =< accident, Inj=<physical_damage,
        X =< person, Y =< person};;

 lr1 :: responsible[agent=A, to=B, for=homicide]
  <=
 Action/[agent=A],
 illegal/[act->Action],
 death/[agent=B, cause->Action],
 || {Action =< intend, A =< person, B =< person};;

 lr2 :: illegal[act->Action]
  <=
Action,
|| {Action =< intend};;

?-responsible/[agent=jim, to=tom, for=homicide].

** 2 answers exist **
** Answer 1 **
 IF mary_case:death.cause =< leave THEN
 YES
** Answer 2 **
 IF mary_case:death.cause =< traffic_accident THEN
 YES

?-mary_case:responsible || {mary_case:death.cause==leave}.

** 1 answer exists **
** Answer 1 **
 IF mary_case:death.cause == leave THEN YES
```

Representation of Legal Knowledge by Logic Flowchart and CPF

Hajime Yoshino*

Abstract

The essential points in developing any method to represent legal knowledge are that: (1) the method is easy for lawyers to understand and use, (2) it has the sufficient ability to express legal knowledge in detail, and (3) it is applicable to formalizing legal reasoning. For (1) I have suggested the legal knowledge representation by logic flowchart. For (2) and (3) I have offered the Compound Predicate Formulas (CPF) and developed it. In this paper I will explain these two methods, illustrating some examples, and also give the rigorous foundation of CPF based on logic i.e. the establishment of its syntax and semantics.

1 Introduction

In order to make up the legal knowledge base, one should abstract legal knowledge from literal sources such as legal articles, judicial precedents or textbook of laws, or from tacit knowledge of lawyers that is not expressed explicitly in the form of letters, and store that knowledge into data base on computers. Knowledge is, however, different from simple data in that knowledge is structured and formalized systematically so that computers can infer by making use of it. Therefore, how to formalize legal knowledge, in other words, how to represent legal knowledge, namely the way for legal knowledge representation, is the crucial problem for establishing legal knowledge base.

The essential points in developing any method to represent legal knowledge are that: (1) the method is easy for lawyers to understand and use, (2) it has the sufficient ability to express legal knowledge in detail, and (3) it is applicable to formalizing legal knowledge reasoning. For (1) I have suggested the legal knowledge representation by logic flowchart. For (2) and (3) I have offered the Compound Predicate Formulas (CPF) and developed it. In this paper I will explain these two methods, illustrating some examples, and also give the rigorous foundation of CPF based on logic i.e. the establishment of its syntax and semantics.

This paper is organized as follows. Chapter 2 attempts to illustrate the method of legal knowledge representation by logic flowchart. Chapter 3 deals with the reasons of the introduction of Compound Predicate Formulas (CPF), practical applications of CPF to legal reasoning, and its syntax and semantics.[1] Chapter 4 is the summary.

*Meiji Gakuin University:1-2-37, shiro-kane-dai, minato-ku, Tokyo, Japan;e-mail:hyoshino@tansei.u-tokyo.ac.jp.

[1] As the axiomatic system we adopt the prevailing standard one. Therefore we will not present it in this paper.

2 Representation of Legal Knowledge by Logic Flowchart

2.1 What is the Logical Flowchart of Legal Norm Sentences

Legal norm sentences have the structure of "legal requirement-legal effect." It means that when the legal requirement is met, the legal effect comes to occur. The legal requirement is composed of logical combinations of some legal requirement factors (legal facts). We can express this structure by a sort of flowchart, to put it more precisely, by a logical flowchart. Figure 0 shows the fundamental structure of legal norm sentence by logical flowchart.

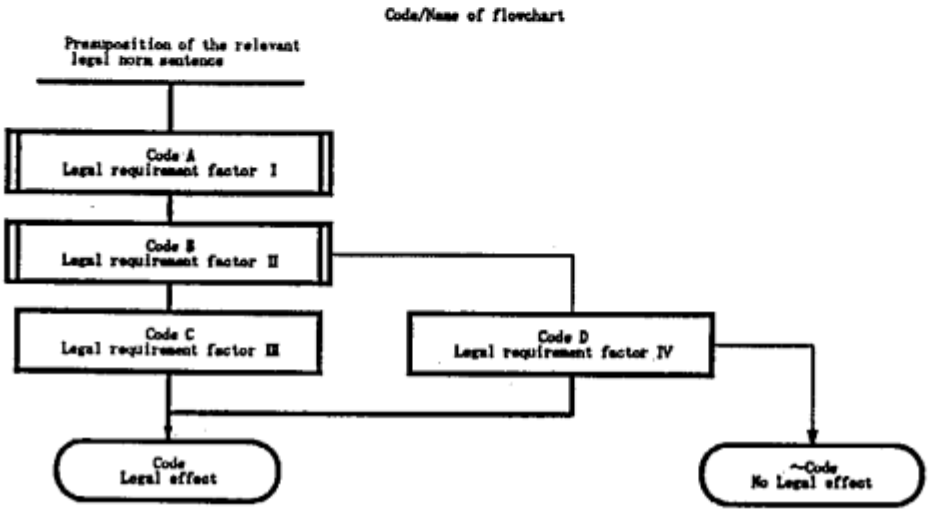


Figure 0. Fundamental Structure of Logic Flowchart of Legal Norm Sentence

In Figure 0 above, rectangular boxes stand for legal requirement factors (legal facts); lines, their logical combinations; ellipses, (no) occurrences of legal effects; part above the starting line, the presupposition of a given flowchart; the header of the flowchart, the code name and its title. Rectangles added vertical lines on both sides show that they have their child flowchart(s) to decide whether the requirement concerned is satisfied or not. When the decision of each legal requirement factor is affirmative, to put it another way, the proposition concerned is proved, we proceed to the next lower rectangle (i.e. the next lower legal requirement) which is continued by a vertical line, in principle. When the decision is negative, we proceed to the next box which is continued by a horizontal line, in principle. This flow of decision from top to bottom is taken to be not only a logical structure but the order of decision, and in many cases the order of time.

