博 士 論 文

対立を用いた法的知識の整合性検証

指導教官　東条 敏 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

萩原 信吾

2008 年 2 月 15 日

要 旨

本稿では法の矛盾を検証する具体的な手法とその実装を提案する．まず人間の直観に基づいた矛盾を検出可能とするため，矛盾の定義に対立 (conflict) を導入する．これは特定の名前の組に対して矛盾を定義するもので，矛盾律 '$A \wedge \neg A$' にみられるような名前一致の制限がない．そのため矛盾律よりも広い矛盾が定義可能である．この対立を導入するためには，対立概念の集合を定義する必要がある．そこで語彙の上位下位関係から，順序関係に対する代数演算の交わり (meet) を用いて対立概念を求める．これにより対立導入における人的負担の軽減を図る．検証に用いる法的知識は，法の法準則を，選言的拡張論理プログラム (EDLP: extended disjunctive logic programming) で形式化したものとする．法準則を対象にするのは規則として解釈可能なことによる．また EDLP を形式化手法として選ぶのは，法準則が選択的帰結を持つことによる．またこの法的知識は，非循環なものに制限される．これは法的知識から得られる結論に対し法的根拠を保証するためである．この法的知識はほぼ規則から構成される．そこで対立の検証では仮説推論を用い，ある法準則が適用される状況を仮定し，その法準則と対立する法準則が同時にその状況に適用されていないかを検査する．これにより論理的には矛盾ではないが，法的知識として矛盾である規則の集合を検出する．本研究が提案する検証手続きは，実装可能で，かつ実用的であることを目的とする．したがって，データの仕様，その具体的な生成方法，そして実用性を勘案した検証手続きを提案する．本稿の実装は前処理器と検証器に分けられる．前処理器は XML で記述された検証の基礎データを検証器に合わせた形式に変換し，そして検証器が求めるデータを抽出する．検証器はそのデータを受け取り，論理推論が関係する処理や代数演算を行う．またこれらの実証実験として，本稿の実装を用いて富山県の条例を検証する．そしてその結果を示す．

目 次

第 1章

はじめに

法規範は整合的であることが求められる [39,85]．規範とは行為や判断や評価を行う際の基準をいい，法規範とは法が定める規範のことをいう．厳密に法規範は「法共同体の成員が自己の行動の基準として受容し，自己の行動の正当化の理由や他人の行動に対する要求・期待あるいは批難の理由として公的に用いるもの」とされる．この規範が相互に整合的であるとは,「規範相互が少なくとも適用事例について対立する結論に至らないこと」とされる [22,37,93]．つまり法が整合的であるとは，ある事例に対し相反する結論を持つ法が同時に適用されないことをいう．

法に整合性が求められるのは，その法規範を前提なしに容認し合う人の共同体を法共同体が社会と呼ばれるからである．社会の成員は社会を規定する法規範を基準に行動している．もし法が整合的でないならば，法は相反する結論を社会にもたらす．例えばある法規範を基準にした社会において，ある成員がある状況である法律の結論を期待して行動をする．しかし法が整合的でなければ，別の成員はその法律の結論とは相反する効果を期待することとなる．それぞれに関係がなく別々の状況に置かれている場合はその問題が顕在化することはない．しかし両者が同一の状況下の当事者である場合，その二人の法共同体の成員は社会的に相反する行為をしたことになる．よって法は整合的でなければ第三者の立場として処決を決定するという能力を持たなくなる．

この法の整合性については，目的が異なる法同士であるならば不整合でもよいという考えもある．事実，法律には一見すると矛盾しているととれるものもあり，またそれが法として現に運用されている．そのような一見対立する法が相互に運

用できている理由は，法解釈という手順が法の適用に際して挟まれることによる．法規範を制定する法令文は文字列であり，それが実社会に対して実行を伴う効力を持つには法令文がどのような意味を表すのか法解釈が行われる．一見対立する法が相互に運用できているのは，法解釈によって結論が対立しないような解釈が与えられているからである．このことは法が法令文の段階で不整合であるがゆえ解釈によってその問題を回避していることを表すものであり，法が不整合でも問題がないことを表すものではない．法が本来整合的であれば，解釈に腐心する必要はない．したがって本稿では，法は理想的には整合性であるべきであるという立場をとる．

以上から法は極力整合的に制定されるが，改正によりその整合性が脅かされる．実例として，本稿の実検対象である富山県の条例が挙げられる．2002 年に富山県では各種申請を電子的手段で行えるよう条例を制定した．以下はその原文である．

> **富山県条例 54 号 第 3 条**
> **(電子情報処理組織による申請等)**
> 県の機関は申請等のうち当該申請等に関する他の条例等の規定により書面等により行うこととしているものについては当該条例等の規定にかかわらず規則で定めるところにより電子情報処理組織 (県の機関の使用に係る電子計算機 (入出力装置を含む以下同じ) と申請等をする者の使用に係る電子計算機とを電気通信回線で接続した電子情報処理組織をいう) を使用して行わせることができる

これは昨今の情報技術の発展にともない，行政手続及び各種の申請を Web 上で可能にするために制定された条例である．当然ながら，この条文の追加により富山県では既存条例との整合性を検証した．そのとき「申請は書面等で行われ，書面等は物理的なものである」ということを前提とする条文や定義などを起因とする不整合が見つかった．このことは，検証なしの改正では法が不整合部分を含む可能性を示唆するものであり，同時に改正時の検証の必要性と重要性を示すものである．

さらにこの法改正は避けることができないことが問題となる．既に制定された法が整合的であるのならば，法の改正は極力しないことがその整合性を保つ上で

は有利である．しかし社会規範でもある法は，変化する社会の要求に応じて改正されなければならない．法共同体である社会は人間が成員として構成され，その時代や情勢によって成員の価値観や法共同体に対する要求が変化する．法とはその改正が行われない限り変化しない静的な存在であるがため，その要求に応えるために改正され続けて行かなければならない．さもなくば社会の実情との間に差異が生まれ，法共同体の基礎であるという能力を失う．

しかしその法検証作業は，複雑で膨大であることが問題である．その検証作業が膨大である理由は，既存の法が大量に存在することが最も大きな理由である．法は時限立法や暫定法でない限り基本的に恒久法である．すなわち既存の法を削除する新法が制定されるかその法を受容する共同体がなくならない限り，既存の法は法としての効果を持ちながら存在し続ける．よって既存の法は累々と積み重ねられ大量に存在する．法の検証作業ではこの既存の法と新しく制定される法が整合的であるかどうかが検証される．つまり既存の法が大量であることは同時に，検証作業が膨大であることを意味する．もう一つの検証作業が複雑である理由は，この既存の法が複雑であることである．法は単独で完結しているものもあるが，その多くは他の法と関係している．法改正における検証では，新しく制定される法と直接的に関係を持つ既存の法のみを取り出し検証できればよいわけではない．その直接的関係を持つ法がさらに関係している法まで検証する必要がある．よって検証作業は，大量にある既存の法の，複雑な関係を紐解きながら新法との関係箇所を洗い出す必要がある．

また検証は，再帰性を持つことも問題になる．この一連の検証作業により非整合的箇所が検出された場合，当初の改正法に加え整合性を保存するための改正法も必要に応じて追加される．しかしこの修正及び改正法の追加が行われると，再び同様の検証作業がそれらに対して必要になることは想像に難くない．

このように法改正時の検証作業とは,「大量」でかつ「複雑」な既存の法に対して行われる上に「再帰的」な検証が求められる．このような作業を人が行うと，人為的ミスによって整合性を失う原因箇所を見逃す可能性は否定できない．しかしながらこれだけ困難な作業にも関わらず，現状は人手によって時間をかけて行われている．そこで本稿では，この法の整合性検証に対する機械的支援を目的とし，具体的な検証方法及び実装を提案する．

法の検証作業を計算機によって支援することが期待できる技術として，法令工学 [94] がある．法令工学は，主に法的知識 (legal knowledge-base) や，法的行為 (legal argumentation) などを，形式的に計算機上であつかう研究分野である [11,24,36,63,84]．このうち法的知識に関する研究は，法的な情報を論理的に用いるためにその形式化を研究する分野である [60]．法的知識の研究では，形式化を行う対象は法令文だけではなく，裁判等での裁判官や弁護士または検察官などがその立論に用いる知識，そして法廷で行われる論争なども対象とされる [10,74]．一方法的行為の研究は，主に論争という行為を形式化する研究である [10,59,69,72,73,76]．論争行為とは，ある立論を否定することでその立論を論破（無効化）するなどのことである．つまり，立論をどのように無効化するか，また知識からどのようにその無効化のための立論を作るかなどが研究の対象となっている．

これらを通して見ることができるのは，これら法令工学の研究分野では論理学的側面からの働きかけが大きいことが挙げられる．それは論理学がいわゆる正しさを形式的に研究する分野であること，そしてこれらの研究が知識からの演繹，矛盾の検出，さらに互いの立論における矛盾点の指摘や否定が焦点となっているからといえる．よって上記法令工学は，法の整合性検証のような「知識の論理的妥当性」を検証することに有用性，適用性があると考えられる [9,38,85]．

本研究ではこれらの研究を応用し，法の整合性の検証を機械的に支援するため，その具体的な検証手続きの提案と実装を行う．ただし現実的な問題として，実際に人間が行う法の検証工程の全てを機械的に行うことは難しい．本研究が対象とすることの詳細は 2，3 章で説明する．

法令工学を応用するには，法を機械的に扱うため法的知識の形式化手法と矛盾の定義，そしてその矛盾の検出方法と法における検証の対象範囲を具体的に決める必要がある．法令文は論理式や論理プログラムなどで形式化され法的知識とされる [59]．この法的知識を作成する際には利用目的に応じた適切な形式化体系を選ばなければならない．形式化手法が適切でない場合，目的とする結果を得られないばかりか，目的以上の形式化により，必要のないコストが発生する．

この論理的形式化手法には様々ある．例えば義務論理 (deontic logic) [4] を採用したもの [35,52] がある．義務論理は，様相演算子を用いて，「義務 ($\mathcal{O}$)」と「権利 ($\mathcal{P}$)」という概念を論理的に取り扱う論理体系である．これは，法の適用，つまり

は権利の行使を演繹する際に用いられることが多い．このほかにも法的知識の形式化には動的論理 (dynamic logic) [40] を用いたものもある [33]．これは動的な変化をともなう知識に対応したもので，特に変遷する議論などをあつかうのに用いられる．また拡張論理を用いたものもある [76]．これは，選択的な帰結を用いて行われる法的論争の研究である．そのほか，この法的知識の形式化には，それぞれの研究で用いられる実装に合わせた，独自の形式化手法を用いたものも多数存在する [9, 36, 81, 82].

本研究では法的知識の形式化手法として選言的拡張論理プログラム (EDLP: extended disjuctive logic program) [31] を採用する．これは本研究の目的である規範相互の矛盾を検出するためには以下のことがその形式化手法に要求される．その要求とは，まず法の規範を形式化可能であること．そして次に，その法が互いに矛盾を引き起こしていないかを表現可能であることである．[76] からも見られるように，EDLP はこれらの要求を充たすと考えた．

次に本研究がその法的知識に対して行う検証について述べる．本研究の検証は，整合性，特に矛盾の検出を行う．まず検出するべきものとして，規則が持つ暗黙的な矛盾が上げられる．この規則の暗黙的な矛盾とは，規則が互いの帰結に相反する結論を持つ場合のことをさし，論理的には矛盾とはならないが，法的知見から不整合として検出する [9, 37, 81]．これら先行研究はその規範の暗黙的な矛盾をいくつかの方法で検出するようにしている．Bench-Capon の研究では，特定の規則の形に対してそれを矛盾に準するものとして特別にあつかっている．しかし特定の形態に対し矛盾と定義することは，その論理体系を複雑にしてしまう問題がある．Sartor の方法では，規則の前提を陽に与えて相反する帰結を明示的に演繹することでその矛盾を検出している．ただこの場合は与える前提を人間が用意しなければならない問題がある．なお Hage の研究はこの規範相互の暗黙的な矛盾を形式化したものである．そこで本稿では，体系そのものを複雑にすることなくまた，人手のコストを最小限にするため，アブダクションを用いて規則の前提を与える．これによって明示的に前提を与える必要もなく，規則が帰結に持つ結論が相互に矛盾を起こしていないか検証可能となる．

そしてこの矛盾検出において，本稿では対立 (conflict) [26] を導入する．上記研究も含め多くの場合矛盾の定義には矛盾律 '$A \wedge \neg A$' が用いられる．しかしこの定

義は，人間の直観や法の矛盾とはそぐわない．それは矛盾律は名前の一致を求めるからである．これにより人間が矛盾と見なすものの中でも一部しか表現することができない．対立は（3.2.1 章）名前に縛られないより柔軟な矛盾の定義が可能である．よって対立を用いることでより人間の直観にそった矛盾を表現し検出することを試みる．なお対立の導入については，実際の運用上用意するデータが増大する問題があることが分かっている．そこで本稿では，語彙の上位下位概念のデータを用いることでその問題を解決する．

なお，本研究のような法から矛盾を取り除くという立場に対し，法は本来矛盾したものでその不整合な法から演繹が可能であればよいとする研究もある [13, 15, 23, 25, 56, 78, 82, 83]．特にこれらにおいては，矛盾許容論理 (paraconsistence logic) [5, 43, 77] を利用したものが多い．矛盾許容論理の多くは，推論規則から矛盾による推論規則 (ex falso quodlibet) '$\varphi \wedge \neg\varphi \vdash \psi$' (ここで，$\varphi, \psi$ は任意の論理式とする) を取り除いたものである．この規則は，矛盾からは任意の論理式を演繹してよいことを意味するため,「論理的爆発」などと呼ばれる．この規則の除外によって，矛盾許容論理では，$\varphi \wedge \neg\varphi \vdash \varphi$ や $\varphi \wedge \neg\varphi \vdash \neg\varphi$ のように，知識に矛盾が含まれている場合でも，知識と関係のある結論は演繹可能になり，また知識とは関係のない論理式 ψ については，$\varphi \wedge \neg\varphi \nvdash \psi$ となる．上記研究を応用することで，法が整合的でなくとも，論理的爆発をともなわずに演繹を行うことは可能ではある．ただしこの立場を認めると，法は矛盾していても問題がなく，ただ任意の状況に対して処決を得ることさえできればよいということになる．法は特定の状況に対してのみ適応できればよいものではない．共同体における共通認識的な規範知識でなければならない．もし矛盾許容論理を用いて矛盾した法から処決を得ることが可能となったとしても，その結論が対立していては上記で述べたように社会規範としての役割を持たない．よって法では，上記例で矛盾するような結論を持つ規則を取り除かなければならない．ゆえに，不整合的でも法律効果を演繹可能かどうかが重要なのではなく，整合的であるというその事実が法では重要であると考え，法に本来あるべき整合性を放棄せずにそれを保つために検証を行う．なお矛盾許容論理は，ただ矛盾を無視したものに過ぎないという批判もある [26]．

以上をふまえ，本研究の目的は以下にまとめられる．

- 法的知識に対して，矛盾律では検出しにくい人間の直観に近い非整合性の検

出を試みる

- 実装不可能な理論ではなく，現実に運用可能な技術として，その具体的仕様と実装を提案する
- 運用可能性の証明として，現実の法令文に対し提案実装を用いて検証を行う

本論文は以下の構成を持つ．まず 2 章 で，本稿の提案手続きで用いる基礎技術を説明する．3 章 では，2 章 で示した技術を本稿に導入する．4 章では，具体的なデータの形式化および，検証手続きについて述べる．そして，5 章で富山県条例に対して，本稿の検証手続きを適応した結果について述べる．そして最後に，6 章でまとめを述べる．

第 2章

検証に関わる基礎技術

本章では，本稿が用いる基礎技術を説明する．ここで示される定義は，知識の形式化に関する定義と推論に関する定義に分類できる．まず知識の形式化に関わる定義として，法規範・法準則・法原則，EDLP，非循環論理プログラムを説明する．そして次に，推論に関わる定義として，交わり (meet)，対立，アブダクティブ論理プログラム (ALP: abductive logic program)，議論 (argument) について説明をする．

2.1　知識の形式化に関する基礎技術

2.1.1　法令文・法規範・法準則・法原則

本章では，法令文・法規範・法準則・法原則といった，法の実体に関する法令用語について説明する．

法は法令文によって規定される．そしてこの法令文とは文章のことを表す．本研究で法令文を考えるとき，この法令文にはその実態が文書として存在しない場合があることに注意する必要がある．その理由は法令文がメタな法令文によって書き換えられている場合があるからである．このメタ的に書き換えるとは，法令文 A が法令文 B の文面について変更を指示することをいう．具体例として「平成十九年十一月三十日法律第百二十号」を挙げる．平成十九年十一月三十日法律第百二十号の法律で書き換えにかかわる部分は以下である．

銃砲刀剣類所持等取締法及び武器等製造法の一部を改正する法律

第三条

組織的な犯罪の処罰及び犯罪収益の規制等に関する法律（平成十一年法律第百三十六号）の一部を次のように改正する。別表第二第十七号中「第三十一条の二第一号（銃砲以外の武器の無許可製造)」を「第三十一条の三第一号（銃砲及び銃砲弾以外の武器の無許可製造）」に改める。

この法律は法令文「組織的な犯罪の処罰及び犯罪収益の規制等に関する法律（平成十一年法律第百三十六号）の別表第二第十七号」について，文字列「第三十一条の二第一号（銃砲以外の武器の無許可製造)」を「第三十一条の三第一号（銃砲及び銃砲弾以外の武器の無許可製造)」に変更する指示である．このようにメタ的な改正法が存在する場合，法令文の実態はその書き換えを反映したものになる．よってこのようなメタな書き換えを受けた法令文の実際の文は，法学上存在しない．この書き換えを指示した法律を反映することは「溶かし込み」などと呼ばれる．

法の整合性検証はこの溶かし込み後の法令文に対して行う必要がある．なぜならばそれが現実の法を表すからである．よって本研究では以後，この溶かし込み作業後に得られる最終的な文章のことを法令文と呼び，それを法令文における検証の対象とする．なおこのような溶かし込みを自動で行うことを試みている研究もある [64]．しかし現段階では実用に至っておらず，この溶かし込み後の法令文を自動で取得することは，現時点では困難であることを述べておく．

法令文が表す意味のことを法規範という．法規範は法令文が定める法の規則や効果や概念などを一般的にさす．つまり法規範は，法令文を人が読み，そして法解釈を行って発現するものといえる．この法規範は大きく 2 つに分けられる．その 1 つは法原則と呼ばれ，残るもう 1 つは法準則といわれる．

法原則とは法令文を解釈する際にその指針となるように，その法を制定した意図や目的などを概括的に表したものである．これは通常抽象的な内容をもって規定される．例えば以下はその法原則の法令文である少年法の総則である．

少年法

第一章 総則

（この法律の目的）

第一条 この法律は、少年の健全な育成を期し、非行のある少年に対して性格の矯正及び環境の調整に関する保護処分を行うとともに、少年及び少年の福祉を害する成人の刑事事件について特別の措置を講ずることを目的とする。

このように総則は具体的な事例に対して何らかの処決を規定するものではない．その法律の制定理由や，またそのほかの条項において何らかの処置が決定されるとき考慮されるべき抽象的事案を記載しているに過ぎない．したがって，この法原則は明確な規則として解釈することが難しい．

もうひとつの法規範が法準則である．これは具体的状況に対する処決を規定した規則である．同少年法では第二十七条がその法準則として挙げられる．

少年法

（競合する処分の調整）

第二十七条

保護処分の継続中、本人に対して有罪判決が確定したときは、保護処分をした家庭裁判所は、相当と認めるときは、決定をもつて、その保護処分を取り消すことができる。

これは「当該少年について，保護処分が継続中であり，かつ有罪判決が確定したならば，家庭裁判所は保護処分を取り消す」というように，規則として解釈が可能である．つまり法準則は概括的な法原則とは異なり具体的な法ということができる．

このような明確な規則の規定方法を「条件プログラム」などという．この条件プログラムとは，二者択一の適用を前提とした形で規定されるものとされ，厳密には「具体的な事例で問題となる人・物・行為などが，あらかじめ定められた一般的なカテゴリーに属するとき，それらに対して画一的に同じ効果がもたらされる形である」とされる [93]．ここで，条件プログラムの適用について満たすべき条件を「要件」，その結果得られる結論を「法律効果」などと呼ぶ．

