Human Conditional Reasoning in Answer Set Programming*

Chiaki Sakama

Wakayama University, 930 Sakaedani, Wakayama 640 8510, Japan

Abstract

Given a conditional sentence $\varphi \Rightarrow \psi$ (if φ then ψ) and respective facts, four different types of inferences are observed in human reasoning. *Affirming the antecedent* (AA) (or *modus ponens*) reasons ψ from φ; *affirming the consequent* (AC) reasons φ from ψ; *denying the antecedent* (DA) reasons $\neg\psi$ from $\neg\varphi$; and *denying the consequent* (DC) (or *modus tollens*) reasons $\neg\varphi$ from $\neg\psi$. Among them, AA and DC are logically valid, while AC and DA are logically invalid and often called *logical fallacies*. Nevertheless, humans often perform AC or DA as *pragmatic inference* in daily life. In this paper, we realize AC, DA and DC inferences in *answer set programming*. Eight different types of *completion* are introduced and their semantics are given by answer sets. We investigate formal properties and characterize human reasoning tasks in cognitive psychology.

Keywords

answer set programming, completion, human conditional reasoning, pragmatic inference

1. Introduction

People use conditional sentences and reason with them in everyday life. From an early stage of artificial intelligence (AI), researchers represent conditional sentences as if-then rules and perform deductive inference using them. Production systems or logic programming are examples of this type of systems. However, human conditional reasoning is not always logically valid. In psychology and cognitive science, it is well known that humans are more likely to perform logically invalid but pragmatic inference. For instance, consider the following three sentences:

S: If the team wins the first round tournament, then it advances to the final round.
P: The team wins the first round tournament.
C: The team advances to the final round.

Given the conditional sentence S and the premise P, *affirming the antecedent* (AA) (or *modus ponens*) concludes the consequence C. Given S and the negation of the consequence $\neg C$, *denying the consequent* (DC) (or *modus tollens*) concludes the negation of the premise $\neg P$. AA and DC are logically valid. On the other hand, people often infer P from S and C or infer $\neg C$ from S and $\neg P$. The former is called *affirming the consequent* (AC) and the latter is called *denying the antecedent* (DA). Both AC and DA are logically invalid and often called *logical fallacies*.

In the pragmatics of conditional reasoning, it is assumed that a conditional sentence is often interpreted as bi-conditional, that is, '*if*' is interpreted as '*if and only if*', and such *conditional perfection* produces AC or DA as *invited inference* [16, 19]. Psychological studies empirically show that a conditional sentence "p *if* q" is rephrased into the form "p *only if* q" with greater frequency for permission/obligation statements [6, 5]. For instance, the sentence "a customer can drink an alcoholic beverage if he is over 18" is rephrased into "a customer can drink an alcoholic beverage *only if* he is over 18". It is also reported that AA is easier than DC when a conditional is given as "*if* p *then* q". When a conditional is given as "p *only if* q", on the other hand, it is rephrased as "*if not* q *then not* p" and this paraphrase yields a directionality opposite which makes DC easier than AA [3]. The fact that people do not necessarily make inferences as in standard logic brings several proposals of new interpretation of conditional sentences in cognitive psychology. *Mental logic* [4] interprets '*if*' as conveying supposition and introduces a set of pragmatic inference schemas for *if*-conditionals. *Mental model theory* [22], on the other hand, considers that the meanings of conditionals are not truth-functional, and represents the meaning of a conditional sentence by models of the possibilities compatible with the sentence. A probabilistic approach interprets a conditional sentence $p \Rightarrow q$ in terms of conditional probability $P(q \mid p)$, then the acceptance rates of four conditional inferences are represented by their respective conditional probabilities [29]. Eichhorn *et al*. [13] use *conditional logic* and define *inference patterns* as combination (AA, DC, AC, DA) of four inference rules. Given a conditional sentence "*if* p *then* q", four possible worlds (combination of truth values of p and q) are considered. An inference in each pattern is then defined by imposing corresponding constraints on

21st International Workshop on Nonmonotonic Reasoning, September 2-4, 2023, Rhodes, Greece

* A longer version of this paper is submitted to a journal and is currently under review.

sakama@wakayama-u.ac.jp (C. Sakama)
http://web.wakayama-u.ac.jp/~sakama (C. Sakama)
0000-0002-9966-3722 (C. Sakama)

the plausibility relation over the worlds.