2.2 Principles of Logic Flowchart of Legal Norm Sentences

Here we will explain the principles that build up logical flowcharts of legal norm sentences in terms of the logical structures of legal norm sentences. The logical norm structure of a unit legal sentence consists in a combination of the legal requirement and legal effect. The relation

obtains that when the former has been met, the latter comes to occur. The combination of both is a logical one. In other words, the logical norm sentence has the logical structure that the requirement is the antecedent and the logical effect is the consequent: these can, therefore, be combined with logical operators, i.e., implication ("if": "$\longrightarrow$"), contra-implication ("only if": "$\longleftarrow$"), equivalence ("if and only if": "$\longleftrightarrow$"). Denoting the legal requirement by V and the legal effect by F, the logical structure of legal norm sentences can be expressed as three types of logical propositions.

(1) $V \longrightarrow F$: (if V, then F)

(2) $V \longleftarrow F$: (only if V, then F)

(3) $V \longleftrightarrow F$: (if and only if V, then F)

In type (1), V is a sufficient condition for F; in type (2), V is a necessary condition for F; in type (3), V is a necessary and sufficient condition for F. Replacing V with F in type (2), (2) is reduced to type (1). Type (2) is logically equivalent to the following:

(4) $\sim V \longrightarrow \sim F$

Type (1), (2) and (3) can be respectively expressed by logic flowcharts (Figure 1).

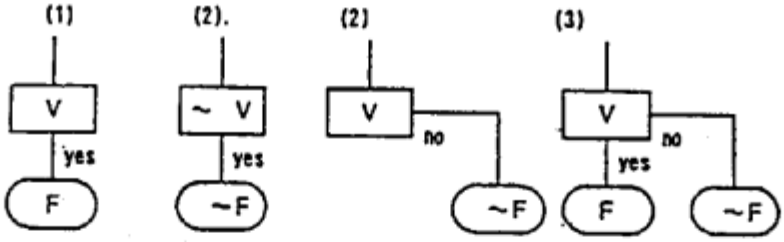


Figure 1. Fundamental logical structure of a unit legal norm sentence

The legal requirement can be analyzed into its legal requirement factors. The legal requirement factors, which are logically combined with each other, constitute one (unit) legal requirement that is combined with one legal effect. They are logically combined by the logical operators: conjunction ("and": " · ") and disjunction ("or": "V"). When the factors are of the legal requirement V1, V2, V3, the logical structure of the legal requirement results in three types (7), (8), and (9). The logical formulas correspond to the following logic flowcharts in Figure 2.

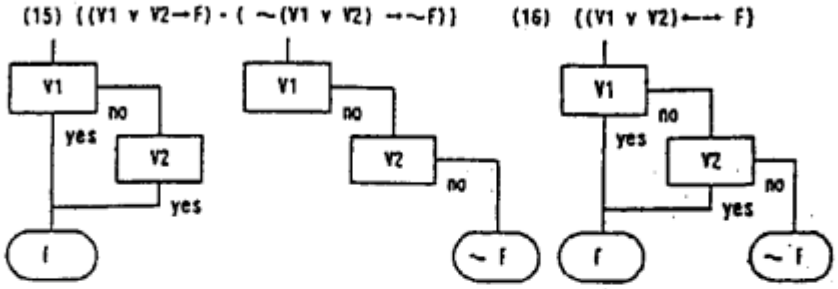


Figure 4, From the fundamental formula of legal norm sentence to a complete formula

The law has a hierarchical systematic structure, in which more abstract concepts involve legal concepts with concrete contents and more abstract concepts are concretized by more concrete concepts. This is also the case with the combination of legal norm sentences, and the latter is much concretized by the former. The principle for integrating legal norm sentences which have a different degree of abstraction (or concretion, as is the same meaning) lies in the definition of a rule. Namely, two norm sentences are combined with each other by the logical operator for equivalence "←→". When a more abstract legal requirement factor V1 consists of more concrete legal factors V1.1 and V1.2 and V1.1 is further concretized by V1.1.1 and V1.1.2, then the logical formulas and logic flowchart will be expressed as in Figure 5.