2.1.2 EDLP

EDLP の説明の前に，簡単に論理プログラムの説明をする．まず論理プログラムの体系はさまざま存在する．そのもっとも基本的な論理プログラムは，ホーン節 (Horn clause) [41] と呼ばれるものである．ホーン節は主に規則 (rule) で規定される．その規則とは以下の形をとる．

$$head \leftarrow body_0, \ldots, body_n. \tag{2.1}$$

このとき，$head$, $body_i$ $(0 \leq i \leq n,\ i, n \in \mathbb{N})$ はそれぞれアトム[1]である．また‘$\leftarrow$’は左向きの含意，‘,’は論理積，‘.’は規則の終端を意味する．なおこのホーン節は，p, q をそれぞれ $body$, $head$ と表すとすると，以下の論理式とほぼ等価に解釈できる．

$$(p_0 \wedge \ldots \wedge p_n) \rightarrow q \tag{2.2}$$

つまり論理プログラムであるホーン節は，知識表現に使用可能な論理式を上記に限定したものであるといえる．

このように制約の大きい論理プログラムが計算機上の知識表現形式として多く用いられる理由は，論理式を証明する困難さにある．自由な論理式は，その証明が非常に複雑で，また計算量が多い．論理体系として広く知られる古典主義論理では Wang のアルゴリズム [34] によってシーケント計算を用いて決定的かつ効率的な証明方法が示されている．しかしながら論理式を知識集合として用いた証明を行う場合一般的にその論理式は少なくない．自由な論理式の証明では知識の量に対して級数的にその証明の工程が増大する．したがって自由な形を許した大きな知識を用いて演繹を行うことは実用上の問題がある．一方論理プログラムを用いた知識での証明とは，限定的な論理式に対する証明方法である SLD 導出によって行われ，その計算量は自由な論理式の証明に比べ低い．したがって論理体系の表現能力と，処理容易さのバランスを取った場合，計算機上における知識表現には，論理プログラムを用いることが現実的であるといえる．

[1]アトムとは一階述語論理において述語記号，引数の変数で構成される最もプリミティブな論理式，または命題論理における命題変数のことをいう．例えば‘$p(x)$’や‘a’がそれに当たる．

ただし上記のようにホーン節が持つ制約は非常に強く，その表現力は低い．また昨今の計算機の性能向上，アルゴリズムの改良により，論理体系の制限を緩和した論理プログラムの体系がいくつか存在する．そのひとつが EDLP である．その EDLP は以下のように定義される．

定義 1 EDLP

EDLP は，以下の規則の集合として定義される．

$$L_0|\ldots|L_i \leftarrow L_{i+1},\ldots,L_j,notL_{j+1},\ldots,notL_k$$

ここで L_n $(0 \leq n \leq k; i \leq j \leq k)$ はリテラル[2]であり，‘$\leftarrow$’ より左側をヘッド部，右側をボディ部という．ヘッド部の ‘|’ は論理和を，ボディ部の ‘,’ は論理積を，‘$\leftarrow$’ は含意を，‘$\neg$’ と ‘*not*’ は否定を表す．この 2 つの否定は ’$\neg$’ が論理否定を表し，‘*not*’ が 失敗による否定 (NAF: Negation as Failure) を表す．なお各々の記号は論理プログラムでの記号であることに注意する．

EDLP の特徴は 2 つある．ひとつはヘッド部に論理和が使用できることである．そしてもうひとつは，2 つの否定を持つ [57] ことである．

ヘッド部における論理和の使用は大きな表現力の拡張となる．ホーン節はヘッド部が高々1 つのリテラルからなるため，高々1 つの正リテラルを持つ節がリテラルの選言標準系と等価である．EDLP ではヘッド部に論理和が許されるので，任意個数の正リテラルを持つという大きな表現能力の拡大がされている．

つぎの特徴として EDLP では 2 つの否定を使用することができる．この 2 つの否定とは上記の論理否定と NAF である．論理否定とは，真偽値を反転させる単項演算子 ‘$\neg$’ を意味する．もう 1 つの否定の失敗による否定は，論理プログラムの特徴的な否定演算子で，一般に ‘*not*’ と記述される．このとき，‘$\varphi \leftarrow not\,\neg\varphi$’, ‘$\neg\varphi \leftarrow not\,\varphi$’ というこれらの規則を採用するかどうかでこれら否定の取り扱いは異なる．これを採用すると NAF は論理否定と等価になり，また論理プログラムに記述されていない事実は全て論理否定の扱いになる（closed world assumption）．

このような EDLP の意味論には安定モデル意味論 (stable model semantics) [30] が用いられる．安定モデルとは，ある論理プログラムの集合に対して常に証明可

[2]リテラルとは，アトムか，または論理否定をともなったアトムをさす．

能な事実の集合をいう．安定モデル意味論では命題の証明について，その証明する命題が与えられた論理プログラムの安定モデルに含まれているかどうかで判別する．

2.1.3 非循環論理プログラム

非循環論理プログラム [45] は，論理プログラムの内部について循環した規則がないように制限するものである．循環した規則とは，論理プログラムの実行順序を考えると分かる．論理プログラムはヘッド部からボディ部へという流れで実行される．ボディ部のある節へ到達すると，またその節をヘッド部に持つ規則が検索され，見つかればその規則が同様にして実行されることになる．循環とはこのような実行の順序において閉路を形成する部分のことをいう．

ただし，その閉路は全ての場合において同一の性質を持つわけではない．例えばある閉路の一部は，否定演算子をともなった節によって形成される場合もある．また正リテラルの呼び出しによって全て構成されている場合もある．論理プログラムの実行に際しその各々について条件によっては循環しても許される場合もある．よってそれらの場合も考慮した循環論理プログラムは以下のように定義されている．

定義 2 非循環論理プログラム
今以下の拡張論理プログラムを考える．

$$A_1|\ldots|A_k|not\, B_1|\ldots|B_l \leftarrow C_1,\ldots,C_m, not\, D_1,\ldots,not\, D_n.$$

ここで A_i, B_j, C_s, $D_t(k,l,m,n,i,j,s,t>0,\ i\leq k,\ j\leq l,\ s\leq m,\ t\leq t)$ はリテラルとする．また合わせて関数 $l : Lit \rightarrow \mathcal{N}$ を与える．論理プログラムの集合 Π に出現するリテラルの集合を Lit としたとき，全ての $L \in Lit$ について，その呼び出しの深さに関し自然数が割り当てられているとする．これを「レベル」と呼ぶ．関数 l は引数のリテラルのレベルを返す関数である．

このとき以下の条件により，論理プログラム Π は 3 つに分類される．

1. 全ての i,s について $l(A_i) > l(C_s)$ であるとき，Π は「正非循環」という．そうでない場合を「正循環」という．

2. 全ての i, j, s について $l(A_i) > l(B_j)$ であり，かつ $l(A_i) \geq l(C_s)$ であるとき，Π は「負非循環」という．そうでない場合を「負循環」という．

3. Π が (item:1), (item:2) を充たすとき，Π を「非循環」という．

つまり簡単に非循環論理プログラムであるとは，ある演繹において一度呼び出された規則へ到達可能な規則が再び呼び出されない，ということを表している．この性質は法的知識において重要な意味を持つ．

2.2 推論に関する基礎技術

2.2.1 対立

論理学における否定は一般的に a を命題変数または論理式としたとき '$\neg a$' で表される．この否定は論理否定と呼ばれる．論理否定は意味論上命題変数の真偽値を反転させる作用を持つ．そして構文上，論理的否定は明確な対象の否定を表現することが分かる．この性質は数学上は問題ないが，自然言語のような対象を形式化する場合，その定義の厳密性が表現力を制限している．

特にその表現力の限界が顕著に表れるものが矛盾の定義である．矛盾の定義である矛盾律は，否定演算子を用いて $\neg A \wedge A \equiv \bot$ と定義される．この定義は，使用している否定が明示的な定義であることから限定的な対象にのみ作用し，それ以外を矛盾とは認めないという利点と欠点を持ち合わせる．つまりこの場合 'A' における正負の組でなければ矛盾にはならない．人の議論やまた自然言語を形式化するためには，この矛盾律では厳密過ぎる．

上記否定演算子は自然言語で「〜でない」に相当する．つまり矛盾律に相当するような自然言語とは,「車でありかつ車でない」のようなものとなる．このような文を矛盾として判断することが妥当であることに疑いはない．しかし自然言語は多様な形を持ちその範囲を容易に越える．例えば「過失でありかつ故意である」などはその典型である．人はこの例が矛盾を含んでいる判断できる．しかし上記の厳密な矛盾律の定義にしたがうならば，この矛盾は矛盾する論理式として表現できない．

これに対して「対立」は，特定の名前の組に対して矛盾を定義するものである．これは矛盾律の定義から否定演算子の厳密さを緩和したものといえる．そのため否定の対象を構文によって固定されることがなく，より自由な対立構造の定義が可能となる．その対立は以下のように定義される．

定義 3 対立
$\perp$ を矛盾，また α, β を命題変数とする．このとき α と β の対立を，$\vdash (\alpha \wedge \beta) \rightarrow \perp$ と書く．またこの α と β を対立概念という．

これは制約論理プログラミング [50] の制約 (constraint) に似たものである．制約は一般に以下のように表される．

$$\perp \leftarrow C_1, \ldots, C_i, \ldots, C_n.$$

この C_i はその制約条件部分である．この制約は主に演繹する値の範囲を制限するために使用される．顕著な例は制約論理プログラムの枠組みを取り入れたアブダクションなどが挙げられる [**?**]．この場合の制約は，得られる仮説を制限する．これから制約は矛盾の定義とは意味が異なることが分かる．これに対して対立は，例えば定義 3について，$\alpha = \neg\beta$ と考えた場合，その定義が矛盾律を内包していることが分かる．本稿ではあくまで矛盾律を基礎として矛盾を拡張する．

2.2.2 交わり

順序関係の定められた集合に対しては演算が定義される．その一つに，「交わり」[16] がある．その交わりは以下の「下界」と「下限」の定義を用いて定義される．なお，それぞれの定義で用いられる順序集合 $\langle \Gamma, \leq \rangle$ の順序関係 $\leq$ は，反射律，推移律，反対称律（定義**??**，**??**，**??**）を満たすものとする．

定義 4 下界 (lower bound)
$\langle \Gamma, \leq \rangle$ を順序集合，Δ を Γ の部分集合とする．Δ の任意の元 δ に対して，$\gamma \leq \delta$ を満たす $\gamma \in \Gamma$ の集合を Δ の「下界」という．

定義 5 最大元 (maximum)
$\langle \Gamma, \leq \rangle$ を順序集合，Δ を Γ の部分集合とする．この Δ のある元 δ が，Δ の任意の元 δ' に対して $\delta' \leq \delta$ を満たすとき，その δ を Δ の「最大元」という．

定義 6 下限 (infimum)
$\langle \Gamma, \leq \rangle$ を順序集合，Δ を Γ の部分集合，Σ を Δ の下界とする．この Σ が最大元 σ を持つとき，σ を，Δ の「下限」という．

以上の定義（定義 4，5，6）を用いて，交わりは以下のように定義される．

定義 7 交わり (meet)
$\langle \Gamma, \leq \rangle$ を順序集合，Δ は $\Delta = \{\alpha, \beta\}$ で，$\alpha, \beta \in \Gamma$ であるとする．このとき，Δ に下限 χ が存在するならば，χ を α, β の「交わり」という．またそれを，$\alpha \sqcap \beta = \chi$ と表す.

交わりの演算例を図 2.1に挙げる．

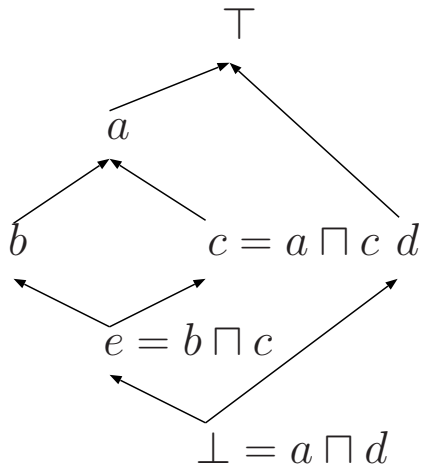


図 2.1: 交わりの例

図 2.1は，順序集合

$$
\begin{aligned}
& \langle A, \leq \rangle \\
A &= \{a, b, c, d, e, \top, \bot\} \\
\leq &= \{{}_aR_\top, {}_dR_\top, {}_bR_a, {}_cR_a, {}_eR_b, {}_eR_c, {}_\bot R_e, {}_\bot R_d\}
\end{aligned}
$$

を表す．たとえば，この順序集合において，$a \sqcap c$ を考える．a 以下の要素は $\{a, b, c, e, \bot\}$ であり，c 以下の要素は $\{c, e, \bot\}$ である．これをそれぞれ δ_a, δ_c とす

る．A における $\{a, c\}$ の下界は，$\delta_a \cap \delta_c = \{c, e, \bot\}$ となる．この集合においては，$\bot \leq c$, $e \leq c$, $c \leq c$ が成り立つ．よって，A における $\{a, c\}$ の下限は c であり，$a \sqcap c = c$ である．

2.2.3 アブダクション・ALP

推論は大きく分けて 3 種類存在する．それは演繹推論 (deduction)，帰納推論 (induction)，そしてアブダクション (abduction) である．演繹推論はある知識集合から帰結を導くことである．例えば知識集合が $\Delta = \{\varphi, \varphi \rightarrow \psi\}$ のとき，$\Delta \vdash \psi$ の ψ を導くことである．一方帰納推論は，事実の集合が存在しそこからある事実が推論されたとき，その結果をもたらす規則を推論することである．上記の例にならうと，$\varphi \rightarrow \psi$ を導くことになる．そしてこのアブダクションは結論を導くために必要な事実を仮定する枠組みである [53]．言い換えるならば，演繹推論の逆向きの推論ともいえる．たとえばそれは，$\Delta = \{\varphi \rightarrow \psi\}$ に対して，ψ を得るにはどの事実が必要なのかを推論することにあたる．この例について仮定可能な事実の集合 '$\{\ldots, \varphi, \ldots\}$' が定義されていたならば，そこから φ を選び出すことを意味する．

このアブダクションに用いられる論理プログラムの枠組みを「ALP」と呼ぶ [54]．一般に，アブダクションは以下のように定義される．

定義 8 アブダクション

ALP $\langle P, A \rangle$ において，P を論理プログラムの集合，A を空でない原子論理式の集合とする．このとき，要求 α に対し

$$P \cup H \vdash \alpha$$

$P \cup H$ is consistent.

$$H \subseteq A$$

を満たすような仮説の集合 H を求めることを，アブダクションという．

このアブダクションを用いるとき課題になることは仮説の選択である．アブダクションでは上記定義に基づき，要求に対して必要な仮説の集合が仮説可能な集

合（定義内では A にあたる）の部分集合として選ばれる．このとき要求を満たす仮説集合は一般には一意に決まらない．例えば $\langle P, A\rangle$ について $P = \{\varphi \to \psi\}$, $A = \{\varphi, \chi\}$ としたとき，要求 ψ を満たす仮説集合は $\{\varphi\}, \{\varphi, \chi\}$ のふたつを考えることが可能である．アブダクションではどのようにして仮説を得るかも重要であるが，どの仮説を採用するかが重要な課題となる．これらのことに関しては多くの研究が存在する [20, 46–48, 51, 80, 87].

また ALP には制約論理プログラムを取り入れた枠組み ACLP(abductive constraint logic program) もあり，その枠組みは $\langle P, A, IC\rangle$ という組で定義される．このとき IC は制約の集合で，一貫性制約 (integrity constraint) と呼ばれる．ACLP では上記定義のように仮説集合を得る場合，IC に反しない範囲で仮説を得る．これは以下のように定義される．

定義 9 発想制約論理プログラミング (ACLP: abductive constraint logic program)
ACLP $\langle P, A, IC\rangle$ があるとする．ここで，P を論理プログラムの集合，A を空でない原子論理式の集合，IC を制約とする．このとき Q を要求としたとき，

$P \cup H \vdash \alpha$

$P \cup H \vdash IC$

$P \cup H$ is consistent.

$H \subseteq A$

を充たす H を選び出す.

2.2.4 議論

議論 [71] はある結論に対する演繹の過程を形式化するものである．人間はある結論を得るためには立論を行う．それは与えられた事実に適用できる規則を選び出し，目的とする事象に到達させることを意味する．論争のように結論を証明できるかどうかよりどのようにしてその結論を得たのかが重要である場合，この立論を議論によって形式化する必要がある．

議論が用いられる顕著な例は法的論争の研究である．法的論争では立論とそれに対する反駁がどのようになされているかが研究される．そのため論争の経過を

形式化するためにこの議論が用いられることが多い．具体的に法的論争では，あるエージェント（便宜上ここでは，発言者をエージェントと一般化して呼称する）が生成した立論に対し，その結論を無効化するために他のエージェントがその立論を否定するよう別の立論を生成する．そしてこれが繰り返されて互いに自分の主張が保全されるように論争を繰り返す．これからもどのように結論を得たのか，そしてその結論を得るまでに用いられる立論はどのようなものかなどが注視され，議論による形式化が有用に働く．本研究ではこの立論の対立を，法的知識に含まれる矛盾点の検出に対して用いる．適用の詳細は 3.2.4 章で述べる．

本研究で用いる議論を，以下のように定義する．

定義 10 議論

P を論理プログラムの集合，F を事実の集合，φ を結論とする．このとき議論 $Arg_{\varphi} = \langle F, P, \varphi \rangle$ は，以下の条件を満たす．

$$F \cup P \vdash \varphi \tag{2.3}$$

$$F \cup P \nvdash \bot \tag{2.4}$$

$$\forall \psi \in (F \cup P), (F \cup P) \setminus \{\psi\} \nvdash \varphi \tag{2.5}$$

この $F \cup P$ は，φ の支持 (support) と呼ばれる．

この定義は議論 $Arg_{\varphi} = \langle F, P, \varphi \rangle$ があるとき，その支持 $F \cup P$ は φ を演繹するための最小限の集合であることを意味する．またその支持は矛盾してはならない．

この定義を用いると，ある議論は以下のように表される．$Arg = \langle F, P, \varphi \rangle$ を議論とし，$F = \{\psi\}$, $P = \{\psi \to \chi, \chi \to \varphi\}$ とする．このとき結論 φ を支えている支持は $F \cup P = \{\psi, \psi \to \chi, \chi \to \varphi\}$ となる．

第 3章

検証手続きへの諸基礎技術の導入

本章では2章の定義を本稿の検証手続きに組み込む意義を述べる．

3.1　知識表現における基礎技術の適用

まず検証対象の法的知識の表現に対して関連研究，法準則，EDLP，非循環論理プログラムを導入する．

3.1.1　法準則

法令文は自然言語で記述される．自然言語はわれわれが持つ最大の表現力を持った記述形式である．したがって，社会の規範を定める法を自然言語で記述することはごく自然であるといえる．しかし自然言語は思想を記述できるほどに自由であることが検証の妨げにもなる．本稿における検証とは，形式的な手法を用いて法的知識を検査し，また不具合が含まれているならばその原因箇所を明らかにする作業である．したがってその検証対象は形式的に取りあつかうことができなければならない．自然言語は曖昧な表現が可能であり，その解釈には人間が持つ能力が必要である．自然言語の法令文をそのまま検証することは現時点では不可能に近い．よって法的知識の検証には，法令文を形式化し，法的知識へと変換する作業が必要となる．

これから検証可能な法は形式化可能な法の要素に限られる．ある要素が形式化可能かどうかは，その際に用いる形式化手法の表現能力に依存するところもあるが，根本的には対象の情報が論理的で明確であるかどうかが最も重要な点である．論理的な構造を持てば規則として記述できる可能性がある．

2.1.1 章で述べたように，法の情報には法原則と法準則がある．このうち法原則は概括的な内容であるため，その形式化が難しい．また仮に形式化した場合でも，形式化とは曖昧性の排除という性質も持つため，法原則が持つ概括性を失うことにもなる．すなわち，現段階では法原則は形式化困難な上，無理に形式化をしても法原則の概括性が失われる．よって法原則は，検証の対象とせず法的知識には含めないこととする．

そこにおいて法準則は規則として記述が可能である．法準則は 2.1.1 章で示したように条件プログラムとして解釈できる．つまり法の情報の中でも形式化されることに向いた情報であるといえる．