In this way, the need of considering the pragmatics of conditional reasoning has been widely recognized in psychology and cognitive science. On the other hand, relatively little attention has been paid for realizing such pragmatic inferences in computational logic or logic programming [33, 24]. From a practical perspective, however, people would expect AI to reason like humans, that is, one would expect AI to conclude P from S and C, or $\neg\, C$ from S and $\neg\, P$ in the introductory example, rather than conclude $unknown$. Logic programming is a context-independent language and has a general-purpose inference mechanism by its nature. By contrast, pragmatic inference is governed by context-sensitive mechanisms, rather than context-free and general-purpose mechanisms [6, 9]. As argued by [11], computational approaches to explain human reasoning should be *cognitively adequate*, that is, they appropriately represent human knowledge (*conceptually adequate*) and computations behave similarly to human reasoning (*inferentially adequate*). Then if we use logic programming for representing knowledge in daily life, it is useful to have a mechanism of automatic transformation of a knowledge base to simulate human reasoning depending on the context in which conditional sentences are used. That is, transform a program to a conceptually adequate form in order to make computation in the program inferentially adequate.

In this paper, we realize human conditional reasoning in *answer set programming* (ASP) [17]. ASP is one of the most popular frameworks that realize declarative knowledge representation and commonsense reasoning. ASP is a language of logic programming and conditional sentences are represented by rules in a program. Inference in ASP is deduction based on *default logic* [31], while modus tollens or DC is not considered in ASP. AC and DA are partly realized by *abductive logic programming* [23] and *program completion* [7], respectively. As will be argued in this paper, however, AC and DA produce different results from them in general. We realize pragmatic AC and DA inferences as well as DC inference in ASP in a uniform and modular way. We introduce the notions of *AC completion*, *DC completion*, *DA completion* and their variants. We investigate formal properties of those completions and characterize human reasoning tasks in cognitive psychology.

The rest of this paper is organized as follows. Section 2 reviews basic notions of ASP programs considered in this paper. Section 3 introduces different types of completions for human conditional reasoning, and Section 4 presents their variance as default reasoning. Section 5 discusses related works and Section 6 summarizes the paper. Due to page limitation, proofs of propositions are omitted in this paper. They are found in the full paper[1].

[1] http://web.wakayama-u.ac.jp/~sakama/hcr2023.pdf

2. Preliminaries

In this paper, we consider logic programs with disjunction, default negation, and explicit negation. A *general extended disjunctive program* (GEDP) [25, 20] is a set of rules of the form:

$$L_1 ; \cdots ; L_k ; not\, L_{k+1} ; \cdots ; not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \quad (1)$$

where L_i's $(1 \leq i \leq n)$ are (positive or negative) literals and $0 \leq k \leq l \leq m \leq n$. A program might contain two types of negation: default negation (or negation as failure) not and explicit negation $\neg$. For any literal L, $not\, L$ is called an *NAF-literal* and define $\neg\neg L = L$. We often use the letter ℓ to mean either a literal L or an NAF-literal $not\, L$. The left of $\leftarrow$ is a disjunction of literals and NAF-literals (called *head*), and the right of $\leftarrow$ is a conjunction of literals and NAF-literals (called *body*). Given a rule r of the form (1), define $head^+(r) = \{L_1, \ldots, L_k\}$, $head^-(r) = \{L_{k+1}, \ldots, L_l\}$, $body^+(r) = \{L_{l+1}, \ldots, L_m\}$, and $body^-(r) = \{L_{m+1}, \ldots, L_n\}$. A rule (1) is called a *fact* if $body^+(r) = body^-(r) = \varnothing$; and it is called a *constraint* if $head^+(r) = head^-(r) = \varnothing$. A GEDP Π is called *not-free* if $head^-(r) = body^-(r) = \varnothing$ for each rule r in Π.