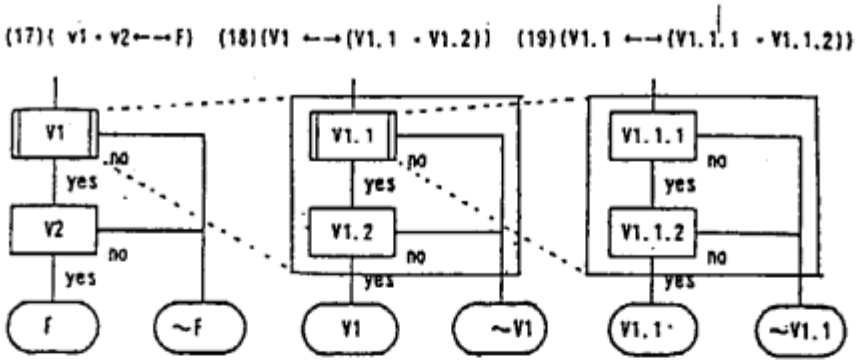
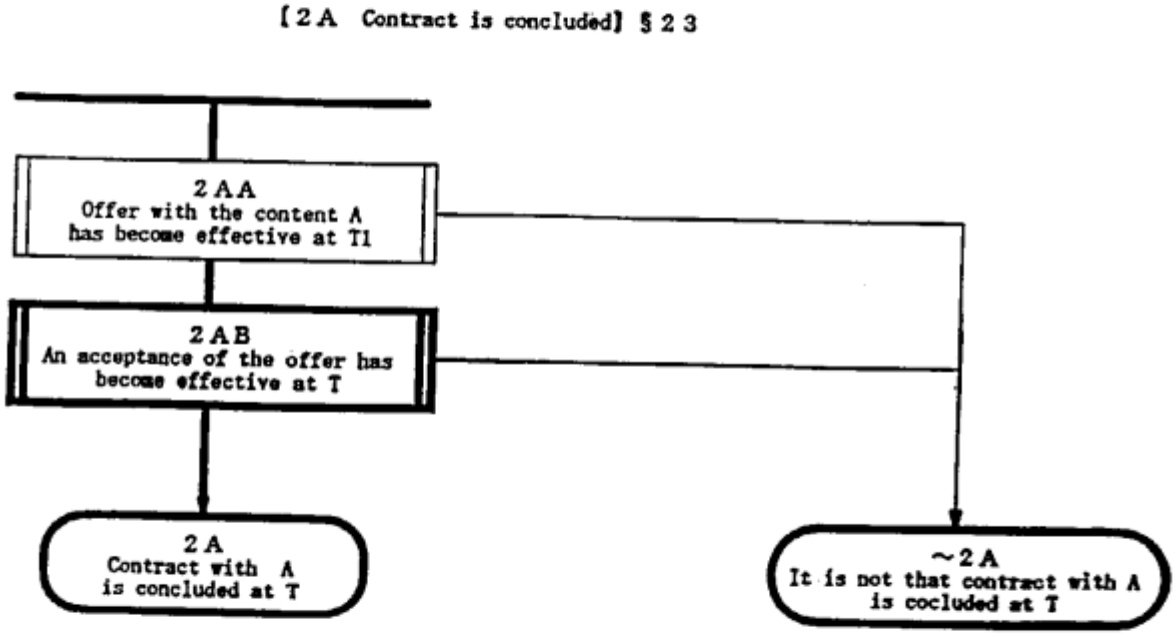


Figure 5. The logical structure of hierarchical connection of legal norm sentences

In Figure 5 above, the relation between (17) and (18) as well as (19) may well be called a main-sub, or parent-child relation. That is, (17) is parent of (18), and (18) is child of (17). The same thing can be said for (18) and (19). Namely, (18) is parent of (19).