以上から本研究が検証対象とする法的知識は，法令文から取り出した法準則を形式化した知識とする．

3.1.2 EDLP

本稿では法準則の条件プログラムを EDLP で形式化する．そのためまず法準則の条件プログラムがどのような構造を持つのか明らかにする．

条件プログラムは [93] の定義を参考に，大まかに以下のような形として表すことができる．

定義 11 法準則の条件プログラム

$$\text{法律効果}_1 | \ldots | \text{法律効果}_m \leftarrow \text{要件}_1, \ldots, \text{要件}_n \tag{3.1}$$

このとき '|' は論理和を表すものとする．ここで法律効果の部分が論理和 '|' によって分割されるのは，法準則の法律効果には選択が存在するからである．法準則の典型的な例である「刑法 199 条」を見るとそのことは明らかである．

刑法 第二十六章 殺人の罪

第 199 条（殺人）

人を殺した者は、死刑又は無期若しくは五年以上の懲役に処する。

この刑法 199 条は以下のような条件プログラムに解釈できる．

死刑 | 無期懲役 | 懲役 5 年以上 ← 人を殺した者 (3.2)

ここでは法令用語「又は」を‘|’で置き換えている．ここで表現された法律効果の選択とは,「死刑」,「無期懲役」，そして「五年以上の懲役」この 3 つである．これが選択であることは明らかであり個別に分ける必要がある．なぜならば「死刑又は無期若しくは五年以上の懲役に処する」という法律効果の部分が一つの命題であることはあり得ない．実際に被適用者はそのいずれかひとつの法律効果を受容している．

これから法律効果は選択的結論を有する．よって条件プログラムを形式化するためには論理プログラムに法律効果の部分，すなわち論理プログラムのヘッド部において「又は（若しくは)」という**選択**を表現できることが最低限求められる．

EDLP は定義 1，11からも分かるように，上記の要求を充たす．さらに EDLP は条件プログラムの論理構造を表現する以上の不必要な複雑さを持たない．この複雑さとは，特異な演算子や場合によって特別視する必要のある携帯を必要とせず，また語彙の様相などの表現粒度を小さくさせる構造などを持たないということを意味する．特異な演算子は計算のアルゴリズムを複雑にする．また表現粒度が小さくなりすぎると，述語記号を不必要なほど複雑な形式で記述する必要が生じる上，実どこまで細かく記述すればよいのかという問題も生じる．本稿が検証の対象とするのは，規則の論理構造であり，対立を基とした不整合の検出である．以上のことから，法準則の論理的構造に関して検証するに際しこの EDLP を形式化手法として用いることは適切であると考え，採用した．

3.1.3　非循環論理プログラム

法的知識は論理プログラムの側面から，そして法の観点からも非循環（定義 2）であることが望ましい．それは，法的知識から得られた結論には法的根拠が求められることが理由になる．法的根拠とは得られた法律効果に対して説明可能なことを表し，またその説明に用いられている事実が現実に得られた事実及び法令文内部で定義された用語に基づいた事柄であることである．法的知識に循環部分が存在するとこれらの法的根拠が得られない恐れがある．

例えば以下の酒税法の例を考える．

> 酒税法
> （納税義務者）
> 第六条 酒類の製造者は、その製造場から移出した酒類につき、酒税を納める義務がある。

この法令文は，以下のような論理プログラムで考えることができる．

$$\text{納める}\,(x,w,y,z) \leftarrow$$
$$\quad \text{酒類}\,(y), \text{製造者}\,(x), \text{製造場}\,(w), \text{酒税}\,(z), \text{製造}\,(x,y,w), \text{移出}\,(y,w).$$

この論理プログラムは「製造者 x が，製造場 w で製造した酒 y を，製造場 w から移出した場合，製造者 x は酒税 z を納める」と解釈できる．このとき仮に結論として「製造者 a は製造場 b で製造した酒 c について酒税 d を納める」（納める`(a,b,c,d)`）が得られた場合，その法的根拠は上記論理プログラムを逆向きにたどり得られる．それは，“酒類`(c)`，製造者`(a)`，製造場`(b)`，移出`(c,b)`，酒税`(d)`，製造`(a,c,b)`” となる．仮に上記酒税法の法準則に加えて，以下のような規則が非循環部分として法的知識内に存在する場合を考える．

$$\text{製造者}\,(x) \leftarrow \text{納税者}\,(x).$$
$$\text{納税者}\,(x) \leftarrow \text{製造者}\,(x).$$

この場合，規則が循環しているため「製造者」の意味が決定できない．結果的にこの場合「製造者」を根拠として得ることができなくなり，納める`(a,b,c,d)`

の法的根拠が決定できなくなる．このように法的観点から法的知識は非循環であること求められる．

また論理プログラムの観点から見てもこの循環は，特別な処理を行わない限り実行時に無限ループを誘発する知識集合であるといえる．無限ループは論理プログラムの処理上制御することは可能ではあるが，その存在が好ましいとはいえない．

以上のことから本稿の法的知識は非循環に制限する．そのため対立に関する整合性検証の前に，前処理として循環部分の検出を行う．

3.2 検証工程における基礎技術の組み込み

本節では矛盾の検証に対して関連研究，対立，交わり，アブダクション，議論を導入する．

3.2.1 対立

通常の場合，矛盾の定義には矛盾律が用いられる．法的知識の矛盾検出に関わるいくつかの研究 [9, 38] でもそれは例外ではない．矛盾律によって矛盾がなりたつためには，定義上論理式 A と $\neg A$ において A が一致しなければならない．

2.2.1 章で提示した対立（定義 3）は名前の組に対して個々に矛盾を定義するものである．これには矛盾律にある名前の一致という構文的な制約がない．よって対立を用いると，矛盾律を内包しつつも矛盾律よりも広い矛盾の定義が可能である．したがって，この対立を矛盾の定義の基礎とすることで，矛盾律で矛盾とできないものを特別扱いすることなく統一的に法的知識の矛盾として表現できる．

矛盾律では表現できないが対立を用いることで表現可能になる対象は，論理式 ‘$A \wedge B$’ において，述語 A, B の述語名，もしくは命題の名前が以下の 3 つの場合 [58] のいずれかの場合である．なお矛盾の基本的な要素である否定の種類にはここで取り扱うもの以外にも多く考えられているが [57]，現時点は観念的なものは取り置き形式化可能性がある語彙として明示的な否定関係をその検証の対象としている．

以下はその対立の採用により表現可能になる矛盾の関係である．

1. 対義語の組である

2. 否定辞がついたものと，それから否定辞を除いたものの組である

3. 排他的な関係にある組である

具体例はそれぞれ以下のようなものになる．

1. 故意 $(a) \wedge$ 過失 (a)（対義語）

2. 可能 $(a) \wedge$ 不可能 (a)（否定辞）

3. 人 $(a) \wedge$ 車 (a)（排他的関係）

これらの例はいずれも述語名が異なるため矛盾律上では矛盾ではない．しかし併記したそれぞれの理由からも法的知識としては矛盾しているとみなすべきである．

ただしこれら 3 つの場合に対しては，以下の手法をとることで矛盾律を用いて矛盾を導く方法も考えることは可能である．

1. 対義語の組の一方を，もう片方の否定として表現する方法

2. 否定辞を否定記号に置き換える方法

3. 付値関数で排他的な関係を定義する方法

これによると，例 (1), (2) は以下に書き換えることになる．

(1):故意 $(a) \wedge \neg$故意 (a)

(2):可能 $(a) \wedge \neg$可能 (a)

これらの手法は法的推論では適切でない．まずこの対義語，否定辞の組をそれぞれお互いの否定として表現する方法について説明する．これは恣意的な表現方法であることが問題である．なぜなら「過失」の対義語は「故意」であるが，‘過失 (x)’ を ‘¬故意 (x)’ と記述することは述語名「過失」の意味を欠落させた表現である．またこの記述は「故意でないなら過失である」とすることと等しい．故意でなく過失のない場合も存在する．つまり過失と故意は完全に補の関係にあるものではない．したがってこの方法は法的知識における表現方法としては適切でない．次

に付値関数を割りあてる方法について述べる．一般に一階述語論理における対象の集合は加算無限の集合である．したがって，対象それぞれに付値を割り当てる方法が適切とは考えにくい．また‘$\forall x[\text{人}(x) \wedge \text{車}(x) \rightarrow \bot]$’と別途一般的に排他的関係を定義することは，対立を導入することと同義となる．このように矛盾律だけでは法的知識の矛盾を適切に表現できないか，またそれが適切な方法とはいえない．

以上から本稿では，矛盾律の代わりに対立を用い，対義語の組，否定辞をともなったものと否定辞を取り払ったもの，排他的関係にあるもの，それぞれを対立上の「対立概念」として統一的に定義する．これにより恣意的に名前を変えることなく矛盾することを定義できるようになる．

3.2.2　交わり

対立を導入するためにはその基準である「対立概念」の集合を定義しておかなければならない．対立概念とは「ある概念名 A と B はともになりたつことはない」ということを表す組のことである（定義 3）．ある知識集合に対してこの対立概念を厳密に定義するためには，その知識集合に出現する命題または述語名の全ての組み合わせに対して，それらが対立概念なのかどうかを人が判別し，記述しておく必要がある．知識集合が小さい場合このことは問題として顕在化しない．しかしある程度の大きさをもった実際の法をあつかう場合，その全ての組み合わせを人手によって判断することは非常に手間がかかり，現実的，実用的な方法とはいえない．本稿が使用した富山県の例でもそれは明らかで，使用した法令文が条例全体の一部でありながら，実験で 30000 以上の組み合わせが検出された．これを人手でそれぞれ判別，記述するのは現実的な方法ではない．

そこで本稿では，語彙の上位下位関係から順序集合に対する演算である「交わり」を用いて対立概念を抽出する方法を提案する．これにより最小限の手間で対立概念を定義することが可能になる．厳密な計算手続きは実装詳細と深く関わるため 4.4.1 章で述べる．

この対立概念は，計算で得ることからその精度に問題が残ることも分かっている．つまりこの計算によって抽出される対立概念は，必ずしも「対立している」と

は限らない．その理由は上位下位概念のデータそのものが知識として完全ではないことが理由として挙げられる．例えば対立していないのに対立概念と誤って認識される場合がある．これは上位下位関係において，下位概念の定義が不足していると考えられる．もしもある概念の組に対して共通下位概念が存在していた場合，それらは交わりの計算上対立概念とはならない．逆に対立概念でありながら対立概念として計算されない場合は間違った下位概念が定義されていると考えられる．

また上位下位関係を記述する言語の表現能力の限界による情報の不完全性もその精度に関係している．現在 OWL や記述論理（description logic）などで盛んに研究されているオントロジー研究では，概念それぞれに「rigid である」などの概念属性を与えて分類するなどの方法が検討されている [91]．この rigid とは，不動的概念か，動的概念かを表す属性である．例えば「学生」という概念は時間的に変動する動的概念で,「人」という概念は不動的概念にあたる．法的知識に関するオントロジーでは物理的か観念的かなどの属性も用いられる．このような厳密な定義を持たない形式で上位下位概念の定義を行った場合，概念のある様相における混同が発生する可能性がある．例えば法律上「法人」は「人」として認識される．また会社組織は「法人」である．したがって単純には 会社 $<$ 法人, 法人 $<$ 人（ここで‘$<$’は上位下位関係を表す）のような概念階層を作ることができるのであるが，この順序関係では計算上「会社」と「人」を対立概念とは見なすことができない．しかし会社と人に関しては一般には対立する概念として扱うべきである．このような点を解決するには，上記のような概念属性を追加し，より厳密な知識を上位下位関係語彙データに記述する必要などがあると考える．

以上のように，交わりによって抽出される対立概念の精度はどれだけ厳密に上位下位概念を作成するのかということとなる．ただし，この精度の高い対立概念を得るために上位下位関係語彙データを厳密に記述する手間は，対立概念の組み合わせを人手で判別する手間よりも少なければならない．より少ない手間でより精度の高い対立概念を抽出する方法は今後の課題である．

3.2.3 アブダクション

本稿の矛盾は対立と対立概念を基に定義されるが，その矛盾を法的知識から検出するにはアブダクションを導入する必要がある．それは法的知識が主に規則から構成されていることと，規則の結論が矛盾するとき法は矛盾しているとみなすべきであることが理由である [9,81].

整合的な法では同じ要件から対立する法律効果を導いてはならない．例えばそれは‘$\{A \to B, A \to C\}$’のような規則があるとき，B, C が対立してはならないことを意味する．もしこの B, C が対立していた場合，法的知識 ‘$\{A \to B, A \to C\}$’は整合的でない法であるとして検出しなければならない.

しかし通常の推論方法では上記の規則が矛盾を起こしているかどうかを判断できない．含意が結論により矛盾を起こすには，その前件が充たされ，後件が明示的に導き出された上で矛盾を起こしている必要がある．これは上記例でいうならば，B, C が対立概念であるとき， A が事実として与えられていなければ対立を起こしているとみなせないことを意味する．したがって法の整合性を検証するためには，この規則の前件 A を仮定する必要がある.

そこで本稿では定義 8のアブダクションを導入する．アブダクションは上記のような場合に要求を B と C とし，それぞれの立論に必要な仮説 A を得る推論である．すなわちこの A を検証時に仮説として得ることで，規則が構成する対立を検出できるようになる．よって検証手続きでは，法準則の要件（前件）をアブダクションで仮定し，その規則の結論が矛盾していないかを検査する.

なお ALP のひとつである ACLP（定義 9）を用いることで，知識が矛盾を含むかどうかに関しては一般化[1]できることも言及しておく．それは，以下の対応をとることによる.

- 対立の対立概念集合を，一貫性制約とする
- 仮定可能事実を，知識集合内でヘッド部に出現しない述語とする
- 要求に対して仮説が得られないときを，矛盾とする

[1]一般化とは本稿の方法論が実現可能であることを意味しない．形式的に表現可能であるという意味で用いている.

ただしACLPをそのまま検証に採用するには以下の2点ついて問題がある．ひとつは本稿の目的が知識集合に矛盾があるかどうかを判別することではなく，矛盾があるときその原因箇所がどこにあるのかを提示するということである．ACLPの一貫性制約は仮説の範囲を制限するための役割を果たす．仮説が得られない場合知識が矛盾を含むと判別可能でも，どの一貫性制約が原因で仮説を得られなかったかは判別できない．ACLP上でこれを実現するには原因箇所を特定するため別途手続きを導入する必要が生じ，それは本稿が導入する各々の技術を用いることと変わらない．もうひとつは実行効率の面である．ACLPの場合，定義上一つの仮説を得るためにはそれぞれの要求に対して全ての一貫性制約を充たしているのか判別する必要がある．本稿の場合，ACLPの一貫性制約に対応する対立概念の集合は小さいものではない．実験では10000組[2]近くがそれであった．実行環境が十分であるならば問題にはなりにくいが，構造として含むには効率上好ましいものではない．

3.2.4 議論

本稿の検証の目的は矛盾の原因をユーザに提示することである．対立を引き起こす規則の集合はすなわち，法が整合性を損なう法令文の箇所であるといえる．それをユーザへ提示することで初めて迅速に法令文の確認すべき場所にたどり着くことができるようになる．

3.2.3章でアブダクションを導入したことにより，対立概念B, Cにおいて$\{ A \rightarrow B, A \rightarrow C \}$のような知識集合から対立を検出することは可能になった．しかしアブダクションは対立の原因箇所を特定するものではない．それはアブダクションが上記例において規則 $A \rightarrow B$ と $A \rightarrow C$ が対立の原因箇所であると特定するものではないからである．

提示部分である対立の原因箇所は，議論により「対立する議論」[75]という形で形式化できる．それは，まず対立概念φ, ψを仮定する．このときそれぞれに対する議論 $Arg_\varphi = \langle F_\varphi, P_\varphi, \varphi \rangle$, $Arg_\psi = \langle F_\varphi, P_\psi, \psi \rangle$ がアブダクションにより得たF_φによって得られたとする．なお，議論 Arg_φ と Arg_ψ において F_φ が共通している

[2] これは上位下位概念から計算された対立概念数である．

のは,「整合的な法は同じ要件から対立する法律効果を導いてはならない」ということを検証するためである．これから対立する議論 Arg_{φ}, Arg_{ψ} が，その P_{φ} と P_{ψ} によりその法的知識内において対立を引き起こしているとできる．さらにこの F_{φ} は，要求を φ として得られる仮説であることから，P_{φ} と P_{ψ} について対立の原因箇所は，仮説を得るために使用されてはいなかった P_{ψ} とできる．よって，P_{ψ} に含まれる規則が再検討するべき法の箇所の候補とでき，ユーザへ提示する箇所の候補となりえる．

第 4章

法的知識の検証手続きと実装

本章では本稿が提案する検証手続きの実装について詳細を説明する．まず見通しを良くするために処理手順の概要を示す．そして本実装が用いる各データの生成方法および仕様に関して述べる．その後それらのデータを使用した具体的な検証手続きを述べる．

4.1 検証手続きと実装の概要

本実装は前処理器と検証器に分けられる．前処理器の役割は入力されたデータから検証器が必要とするデータを抽出し，またそれを検証器に合わせた記述に変換することである．この入力データとは，図 4.1に示されるように，検証対象となる法準則のデータ，及び対立概念抽出のための上位下位概念語彙データ，これら 2つである．この入力データはいずれも XML で記述している．これらの入力データを処理する前処理器は Ruby [1] で実装されている．この言語の選択理由は Ruby がマルチバイト文字の文字列処理が容易なことに加え，オブジェクト指向言語であること，軽量言語であり実装コストが低いこと，そして XML 関連のライブラリが充実していることなどが理由に挙げられる．これら入力データからは前処理器により検証に必要なデータが抽出，変換される．そこで生成される中間データは図 4.1に示されるように，原文参照データ，検証用法準則データ，アリティデータ，語彙グラフデータ，補助データ，の 5つである．いずれも，検証器に合わせて，Prolog の構文で出力される．検証器は上記の前処理器によって変換，及び生成さ

れたデータを元に，推論を必要とする部分，主に法的知識の検証を行う．それは語彙グラフデータとアリティデータを用いて検出する矛盾のために対立概念を抽出し，そして補助データ，検証用法準則データに対してその矛盾の検証を行う．このように検証器では，代数演算および矛盾の検証などの，推論的処理を行う．よって推論処理の実装に適している Prolog を実装言語とした．また処理系としては広く知られる *SWI-Prolog* [2] を使用した．

システム全体の行う処理概要は以下のようになる．以後の節でこれら概要の項目についてその仕様などの詳細を説明していく．

1. 「法準則データ」は，法準則を EDLP で形式化したものを XML で記述したものである．この法準則データを検証器用に合わせた記述形式の「検証用法準則データ」へと変換する．

2. 「上位下位関係語彙データ」は法令文に出現する語彙の上位下位関係を XML で記述したものである．この上位下位関係語彙データから 2 種類のデータを取り出す．ひとつめは検証用法準則知識を補う「補助データ」，そしてもうひとつは語彙の上位下位関係をグラフデータへと変換した「語彙グラフデータ」である．

3. (2) の語彙グラフデータから「対立概念データ」を取り出す．

4. (1) の検証用法準則データと，(2) の補助データ，これらを合わせたものを「検証用法的知識データ」と呼ぶ．この検証用法的知識データが非循環であるか検査をする．そして循環箇所が検出された場合，その結果を「循環部分検出結果」として出力する．

5. (3) の対立概念データを用い，検証用法的知識データが対立を起こしていないか検査する．そして対立が検出された場合，その箇所を「対立部分検出結果」として出力する．

なお，上記概要においては，処理概要項目 (1)–(2) を，前処理器が担当する．残りの，処理概要項目 (3)–(5) は，検証器が担当する．上記概略の各項目では，各処理項目で主に利用されるデータのみ示している．よって，以後の対応する節での詳