A GEDP Π coincides with an *extended disjunctive program* (EDP) of [17] if $head^-(r) = \varnothing$ for any rule r in Π. An EDP Π is called (i) an *extended logic program* (ELP) if $|\, head^+(r)\,| \leq 1$ for any $r \in \Pi$; and (ii) a *normal disjunctive program* (NDP) if Π contains no negative literal. An NDP Π is called (i) a *positive disjunctive program* (PDP) if Π contains no NAF-literal; and (ii) a *normal logic program* (NLP) if $|\, head^+(r)\,| \leq 1$ for any $r \in \Pi$. In this paper, we consider ground programs containing no variable and a *program* means a (ground) GEDP unless stated otherwise.

Let Lit be the set of all ground literals in the language of a program. A set of ground literals $S \subseteq Lit$ *satisfies* a ground rule r of the form (1) iff $body^+(r) \subseteq S$ and $body^-(r) \cap S = \varnothing$ imply either $head^+(r) \cap S \neq \varnothing$ or $head^-(r) \not\subseteq S$. The *answer sets* of a GEDP are defined by the following two steps. First, let Π be a not-free GEDP and $S \subseteq Lit$. Then, S is an *answer set* of Π iff S is a minimal set satisfying the conditions: (i) S satisfies every rule from Π, that is, for each ground rule: $L_1 ; \cdots ; L_k \leftarrow L_{l+1}, \ldots, L_m$ from Π, $\{L_{l+1}, \ldots, L_m\} \subseteq S$ implies $\{L_1, \ldots, L_k\} \cap S \neq \varnothing$. In particular, for each constraint $\leftarrow L_{l+1}, \ldots, L_m$ from Π, $\{L_{l+1}, \ldots, L_m\} \not\subseteq S$. (ii) If S contains a pair of complementary literals L and $\neg L$, then $S = Lit$.

Second, let Π be any GEDP and $S \subseteq Lit$. The *reduct* Π^S of Π by S is a not-free EDP obtained as follows: a rule r^S of the form: $L_1 ; \cdots ; L_k \leftarrow L_{l+1}, \ldots, L_m$ is

in Π^S iff there is a ground rule r of the form (1) from Π such that $head^-(r) \subseteq S$ and $body^-(r) \cap S = \varnothing$. For programs of the form Π^S, their answer sets have already been defined. Then, S is an *answer set* of Π iff S is an answer set of Π^S.

When a program Π is an EDP, the above definition of answer sets coincides with that given in [17]. It is shown that every answer set of a GEDP Π satisfies every rule from Π [20]. An answer set is *consistent* if it is not Lit. A program Π is *consistent* if it has a consistent answer set; otherwise, Π is *inconsistent*. When a program Π is inconsistent, there are two different cases. If Π has the answer set Lit, Π is called *contradictory*; else if Π has no answer set, Π is called *incoherent*. The difference of two cases is illustrated by the following example.

Example 2.1. The program $\Pi_1 = \{\, p \leftarrow not\, q,\ \ \neg p \leftarrow \,\}$ is incoherent, while $\Pi_2 = \{\, p \leftarrow q,\ \ q \leftarrow,\ \ \neg p \leftarrow \,\}$ is contradictory. Note that Lit is not the answer set of Π_1 because Lit is not the answer set of $\Pi_1^{Lit} = \{\, \neg p \leftarrow \,\}$.