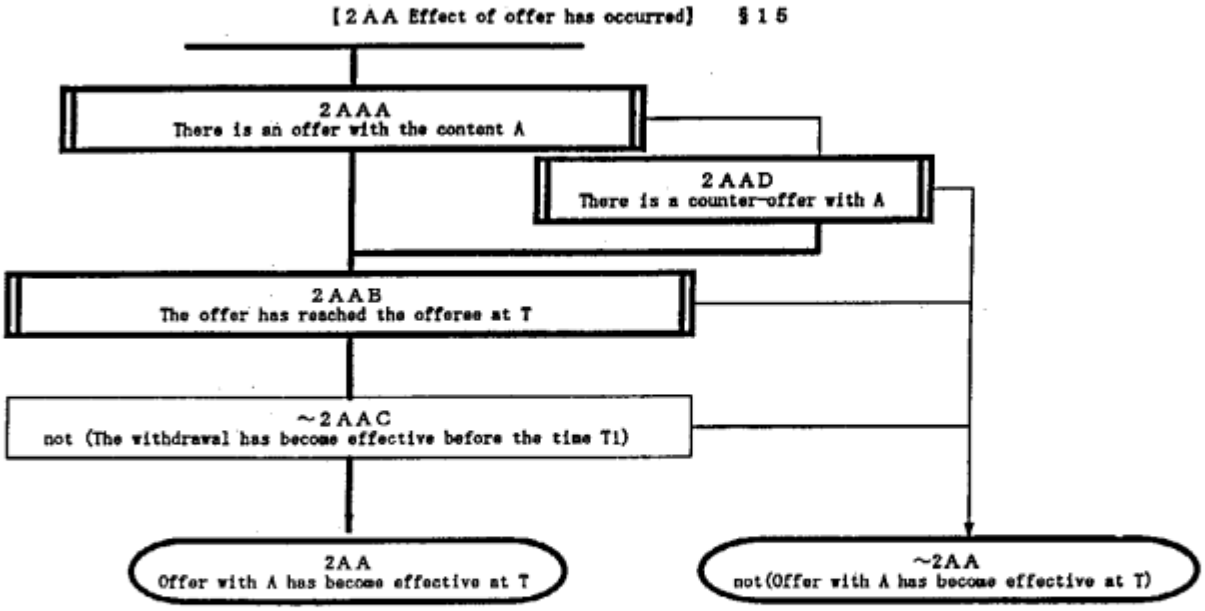
2.3 Examples of Logic Flowcharts of Legal Norm Sentences

I will show two examples of logic flowcharts of legal norm sentences in Figure 2A and 2AA. Both are taken from United Nations Convention on Contracts for the International Sale of Goods(CISG). In Figure 2A we can see the legal effect of "formation of contract" and the legal requirement which has the effect occur. This figure corresponds to Art 23 of CISG. In this figure, the thick rectangle indicates that the concepts which are expressed in rectangles or ellipses are described in some place of the articles concerned, and the thin rectangle indicates that the concepts are not contained in any place of the articles in question. Figure 2AA is the child flowchart to decide whether the first legal requirement factor 2AA "An offer of A has become effective at T1" is satisfied or not. Figure2AA is based on Article 15 in the said law.



Article 23

A contract is concluded at the moment when an acceptance of an offer becomes effective in accordance with the provisions of this Convention.



2.4 Advantages of Legal Knowledge Representation by Logic Flowchart

The method of legal knowledge representation by logic flowchart has the following advantages. Firstly, lawyers can easily build up this flowchart by themselves and understand the contention of it. Secondly, for lawyers to build up this flowchart makes clear the order between lawyers' judgments, which they make tacitly, and the time order among legal facts. Thirdly, to build up this flowchart is effective for systematizing legal knowledge. In doing so, we may discover or confirm legal common knowledge that is presupposed among lawyers, and, above all, we may abstract knowledge as a frame for systematizing knowledge. However, this flowchart has its limit in that the formulation is basically in the level of propositional logic. To obtain more precise representation of legal knowledge dependent upon the inner structure of sentences, we need to formalize this knowledge in terms of predicate logic. I have developed the Compound Predicate Formulas, which is a conservative extension of predicate logic. Next chapter is concerned with this.

3 Representation of Legal Knowledge by Compound Predicate Formulas (CPF)

3.1 Why CPF?

In this section I will explain the reason why I have introduced CPF, not others. To say the reason in brief phrases, in order to represent legal knowledge adequately and plainly. To

clarify this point, we will take a simple example from legal sentences and present the difficulty of representing such a sentence by the standard first order logic. Consider this:

Ex. John made an offer to Mary, and it was accepted.

It is difficult to express the whole sentence in the standard first order language, for the standard first order language does not contain any device for representing the referential expression "it". We can symbolize, in the above sentence, "John made an offer to Mary" as "offer (John, Mary)," but how can we symbolize "it was accepted"? We all agree that the referential pronoun "it" that is part of the above sentence refers "an offer of John to Mary." Unfortunately the first order language has the ability to refer only individual entities but not any state of affair such as an offer made of John to Mary. As far as we are in the standard first order language, we must content ourselves with symbolizing the above example like, using a predicate $p(X_1, X_2)$, $offer(John, Mary)$. Though, this symbolizing does not reflect the inner structure this sentence has. This fact implies that the standard first order language is not rich enough to adequately represent legal sentences, and therefore to describe legal reasoning.

In short, the standard first order language lacks the means to refer to each legal act, which involves "that contract" or "the trade at 15:00 on Feb. 3rd, 1994." Moreover the standard language has no device to represent referential pronouns, say "that." What we really need is some richer language that enables us to deal with these expressions.

3.2 Some Precedent Approach

3.2.1 Lambda Abstraction

For instance, if the sentence "X contracts with Y" is formalized as "contract," then the relation of contracting is , according to lambda notation, described as:

$$\lambda X \lambda Y(contract(X, Y))$$

In general given a predicate, lambda operators form the expression that refer to the concept the predicate denotes. However, by this lambda abstraction it is very difficult to designate any particular contract e.g. "the contract made between John and Mary at that time." Again what we would like to obtain is the method for referring each concrete instantiation of given legal relations rather than one for any abstract concept.