細な説明では，当概要の項目に現れていないデータが使用されることもある．データの完全な種類，それを生成するプログラム，そしてその依存関係は図 4.1に示される．

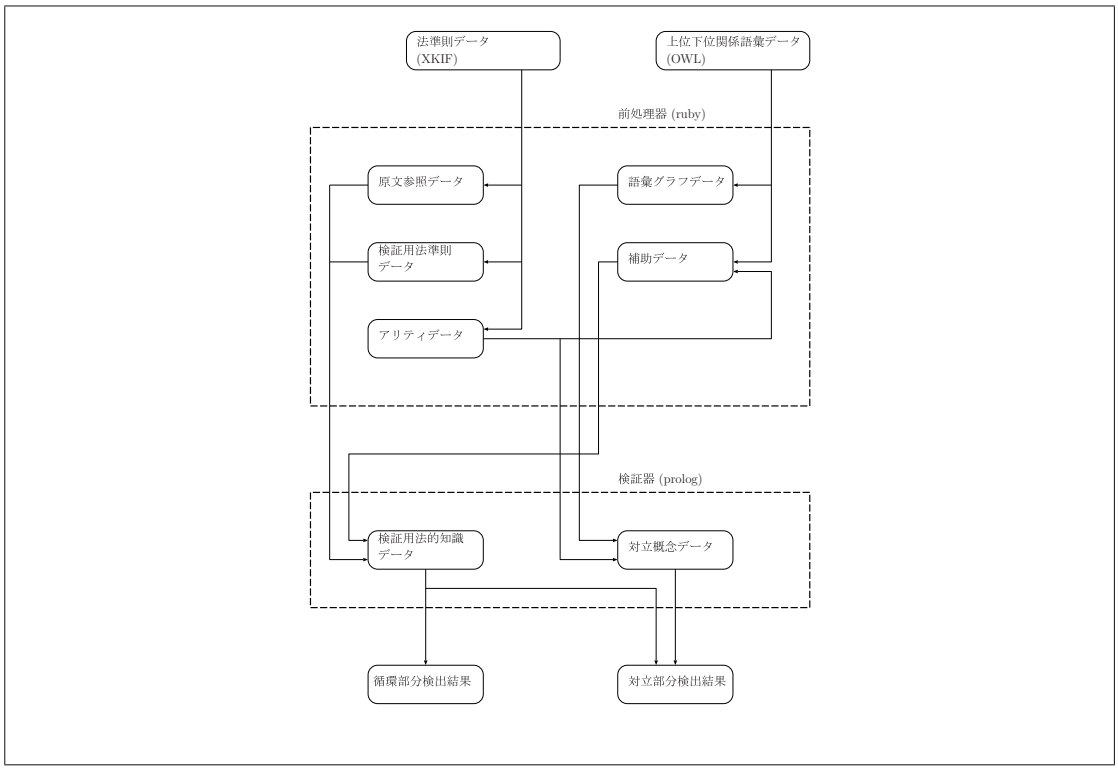


図 4.1: 各データの依存関係と生成箇所

4.2 前処理器への入力データ類

本節では前処理器に入力される，法準則データ，上位下位関係語彙データ，これら 2 種類のデータについて述べる．なお，これらのデータは，人手で作成されるものであるので，その記述の仕様，作成の方法などについて説明する．

4.2.1 法準則データ

法準則データは以下の手順によって作られる．なお以下の項目（1）– (3) に関しては，3.1.1，3.1.2 章で述べたとおりである．

1. 法令文が定める法規範から法準則を取り出す

2. 法準則を条件プログラムとして解釈する

3. 条件プログラムを EDLP で形式化する

4. 本稿が定める XKIF 形式でファイルに記述する

ここで項 (4) の XKIF とは，本稿が KIF (knowledge interface format) [32] を参考にして策定した知識記述用の XML 形式である．なお KIF とは知識の相互運用性を確保するために策定された記述仕様で，LISP に似た構文を持つ．この KIF は基本的に一階述語論理の記述を目的としている．ただし厳密な一階述語論理に則するわけではなく，より柔軟な記述ができるようになっている．

法準則データをこの XML である XKIF で記述する理由はソフトウェアの読み込み汎用性を確保するためである．読み込み汎用性とはデータに特別な加工を施すことなく複数のソフトウェアが読み込み可能かどうかを意味する．XML は広く知られたデータ記述形式でこの読み込み汎用性が高い．XML を採用することでそのデータを記述，編集する際に，特定のソフトウェアを新たに用意する必要がなくなる．事実 XML に対応したソフトウェアは多数存在する．それらの既存の資産を利用することで，独自のデータ形式を知識のデータに採用しながらも，低コストで容易な編集および作成が可能となる．XML を採用しない場合，低コストのデータ作成方法として記述用ソフトウェアの開発せずテキストデータを直接エディタなどで記述するという方法も考えられる．しかしそれよりも既存の XML 記述用ソフトウェアを用いた方が，ユーザーインターフェースが整備されていることからより効率的なデータの作成が可能であることは明らかである．

またデータ処理の観点からも XML の採用には利点が多い．本稿が前処理器の実装言語として採用している Ruby 言語では，XML の読み込み，処理するためのライブラリが充実している．このように，XML を読み込むためのインターフェースは現時点でほとんどの言語が備えており，様々な扱いが簡易になる．

さらに XML は設定自由度が高いことが知られている．XML はタグと呼ばれる対のマークアップで構造化データを記述，定義するものである．この構造化データとは，データに対して木構造を付与することである．その場合用いられるタグ

は限定的なものではなく，ユーザが自由に設定可能である．したがって，XML という形式に則りながらも，その構造に対する意味づけの部分でユーザの自由度は高い．

これらデータの読み込み汎用性，ユーザー定義の自由度，処理環境などの点から XML は知識の記述に適していると考えられた．したがって本稿ではこの KIF を XML で策定した XKIF を法準則データの記述形式とした．その XKIF の仕様を以下に示す．

まず法準則データを記述するためには EDLP が記述できなければならない．したがって XKIF では EDLP の論理プログラムを記述するタグを定義した．そのときに使用されるタグ，そして対応する EDLP の論理記号は表 4.1となる．

なお本稿が扱うデータは，純粋な EDLP の論理プログラムという性格のみを持っているわけではない．データは EDLP の論理プログラムであると同時に，法的知識という側面も持ち合わせている．そこで XKIF には法準則の論理プログラム以外に，条文名，条数，章名，そして項数などの法的知識特有の情報も記述することができるようにタグを定義した．その記述に使用されるタグは表 4.2に示される．

タグ名	意味	
<logic>	論理プログラム部分	-
<implies>	含意	$\leftarrow$
<and>[1]	ボディ部論理積	,
<or>	ヘッド部論理和	|
<not>	論理否定	$\neg$
<unp>	失敗の否定	*not*
<clause>	原子リテラル部分	-
<predicate>	述語名	-
<argument>	引数領域	-
<var>	変数記号	-

表 4.1: EDLP 記述用 XKIF のタグと対応

[1] これは，明記が必要なときに限って使用される．

タグ名	意味
<ordinance>	条文名
<subject>	項目名
<section>	条数
<paragraph>	項数

表 4.2: 法律情報記述用の XKIF のタグ

つぎに XKIF を用いた論理プログラムの記述方法を説明する．まずある論理式を P とする．この P は論理プログラムのひとつにあたるものとする．そのとき，P の部分論理式を φ, ψ, θ とする．また，$K(\varphi)$, $K(\psi)$, $K(\theta)$ は論理式をそれぞれ XKIF で表したものであるとする．なお $K(\varphi)$ は

<X>
$K(\gamma)$
$\vdots$
$K(\sigma)$
</X>

のように，あるタグ（ここでは便宜上‘<X>〜</X>’）で囲まれたものとする．これらから，

$\varphi \wedge \psi \wedge \theta$

という論理式は

<and>
$K(\varphi)$
$K(\psi)$
$K(\theta)$
</and>

と記述される．なお，選言‘∨’についても，同様の形式で，<or>を用いて記述される．

含意を表す‘<implies>’タグは，DOM (Document Object Model) 上の第 1 要素がヘッド部，第 2 要素以降がボディ部に相当するようになっている．これは以下の論理プログラムがあるとき，

$$\varphi \leftarrow \psi, \theta$$

以下のように XKIF で記述されることになる．

<implies>
$K(\varphi)$
$K(\psi)$
$K(\theta)$
</implies>

最後に‘not’, ‘unp’タグのような単項論理演算子を表すものは要素を 1 つだけ持つものとする．よって‘$not\,\varphi$’や，‘$\neg\varphi$’は以下のように記述する．

<not>
$K(\varphi)$
</not>

よって

<not>
$K(\varphi)$
$K(\psi)$
</not>

のようなデータは XKIF の XSD (XML schema definition) ファイルにおいて妥当ではない．

つぎに演算子以外の，要素について説明する．この XKIF で記述される論理プログラムのアトムは，‘<clause>’,‘<predicate>’,‘<argument>’,‘<var>’タグを用い

て構成される．<clause> タグは節を表す．簡単には，アトム 1 つに相当すると考えてよい．<predicate>は述語記号を定義するタグである．述語記号の定義は，<predicate>タグの内部の属性‘value’に記述する形でされる．<argument>タグはアトムの引数を表すタグである．その子要素には後述の変数名を定義する<var>タグを持つ．<argument>タグの主な役割は変数の順序を保存することである．XML では DOM 構造上親子関係しか定義されない．子の要素が順序を持つ場合，別途その情報を書く必要がある．アトムは変数の順序が保存されなければならないので，この<argument>タグの属性‘number’で変数がアトムの何番目の変数なのかという情報を保持する．最後に<var>タグは変数名の宣言を属性‘name’によって定義する．

これを用い，‘行政庁 (x)’のような述語は，以下のように XKIF で記述されることとなる．

```
<clause>
  <predicate value=’行政庁’/>
  <argument number=’1’>
    <var name=’x’/>
  </argument>
</clause>
```

なおタグ<var />のようにタグの最後に‘/’を書くのは終了タグの略記で，これは XML の仕様である．

そのほかの法的知識特有のデータの記述，論理演算子なども合わせ，この XKIF の記述例として実際のデータを以下に挙げる．なおこの例は，本稿が実験に用いたデータの一部である．

まず以下の EDLP の論理プログラムは，法律名「富山県行政手続条例 」，条数「第 8 条」，項数「第 1 項」として出現していた．

申請行為 $(x, y, z) \leftarrow$ 許認可等 (z), 申請者 (x), 行政庁 (y).

この規則を上記の法的な情報と合わせ XKIF で表すと図 4.2のようになる．

4.2.2 上位下位関係語彙データ

前処理器に入力される上位下位関係語彙データは人手により OWL [86] で記述される．OWL は代表的なオントロジ記述言語で，XML 上で策定されている．本稿はデータの汎用性を確保するために OWL を上位下位関係語彙データの記述に採用した．この汎用性とは，XML による読み込み汎用性の確保に加え，この上位下位関係を本稿の検証のみの使用を限定しないことを意味する．

この OWL の採用は OWL が持つ意味論を必要としたものではない．OWL は記述論理 (description logic) [6, 61, 89] を基礎として成り立っており，その解釈は厳格なものとなっている．その意味論に厳密にしたがうと本稿が目的とすることにそぐわない箇所がある．具体的にそれは，オントロジーに記述された語彙を一階述語論理へ読み替えるとき用いられる写像方法に関してである．さらに OWL は豊富な機能を持つが，その全てを本稿は必要としない．したがって上位下位関係語彙データは OWL の Scheme について妥当である（つまり OWL を編集可能なソフトウェアについてはその読み込み可能性が保障される）が，その利用については OWL の意味論に完全には準拠しない方法で一部用いるものとする．

本稿が使用している OWL のタグは上位下位関係を記述するための ‘Class’ と ‘subClassOf’ タグである．ある語彙項目 p1 と p2 が「 p1 は p2 の下位語である (p2 は p1 の上位語である)」という関係にあるとき，それを以下のように記述する．このような記述の集合が上位下位関係語彙データとなる．

```
<owl:Class rdf:ID="p1">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="p2"/>
  </rdfs:subClassOf>
</owl:Class>
```

なおここでは限定的にデータの一部を示しているが，実際にはそのほかも記述がある．ただしそれらは OWL の Scheme で妥当であるためだけに必要な記述であるため割愛した．

4.3 前処理器による中間生成データ類

本章では法準則データと上位下位関係語彙データそれぞれから前処理器が生成するデータ類についてその方法と仕様を説明する.

4.3.1 法準則データから生成されるデータ類

前処理器は法準則データから以下のデータを検証器に合わせて Prolog のコードとして取り出す.

- アリティデータ：述語のアリティを記録したデータ
- 原文参照データ：元の法準則データを Prolog の構文に直したデータ
- 検証用法準則データ：検証可能な形に法準則データを変換したデータ

以下の章でそれぞれの仕様，生成方法について説明する.

アリティデータ

アリティデータは法準則データに出現する述語の引数の数を取り出しそれを記録したものである．これは後述するが，上位下位関係語彙データから補助データ（4.3.2，4.3.2 章）や対立概念（4.4.1 章）を抽出する際に使用される.

アリティデータの形式は以下になる.

`arity(`*predicate_name*`,[`$arity_1$`, ...]).`

predicate_name は述語記号を，$arity_i$ はその述語記号が持つアリティを表す．アリティは複数種類の可能性があることからリスト形式で記録される．例えば述語記号 p に関して法準則データ内に‘`p(X)`’と‘`p(X,Y)`’というような出現があった場合，アリティデータは‘`arity(p, [1, 2])`’となる.

以下にその抽出例を示す.

例 1 以下の規則が法準則データに記述されていたとする．

$$P_0(x,y)|P_1(x,y) \leftarrow P_2(x), notP_3(x), \neg P_4(y)$$
$$P_0(x) \leftarrow P_2(x)$$

これから得られるアリティデータは以下になる．

```
arity(p0,[1,2]).
arity(p1,[2]).
arity(p2,[1]).
arity(p3,[1]).
arity(p4,[1]).
```

原文参照データ

原文参照データはXKIF のタグの入れ子構造をPrologのリストの入れ子構造で記述したものである．これは検証中にもとの法準則の形を確認する必要が生じたときに用いられるもので，これそのものを検証するわけではない．なおXKIF からPlorogの構文に変換するのは，検証器から法準則データを利用しやすくするためである．以下に4.2.1 章のXKIF を原文参照データに変換した例を簡単に挙げる．

```
ordinance(富山県行政手続条例,[
    section(第８条,subject(理由の提示),[
        paragraph(第１項,[
            logic([
                implise(申請行為(X,Y,Z),[
                    許認可等(Z),
                    申請者(X),
```

```
                    行政庁(Y)
                ]),
            ]),
        ]),
    ]),
]).
```

検証用法準則データ

検証器は法準則の規則を Prolog の規則として表したものを要求する．これは，法準則が‘H:-B1,B2, ···’のような Prolog のコードの規則として入力されることを意味する．したがって EDLP の法準則はこの前処理器によって，Prolog のコードの規則へと変換される．この方法をとる理由は小さな手間で検証の実行速度を上げるためである．

本稿で使用している Prolog 処理系 SWI-Prolog はコードを読み込むとそれを WAM (Warren's abstraction machine) 用の中間言語にコンパイルする．このコンパイルではコードの実行速度最適化も同時にされる．つまり法準則を Prolog の規則とすることで，自動で検証実行速度に関わる最適化が見込まれる．

通常このような検証プログラムを実装する場合，検証データは文字列として読み込み，文字列処理上で推論をする場合が多い．それは柔軟な記述能力と推論機構を支えるためで，検証系の Isabelle [62] や，Coq [14] などにも表される．しかし文字列処理で推論をする場合，その実行速度を上げるためには非常に手間がかかる．それは上記で示した検証系に見られるように，それだけで 1 つの研究分野をなす．

検証など計算量が概して大きい研究対象 [21] の場合，実装における高速化は無視できない問題である．しかしその場合本来の目的と同時に，実装，処理，双方のコストを考える必要がある．本稿の場合検証器実装言語として Prolog を採用している．推論処理における Prolog の記述能力は高く，その実装コストは他の言語を大きく上回る．ただし Prolog は実行速度がさほど高速ではない言語としても知

られている．本稿はその実装のためのコストと，高速化のためのコスト双方を勘案して現在の実装形態を採用している．この規則を Prolog の規則へ変換する現在の検証方法の実行速度は，処理系の速度と同等である．これ以上の速度を求める場合 Prolog の処理系を作り直す必要がある．それは上記理由から本論から外れることになる．

以上の理由から，法準則は Prolog のコードに変換される．しかしこのことは，同時に法準則の表現形式が Prolog の持つ制約に縛られてしまうことも意味する．法準則データは EDLP で形式化されており，ヘッド部に論理和を含めることが許されている．しかし Prolog はホーン節を基礎としていることから，基本的にはヘッド部に論理和を含めることはできない．この点に関してはより効果的な Prolog 上での知識集合の実装 [27–29] を考慮する必要があると考えるが，それは今後の課題である．

よって本稿では，用いるデータが純粋な EDLP ではなく法準則という法的知識であり，また検証用のデータであるという特性を用いて，その Prolog の制約が検証上問題にならないよう処理を行う．そのためにはまず法準則データを Prolog の構文に納めなければならない．そこで [8, 79] を参考に，ヘッド部の論理和に応じて規則を分割しホーン節に収める．その分割の例を以下に挙げる．

例 2 以下の EDLP の規則があると仮定する．

$$p_0(x)|p_1(x) \leftarrow p_2(x), p_3(x).$$

これを，ヘッド部の論理和にしたがって，以下のコードへ変換する．

$$p_0(x) \leftarrow p_2(x), p_3(x).$$
$$p_1(x) \leftarrow p_2(x), p_3(x).$$

この規則の分割により検証用法準則データは全てヘッド部に論理和が存在しない論理プログラムへと変換される．このとき変換がもとの規則と等価である変換をしたものではないことに注意する．等価な変換を行うには，[**?**] にあるような分割時に除外したヘッド部の要素の否定節をボディ部に含めなければならない．本稿ではその点を参照用データを適宜用いることで補っている．よってこの等価で

ない分割された規則を検証することは，本来の規則を検証することとは異なってしまうという懸念がある．しかし本稿ではそれは問題とはならない．その理由は以下の 2 つである．

1. 自由な演繹ではなく，本稿が定める方法で「検証」をするという特性
2. 法的知識における法律効果の選択を表す「又は（若しくは）」は論理学上の論理和ではないこと

まずひとつめの項 (1) である．演繹とは知識の集合を Δ とし，証明したい命題を φ としたとき，システムが $\Delta \vdash \varphi$ が充足されるかどうかを推論規則を知識集合に適応して検査することである．ここで自由な演繹とは，ユーザが φ，つまりクエリを任意に設定してよいことを意味する．もしこの自由な演繹を許容するならばシステムはその知識集合が基礎としている論理体系の証明機構を完全に備える必要がある．またその証明機構を備えたシステム上では，式変形が同値類でなければならない．すなわちその場合，上記で述べた等価でない EDLP の規則の分割は許されるものではない．しかしながら本稿では検証することが目的で，それ以外の効果は期待しない.「ある法律効果をもつ規則から，どのような事実が導かれるか．またそれが導かれると仮定した場合，対立を起こしていないか」，この点のみが本稿がシステムに求める要求である．つまり ‘$\varphi|\psi \leftarrow \chi_1, \ldots, \chi_n$.’ のような規則から，$\varphi|\psi$ を導くことや，任意の論理式が知識集合で証明可能かどうかということは本研究の検証では重要ではない．実際に，φ および ψ が導かれたと仮定したとき，どの規則が使用されたのか，そしてそれによって法の内部で対立概念データが規定する対立が演繹可能かを調べることができれば十分である．したがって，規則を分割し，ヘッド部に現れていたそれぞれの述語記号について，その対立の検証が可能であればこの規則分割は問題がない．これらの詳細に関しては 4.5 章で述べる．このような検証だけに注目して特殊化された演繹体系をもつ研究は同様の検証に関わる研究でも見られる．なお上記のような論理和を含む命題を拡張論理プログラムから演繹する研究もある [47] が，上記のことから現時点では考慮する必要はないと考える．

次に項 (2) である．法的知識における法令用語「又は」などは，一般に論理学でいう「論理和」とは厳密には異なる．論理学における論理和は ‘$\varphi \vee \psi$’ とあった場

合，φ または ψ が真であるときと，それに加え同時に真であるとき $\varphi \vee \psi$ は真になるときを表す．このことは集合論的には妥当なものである．しかしながら本稿が検証する知識は論理学からなりたつものではなく法的知識である．法的な解釈と論理学的な解釈のどちらを優先するべきか考えた場合，その答えは自明で法的解釈を優先するべきである．法令用語の「又は」は選択を表すとある．つまり論理学で論理和の式が真となる条件のひとつである項の 2 つが同時に真であるときを考慮する必要がない．この点からも法的知識においては論理和の項のそれぞれを別々に検査することで必要十分であることが分かる．なお拡張論理プログラムの論理和による帰結に対して，優先順位を設ける研究もあるが，今回はその全てが検証対象になるのでそれは考えない [18].