We write $\Pi \models_c L$ (resp. $\Pi \models_s L$) if a literal L is included in some (resp. every) consistent answer set of Π. Two programs Π_1 and Π_2 are *equivalent* if they have the same set of answer sets. Two programs Π_1 and Π_2 are *strongly equivalent* if $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent for any program Π [26]. In particular, two rules r_1 and r_2 are *strongly equivalent* if $\Pi \cup \{r_1\}$ and $\Pi \cup \{r_2\}$ are equivalent for any program Π.

An answer set of a GEDP is not always minimal, i.e., a program Π may have two answer sets S and T such that $S \subset T$ [25, 20]. This is in contrast with the case of EDPs where every answer set is minimal.

Example 2.2. Let Π be the progam:

$$p\,;\, not\, q \leftarrow, \quad q\,;\, not\, p \leftarrow .$$

Then Π has two answer sets $\varnothing$ and $\{p, q\}$.

Suppose a rule r such that $head^+(r) = \varnothing$:

$$not\, L_{k+1}\,;\cdots;\, not\, L_l \leftarrow\ L_{l+1},\ \ldots,\ L_m,\ not\, L_{m+1},\ \ldots,\ not\, L_n. \quad (2)$$

Define a rule $\eta(r)$ of the form:

$$\leftarrow\ L_{k+1},\ \ldots,\ L_l, L_{l+1},\ \ldots,\ L_m,\ not\, L_{m+1},\ \ldots,\ not\, L_n \quad (3)$$

that is obtained by shifting $not\, L_{k+1}\,;\cdots;\, not\, L_l$ in $head^-(r)$ to $L_{k+1}, \ldots, L_l$ in $body^+(\eta(r))$. The two rules (2) and (3) are strongly equivalent under the answer set semantics.

Proposition 2.1 ([20]). *Let Π be a program and $\Phi = \{\, r \mid r \in \Pi \ \ and \ \ head^+(r) = \varnothing \,\}$. Also, let $\Pi' = (\Pi \setminus \Phi) \cup \{\, \eta(r) \mid r \in \Phi \,\}$. Then Π and Π' have the same answer sets.*

Proposition 2.2. *Let Π be a program and $\Psi = \{\, r \mid r \in \Pi,\ head^+(r) = \varnothing \ and \ head^-(r) \cap body^-(r) \neq \varnothing \,\}$. Then, Π and $\Pi \setminus \Psi$ have the same answer sets.*

Example 2.3. For any Π, $\Pi \cup \{\, not\, p \ \leftarrow\ q, not\, p \,\}$ is equivalent to $\Pi \cup \{\, \leftarrow p, q, not\, p \,\}$ (Proposition 2.1), which is further simplified to Π (Proposition 2.2).

Proposition 2.3. *Let Π be a not-free GEDP. If there is a constraint in Π, then Π is not contradictory.*

3. Human Conditional Reasoning in ASP

3.1. AC Completion

We first introduce a framework for reasoning by *affirming the consequent* (AC) in ASP. In GEDPs, a conditional sentence $\varphi \Rightarrow \psi$ (*if φ then ψ*) is represented by the rule $\psi \ \leftarrow\ \varphi$ where ψ is a disjunction "$L_1\,;\cdots;\, L_k\,;\, not\, L_{k+1}\,;\cdots;\, not\, L_l$" and φ is a conjunction "$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$". To realize reasoning backward from ψ to φ, we extend a program Π by introducing new rules.

Definition 3.1 (AC completion). Let Π be a program and $r \in \Pi$ a rule of the form (1). First, for each disjunct in $head^+(r)$ and $head^-(r)$, converse the implication:

$$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \leftarrow\ L_j \quad (1 \le j \le k), \quad (4)$$

$$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \leftarrow\ not\, L_j \quad (k+1 \le j \le l). \quad (5)$$