3.2.2 Class Notation

Then what if the class notation ? In this notation, relations corresponds to classes and each instance, for example:

$$A = \{X : acceptance(X)\}$$

$$O = \{Z : \exists X \exists Y(offer(X, Y) \wedge Z =< X, Y >\}$$

$$\exists Z(Z \in O \wedge Z \in A)$$

A and O show (the extension of)a concept of "acceptance" and a concept "offer," respectively. Then the below (1) is obtained as the translation of a legal sentence "X made an offer to Y and it was accepted":

(1) $\exists Z \exists X \exists Y (offer(X,Y) \wedge Z =< X,Y > \wedge acceptance(Z))$

As (1) shows, the class notation is adequate for denoting a concept itself while this method is clearly not satisfactory for denoting its particular instance. Indeed each instance is a certain element of a given class, as already noted. Many of expressions in natural languages, however, involve ample pragmatical i.e. contextual information, and it is hard to specify a purposed set, considering much contextual information, and even though the specification is acquired, the formula is all too often complex for us to understand. Also the class notation does not have any type of apparatus to represent referential pronouns. In legal sentences and legal reasoning, one does meet with referential expressions frequently.

3.3 A New Device–ID-symbols

In the previous section we have recognized that some approaches to overcoming difficulties as to the standard first order language are inadequate to express each individual legal act or to describe the sentences with referentials in the form which reflects the inner structure of them. Now we are in a stage to offer a new device for coping with these puzzles: ID-symbols. Though, before introducing ID-symbols we will account for the rough idea of them from which they stem.

Let us suppose that:

1. $offer(Z,X,Y)$:Z is an offer of X to Y.
2. $acceptance(W,Z)$:W is the acceptance of Z.

If we assume these formulas, then a sentence "An offer of X to Y was accepted." would be formalized as follows:

(2) $\exists Z(offer(X,Y,Z) \wedge acceptance(Z,W))$

Compared (2) with (1), we find that (2) is more simple and plain, for in the assumptions expressions within which they contain the way for referring to a certain specified individual legal act or relation had been posited. And this is how we contrive a device called ID-symbols.

In general for any predicate $p(X_1,\ldots,X_n)$,

$$ID - p(X_1,\ldots,X_n)$$

is the predicate in question. For instance, for a predicate "contract(Mary,John),"

$$ID - contract(Mary,John)$$

expresses a contract between Mary and John. In other words, it is the name of that contract. Using ID-symbols, the formula (2) is more concisely rewritten as

(3) $acceptance(-,ID-offer,-)$

Here I would like to emphasize, as the characteristics of ID-symbols, this sort of nominalization, i.e. an ID-symbol forms the name of a particular instance in a given concept. (Recall that notations by lambda operators or classes form the name of a concept itself.) As one more example, "the reject of that offer" is, by ID-symbols, formalized as

(4) $ID-reject(-, ID-offer(-), -)$

Using ID-symbols, we can easily deal with such a relatively complex case.

We may note, in passing, that, for ID-symbols to function as names, it is necessary that the obvious identity criterion for the referents of Id-symbols is given. It means that for particular instances of a concept the condition of continuity through time must be defined. So as to define that condition, we ought to define each legal concept strictly. But this problem is the matter of law and not that of logic.

3.4 Outline of Syntax of CPF

The following attempts to define the syntax of CPF. Most portions are the same as the standard first order language, CPF is highly different from the language so far in that CPF has new devices such as case symbols[2] and ID-symbols. We have to define an describe the syntactical behaviors of ID-symbols more fully than here. But our concern here restricted only to program clauses. So we will think only quantifier free part i.e. Horn clause:

$$B \longleftarrow A_1, \ldots A_n$$

where $B, A_1, \ldots, A_n$ are literals.

The syntax of CPF is as follows:

1.Basic Vocabulary:

1.1.individual variables: $X_1, X_2, \ldots T_1, T_2, \ldots$

1.2.individual constants: $a_1, a_2, \ldots$

1.3.case symbols $agt:, obj:, goa:, tim:, \ldots$

1.4.predicate letters: $p_1, p_2, \ldots$

1.5.list symbols [,]

1.6.logical constants: $\neg, \longleftarrow, \forall$

1.7.commas, parentheses: ( ) ,

2.terms and formulas:

2.1. Variables, individual constants and ID-symbols are terms.

2.2. If t is an individual constant, an individual variable or an ID-symbol, then $agt:t$, $obj:t$, $tim:t$, $goa:t$ are terms. ($c_1, c_2, \ldots$.stand for case symbols.)

2.3. $[t_1, \ldots, t_n](t_i(1 \leq i \leq n))$ is a list.

2.4. $p([t_1, \ldots, t_n])$ is a formula.

[2]Case symbols are a notation contrived to express the inner structure of predicate explicitly. On the syntactic and semantic status of case symbols, we may leave to another occasion.