法準則データから検証用法準則データを生成する手順を以下に説明する．

前処理器は検証用法準則データの生成において 2 つの加工を施す．ひとつは検証用の述語記号を埋め込むこと，もうひとつは先ほど述べた規則の分割処理である．

まず検証用の述語の埋め込みは，検証述語 ‘`head_hook`’ と ‘`body_hook`’ で規則に現れる述語を覆うことを行う．具体的には以下のようなリテラルがある規則のボディ部に含まれていたとする．

行政庁 (x)

これをその検証述語記号で覆い，以下のようにされる．

`body_hook(`行政庁 (x)`, Ruleinfo, Opt)`

ここで `Ruleinfo` は，そのアトムが何号何条何項の法令文の規則に含まれていたかという情報を格納している．そして `Opt` は，動作の命令を格納する．この動作の命令は，検証述語がアトム実行前や後に任意の割り込み処理を挿入する役割を担うことに関係する．本稿ではこれまで説明したように，法準則データは Prolog で実行可能な規則として記述されている．つまり検証するコードと，検証されるコードには基本的に区分がない．検証を行うには被検証コードに対して何らかのメタ的な区分を実現する機構が必要となる．それがこの検証用述語記号で検証さ

れる規則に出現するアトムを覆うということである．ただし検証の作業というものはある一過性の処理で終えることはできない．それは3.2.3章でも分かるように，検証するために一度知識を調べる作業などが必要となる．もしもその作業ごとに検証用述語を用意したならば，それぞれ別の述語で覆ったデータを個別に用意する必要があり非効率である．したがってこのOptによってそのつどの処理内容を切り変えている．なおこの割り込み処理で検証および結果の記録が行われている．なおもうひとつの前処理器で行われる加工，規則の分割分割処理については，上記で述べた通りである．

こうして検証用述語記号に覆われた検証用法準則データは，検証対象のアトムPに対し，以下のような処理の流れを持つことになる．

1. head_hook(P , Ruleinfo, Opt)が実行される．
2. head_hook(P , Ruleinfo, Opt)をヘッド部に持つ規則が選ばれる
3. 選ばれた規則のボディ部，たとえばbody_hook(Q , Ruleinfo, Opt)が実行される
4. 述語body_hookは，Optの内容によって，法準則の述語Qに対して処理を行う
5. 述語body_hookは，内部処理が終了した後，head_hook(Q , Ruleinfo, Opt)を実行する

このようにして，Prolog処理系の機能を最大限利用しながら法準則を実行し割り込み処理内部で検証が行われていく．

法準則データから検証用法準則データを生成する実際の例を以下に挙げる．なお例には規則分割はすでに行われたものとする．

例 3 XKIFの法準則データに,「法律名 a，項目名 b，条数 m，項数 n」で，以下の規則があるとする．

$$p_0(x, y) \leftarrow p_1(x), p_2(y)$$

この規則を変換して得られる検証用法準則データは以下のコードになる．

```
head_hook(p0(X,Y), Ruleinfo, Opt):-
    Ruleinfo = [
        ordinance('a'),
        section('m'),
        subject('b'),
        paragraph('n')
    ],
    body_hook(p1(X), Ruleinfo, Opt),
    body_hook(p2(Y), Ruleinfo, Opt).
```

検証でアトムが検証項目に反した場合は割り込み処理内部で Ruleinfo の値が記録される．すなわち検証後に提示される議論の情報はこの条文の情報 Ruleinfo となる．

4.3.2 上位下位関係語彙データから生成されるデータ類

上位下位関係語彙データからは前処理器により以下の 2 つのデータが生成される．

- 上位下位関係グラフデータ：上位下位関係を有向グラフに変換したもの
- 補助データ：法準則データに暗黙知を補うもの

以下それぞれを説明する．

語彙グラフデータ

語彙グラフデータは上位下位関係語彙データにおける語彙の順序 2 項関係を有向グラフとして Prolog のコードに変換したものである．その変換方法は,「 p1 は p2 の下位語である（p2 は p1 の上位語である）」という上位下位関係を以下のようなコードへ単射したものである．

```
relation(p1, p2).
```

検証器はこの半順序集合から対立概念を計算する．ここで注意することは，本データは語彙グラフデータに含まれる上位下位関係をそのまま順序関係に書き換えたデータであるということである．よって，本データは束としての性質を保証していない．つまりこのデータにおいて，任意の要素の組に対して交わりが存在するとは限らない．

補助データ

法令文には語彙の関連などの常識の範疇に分類されるような暗黙的な知識は，特定の語彙の定義を除いて明記されない．これらは一般知識 (normative knowledge) と呼ばれる．例えばそれは「車は乗物である」といったようなものである．上位下位関係語彙データはその暗黙知を一部集めたものといえる [3,17,81]．それは，上位下位関係語彙データの順序関係が含意の意味を持つからである．

このことから上位下位関係語彙データは，規則に読み替えることができる．例えば概念名 p_1 と p_2 に「 p1 は p2 の下位語である（p2 は p1 の上位語である）」という上位下位関係があるとする．このとき「あるものが 下位語 p1 であるなら，それは 上位語 p2 でもある」という含意の論理式 '$p_1(x) \to p_2(x)$' に読み替えることができる [7].

したがって，上位下位関係語彙データの順序関係を検証用法準則データ同様の規則に変換し，その規則を検証用法準則データに追加することで，検証用法準則データに対して法令文には明記されていなかった一般知識の一部を補完できることが期待される．この一般知識を検証用法準則データに追加する意義は大きい．なぜならばその知識の追加によって，以下のような新たに検出可能な検出するべき対立が存在するからである．

たとえば検証用法準則データに '$\{\top \to p, p \to q, r \to s, r \to t\}$' という規則が存在し，$s$ と t が対立概念であったとする．このときもしも q と r の間に暗黙的な上位下位の関係があったならば，この法準則の集合には対立が含まれ，それを検

出することが望まれる．ここで暗黙知 $q \to r$ という知識を検証用法準則データに補うことができた場合，この上記規則集合に含まれる s, t に関する対立を検出できる．

上位下位関係を検証用法準則データに加えるためにはその順序関係を論理プログラムの規則に直す必要がある．そのためには順序関係が定義されている語彙名を，その変換後の論理プログラムに現れる述語記号とみなすことを行う．しかしそのとき述語のアリティをどうするべきか注意を払う必要がある [88]．

上位下位関係語彙データを記述している OWL の意味論では，クラス名のアリティは 1，ロール名は 2 に限定される．クラス名とは一般にものの性質を表す．例えばそれは「人」や「車」といったようなものである．そしてロール名とは役割をなどを表す．このように OWL 上のデータを一階述語論理に読み替える定義が OWL には存在している．これは OWL が記述論理を基礎としていることに由来するものである．

OWL の意味論を使用する目的のもとで厳密に語彙が定義されている場合は，その解釈方法は妥当である．しかし本稿は検証に対して上位下位関係語彙データだけを使用するものではなく，またこのデータが検証する対象であるわけでもない．本稿では入力としてこの上位下位関係語彙データに加え，法準則データが存在する．そして基軸となる知識集合は法準則データである．具体的には，もしある語彙名 p が上位下位関係語彙データに存在し，法準則データにもその p を述語記号とするアトムが存在した場合，優先順位として優先されるべきその述語記号が持つ情報は，法準則データに含まれるものである．よって本稿では上位下位関係語彙データを規則へ変換する際，その述語のアリティは法準則データから得たアリティデータを用いて決定する．

ただしアリティデータを用いた論理プログラムへの変換には一定の制限を設けている．その制限とは，上位下位関係語彙データで上位下位関係のある語彙名 p_1 と p_2 が共通したアリティをアリティデータ上で持っている場合のみそのアリティを用いて論理プログラムに変換するというものである．これは規則に自由な変数が出現することを防ぐためでもある．なおこのような OWL の語彙に対して 3 以上のアリティを与える研究も他に存在する [42, 66, 67]．

手続き 1と手続き 2に上位下位関係知識から補助知識への具体的変換手続きを示

す．なお手続き 1における ‘$p \leq q$’ は「p は q の下位語である」を表すものとする．

```
Input: A:アリティデータ
Input: T:上位下位関係データ
Output: K^N:補完知識
begin
    K^N := ∅ ;
    forall p1 ≤ p2 ∈ T do
        if arity(p1, A) ∩ arity(p2, A) ≠ ∅ then
            forall n ∈ arity(p1, A) ∩ arity(p2, A) do
                K^N := K^N ∪ {p2(x1, ..., xn) ← p1(x1, ..., xn)} ;
            end
        end
    end
end
```

手続き 1 上位下位関係から論理プログラムへの変換

```
Input: A:アリティデータ
Input: p:述語名
Output: N:アリティの集合
begin
    if arity(p, N) ∈ A then
        return N;
    else
        return ∅;
    end
end
```

手続き 2 関数 arity()

手続き 1と手続き 2による処理内容は以下となる．

まず手続き 1で上位下位関係語彙データから任意の順序 2 項関係を取り出す．次にその順序 2 項関係にある 2 つの語彙を述語記号とみなし，それらが持つアリティを関数 *arity* を用いてアリティデータから調べる．そしてその 2 つの述語記号が共通したアリティを持つとき，語彙をそのアリティにしたがった述語へ変換し，検証用法準則データで述べたような論理プログラムへ変換する．

なお語彙を述語に変換する際の変数名は，その 2 つの述語に対して同じ順序で与えている．これは，アリティが等しくそこに上位下位関係が存在すれば引数の型の順序も同じである，という仮定を置いたためである．例えば p_1 と p_2 に順序関係 $p_1 < p_2$[2]があり，アリティが 3 で一致していたとする．このときそれを以下の論

[2] $x < y$ は y は x の上位語を意味するものとする．

理式として変換することを意味する．なお変数の名前はそれぞれ違ったものを生成する．

$$p_2(X1, X2, X3) \leftarrow p_1(X1, X2, X3).$$

この変換で誤った論理プログラムが生成される可能性として否定できない．それは p_1 と p_2 のアリティが一致していて，変数の順序が異なる場合である．例えば以下のような論理プログラムが正しいとされる場合などである．

$$p_2(X1, X2, X3) \leftarrow p_1(X1, X3, X2).$$

この変数の順序を厳密に解決するためには，変換する情報のもとである法準則データの述語記号の引数に型の情報が必要になる．例えば「運転する (x,y)」という述語があった場合，その述語の引数に対して「運転する (x:人, y:乗物)」といったように変数それぞれにどのような概念が入るか明示することを意味する．これは型階層論理などで研究されていることでもある．しかしこのような述語の引数に型を与えることには解決しなければならない問題がある．型を与える方法として考えられるのは 2 つある．ひとつが，法準則データで用いられる述語を全て正確に OWL 上で定義してしまう方法である．しかしこの方法は上記で述べたようにアリティの制限が厳しく，現時点で OWL が法的知識に用いられる述語記号の情報を記述する能力を持ち合わせているとはいいにくい．また OWL を拡張してその述語を定義することも考えられるが，その場合は拡張 OWL を新たに定義する必要が生じ，語彙関連項目データとしての読み込み汎用性を欠くこととなる．その場合の不利益は 4.2.1 章 で述べた通りである．よって型付きの記述をそのまま採用することはできない．

またもうひとつの方法は，法準則データに記述されている論理プログラムに型情報を与えることである．この場合は法を論理プログラムに直す際に人がその引数に対して与えるべき妥当な型を推測して与えることとなる．このことは記述コストの増大と共に別の問題も引き起こす．それは [90] で用いられているように，その型の単一化子が型階層として存在するか導く機能が実装に必要となることである．たとえば，上記の例で述べると，p_1 に

$p_1(x:人, y:車, z:場所)$

という型が与えられているとする．また p_2 には

$p_2(x:人, y:駐車場, z:乗物)$

のような型が与えられていると仮定する．この場合人間であるならば，通常 p_1 の「車」の型と p_2 の「乗物」という型は関係があると見なし，その変数がそれぞれに対応すると判別できる．つまりここには単一化子が存在する．しかしこれを機械的に行う場合は，型階層においてそれぞれの型が関係を持つか調べる必要がある．一般にはこれは型推論と呼ばれる．このように法準則データに対して型情報をあたえることは，データ作成時の人的コストの増大と，より複雑な演繹システムが必要となる．

以上のことから現時点では，型は順序のまま一致するという仮定をおいた．この点に関する厳密な解決は今後の課題でもある．

以下に，補助データの生成例を挙げる．

例 4

今，上位下位関係語彙データに

$p_1 \leq p_2, p_1 \leq p_3$

が含まれ，アリティデータに

```
arity(p1, [1, 2, 3]).
arity(p2, [1, 2]).
```

が含まれていたとする．これを用いて，$p_1 \leq p_2, p_1 \leq p_3$ から，p1 と p2，p1 と p3 のアリティが比較される．そのとき，p1 と p2 にはアリティ1，2 で一致が確認できる．したがって，以下の 2 つの規則が補助データとして生成される．

```
head_hook(p2(X1), RuleInfo, Opt):-
  RuleInfo = [normative],
  body_hook(p1(X1), RuleInfo, Opt).
head_hook(p2(X1, X2), RuleInfo, Opt):-
  RuleInfo = [normative],
  body_hook(p1(X1, X2), RuleInfo, Opt).
```

このとき生成されるコードは，4.3.1 章で示した検証用法準則データとほぼ同様である．異なる部分は，条文の名前や条数などを格納している RuleInfo の値が，補助データを表す値 ‘normative’ になっていることである．

現時点では補助データに含まれている上位下位関係から作られた規則を個別に識別するための情報はない．したがって，もしも対立がこの補助データによって検出された場合，その該当する補助データの規則を記録する手段としては，その規則が補助データに含まれていたというほかない．この点に関しては，今後それら補助データの規則に対してユニークな番号を振るなどして対応することが考えられるが，その情報は OWL 側にも含める必要がある．それは OWL の拡張も伴うため今後の課題としている．

4.4 検証器での前処理

検証器は対立による矛盾検査の前に 2 つの処理をする．そのひとつは対立概念データの計算である．そしてもうひとつの処理は循環検査である．

4.4.1 対立概念データ

まず対立概念を上位下位関係語彙データとアリティデータに関連させて以下のように定義する．

定義 12 対立概念データに含まれる対立概念
Δ を論理プログラムの集合，Δ_A を Δ から得られたアリティデータ，$G = \langle V, E \rangle$ を上位下位関係を表す有向グラフとする．このとき，V のある要素 v_i, v_j $(i \neq j)$ が G 上で $v_i \sqcap v_j = \bot$ であり，かつ $\mathtt{arity}(v_i, \Delta_A) \cap \mathtt{arity}(v_j, \Delta_A) \neq \emptyset$ であるとき，この (v_i, v_j) を G による Δ の対立概念とする．

この対立概念の定義は，前処理器の用意した語彙グラフデータで交わりが $\bot$ になり，さらにアリティデータで共通アリティを持つような語彙の組の集合とすることを意味する．これは型階層論理 [7] における排他的ソート [55] を参考にしたものである．

兼岩ら [55] の排他的ソートは型階層論理の意味論に基づいた概念同士の排他関係である．型階層論理においてその概念名は,「ソート (sort)」と呼ばれる．ソートとは，対象の集合が用意されているときにその部分集合に対して名前を付けたものとして扱われる．つまり，対象の集合 U が存在したとする．このときソート s では，ソートの解釈関数を $[\![\cdot]\!]$ を用いて，関係 $[\![s]\!] \subseteq U$ が成り立つ．そしてそのソート同士が構成する型階層とは，その対象の集合 $[\![s]\!]$ の包含関係によって意味づけられている．型階層論理では，ソート s と s' 間にある包含関係を $s \sqsubseteq s'$ と記述するが，この関係は意味論上 $[\![s]\!] \subseteq [\![s']\!]$ が満たされることになる．したがって，この型階層論理の意味論に基づくソート間の排他的関係とは，$[\![s]\!] \cap [\![s']\!] = \emptyset$ が満たされるソート s と s' に対して定義されている．

本来ならばこの排他的ソートの定義をそのまま採用することが望まれるが，本稿においては以下の理由からそれはできない．まず上記の型階層論理では，ソートが単項述語に変換できることが前提である．本稿が用いる概念名は補助データ生成時に説明したように，アリティが法準則データに依存して決定されるため単項述語であることが保証されていない．次に，排他的ソートはその排他関係が対象の集合を用いる意味論に基づいている．つまり，そのソートや概念名が写像される対象の集合が定義されていなければ排他的関係にあるのかどうか判別できな

い．事実上記研究では，その排他的関係は計算によって求めるものではなく，概念名にあらかじめ定義される．つまりこれは，排他的関係をそれぞれ人手で定義する必要があることを意味する．このことは，対立概念を，概念名の組み合わせ全てについて人手で定義することと同じである．

よって本稿では，いくつかの仮定を置くことにより，完全ではなく近似的にこの対立概念を計算によって求める方法を提案する．それが以下で説明する仮定と，アリティ制限を伴いながら順序関係に対する演算「交わり」によって求めることである．

まず仮定とは,「任意の 2 つの概念において，同時に成り立つ対象が存在する場合，その共通対象の集合には必ず概念名が与えられている」という仮定である．上記研究においては，もし $[\![s]\!] \cap [\![s']\!] = \Delta, \Delta \neq \emptyset$ であるとき，s と s' に対する交わりは $\bot$ になる可能性がある．それは Δ に明示的に概念名が与えられおらず，上位下位関係上ではその集合が体現化されていない場合である．そこでもしその Δ に概念名 s'' が与えられていたならば，s, s', s'' に対して，$s'' \sqsubseteq s$, $s'' \sqsubseteq s'$ という順序関係が存在するため，s と s' の交わりは，$\bot$ にならず，s'' になる．したがって，このような共通対象の集合には必ず概念名が定義されているという仮定をおくと，対立概念であるとできる概念の組は同時に成り立つことがないとできる．なお型階層論理の上位下位関係の記号 $\sqsubseteq$ は推移律を少なくとも満たしているものとする．よって本稿では，共通下位概念がある場合は必ずその名前が上位下位関係語彙データで定義され，またそれが定義されていない場合は，その 2 つの概念名に対して共通した対象は「知られている範囲では存在しない」とする．

次に，本稿に対して排他的関係がそのまま採用できない理由は対象集合の問題である．型階層論理で用いられる解釈関数 $[\![\cdot]\!]$，これは，ソート名を対象集合 U の部分集合へ写像する関数である．したがって，U が，$\{a_1, a_2, \dots\}$ のような対象の集合である場合，当然ソートは単項述語と等しい意味になる．本項で用いる概念名は，上記で述べたように単項であるとは限らない．そこで本稿では，その対象集合と解釈関数について以下のようにする．それは，対象集合 U が存在したとき，その U のべき集合 $\mathcal{B}(U)$ を考える．そして，上位下位関係語彙データに含まれる概念名に対する解釈関数を $[\![\cdot]\!]^{\mathcal{B}}$ とした場合，この $[\![\cdot]\!]^{\mathcal{B}}$ は概念名を $\mathcal{B}(U)$ に写像する関数であるとする．このように考えることで，概念名が単項である必要はなくな

る．ただしこのままでは，もしある概念名 p_1 と p_2 が存在し，それぞれのアリティが，n, m $(n \neq m)$ であったならば，p_1 と p_2 は意味論上対立概念であるとするのが妥当となってしまう．しかしこのようなアリティが異なる概念名同士が対立概念であるとすることには検証上意義がない．それは，アリティが異なればそれらは 3.2.1 章で述べた，反意語，否定辞の含有語彙，名詞の意味上の排他的関係，いずれにも該当しないからである．そこで本稿では，まず交わりを計算する前にそれら概念名のアリティが等しい場合という制約を設けている．これにより無意味な対立概念を計算することを押さえる．

これら仮定のもと，交わりを用いて対立概念を計算する．

対立概念の具体的な計算手続きを手続き 3に示す．なお手続き 3において，$G = \langle V, E \rangle$ は語彙グラフデータ，V は上位下位関係語彙データに含まれる語彙の集合，E は V の要素に対する順序関係の集合とする．また，‘$G : v \sqcap w = u$’ は「グラフ G 上で $v \sqcap w = u$ をみたす」と解釈する．