In (4) and (5), the conjunction "$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$" appears on the left of $\leftarrow$. The produced (4) (resp. (5)) is considered an abbreviation of the collection of $(n - l)$ rules: $(L_{l+1} \leftarrow L_j), \ldots, (not\, L_n \leftarrow L_j)$ (resp. $(L_{l+1} \leftarrow not\, L_j), \ldots, (not\, L_n \leftarrow not\, L_j)$)[2], hence we abuse the term 'rule' and call (4) or (5) a rule. In particular, (4) is not produced if $head^+(r) = \varnothing$ or $body^+(r) = body^-(r) = \varnothing$; and (5) is not produced if $head^-(r) = \varnothing$ or $body^+(r) = body^-(r) = \varnothing$. The set of all rules (4) and (5) is denoted as $conv(r)$. Next, define

$$ac(\Pi) = \{\, \Sigma_1\,;\cdots;\, \Sigma_p \ \leftarrow\ \ell_j \ \mid\ \Sigma_i \leftarrow \ell_j\ (1 \le i \le p)\ \text{ is in } \bigcup_{r \in \Pi} conv(r) \,\}$$

where each Σ_i $(1 \le i \le p)$ is a conjunction of literals and NAF-literals, and ℓ_j is either a literal L_j $(1 \le j \le k)$ or an NAF-literal $not\, L_j$ $(k+1 \le j \le l)$. The *AC completion* of Π is then defined as:

$$AC(\Pi) = \Pi \ \cup\ ac(\Pi).$$

[2]We often use the parenthesis '()' to improve the readability.

(4) and (5) in $conv(r)$ represent converse implications from the disjunction in the head of r to the conjunction in the body of r. $ac(\Pi)$ collects rules $\Sigma_i \leftarrow \ell_j$ $(1 \leq i \leq p)$ having the same (NAF-)literal ℓ_j on the right of $\leftarrow$, and constructs "$\Sigma_1 ; \cdots ; \Sigma_p \leftarrow \ell_j$", which we call an *extended rule*. Introducing $ac(\Pi)$ to Π realizes reasoning by AC in Π.

The set $ac(\Pi)$ contains an extended rule having a disjunction of conjunctions in its head, while it is transformed to rules of a GEDP. That is, the extended rule:

$$(\ell_1^1, \ldots, \ell_{m_1}^1) ; \cdots ; (\ell_1^p, \ldots, \ell_{m_p}^p) \leftarrow \ell_j$$

is identified with the set of $m_1 \times \cdots \times m_p$ rules of the form: $\ell_{i_1}^1 ; \cdots ; \ell_{i_p}^p \leftarrow \ell_j$ $(1 \leq i_k \leq m_k;\ 1 \leq k \leq p)$. By this fact, $AC(\Pi)$ is viewed as a GEDP and we do not distinguish extended rules and rules of a GEDP hereafter. The semantics of $AC(\Pi)$ is defined by its answer sets.

Example 3.1. Let $\Pi = \{p ; not\, q \leftarrow r,\ not\, s,\ \ p \leftarrow q\}$. Then, $ac(\Pi) = \{ (r,\ not\, s) ; q \leftarrow p,\ \ r,\ not\, s \leftarrow not\, q \}$ where the 1st rule "$(r,\ not\, s) ; q \leftarrow p$" is identified with "$r ; q \leftarrow p$" and "$not\, s ; q \leftarrow p$"; and the 2nd rule "$r,\ not\, s \leftarrow not\, q$" is identified with "$r \leftarrow not\, q$" and "$not\, s \leftarrow not\, q$". $AC(\Pi) \cup \{p \leftarrow\}$ has two answer sets $\{p, q\}$ and $\{p, r\}$.