2.5. If A and B are formulas, then $\neg A, A \longleftarrow B$ are formulas.

2.6. If $A(X)$ is a formula, then $\forall X A(X)$ is a formula.

2.7.The definition of ID-symbols: For the predicate representing legal concept $p([t_1, \ldots, t_n])$, $ID - p([t_1, \ldots, t_n])$ is an ID-symbols of its predicate.[3]

2.8. For a predicate $p([t_1, \ldots, t_n])$, $p(ID - p, [t_1, \ldots, t_n])$ is a formula as well.[4]

2.9. An expression is a formula only if it can be shown to be a formula on the basis of conditions 2.4-2.6, 2.8.

Other logical constants are introduced by the definitions below:

$$A \wedge B \stackrel{\Delta}{=} \neg(\neg B \longleftarrow A)$$

$$A \vee B \stackrel{\Delta}{=} B \longleftarrow \neg A$$

$$\exists X A X \stackrel{\Delta}{=} \neg(\forall X \neg A X)$$

3.5 Legal Knowledge Representation in terms of CPF

Having defined syntax of CPF, we state legal knowledge representation using CPF. Here we cite an article and show how it can be translated into a formula of the language of CPF.

CISG article 23: A contract is concluded when an acceptance of an offer becomes effective.

1. $contract(ID - co, [agt : [X, Y], obj : C])$: A contract C was made between X and Y.

2. $acceptance(ID - ac, [agt : X, obj : ID - of, goa : Y])$: X accepted $ID - of$ to Y

3. $offer(ID - of, [agt : X, goa : Y, obj : C])$: An offer C was made to Y by X.

4. $be - concluded(ID - bc, [obj : ID - co, tim : T])$: $ID - co$ was concluded at time T.

5. $become - effective(ID - bc, [obj : ID - ac, tim : T])$: $ID - ac$ became effective at time T.

Based on these, the above article is translated into the formula below.

$$\begin{aligned} & be - concluded(ID - bc, [obj : ID - co, tim : T_1]) \\ & \quad \wedge contract(ID - co, [agt : [X, Y], obj : C]) \longleftarrow \\ & \quad become - effective(ID - be, [obj : ID - ac, tim : T_1]) \\ & \quad \wedge acceptance(ID - ac, [obj : ID - of]) \\ & \quad \wedge offer(ID - of, [agt : X, goa : Y, obj : C]) \end{aligned}$$

Such a formula is called Flattized CPF formula(FCPF), and it is equivalent to the CPF formula below:

[3]We will omit the arguments in ID-symbols unless leading to misunderstanding. Derivatively we will define ID-symbols about predicate symbols as well.

[4]The former is called the formula without ID-symbol, the latter the formula with ID-symbol as a matter of convenience. As easily seen, we need a formula we need a formula to assure $p(t_1, \ldots, t_n) \Longleftrightarrow p(ID - p, t_1, \ldots, t_n)$,if we make an axiomatic system for a CPF.

$be - concluded(ID - bc, [obj : contract(ID - co, [agt : [X, Y], obj : C]), tim : T_1]) \longleftarrow$
$become - effective(ID - be, [obj : acceptance(ID - ac, [agt : X, goa : Y,$
$obj : offer(ID - of, [agt : X, goa : Y, obj : C])])], tim : T_1])$

This formula is an abbreviation of the above FCPF formula. Legal sentences are described and stored into knowledge base in this form. To execute the predicational reasoning, these formulas are compiled (flattized) into FCPF above.[5]

Next is the outline of procedure of the flattization. Any CPF formula A is flattized into an FCPF formula, i.e., for any CPF formula A,

1. if A contains no formulas which have the form of $p(ID-p, [c_1 : t_1, \ldots, c_i : q(ID-q, []), \ldots, c_n : t_n](1 \leq i \leq n)$ in A , the formula is not flattized.

2. if A contains any formulas described in 1, choose the left-most one in that formulas, replace $c : q(ID - q, [\,])$ with $c : ID - q$, and replace the original formula with the below one,

$$p(ID - p, [\ldots, c : ID - q, \ldots]) \wedge q(ID - q, [])$$

3. Repeat the procedure of 2 until it is not applicable.

3.6 Application of CPF to Legal Reasoning

In CPF ID-symbols play an important role. We have already seen some advantages of ID-symbols. In this section I will expand an advantage by the introduction of ID-symbols in legal reasoning. Since our CPF has its basis on the standard first order language, we can use inference rules of it. Besides that, ID-symbols increased the power of our language so that we could deal with some legal reasoning cases that have been difficult to cope with so far. For example, let us consider an inference like this:

Premise1. A made a contract with B.

Premise2. If that contract is effective, then A can claim to payment to B.

Premise3. That contract is effective.

Conclusion. A can claim payment to B.