Input: $G = \langle V, E \rangle$: 語彙グラフデータ
Input: A: アリティデータ
Output: C: 対立概念データ
begin
  $C := \emptyset$;
  $G' = complement\_graph(G)$;
  forall $v \in V$ **do**
    forall $u \in V$ **do**
      if $arity(v, A) \cap arity(u, A) \neq \emptyset, G' : v \sqcap u = \bot$ **then**
        $C := C \cup \{\texttt{conflict}(v, u)\}$;
      end
    end
  end
  return C;
end

手続き 3 対立概念抽出

手続き 3と，それにかかわる手続き 4について説明する．

手続き 3はグラフデータが束であることを保障しないため，関数 `complement_graph` を用いてグラフデータの順序関係に対し上と下に $\top$ と $\bot$ で閉じる補完をする．グラフ $G = \langle V, E \rangle$ が $\top$ と $\bot$ に対して上と下に閉じているとは，V の任意の要素 v に対し，$v \leq \top$ かつ $\bot \leq v$ が E 上でなりたつことをいう．このとき $\leq$ は推移律を満たすものとする．対立概念は $v \sqcap w = \bot$ を満たす v と w を捜すことである．つまりグラフは特に下に閉じる必要がある．さもなければ，下界が存在し最大限が存

```
Input:  G = ⟨V, E⟩: グラフ
Output:  G′ = ⟨V′, E′⟩: 補完済みグラフ
begin
    V′ := V ∪ {⊥, ⊤};
    E′ := E;
    forall v ∈ V do
        if not (∃u ∈ V, v ≠ u, v ≤ u ∈ E) then
            E′ := E′ ∪ {v ≤ ⊤};
        end
        if not (∃u ∈ V, v ≠ u, u ≤ v ∈ E) then
            E′ := E′ ∪ {⊥ ≤ v};
        end
    end
    return G′;
end
```

手続き 4 関数 complement_graph()

在しない場合と，下界が空であることの区別がつかなくなる．なお $\top$ に関する補完は今後の拡張を考えてのことであり，本稿では特に使用しない．

次に手続き 3は補完されたグラフ G' の頂点集合 V' から任意に 2 つの概念名 $v, w (v, w \neq \bot, v \neq w)$ を選ぶ．そして，v, w が互いに共通したアリティを持つか検査を行う．もし共通アリティが存在した場合，それら概念名 v, w が $v \sqcap w = \bot$ を満たすか検査する．もし満たせばその組を対立概念とし記録する．

4.4.2 循環検査

検証器は矛盾検証の前にまず検証用法的知識データが非循環か検査をする．その必要性は**??** 章で述べたとおりである．

循環検査は有向グラフ上で行われる．この検証用法的知識データとは，検証用法準則データと補助データを合わせたものを意味する．補助データは語彙グラフデータとして等価な有向グラフがすでにある．したがって循環検査は，検証用法準則データを有向グラフにしたものと，語彙グラフデータとを合わせたものに対して行われる．

検証用法準則データを有向グラフへ変換する手続きは以下のように表される．$L_1, \ldots, L_n, L_m$ をリテラル，Δ を '$L_m \leftarrow L_1, \ldots, L_n$' のような要素を持つ論理プログラムとする．なお規則 $L_m \leftarrow L_1, \ldots, L_n$ に対し，$L_1, \ldots, L_n$ をボディ部，また L_m をヘッド部と呼ぶ．このとき，*body* をボディ部に含まれるリテラルの集合を

返す関数，$head$ をホーン節のヘッド部に含まれるリテラルの集合を返す関数，lit を論理プログラムに含まれるリテラルの集合を返す関数とする。そこで，以下の有向グラフ $G = \langle V, E \rangle$ を，論理プログラムから変換した有向グラフとする．なお，E の要素である辺 (L_i, L_j) は L_i から L_j への方向を表す.

$$V = \{L_i | C \in \Delta, L_i \in lit(C)\},$$
$$E = \{(L_i, L_j) | C \in \Delta, L_i \in body(C), L_j \in head\},$$

例えば，

納める $(x, w, y, z) \leftarrow$
　酒類 (y), 製造者 (x), 製造場 (w), 酒税 (z), 製造 (x, y, w), 移出 (y, w).

のような論理プログラムが検証用法準則データにあるなら，これは

$V =$ { 納める, 酒類, 製造者, 製造場, 酒税, 製造, 移出 }
$E =$ {(酒類, 納める), (製造者, 納める), (製造場, 納める),
　(酒税, 納める), (製造, 納める), (移出, 納める)}

という有向グラフに変換される．

この有向グラフを用いた循環検査の具体的な手続きを 手続き 5 に示す．なお手続き 5で使用されている順序関係 '$<$' は，推移律を充たし，反射律を充たさないものとする．これは各頂点について自己到達可能かどうかで閉路が存在するか判断するからである．また '$G : x < y$' を「グラフ G 上で $x < y$ が充たされる」とする．

手続き 5では，まずグラフの任意の頂点を 1 つ選び出す．次に，その頂点が反射律を満たさない順序関係上で自己到達可能かどうかを検査する．反射律を持たない順序関係で自己到達可能な頂点は，ある閉路上にある頂点である．

本稿ではその自己到達可能な頂点の集合を選び出すだけで循環検査を終わらない．その頂点がどのような循環を形成しているかを特定する．閉路は複数存在する可能性があり，またそのそれぞれの閉路は交差していたり，素であったりする．そのような閉路の状態を解析し提示することにより，ユーザはどのように知識を修正するべきか指針にできる．その閉路構成の特定方法は以下になる．

まず循環検査によって集められた自己到達可能な頂点集合から，任意の 2 つの頂点を選び出す．そしてその 2 頂点が互いに到達可能か検査する．この場合も，その順序関係は反射律を充たさない．これはある経路の閉路と別の経路の閉路が共通した頂点を含んでいる場合，そのそれぞれの閉路上にある任意の頂点は，同じそれぞれの閉路上の任意の頂点へ到達可能であることを利用し，互いに素な閉路へと自己到達可能な頂点集合を分割するためである．このようにしてそれぞれの閉路を区別する．このときこれら頂点は，検証用法準則データと，語彙グラフデータに含まれるものであるので，頂点がどの法準則であったか，または上位下位関係語彙データに含まれていたものかどうか，という情報を持っている．したがって，ユーザには,「この閉路には，XX 法 第 n 条 m 項の述語 p が含まれる」などという情報が示される．

なおこのような閉路を含む有向グラフに対し，その閉路を解消するため削除する辺を最小化する数学的な研究はある．しかし本稿の場合はもととなる知識集合が法的知識であるため，循環箇所について削除，修正すべき部分がどこかは法の意味を考慮する必要がある．よって数学的にそれは決定できないので，自動でこの閉路を解消することは試みない．あくまで，ユーザが修正するために必要となる情報を提示するに止める．

```
Input:  K : 検証用法準則知識データ
Input:  G^t = ⟨V^t, E^t⟩ : 語彙グラフデータ
Output: U : 循環箇所
begin
    /* 初期設定 */
    V := V^t ;
    E := E^t ;
    G = ⟨V, E⟩ ;
    L := ∅ ;
    /* 規則から有向グラフへの変換 */
    forall R ∈ K do
        R = L_0 ← L_1, ..., L_n;
        V := V ∪ {L_0} ;
        for i = 1 to n do
            V := V ∪ {L_i};
            E := E ∪ {L_i ≤ L_0};
        end
    end
    /* 循環部分の頂点収集 */
    forall v ∈ V do
        if G : v < v then L := L ∪ {v} ;
    end
    /* 疎な集合へ分割 */
    forall v ∈ L do
        U' := {v} ;
        forall v' ∈ L do
            if v ≤ v' ∈ E & v ≠ v' then
                L := L \ {v'};
                U' := U' ∪ {v'};
            end
        end
        U := U ∪ {U'} ;
    end
    U := U ∪ L ;
    return U ;
end
```

手続き 5 循環箇所検出手続き

4.5 法的知識の検証

本章では検証用法準則データと，補助データを合わせた検証用法的知識データに対する検証の詳細を説明する．検証ではアブダクションを用いて検証用法的知識データに対立が含まれていないかを検査する．対立が検出された場合，その対立はある 2 つの概念によって成り立っている．そこでその対立概念それぞれに対する議論を求める．対立する議論は，法的知識内部に含まれる対立の構造を表す．

よってこの対立する議論をユーザに提示することにより，ユーザは修正するべき法的知識へ迅速にたどり着け，検証作業の軽減となる．

まず検証の具体的手続き，手続き 6, 7を示す．この手続き 7，6が処理する概要は以下になる．

1. ある法が成立する状況を仮定する

2. 仮定した状況に対して，対立する法が同時に適用されないか検査する

以下の節で，この 2 つの工程について説明する．

```
Input: C: 対立概念集合
Input: K^v: 検証用法的知識データ
Input: K^o: 原文参照データ
Output: C^p: 証明済み対立概念集合
Output: R^c: 検出された対立に使用されていた規則の集合
begin
    C^p := ∅;
    /* 対立概念を選ぶ */
    forall (α, β) ∈ C do
        if A := arity(K^v, α) = arity(K^v, β) ≠ ∅ then
            forall n ∈ A do
                /* ここで c_1, ..., c_n は定数 */
                P_α = p_α(c_1, ..., c_n) ;
                P_β = p_β(c_1, ..., c_n) ;
                K'^v := K^v ;
                if P_α|P_β ··· ← L_1, ..., L_n ∈ K^o then
                    /* OR の規則削除 */
                    K'^v := K'^v \ {P_α ← L_1, ..., L_n};
                    K'^v := K'^v \ {P_β ← L_1, ..., L_n};
                end
                E := get_evidence(K'^v, P_α) ;
                E' := ∅;
                forall E_s ∈ E do
                    /* not 付きの要素削除 */
                    E'_s := ∅;
                    forall f ∈ E_s do
                        if f ≢ not lit then
                            E'_s := E'_s ∪ {f} ;
                        end
                    end
                    E' := E' ∪ {E'_s} ;
                end
                if E' ≠ ∅, ∀E'_s ∈ E'[E'_s ≠ ∅, E'_s ∪ K^v ⊢ P_β] then
                    /* 証明済み対立概念の記録 */
                    C^p := C^p ∪ {(α, β)};
                    /* β 証明時に使用されていた規則はマークされている．関数 find はその記録情報をただ集めてくる．マークは証明の
                       たびに毎回クリアされている． */
                    R^c_(α,β) = find();
                    R^c := R^c ∪ {((α, β), R^c_α,β)};
                end
            end
        end
    end
end
```

手続き 6 法的知識検証手続き

```
Data: G: 要求
Data: P: 論理プログラム
Result: A^main: 仮説集合
begin
    A^main := ∅ ;
    if G ≡ not G' then G^lit := G' else G^lit := G;
    if G^lit ← L_1, ..., L_n ∉ P then                                   /* 末端処理 */
        A^main := {{G}} ;
    else                                                                /* 規則処理：再帰呼び出しによって仮説を収集 */
        forall G^lit ← L_1, ..., L_n ∈ P do
            A^rule := ∅;
            for i = 1 to n do
                A^rule_temp := ∅ ;
                E^clause := get_evidence(P, L_i) ;
                forall A_s ∈ A^rule do
                    forall E_s ∈ E^clause do
                        A^rule_temp := A^rule_temp ∪ {A_s ∪ E_s};
                    end
                end
                A^rule := A^rule_temp;
            end
            A^main := A^main ∪ A^rule ;
        end
    end
    /* 仮説生成中の矛盾検査 */
    forall A_s ∈ A^main do
        if (α, not α ∈ A_s) | (α, ¬α) ∈ A_s then
            A^main := A^main \ {A_s};
        end
    end
    /* ヘッドが NAF で呼び出されたときの仮説の展開 */
    if G ≡ not G' then
        A^dir := ∅;
        forall A_s ∈ A^main do
            m := length(A_s);
            S_1 := ··· := S_m := ∅;
            i := 1;
            forall f ∈ A_s do
                if f ≡ not f' then
                    S_i := {not f', f'}
                else
                    S_i := {not f, f}
                end
                i := i + 1;
            end
            /* '×̇' は非順序対に関する直積 */
            A^dir := A^dir ∪ {S_0 ×̇ ... ×̇ S_m};
        end
        A^main := A^dir \ A^main ;
    end
    return A^main;
end
```

手続き 7 関数 get_evidence()

4.5.1 状況の仮定

法準則はある状況に対して画一的な適用を目的として条件プログラムの形で記述される．そして整合的な法とは，ある事実から対立する結論を生み出さないことである．条件プログラムは，前提条件が存在した場合，無条件で適用されるも

のである．したがって，もしある状況に対して対立するような法準則が同時に適用可能であるならば，それは法が対立を含むことになり，これを排除する必要がある．

この法の整合性を検証するためには上記に表されているように，ある状況に対して対立する法準則が同時に適応されていないか調べる必要がある．この「ある状況」を得るために，アブダクションを用いる．具体的には，まず検証する対立概念を決定する．次に片方の対立概念の概念名を結論に持つような法準則をその検証の軸とし，その法準則を成り立たせるような仮説を得る．例えば法準則 A と B が存在し，それぞれが結論において互いに対立するものを持っていたとする．その場合片方の法準則 A を一時的に「正しい法準則である」と仮定する．そして法準則 A の結論を要求としてアブダクションを行う．これにより法準則 A が適用される仮説，すなわち A が適用される「状況」を得る[3]．

この得られた仮説について，法準則 A に対し対立するような法準則が成り立たないことを検査をする．もし対立する法準則が成り立たなければ法準則 A が適用される状況に対して，法は少なくとも検証用法的知識データ上本稿が検証する基準について整合的であったといえる．この手続きを以下で手続き 6，7に準じて説明する．

まず手続き 6で対立概念データから検証する対立概念を 1 つ選ぶ．このとき取り出される対立概念は，‘(α, β)’ のような概念名の対である．なお検証作業は対立概念全てに対して実行されるので，どれを選ぶかを気にする必要はない．

次に対立概念 (α, β) から仮説を得るためどちらを軸とするかを選ぶ．ここではその軸を α にすることにする．なお対立概念の組 1 つに対する検証は，α を軸にした場合と，β を軸にした場合それぞれに対して行われるので，この決定の方法は気にする必要はない．

この α は概念名である．よって手続き 6でアリティデータを用い α からアトムを生成する．その変換で，α のアリティが n とするとき，$\alpha(x_1, \ldots, x_n)$ のようなアトムが作られる．このアトムへの変換方法は 4.3.2 章での補助データ生成に使われたものとほぼ同様である．ただし引数には変数ではなく，定数が割り当てられる

[3]こうして仮説もとにした知識の無矛盾性維持システムの研究もある [48, 51].

点が異なる[4]．以後この概念名から作られたアトムを $Pred_\alpha$ と記述する．

生成されたアトム $Pred_\alpha$ は手続き 7に渡され，$Pred_\alpha$ の仮説とそれにより成り立つ事実の集合を得る．その手続き 7の処理概要は以下になる．

1. $Pred_\alpha$ の証明に用いられる規則 P_α を検証用法的知識データから選定する．このときヘッド部の論理和によって利用可能な規則とそうでないものが存在する．

2. P_α^B を P_α のボディ部に含まれるアトムの集合，P_α^H を P_α のヘッド部に含まれるアトムの集合とする．このとき，$Ab_\alpha = P_\alpha^B \setminus P_\alpha^H$ を仮定可能な集合とする．

3. P_α から $Pred_\alpha$ を演繹するために必要な事実 F_α を Ab_α から選び，$P_\alpha \cup F_\alpha$ で成り立つ事実の集合を求める．

項 (1) では仮説を得るために P_α を検証用法的知識データから選定する．このとき P_α は自由に検証用法的知識データから選ばれない．P_α の選定は 4.3.1 章で述べたように，原文参照用データを用いてヘッド部の論理和が考慮される．これによって制限された検証用法的知識データから選ばれることになる．具体的には検証対象の対立概念の組が (α, β) であるとする．このとき以下のように，アトム $Pred_\alpha$ を論理和でヘッド部に含む規則が原文参照データに存在すると仮定する．

$$Pred_\alpha \,|\, Pred_\theta \cdots \leftarrow L_1, \ldots, L_n \tag{4.1}$$

このような規則が原文参照データにある場合，4.3.1 章の規則分割によって以下の規則が検証用法的知識データに含まれている．

$$\left\{ \begin{array}{c} Pred_\alpha \leftarrow L_1, \ldots, L_n \\ Pred_\beta \leftarrow L_1, \ldots, L_n \\ \vdots \end{array} \right\} \tag{4.2}$$

[4] これは得られる状況の変数を定数で束縛するためである．

以上のように検証対象である対立概念 $Pred_\alpha$ をヘッド部に論理和で含む規則（規則 4.1）が原文参照データにあるとき，それを分割して得られた規則（規則 4.2）は P_α に含めない．それは，ヘッド部の論理和のもととなる法令用語「又は（若しくは）」の意味に起因する（4.3.1 章）．法令用語の「又は」,「若しくは」は，法律効果の**選択** [92] を記述するために用いられる．よってそれらを用いて並記された法律効果は複数を同時に適応することは前提としない．このことから法準則でのヘッドの論理和は，論理学上の論理和ではなく，排他的論理和と見なすことが妥当である．つまり‘$Pred_\alpha | Pred_\gamma \leftarrow L_1, \ldots, L_n$’という規則を用いて法律効果を得る場合，ヘッド部にある法律効果 $Pred_\alpha$ と $Pred_\gamma$ は排他的論理和によって選択的に選ばれ適用されることになる．すなわちこの規則のヘッド部にある対立概念はすでに排他的な関係であり，この規則から分割して得られた規則に関しては検証する必要がない．例えば，

$$\text{拒否} \mid \text{許可} \leftarrow \text{申請} \tag{4.3}$$

などという法準則が仮に存在した場合，それを分割して得られる規則

$$\text{拒否} \quad \leftarrow \text{申請} \tag{4.4}$$

$$\text{許可} \quad \leftarrow \text{申請} \tag{4.5}$$

これらが対立を起こしていないか検査する必要はないという意味である．ただし，検証が (α, β)（上記例では，‘拒否’ または ‘許可’ を含む対立概念）以外の対立概念について行われるときは，これらの規則も用いて検証される．なお実際の P_α の選定と規則の除外は，検証用コード `head_hook` と `body_hook` による内部割り込み処理によって行われている．

次に (2) で仮定可能集合 Ab_α が，P_α に含まれるアトムの集合から選ばれる．理論的には以下のような範囲がその仮定可能集合になる．まず P_α に含まれる規則のヘッド部に出現するアトムの集合を P_α^H，またボディ部に出現するアトムの集合を P_α^B とする．このとき Ab_α は $Ab_\alpha = P_\alpha^B \setminus P_\alpha^H$ となる．つまり Ab は P_α を用いて $Pred_\alpha$ を証明するとき用いられるアトムであり，かつそれ自身をヘッドに持つ規則がないアトムの集合といえる．実際の処理ではヘッド部に存在しないアトムの実行が行われたとき，検証用コードによって仮説としてそのアトムが収集されることになる．

最後に項 (3) について説明する．手続き 7は仮説を求めた後，それによって証明可能になる事実の集合を集める．これを求めるのは，検証する際の実装上の問題である．検証の論理的構造は，要求 $Pred_\alpha$ に対する仮説を検証用法的知識データに加え，要求 $Pred_\alpha$ と対立する概念から作られたアトム $Pred_\beta$ が同時に証明可能かを検査することと等しい．しかし $Pred_\beta$ を証明するとき，規則を使用して仮説に帰着させるよりも証明途中で証明可能な事実集合に帰着させた方が効率がよい．その観点から現時点ではこのように証明可能事実の集合を得ている．また規則の利用がシーケンシャルな方法ですむことも理由としてあげられる．なおこの方法は，計算量を減らすかわりにメモリ効率を犠牲にしている．この点のより効率的な実装方法は今後の課題でもある．

以下にこれらの手順の実行例を挙げる．

例 5 今原文参照データ $L^{\mathcal{O}}$ が以下のようになっているとする．

$$L^{\mathcal{O}} = \left\{ \begin{array}{l} \alpha \leftarrow \beta, not\, \chi \\ \alpha \,|\, \varphi \leftarrow \gamma \\ \chi \leftarrow \neg\delta, not\, \varepsilon \end{array} \right\} \tag{4.6}$$