By definition, if there is more than one rule having the same (NAF-)literal in the heads, they are collected to produce a single converse rule. For instance, $\Pi = \{ p \leftarrow q,\ \ p \leftarrow r \}$ produces $ac(\Pi) = \{ q ; r \leftarrow p \}$ but not $\Lambda = \{ q \leftarrow p,\ \ r \leftarrow p \}$. Then, $AC(\Pi) \cup \{p \leftarrow\}$ has two answer sets $\{p, q\}$ and $\{p, r\}$. Suppose that the new fact $\neg q \leftarrow$ is added to Π. Put $\Pi' = \Pi \cup \{ \neg q \leftarrow \}$. Then $AC(\Pi') \cup \{p \leftarrow\}$ has the answer set $\{p, r\}$, which represents the result of AC reasoning in Π'. If Λ is used instead of $ac(\Pi)$, however, $\Pi' \cup \Lambda \cup \{p \leftarrow\}$ has the answer set Lit. The result is too strong because r is consistently inferred from $\Pi' \cup \{p \leftarrow\}$ by AC reasoning. As a concrete example, put $p = wet\text{-}grass$, $q = rain$, and $r = sprinkler\text{-}on$. Then $AC(\Pi') \cup \{ wet\text{-}grass \leftarrow \}$ has the answer set $\{ wet\text{-}grass, \neg rain, sprinkler\text{-}on \}$, while $\Pi' \cup \Lambda \cup \{ wet\text{-}grass \leftarrow \}$ has the answer set Lit. AC completion derives an antecedent from a consequent, but it does not derive negation of antecedent by its nature. For instance, given $\Pi = \{ p ; q \leftarrow r,\ \ p \leftarrow \}$, $AC(\Pi) \models_s r$ but $AC(\Pi) \not\models_c \neg q$.

Note that in Definition 3.1 the converse of constraints and facts are not considered. When $head^+(r) = head^-(r) = \varnothing$, r is considered a rule with *false* in the head, then (4) and (5) become "$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \leftarrow false$" which has no effect as a rule. On the other hand, when $body^+(r) = body^-(r) = \varnothing$, r is considered a rule with $true$ in the body, then (4) and (5) become "$true \leftarrow L_j$ $(1 \leq j \leq k)$ and "$true \leftarrow not\, L_j$ $(k+1 \leq j \leq l)$, respectively. We do not include this type of rules for constructing $ac(\Pi)$ because it would disable AC reasoning. For instance, transforming $\Pi = \{ p \leftarrow q,\ \ p \leftarrow \}$ into $\Pi \cup \{q ; true \leftarrow p\}$ produces the answer set $\{p\}$, then q is not obtained. With this reason, constraints and facts are not completed at the first step of Definition 3.1. Note also that the result of AC completion is syntax-dependent in general. That is, two (strongly) equivalent programs may produce different AC completions.

Example 3.2. Let $\Pi_1 = \{ not\, p \leftarrow q \}$ and $\Pi_2 = \{ \leftarrow p, q \}$. By Proposition 2.1, Π_1 and Π_2 are strongly equivalent, but $AC(\Pi_1) = \Pi_1 \cup \{ q \leftarrow not\, p \}$ and $AC(\Pi_2) = \Pi_2$. As a result, $AC(\Pi_1)$ has the answer set $\{q\}$ while $AC(\Pi_2)$ has the answer set $\varnothing$.

In the above example, "$not\, p \leftarrow q$" is a conditional sentence which is subject to AC inference, while "$\leftarrow p, q$" is a constraint which is not subject to AC inference by definition. For instance, given the conditional sentence "if it is sunny, the grass is not wet" and the fact "the grass is not wet", people would infer "it is sunny" by AC inference. On the other hand, given the constraint "it does not happen that wet-grass and sunny-weather at the same time" and the fact "the grass is not wet", the number of people who infer "it is sunny" by AC would be smaller because the cause-effect relation between "sunny" and "not wet" is not explicitly expressed in the constraint.

Reasoning by AC is nonmonotonic in the sense that $\Pi \models_c L$ (or $\Pi \models_s L$) does not imply $AC(\Pi) \models_c L$ (or $AC(\Pi) \models_s L$) in general.