We will be in trouble with this inference if we have to formalize this within the standard first order language. The trouble is derived from that the standard predicate logic has no device for referring to any particular instance like "that contract". On the other hand, CPF tells us that the above inference is valid. The formalization by CPF is below:

Premise1'. $contract(ID - co, A, B)$

Premise2'. $claim - payment(A, B) \longleftarrow is - effective(ID - ic, ID - co)$

Premise3'. $is - effective(ID - ic, ID - co)$

[5]This flattization is, substantially, the procedure of converting a many-sorted formula into a one-sorted one.

Conclusion'. $claim - payment(A, B)$

Notice that we can deal with this inference not because we have extended the inference rules of the standard first order logic (in fact we have not extended them), but because we have introduced ID-symbols, which enables us to refer to particular instances of a given act. That is to say, in the case of quantifier-free part, CPFL is an extension of the standard first order language.

I would like to suggest more two points about the usage of ID-symbols: in an inference (1) even if a particular ID-symbol is used with its argument not specified, we may identify the ID-symbol safely, and (2) when a particular ID-symbol is used and embedded in another ID-symbol such as a case , $ID - r(t_1, \ldots, ID - contract, \ldots, t_n)$, we might be in trouble as to the identification of given plural ID-symbols.

3.7 Semantics of CPF

We can define semantics of CPF as usual. Only difference between usual first order language and that of CPF is the introduction of ID-symbols in the latter.

The definition of a model of CPF is as follows.

Definition 3.7.1 (Level of ID-symbols) *Given an ID-symbol $ID - P(t_1, \ldots, t_n)$, the number of ID-symbols in $ID - P(t_1, \ldots, t_n)$ is called a level of the ID-symbol, and we express it as $LEVEL(ID - P(t_1, \ldots, t_n))$.*

Definition 3.7.2 *$M =< D(= D_1 \cup D_2 \cup \{\perp\}), I >$ is a model of CPFL with respect to $g(= g_1 \cup g_2)$ $\Longleftrightarrow$*

1. *D_1 and D_2 are a class of individuals and a class of time points respectively. We stipulate that $D_1 \cap D_2 = \emptyset$. $\perp \notin D_1 \cup D_2$.*

2. *$g_1 : INDVAR^6 \longmapsto D_1$ and $g_2 : TIMVAR \longmapsto D_2$.*

3. *If t is an individual constant, $I(t) \in D_1$.*

4. *Given an n-place predicate p, $I(p) \subseteq D^n$ where D^n is a Cartesian product of D with n times.*

5. **interpretation of ID-symbols** *Given a predicate $p(t_1, \ldots, t_n)$, the interpretation of its ID-symbol $I_g(ID - p(t_1, \ldots, t_n))$ is defined by the induction on the level of the ID-symbol.[7] $I(ID - p)$ is a function defined on D. [8] If $LEVEL((ID - p(t_1, \ldots, t_n))) = n \geq 2$, and the interpretation of ID-symbols of the level less than n have been defined, then $I_g(ID - p(t_1, \ldots, t_n))$ is defined as follows:*

   1. *If $(I(ID - p) \neq \emptyset$, pick up an a such that $a \in I(ID - p)$ so that $I_g(ID - p(t_1, \ldots, t_n)) = a$.*

[6] $INDVAR$, $TIMVAR$ are the set of the individual variables and the set of the time variables respectively.

[7] I_g is a function such that for a variable X, $I_g(X) = g(X)$ and for a term t of the other kind, $I_g(t) = I(t)$.

[8] In a special case, $I(ID - p)$ is a function. As to the reason why we have defined in a more general way, we will explain later. The meaning of an ID-symbol is substantially identical to the meaning of constant in the case of quantifier-free formulas.

2. *Otherwise,* $I_g(ID-p((t_1),\ldots,(t_n))) = \perp$.

Definition 3.7.3 (Satisfaction of the Formulas) *We define the satisfaction of the formulas of CPF by the induction on the complexity of them. Namely, for any model* $M =< D, I >$ *and any assignment* g,

1. $M \models_g p(t_1,\ldots,t_n) \Longleftrightarrow < I_g(t_1),\ldots,I_g(t_n) > \in I(p)$[9]

1.1. $M \models_g p(ID-p,t_1,\ldots,t_n) \Longleftrightarrow < I_g(t_1),\ldots,I_g(t_n) > \in I(p)$[10]

2. $M \models_g \neg A \Longleftrightarrow M \not\models_g A$[11]

3. $M \models_g B \longleftarrow A \Longleftrightarrow M \not\models_g A$ *or* $M \models_g B$

4. $M \models_g \forall X A(X) \Longleftrightarrow \forall g' s.t. g' =_X g, (M \models_{g'} A(X))$[12].

We should explain the idea behind the model construction above. Interpretation of ID-symbols needs special explanation. If we understand an ID-symbol by analogy to a function,

$$ID-r :< X_1,\ldots,X_n > \longmapsto X_{n+1}$$

Namely,

$$ID-r(X_1,\ldots,X_n) = X_{n+1}$$

Therefore, we are tempted to define as follows: given an assignment g,

$$I_g(ID-r(X_1,\ldots,X_n)) = I_g(ID-r)(< g(X_1),\ldots,g(X_n) >)$$

where $I_g(ID-r)$ is a function.