このとき検証用法的知識データ $L^{\mathcal{V}}$ は以下のようになっている．

$$L^{\mathcal{V}} = \left\{ \begin{array}{l} \alpha \leftarrow \beta, not\, \chi \\ \alpha \leftarrow \gamma \\ \varphi \leftarrow \gamma \\ \chi \leftarrow \neg\delta, not\, \varepsilon \end{array} \right\} \tag{4.7}$$

この検証用法的知識データ $L^{\mathcal{V}}$ において，対立概念を (α, β)，要求を α とし，議論 $A_\alpha = \langle P_\alpha, F_\alpha, \alpha \rangle$ を得ることを考える．議論 A_α は，支持 P_α が要求の証明に利用される規則の集合，事実 F_α が仮説に相当する．証明可能事実はこの P_α と F_α から証明可能な命題に相当する．まずは支持である規則の集合 P_α を選ぶ．支持は上記で述べたようにProlog処理系の実行により選択されるので，ここでは知識 4.7からその除外規則を選定することになる．そこで原文参照データ $L^{\mathcal{O}}$ を調べる．$L^{\mathcal{O}}$ には規則 ‘ $\alpha \,|\, \varphi \leftarrow \gamma$ ’ が含まれている．これから，アトム α, φ がその規則の同一

のヘッド部に存在することが分かる．したがって $L^{\mathcal{V}}$ に含まれている，規則分割で生成された規則 '$\alpha \leftarrow \gamma$' と '$\varphi \leftarrow \gamma$' が議論 A_α における支持の対象からは外される．よって，A_α における仮説を得るために用いられる規則の集合 P_α は以下のようになる．なおその P_α を $L^{\mathcal{V}}_\alpha$ と記述する．

$$L^{\mathcal{V}}_{(\alpha,\varphi)} = \left\{ \begin{array}{l} \alpha \leftarrow \beta, not\, \chi \\ \chi \leftarrow \neg\delta, not\, \varepsilon \end{array} \right\}$$

規則の選別が行われた後，この知識から仮説を得てそれを基にした証明可能事実の集合を得る．まずこの知識集合に対し要求 α が実行される．ここで規則 $\alpha \leftarrow \beta, not\, \chi$ がヘッド部に α を持つので，そのボディ部が実行される．ボディ部の実行順序は処理系によるが，ここでは $not\, \chi$ が呼び出されるものとする．not は述語記号のひとつとして実装されており，その引数 χ をボディ部とする規則と同じ実行を行う．したがって次に χ をヘッド部に持つ規則が実行されることになる．

呼び出された χ の規則はボディ部それぞれがそれらをヘッド部に持つ規則を持たない．したがって仮説可能な事実になる．そこでそのボディ部の節それぞれに仮説を埋め込む．その結果は以下のようになる．なお χ をヘッド部に持つ規則は not によって呼び出されている．そこでヘッド部には「$not\, \chi$ でこの規則は呼び出された」という印を便宜上つけている．実際の実行もこれとほぼ同様に not で呼び出されたことを記録している．

$$\alpha \leftarrow \beta, not\, \chi$$
$$\chi^{not} \leftarrow \{\{\neg\delta\}\}, \{\{not\, \varepsilon\}\}$$

上記のようにボディ部にはそれぞれの節にその節を充足するための仮説が集められる．上記例では便宜上それを集合の形で記述している．よって議論 A_α における事実 F_α と支持の規則集合 P_α を得たことになる．

次にこれを用いて順次証明可能事実の集合をボトムアップに構築していく．まずはヘッド部 χ を充足するための事実を得るため節の仮説の直積をとる．それは以下のようになる．

$$\alpha \leftarrow \beta, not\, \chi$$
$$\{\{\neg\delta, not\, \varepsilon\}\}^{not} \leftarrow \{\{\neg\delta\}\}, \{\{not\, \varepsilon\}\}$$

ヘッド部の χ 部分はこのように集合族になる．この集合の論理的意味は，集合族の要素であるそれぞれの集合は論理和でつながれる．またその集合の要素同士は論理積でつながれているものとする．たとえば，

$$\{\{a, b\}, \{c\}\}$$

これは，

$$(a \wedge b) \vee c$$

を表す.

この χ の規則は $not\, \chi$ として呼び出されていた．そのため以下のようにその集合は展開が行われる．

$$\alpha \leftarrow \beta, not\, \chi$$
$$\{\{not\, \neg\delta, not\, \varepsilon\}, \{\neg\delta, \varepsilon\}, \{not\, \neg\delta, \varepsilon\}\}$$
$$\leftarrow \{\{\neg\delta\}\}, \{\{not\, \varepsilon\}\}$$

これはもとの集合に対してべき集合をとるものとほぼ等しい．ただ異なる点は，そのべき集合は各要素に対して not をつけるか否かによって行われる．これによって α の規則のボディ部に $not\, \chi$ の証明可能事実集合が与えられる．これから β の仮説と合わせ以下のようになる．

$$\alpha \leftarrow \{\{\beta\}\}, \{\{not\, \neg\delta, not\, \varepsilon\}, \{\neg\delta, \varepsilon\}, \{not\, \neg\delta, \varepsilon\}\}$$

最後にそれらがまとめられ，α に対する仮説を用いた証明可能事実集合は以下になる．

$$\left\{\begin{array}{r}\{\beta, not\ \neg\delta, not\ \varepsilon\},\\ \{\beta, \neg\delta, \varepsilon\},\\ \{\beta, not\ \neg\delta, \varepsilon\}\end{array}\right\} \leftarrow \left\{\begin{array}{r}\{not\ \neg\delta, not\ \varepsilon\},\\ \{\neg\delta, \varepsilon\},\\ \{not\ \neg\delta, \varepsilon\}\end{array}\right\}$$

得られた証明可能事実集合からは not のつく事実は捨てられる．結果，$\{\beta\}$, $\{\beta, \neg\delta, \varepsilon\}$, $\{\beta, \varepsilon\}$ が α の証明可能事実集合として得られる．

4.5.2 検証の工程

対立概念の片方の議論 Arg_α を得た後，検証用法的知識データと Arg_α から対立概念のアトム $Pred_\beta$ が証明できるかを調べる．もし証明が可能であった場合，$Pred_\alpha$ を結論に持つような法準則が適用される状況下では，対立する $Pred_\beta$ を結論に持つ法準則もまた画一的に適用されることを意味する．その場合は，検証用法的知識データが対立概念 (α, β) において対立を含むことを示唆する．

この検証は以下のように表される．なおこのとき $L^{\mathcal{V}}$ は検証用法的知識データとする．

$$F_\alpha \cup L^{\mathcal{V}} \vdash Pred_\beta$$

これにより対立が検出されると，少なくともその法令文には，人間が確認すべき条項が存在する．したがって，その条項の情報がユーザに提示される．

その提示箇所とは，$Arg_\alpha = \langle F_\alpha, P_\alpha, Pred_\alpha\rangle$ と，それに対立する議論 $Arg_\beta = \langle F_\alpha, P_\beta, Pred_\beta\rangle$ において，$P_\beta \setminus P_\alpha$ の規則であり，さらにボディ部に証明可能事実の要素を含むものとしている．これを提示部分とする理由は，議論 Arg_α と Arg_β の支持が F_α に依拠していることによる．F_α は Arg_α の $Pred_\alpha$ が適用される状況を仮定したものである．つまり F_α を得るためには，P_α，つまり，$Pred_\alpha$ を結論に持つ法準則と，それにかかわる法が正しくなければならない．さもなければ，その得た仮定そのものが誤った法を基にして得たこととなり，その仮定を基礎に検証を行った結果そのものの正当性，妥当性が失われる．したがって，仮定された状況が F_α であるならば，P_α はそのときの仮定を用いる検証時では暫定的に正しい

とする必要がある．つまり対立する法準則がなりたった場合，誤りを含んでいるのは仮定を得ることに使用しなかった対立する法準則にかかわる部分であるということである．以上の理由から，対立する議論 Arg_β の当該箇所を要確認箇所とした [75]．なおこのことから分かるように，本研究が検出する対立には方向性がある．これは矛盾の原因に順序をつけられる可能性があるため，これを利用することで提示箇所をより絞ることが期待できる．

本稿では矛盾の検出にこのような対立する議論の構造を利用しているが，矛盾する知識から議論を得る研究 [12] などもある．今後はこのような研究を採用し，対立する議論をどちらが基軸ということなく取り出すことが必要であるとも考える．それにより対立概念の組の要素を区別することなく矛盾の箇所を提示できる可能性がある．

最後に検証に用いた仮説について述べる．仮説は要求に対して複数得られる可能性がある．前節で証明可能事実の集合が複数あることがそれを示している．本稿ではこの全ての証明可能事実に対して検証を行った．そしてもし全ての証明可能事実について対立が含まれるならば，その結果を提示することにしている．これは [49] に見られるような「懐疑的な結論」を採用した結果である．

例 6 対立に関する検証の例を図 4.3を用いて説明する．図 4.3は対立の検証工程を表す図であり，矢印は論理プログラムの含意を表す．検証用法的知識データ $L_\mathcal{V}$ が以下の論理プログラムの集合であるとする．

$$L_\mathcal{V} = \left\{ \begin{array}{r} P_0(X,Y) \leftarrow P_1(X,Y). \\ P_1(X,Y) \leftarrow P_2(X,Y). \\ P_2(X,Y) \leftarrow P_3(X), P_4(Y). \\ P_6(X,Y) \leftarrow P_3(X), P_5(Y). \\ P_5(X) \leftarrow P_4(X). \end{array} \right\}$$

この検証では対立概念の集合から，(P_0, P_6) の組が選ばれたと仮定している．まず仮説を得るために軸を P_0 にしたものとする．概念名 P_0 からはアトムが作られる．この場合アリティデータから得られたアリティが 2 であったとすると $P_0(X,Y)$ が作られる．なお $P_0(X,Y)$ の引数には，適当な定数が割り振られるため，例えば

$P_0(a,b)$ のようなアトムが実行される．その $P_0(a,b)$ を実行中に，末端のアトムが仮説として得られる．それにより仮説を F_{P_0}，使用される規則 P_{P_0}，要求 $P_0(a,b)$ として，議論 $Arg_{P_0} = \langle F_{P_0}, P_{P_0}, P_0(a,b) \rangle$ が構成される．なおその仮説 F_{P_0} は，以下になる．

$$F_{P_0} = \left\{ \begin{array}{l} P_3(a) \\ P_4(b) \end{array} \right\}$$

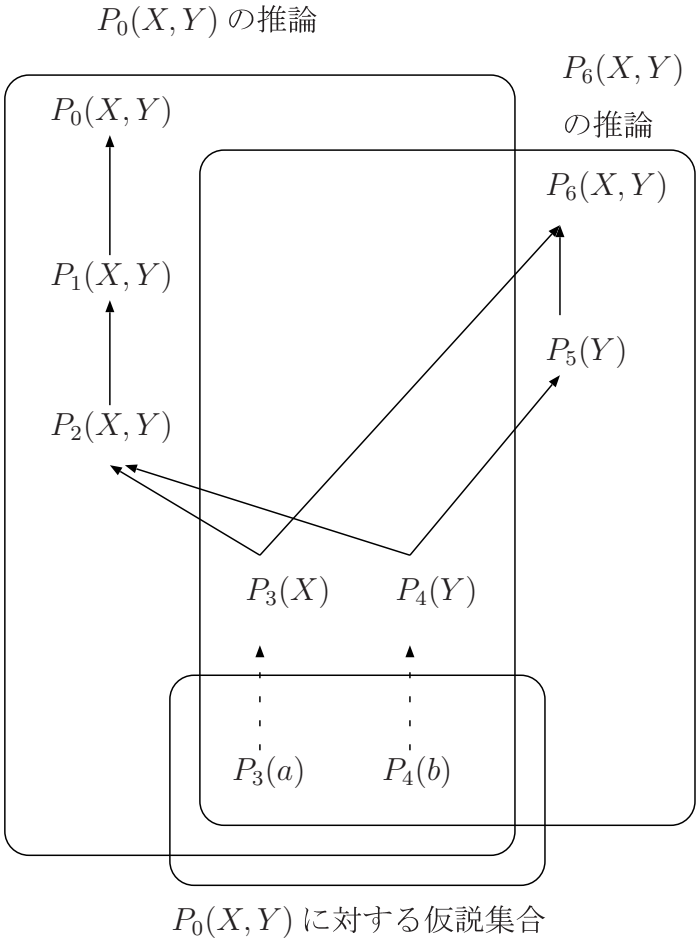
この議論に対する証明可能事実 Ans_{P_0} を得る．それは以下になる．

$$Ans_{P_0} = \left\{ \begin{array}{r} P_0(a,b) \\ P_1(a,b) \\ P_2(a,b) \\ P_3(a) \\ P_4(b) \end{array} \right\}$$

次に対立概念のもう片方の概念名 P_6 からアトム $P_6(X,Y)$ が作られ，それが成り立つか検証が行われる．この生成されるアトムには，$P_0(a,b)$ によって作られた証明可能事実に単一化するため，引数に変数が置かれる．そして $P_6(a,b)$ がもし Arg_{P_0} によってなりたてば，この知識集合は対立概念 (P_0, P_6) について P_0 から P_6 への対立を含んでいることと判断される．よってユーザには結論 P_6 にかかわる規則として $P_3 \rightarrow P_6$, $P_4 \rightarrow P_5$ が要確認箇所として提示されることになる．なお同様の検証が P_0 と P_6 を入れ替えてもう一度行われる．

```
<?xml version="1.0" encoding="utf-8" ?>
<ordinance id="富山県行政手続条例" xmlns="http://www.webgen.co.jp/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.webgen.co.jp/ ordinance-logic.xsd">
  <section id="第 8 条">
    <subject>理由の提示</subject>
    <paragraph id="第 1 項">
      <logic lang="xkif">
        <implies>
          <clause>
            <predicate value="申請行為"/>
            <argument number="1"><var name="x"/></argument>
            <argument number="2"><var name="y"/></argument>
            <argument number="3"><var name="z"/></argument>
          </clause>
          <clause>
            <predicate value="許認可等"/>
            <argument number="1"><var name="z"/></argument>
          </clause>
          <clause>
            <predicate value="申請者"/>
            <argument number="1"><var name="x"/></argument>
          </clause>
          <clause>
            <predicate value="行政庁"/>
            <argument number="1"><var name="y"/></argument>
          </clause>
        </implies>
      </logic>
    </paragraph>
  </section>
</ordinance>
```

図 4.2: XKIF の実例



対立概念の組: (P_0, P_6)

図 4.3: 検証過程の例

第 5 章

検証結果

使用したデータはXML化した富山県の条例（富山県条例54号第1条－第10条，富山県条例55号第1条－第7条，富山県行政手続条例，富山県手数料条例，富山県職員旅費条例，富山県税条例登山届出条例）とそれを基に作られた語彙の上位下位関係を用いた[1]．

このデータから作られた検証用法準則データには規則が 278 個があった．上位下位関係語彙データからは対立概念 11602 組が計算された．

実行環境は，CPU:Intel(R) Xeon(R) CPU X3350 @ 2.66GHz，Memory:3.2GB，OS: WindowsXP SP3，前処理器:ruby 1.8.7 (2008-06-20 patchlevel 22) [i386-cygwin]，検証器:SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.52) である．実行時間は本実験規模において全てを通し約 1 分程度である．

実験結果について説明する．循環部分として 2 つの箇所が検出された．以下にその出力結果を載せる．

[1]本データの作成は（株）インテック・ウェブ・アンド・ゲノム・インフォマティックスに依頼した [95].

```
Cycle:[人事委員会の規則, 公安委員会の規則, 収用委員会の規則, 委員会の規
則, 教育委員会の規則, 規則]
人事委員会の規則 -> 委員会の規則 :is OWL.
公安委員会の規則 -> 委員会の規則 :is OWL.
収用委員会の規則 -> 委員会の規則 :is OWL.
委員会の規則 -> 規則 :is OWL.
教育委員会の規則 -> 委員会の規則 :is OWL.
規則 -> 人事委員会の規則 :is FOL relation.
条例: 富山県条例第 54 号, 第 10 条
名前: 規則
号数: 第 1 項

規則 -> 公安委員会の規則 :is FOL relation.
条例: 富山県条例第 54 号, 第 10 条
名前: 規則
号数: 第 1 項

規則 -> 収用委員会の規則 :is FOL relation.
条例: 富山県条例第 54 号, 第 10 条
名前: 規則
号数: 第 1 項

規則 -> 教育委員会の規則 :is FOL relation.
条例: 富山県条例第 54 号, 第 10 条
名前: 規則
号数: 第 1 項


Cycle:[副書, 申請内容, 申請書]
副書 -> 申請書 :is OWL.
申請内容 -> 副書 :is OWL.
申請書 -> 申請内容 :is FOL relation.
条例: 富山県行政手続条例, 第 8 条
名前: 理由の提示
号数: 第 1 項
```

図 5.1，5.2は出力結果から得られたそれぞれの閉路を図示したものである．

図 5.1では規則にある

- 規則 → 人事委員会の規則

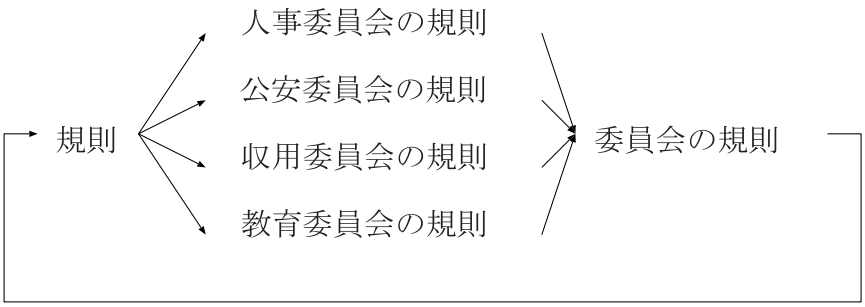


図 5.1: 閉路 1

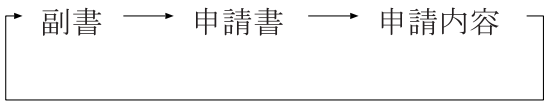


図 5.2: 閉路 2

- 規則 → 公安委員会の規則
- 規則 → 収用委員会の規則
- 規則 → 教育委員会の規則」

に関係する包含関係が誤りだと判別された．出力結果から分かるようにいずれも法準則が誤りの原因であった．ただしそれぞれ該当する原文を参照したところ原文に問題は見つからなかった．したがってこれは原文を法準則知識に直す時点で混入した法準則データの記述誤りであると判断される．

図 5.2では上位下位関係知識

- 申請内容 → 副書

が誤りであると判別された．上位下位関係語彙データは人手によって作成されたもので，さらに法準則データのような基礎となるものを用いていない．よって作成した人間が誤ってこの関係を作成したものと考えられる．

上記で検出された循環を取り除いた法的知識について対立の検出を行ったがこれは検出されなかった．よって法の論理的構造がもたらす対立による矛盾は法にはない可能性が高いと考えられた．ただし完全に法に矛盾がないと保証できるものではない．これはあくまで本稿の手続きでモデル化した法的知識に対して検出

可能な対立が存在しなかったことを意味するものである．しかしながら本稿がこの検証に用いた法令文は当然ながら公布されているものであり，矛盾がないよう専門家により検査が行われたものである．よって本件で対立が検出されなかったことは妥当とも考えられる．

なおシステムが正常に動作しているか確認のため人為的に誤りを含んだ法準則データも用意し，それについて実験も行った．当該データにおいてその対立部分が正常に検出することができた．これにより，検証に用いた法準則データから対立が検出できなかったことは，システム上の誤りではないことが確認できた．

第 6章

まとめと課題

本論文の寄与するところは以下の 3 点にまとめられる.

1. 法的知識の矛盾検出に対し対立の定義を導入し，またそれを実用性のある手法とするため上位下位関係から対立概念を抽出しそれを使用した.
2. 検証システムとして XML データから必要な情報を取り出し形式を変換する前処理器と，実際に検証を行う検証器を実装した.
3. 論理プログラムの技術を実際に応用し，現実の問題として富山県条例の検証をした.

項 (1) について述べる．まず矛盾の定義に対立を導入したことが挙げられる．対立の導入により論理学上の矛盾である矛盾律を包括しながらも，より人間の直観に近い矛盾も検出できるようにした．ただしこの対立の導入には，基準となる対立概念を定義する必要がある問題が明らかになった．そこで上位下位概念データを用いて近似的にその対立概念を計算で求める方法を提示した．これにより，対立概念をすべて人の手により定義する必要がなくなり，少ない労力で対立概念を用いることができた.

矛盾の検証方法に関してはアブダクションや対立する議論などの導入を行った．この導入により法的知識が内部で矛盾を起こしているかどうかの判定のみならず，具体的に矛盾の原因箇所をユーザに提示することができた．なお対立は実験で検出されなかったので具体的な事例を挙げることができない．しかし恣意的に用意したデータを用いてその原因箇所を提示できることを確認した.

項 (2) について述べる．本稿は提案した検証手続きに対する具体的な実装を示した．ここでは提案手続きの実装だけではなく，用いるデータの記述形式や形式化方法，また検証可能な法の要素これらの基礎的なことについても提案した．これらにより手続きの実装可能性と実効性を示したことに加え，実装上問題になることを明らかにもした．具体的には，概念名から述語を生成する際に引数に型情報が有用であることや，計算によって求められる対立概念の精度，また法準則の記述表現力と実行速度の関係などである．

項 (3) について述べる．本稿の実装はある程度の大きさの法的知識に対して，現実的な時間で検証を実行できることを示した．ただし正確な計算量などは現時点では求めていない．ただし級数的に計算量が増えることは自明である．

実験結果については，対立は検出されなかったが恣意的なデータでは対立箇所を提示することができた．さらに，循環する規則の具体的な箇所をユーザに提示することができた．この検出された箇所は，上位下位関係語彙データから追加された補助データによるもの，また法準則データが原因になった箇所であった．これにより，本手続きおよびシステムは法令文の検証において補助として機能できることが示唆される．

今後の課題としては以下が挙げられる．

まず知識記述方法における表現能力の拡大である．現時点では対立の導入により人の直観に近い矛盾の検出には対応した．しかしより厳密に法の矛盾を検出するためには，規則に対し表現力の高い形式化方法，またそれと実用性を両立させた検証手続きを検討する必要がある．そのためには，実際に法的知識から法的結論を導くような研究 [65,68] でどのような情報が用いられるのか，また要求されるのか，これらについてさらに詳しく調べる必要がある．また論理プログラムそのものに対して矛盾を検出する研究もある [19]．Prolog の構文を知識の表現として用いるのであれば，このような技術も取り入れることが必要である．

法的知識が起こす対立には方向性がある．これは 4.5.2 章で述べたように，検証をする際にある法律効果が適応されることを前提として状況を仮定し，そしてその状況をもとに検証が行われることによる．この方向性は議論の起こす対立の方向とその対立の順序関係を表すことができる．このため対立が検出されたとき，ユーザに提示される修正点の候補，これを絞ることに今後利用可能ではないかと考える．

法の検証結果はその法の修正を促すものである．もし提示箇所が実際に修正を必要とする部分であったならば，それは同時に改正法の必要性または，その追加法の修正を意味する．これは１章でも述べたが，検証結果を受けた修正に対しては再帰的な検証が必要となる．このような改修される知識集合の検証において，非単調推論における知識集合の仮説推論やまた対立議論の生成などが考慮できる [44,70]．今後このような研究を採用し，時系列的な検証も考える必要がある．

現時点では検証を法令文そのものに対して行っていない．あらかじめ人が法令文を読み，それを法的知識として実装したものに対して行っている．法令文は比較的，自然言語でありながらも形式的な記述に則っている．よって今後は自動で論理的構造を抽出できるような研究が必要と考える．

謝辞

本研究を行なうに当たり, 終始御指導を賜わった東条 敏教授に深謝致します.

最後に, 本論文をまとめるに当たって御協力いただいた東条研究室の諸兄に厚く御礼申し上げます.

参考文献

[1] *ruby 1.8.7*. http://www.ruby-lang.org/ja/.

[2] *SWI-Prolog Version 5.6.52*, 2006. University of Amsterdam, http://www.swi-prolog.org/.

[3] Abdullatif A.O. Elhag, Joost A.P.J. Breuker, and Bob W. Brouwer. On the formal analysis of normative conflicts. *Information & Communications Technology Law*, 9(3):207–217(11), 2000.

[4] Lennard Aqvist. Deontic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic: Volume II Extensions of Classical Logic*. Kluwer, 1994.

[5] B. H. Slater. Paraconsistent logic? *Journal of Philosophical Logic*, 25:451–454, 1995.

[6] Franz Baader and Werner Nutt. *Basic Description Logics. In the Description Logic Handbook*. Cambridge University Press, 2002.

[7] Christoph Beierle, Ulrich Hedtstück, Udo Pletat, Peter H. Schmitt, and Jörg H. Siekmann. An order-sorted logic for knowledge representation systems. *Artificial Intelligence*, 55:149–191, June 1992.

[8] Rachel Ben-eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[9] Trevor Bench-Capon and Reans Coenen. The maintenance of legal knowledge based systems. *Artificial Intelligence Review*, 6:129–143, 1992.

[10] Trevor Bench-Capon and Henry Prakken. Argumentation. In A.R. Lodder and A. Oskamp, editors, *A.R. Lodder & A. Oskamp (eds.): Information Technology & Lawyers: Advanced technology in the legal domain, from challenges to daily routine, Berlin*, pages 61–80. Springer Verlag, 2006.

[11] Trevor Bench-Capon and Henry Prakken. Introducing the logic and law corner. *Journal of Logic and Computation*, 18:1–12, 2008.

[12] Salem Benferhat, Didier Dubois, and Henri Prade. Argumentative inference in uncertain and inconsistent knowledge bases. In *In Proceedings of the 9th Uncertainty in Artificial Intelligence*, pages 411–419. Morgan Kaufmann, 1993.

[13] Leopoldo Bertoss, Anthony Hunter, and Torsten Schaub. Introduction to inconsistency tolerance. In *LNCS3300*. Springer-Verlag Berlin Heidelberg, 2004.

[14] Yves Bertot and Pierre Castan. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. EATCS Texts in Theoretical Computer Science. Springer Verlag, 2004. ISBN 3-540-20854-2.

[15] Philippe Besnard, Luis Fariñas del Cerro, Dov M. Gabbay, and Anthony Hunter. *Logical Handling of Inconsistent and Default Information*, chapter 11: Logical Handling of Inconsistent and Default Information, pages 325–342. Springer, 1996.

[16] Garrett Birkhoff. *Lattice Theory*. AMS Bookstore, 3 edition, 1979. ISBN 0821810251, 9780821810255.

[17] A. Borgida and R. J. Brachman. *Conceptual Modelling with Description Logics. In the Description Logic Handbook*. Cambridge University Press, 2002.

[18] Gerhard Brewka. Logic programming with ordered disjunction. In *In Proceedings of AAAI-02*, pages 100–105. Morgan Kaufmann, 2002.

[19] Jan Chomicki. Conflict resolution using logic programming. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 15:244–249, 2003.

[20] Neophytos Demetriou and Antonis Kakas. Argumentation with abduction. In *the 4th Panhellenic Symposium on Logic*, 2003.

[21] Uwe Egly and Hans Tompits. Proof-complexity results for nonmonotonic reasoning. *ACM Trans. Comput. Logic*, 2(3):340–387, 2001.

[22] Christoph Engel. *Inconsistency in the Law.* 2004. http://www.coll.mpg.de/pdf_dat/2004_16online.pdf.

[23] Anthony C. W. Finkelstein, Dov Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. Inconsistency handling in multiperspective specifications. In *IEEE Transactioys on Software Engiyeeking*, volume 20, 1994.

[24] Kathleen Freeman and Arthur M. Farley. A model of argumentation and its application to legal reasoning. *Artificial Intelligence and Law*, 4(3–4):163–197, 1996.

[25] Dov Gabbay and Anthony Hunter. Making inconsistency respectable 1: A logical framework for inconsistency in reasoning. *Fundamentals of Artificial Intelligence*, 1991.

[26] Dov M. Gabbay and A. Hunter. Negation and contradiction. In Dov Gabbay and Heinrich Wansing, editors, *What is Negation?*, pages 89–100. Kluwer Publishers, 1999.

[27] Michael Gelfond. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 12(1–2):89–116, 1994.

[28] Michael Gelfond. *Handbook of Knowledge Representation*, chapter Answer sets, pages 285–316. Elsevier Science Ltd, 2008.

[29] Michael Gelfond and Nicola Leone. Logic programming and knowledge representation - the a-prolog perspective. *Artificial Intelligence*, 2002.

[30] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP-88*, pages 1070–1080, 1988.

[31] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. In *New Generation Comput*, volume 9, pages 365–386, 1991.

[32] M. R. Genesereth and R. E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical report, Technical Report Logic-92-1, 1992.

[33] Nuno Graça and Paulo Quaresma. How to model legal reasoning using dynamic logic programming: Apreliminary report. In D. Bourcier, editor, *Legal Knowledge and Information Systems. Jurix2003: The Sixteenth Annual Conference*, 2003.

[34] H. Wang. Toward mechanical mathematics. *Automation of Reasoning*, 1:244–264, 1960.

[35] Jaap Hage. An alternative for deontic logic. *Legal Knowledge Based Systems. Aims for Research and Development*, pages 59–69, 1991.

[36] Jaap Hage. Monological reason based reasoning. *Legal Knowledge Based Systems. Model-based reasoning*, pages 77–91, 1991.

[37] Jaap Hage. Rule consistency. In *Legal Knowledge Based Systems. Jurix 1999: The Twelfth Conference*, 1999.

[38] Jaap Hage. Goal-based theory evaluation. In *Legal Knowledge and Information Systems. Jurix 2000: The Thirteenth Annual Conference*. IOS-Press, 2000.

[39] Jaap C. Hage. Formalizing legal coherence. In *ICAIL-2001*, 2001.

[40] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. pages 497–604, 1984.

[41] Alfred Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16:14–21, 1951.

[42] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. pages 723–731, 2004.

[43] Anthony Hunter. Paraconsistent logics. In D. Gabbay and Ph. Smets, editors, *Defeasible Reasoning and Uncertain Information*, volume 2. Kluwer, 1998.

[44] Katsumi Inoue and Chiaki Sakama. Abductive framework for nonmonotonic theory change. In Morgan Kaufmann, editor, *the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 204–210, 1995.

[45] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Logic Programming*, 35:39–78, 1998.

[46] Katsumi Inoue and Chiaki Sakama. Computing extended abduction through transaction programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):339–367, 1999.

[47] Katsumi Inoue and Chiaki Sakama. Disjunctive explanations in abductive logic programming. In Stephen Muggleton, editor, *Special Issue on Machine Intelligence 19, Electronic Transactions on Artificial Intelligence*, volume 7, 2004.

[48] Noboru Iwayama and Ken Satoh. Computing abduction by using tms with top-down expectation. *Journal of Logic Programming*, 44:179 – 206, 2000.

[49] J. F. Horty. Skepticisim and floating conclusions. *Artificial Intelligence*, 135:55–72, 2002.

[50] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Annual Symposium on Principles of Programming Languages Proceedings of the*

14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages t. ACM New York, NY, USA, 1987.

[51] Johan de Kleer. An assumption-based tms. *Artif. Intell.*, 28(2):127–162, 1986.

[52] Andrew J. I. Jones and Marek Sergot. Deontic logic in the representation of law: Towards a methodology. *Artificial Intelligence and Law*, 1(1):45–64, 1992.

[53] Antonis C. Kakas and M. Denecker. Abduction in logic programming, 2002.

[54] Antonis C. Kakas and Paolo Mancarella. Abductive logic programming. In *LPNMR*, pages 49–61, 1990.

[55] Ken Kaneiwa and Satoshi Tojo. An order-sorted resolution with implicitly negative sorts. In *International Conference on Logic Programming*, pages 300–314. Cyprus, 2001.

[56] Michael Kifer and Eliezer L. Loziniskii. A logic for reasoning with inconsistency. *Automated Reasoning*, 9:179–215, 1992.

[57] L. R. Horn. *A Natural History of Negation*, chapter 5, pages 268–273. CSLI Pblications, 2001.

[58] Adrienne Lehrer and Keith Lehrer. Antonymy. In *Linguistics and Philosophy*, volume 5, pages 483–501. Springer Netherlands, 1982.

[59] Catherine C Marshall. Representing the structure of a legal argument. *ICAIL '89: Proceedings of the 2nd international conference on Artificial intelligence and law*, pages 121–127, 1989.

[60] L. Thorne McCarty. A language for legal discourse i. basic features. In *ICAIL '89: Proceedings of the 2nd international conference on Artificial intelligence and law*, pages 180–189, New York, NY, USA, 1989. ACM.

[61] D. Nardi and R. J. Brachman. *The Description Logic Handbook Theory, Implementation and Applications*, chapter 1, pages 1–39. Cambridge University Press, 2003.

[62] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[63] Katsumi Nitta, Stephen Wong, and Yoshihisa Ohtake. A computational model for trial reasoning. In *ICAIL '93: Proceedings of the 4th international conference on Artificial intelligence and law*, pages 20–29, New York, NY, USA, 1993. ACM.

[64] Yasuhiro Ogawa, Shintaro Inagaki, and Katsuhiko Toyama. Automatic consolidation of japanese statutes based on formalization of amendment sentences. In *New Frontiers in Artificial Intelligence: JSAI 2007 Conference and Workshops*, 2007. Lecture Notes in Computer Science, Vol.4914, pp.363-376, Springer (2008. 2.).

[65] Kamalendu Pal and John A. Campbell. An application of rule-based and case-based reasoning within a single legal knowledge-based system. *The DATA BASE for Advances in information systems*, 28(4):48–63, 1997.

[66] Jeff Z. Pan and Ian Horrocks. Extending datatype support in web ontology reasoning. In *Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002)*, 2002.

[67] Jeff Z. Pan and Ian Horrocks. Extending datatype support in web ontology reasoning. In *workshop on OWL: Experience and Directions (OWL-ED2005)*, 2005.

[68] John L. Pollock. *Knowledge and Justification*. Princeton University Press, 1974.

[69] John L. Pollock. How to reason defeasibly. *Artificial Intelligence*, 57:1–42, 1992.

[70] John L. Pollock. Justification and defeat. *Artificial Intelligence*, 67:377–408, 1994.

[71] Henry Prakken. A logical framework for modelling legal argument. In *ICAIL '93: Proceedings of the fourth international conference on Artificial intelligence and law*, pages 1–9. ACM Press, 1993.

[72] Henry Prakken. Modelling reasoning about evidence in legal procedure. In *ICAIL-2001*, 2001.

[73] Henry Prakken. Analysing reasoning about evidence with formal models of argumentation. *Law, Probability and Risk*, 3:33–50, 2004.

[74] Henry Prakken. Ai & law, logic and argument schemes. In *Argumentation*, volume 19, pages 303–320. Springer Netherlands, 2005.

[75] Henry Prakken and Giovanni Sartor. A dialectical model of assessing conflict arguments in legal reasoning. *Artificial Intelligence and Law*, 4:331–368, 1996.

[76] Henry Prakken and Giovanni Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.

[77] Greg Restall. Paraconsistent logics! *Bulletin of the Section of Logic*, 26:156–63, 1997.

[78] Nico Roos. A logic for reasoning with inconsistent knowedge. *Artificial Intelligence*, 57:69–103, 1992.

[79] Kenneth A. Ross and Rodney W. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4(4):397–424, 12 1988.

[80] Fariba Sadri and F. Toni. *Abduction with Negation as Failure for Active and Reactive Rules*, volume LNAI 1489. Springer Verlag, 2000.

[81] Giovanni Sartor. Normative conflicts in legal reasoning. *Artificial Intelligence and Law*, 1(2–3):209–235, 1992.

[82] Giovanni Sartor. A simple computational model for nonmonotonic and adversarial legal reasoning. In *Proceedings of the 4th international conference on Artificial intelligence and law*, pages 192–201, 1993.

[83] Iikuo Tahara and S. Nobesawa. Reasoning from inconsistent knowledge base. *The IEICE Transactions on information and systems, PT.1*, J87-D-I(10):931–938, 2004.

[84] Yao Hua Tan and Leendert W.N. van der Torre. Representing legal knowledge in a diagnostic framework. 1996.

[85] Priyamvadha Thambu, Vasant Honaver, and Thomas Barta. Knowledge-base consistency maintenance in an evolving intelligent advisory system. In *FLAIRS 1998*, 1993.

[86] W3C. Owl web ontology language reference, 2004. http://www.w3.org/TR/owl-ref/.

[87] Kewen Wang. Argumentation-based abduction in disjunctive logic programming. *Journal of Logic programming*, 45:1–3, 2000.

[88] 兼岩 兼 and 東条 敏. イベントとプロパティの区別を導入した型階層論理. **日本ソフトウェア科学会**, 17(2):118–132, 2000.

[89] 兼岩 憲 and 佐藤 健. Dl: Description logic. **人工知能学会誌**, 18(1):73–82, 2002.

[90] 兼岩 憲 and 東条 敏. 法律知識の事象的/属性的読みを区別した推論システム. **情報処理学会論文誌**, 40:2892–2904, 1999.

[91] 兼岩 憲 and 溝口 理一郎. 形式オントロジーと順序ソート論理の拡張. **人工知能学会論文誌**, 20:387–395, 2005.

[92] 林 修三. **法令用語の常識 改訂版**. 日本評論社, 1975.

[93] 平野 仁彦, 亀本 洋, and 服部 高宏. **法哲学**. 有斐閣アルマ, 2002.

[94] 片山 卓也, 島津 明, 東条 敏, 二木 厚吉, and 落水 浩一郎. 電子社会と法令工学. **人工知能学会誌**, 23(4):529–536, 7 2008.

[95] 東条 敏 and 萩原 信吾. **オントロジーを用いた法的知識からの不整合の検出**, volume 3 of *COE research monograph*. JAIST Press, 2008. ISBN 978-4-903092-13-3.

本研究に関する発表論文

[1] Mikito Kobayashi, Shingo Hagiwara, Satoshi Tojo: “Analysis of Miscommunication in Legal Cases”，the 2nd International Workshop on Juris-informatics (JURISIN)，2008.

[2] Shingo Hagiwara, Satoshi Tojo, Mikito Kobayashi: “Belief Updating by Communication Channel”，Seventh Workshop on Computational Logic in Multi-Agent Systems CLIMA-VII, pp.211–225, Springer, Lecture Notes in Computational Logic in Multi-Agent Systems Vol.4371, 2006.

[3] Shingo Hagiwara and Satoshi Tojo: “Discordance Detection in Regional Ordinance: Ontology-Based Validation”, Legal Knowledge and Information Systems (JURIX 2006), Tom M. van Engers editor, IOS Press, ISBN 1-58603-698-X.

[4] Shingo Hagiwara and Satoshi Tojo: “Stable Legal Knowledge with Regard to Contradictory Arguments”，Artificial Intelligence and Applications 2006, pp.323–328.

[5] Shingo Hagiwara and Satoshi Tojo: “Efficient extraction of minimal inconsisntent sets from a legal knowledge” JAIST 21 世紀 COE シンポジウム 2005「検証進化可能電子社会」GRP 研究員発表会, 2005.

[6] Shingo Hagiwara and Satoshi Tojo: “Stable Legal Knowledge with Regard to Contradictory Arguments” JAIST 21 世紀 COE シンポジウム 2005 「検証進化可能電子社会」GRP 研究員発表会, 2006.

[7] Shingo Hagiwara and Satoshi Tojo: “Discordance Detection in Regional Ordinance:

Ontology-based Verification” JAIST 21 世紀 COE シンポジウム 2005 「検証進化可能電子社会」GRP 研究員発表会, 2007.

[8] 小林 幹門, 萩原 信吾, 東条 敏: “エージェント間通信を考慮した論理モデルに基づいたエージェントコミュニケーションの表現”，第 21 回人工知能学会全国大会,2007.

[9] 萩原 信吾, 東条 敏: “矛盾知識集合における極小矛盾集合の抽出”，第 19 回人工知能学会全国大会.

[10] 萩原 信吾, 東条 敏: “拡張論理プログラミングを用いた法的知識の整合性検証”，人工知能学会論文誌（投稿中）.

[11] 東条 敏, 萩原 信吾: “オントロジーを用いた法的知識からの不整合の検出”, COE research monograph Vol.3, JAIST Press, ISBN 978-4-903092-13-3