If $I(ID-r)$ is a function with non-empty range, its corresponding role of $I(ID-r)$ in ordinary language is that of a singular term. We have expressions of this kind. For example, "that contract between a and b"is such an expression. But in any case, can we say that the above mentioned contract is unique or many or none? If there are several contract between A and B, and we can't determine the special one by the lack of information, then the decision remains obscure. Suppose that there are several contracts between A and B such as the contract on March 17, and the contract on November 16. For this reason, it is possible that when debating, they misunderstand each other what contract they are talking about. And it is also possible that when we infer about some contract, we do so without complete knowledge of the contract. Rather it seems that such a reasoning is typical in our ordinary life.

We would like to comment on the interpretation of ID-symbols.

- If we don't have enough information of ID-r to make it function, then $I(ID-r)$ would be a correspondence. But if it is a function, then there will be no problem of misidentification. In this case we can think with appropriate objects.

[9] $M \models_g A$ should be read as: g satisfies A in M. In CPF predicate of the original form, every argument of the predicate appear within a list, but for the sake of convenience, we employ predicates in a standard form. There is no essential difference between them.

[10] The satisfaction is defined in the same way for atomic formula without ID-symbols $p(t_1,\ldots,t_n)$ and atomic formula with ID-symbols $p(ID-p,t_1,\ldots,t_n)$.

[11] $M \not\models_g A$ should be read as: g does not satisfy A in M.

[12] $g =_X g' \overset{\Delta}{\Longleftrightarrow}$ for any Y *s.t.* $Y \neq X g(Y) = g'(Y)$

4 Conclusion

We have so far been stating the quantifier-free part of CPF (syntax, legal knowledge representation using it, semantics, and so on). At the end we are going to summarize in short merits of introducing CPF, especially ID-symbol.

- CPF makes expressive capacity richer, and can mention each legal acts of buying and selling.
- CPF makes legal knowledge representation which has close form to natural language.

In this way CPF has big advantages. Above all the best significance of this paper is to give a logical basis of CPF.

I would like to state further tasks. I have hesitated offering the explanation of case symbols in order to avoid making obscure the forms of argument, but they are an important tool. Case symbols are a device for clarifying that which role terms play in a predicate. It is interesting to give semantics to such a category of grammar.

I then have often mentioned the characteristics of ID-symbols as a demonstrative pronoun. Next steps, we must consider other kinds of demonstrative pronoun (indexicals such as " I ", demonstratives such as " the book I have " and so on) from the wider point of view. Now I present two points to consider in future.

- How to formally identify objects that are represented by making some extension of first order language
- How to combine ID-symbols and many sorted language

References

[1] Ebbinghaus, H.D., J.Flum and W.Thomas, *Mathematical Logic*, Springer,1984.

[2] Gupta, A., *The Logic of Common Noun*, Yale University Press, 1980.

[3] Rödig. J, Über die Notwendigkeit einer besonderen Logik der Normen, in Rechtstheorie als Grundlagenwissenschaft der Rechtwissenschaft, hrsg.v. Albert, H.,Luhmann, N.,Maihofer,W.,Weinberger, O.,Jahrbuch für Rechtssoziologie und Rechtstheorie Bd,2(1972)

[4] Smullyan, R., *First-order Logic*, Springer,1968.

[5] Yoshino, Hajime, Über die Notwendigkeit einer besonderen Normenlogik als Methode der juristischen Logik in Gesetzgebungstheorie, Jurisitische Logik, Zivil- und Prozeßrecht Gedächtnisschricht für Jürgen Rödig, Springer-Verlag Berlin Heidelberg 1978

[6] Yoshino, Hajime, Possibility of Applying Computers to Judicial Process, (the prize paper of The YOMIURI Newspaper Company). The YOMIURI, evening edition, 22nd December 1983, page 9.

[7] Yoshino, Hajime, 'Application of computer to reasoning in legal adaptation process, in *Law and Computer*, No.3, June, 1985, pp.77-94 (in Japanese)

[8] Yoshino, Hajime, 'Logical Structure of Law and the Possibility of Computer Aided Legal Reasoning' in: ARSP(Archiev für Rechts- und Sozialphilosophie) Beihefte Nr.30, 1986, pp. 185-202

[9] Yoshino, Hajime, et al. 'Legal Expert System Les-2' in: Proc. of "The Logic Programming Conference '86", 1986, pp. 68ff. (in Japanese)

[10] Yoshino, Hajime, et al. 'Legal Expert System Les-2', in: Wada, E. (Ed.)Logic Programming '86 (Lecture Notes in Computer Science 264), 1987, p.36ff.

[11] Yoshino, Hajime, A research report on Legal Expert System, March, 1989, Association for Machine System Promotion, pp.51-81 (in Japanese)

[12] Yoshino, Hajime, A research report on explication of legal knowledge structure and development of legal knowledge-base. March, 1990, Association for Machine System Promotion, pp.27-32, pp.41-55 (in Japanese)