Example 3.3. $\Pi = \{ p \leftarrow not\, q,\ \ r \leftarrow q,\ \ r \leftarrow \}$ has the answer set $\{p, r\}$, while $AC(\Pi) = \Pi \cup \{ not\, q \leftarrow p,\ \ q \leftarrow r \}$ has the answer set $\{q, r\}$.

In Example 3.3, reasoning by AC produces q which blocks deriving p using the first rule in Π. As a concrete example, an online-meeting is held on time if no network trouble arises. However, it turns that the web browser is unconnected and one suspects that there is some trouble on the network. Put p="online-meeting is held on time", q="network trouble", r="the web browser is unconnected". In this case, one may withdraw the conclusion p after knowing r. As such, additional rules $ac(\Pi)$ may change the results of Π. One can see the effect of AC reasoning in a program Π by comparing answer sets of Π and $AC(\Pi)$.

A consistent program Π may produce an inconsistent $AC(\Pi)$. In converse, an inconsistent Π may produce a consistent $AC(\Pi)$.

Example 3.4. $\Pi_1 = \{ p \leftarrow \neg p,\ \ p \leftarrow \}$ is consistent, but $AC(\Pi_1) = \Pi_1 \cup \{ \neg p \leftarrow p \}$ is contradictory. $\Pi_2 = \{ \leftarrow not\, p,\ \ q \leftarrow p,\ \ q \leftarrow \}$ is incoherent, but $AC(\Pi_2) = \Pi_2 \cup \{ p \leftarrow q \}$ is consistent.

A sufficient condition for the (in)consistency of $AC(\Pi)$ is given below.

Proposition 3.1. *If a PDP* Π *contains no constraint, then* $AC(\Pi)$ *is consistent. Moreover, for any answer set* S *of* Π*, there is an answer set* T *of* $AC(\Pi)$ *such that* $S \subseteq T$.

Proposition 3.2. *If a program* Π *is contradictory, then* $AC(\Pi)$ *is contradictory.*

3.2. DC Completion

We next introduce a framework for reasoning by denying the consequent (DC) in ASP. There are two ways for negating a literal—one is using explicit negation and the other is using default negation. Accordingly, there are two ways of completing a program for the purpose of reasoning by DC.

Definition 3.2 (DC completion). Let Π be a program. For each rule $r \in \Pi$ of the form (1), define $wdc(r)$ as the rule:

$$not\ L_{l+1}; \cdots; not\ L_m\ ;\ L_{m+1}; \cdots; L_n \leftarrow not\ L_1, \ldots, not\ L_k,\ L_{k+1}, \ldots, L_l \quad (6)$$

and define $sdc(r)$ as the rule:

$$\neg L_{l+1}; \cdots; \neg L_m\ ;\ L_{m+1}; \cdots; L_n \leftarrow \neg L_1, \ldots, \neg L_k,\ L_{k+1}, \ldots, L_l. \quad (7)$$

In particular, (6) or (7) becomes a fact if $head^+(r) = head^-(r) = \varnothing$; and it becomes a constraint if $body^+(r) = body^-(r) = \varnothing$. The *weak DC completion* and the *strong DC completion* of Π are respectively defined as:

$$WDC(\Pi) = \Pi \cup \{\, wdc(r) \mid r \in \Pi \,\},$$
$$SDC(\Pi) = \Pi \cup \{\, sdc(r) \mid r \in \Pi \,\}.$$

By definition, $WDC(\Pi)$ and $SDC(\Pi)$ introduce contrapositive rules in two different ways. In (6), literals L_i $(1 \leq i \leq k;\ l+1 \leq i \leq m)$ are negated using default negation not and NAF-literals $not\ L_j$ $(k+1 \leq i \leq l;\ m+1 \leq i \leq n)$ are converted to L_j. In (7), on the other hand, literals L_i $(1 \leq i \leq k;\ l+1 \leq i \leq m)$ are negated using explicit negation $\neg$ and NAF-literals $not\ L_j$ $(k+1 \leq i \leq l;\ m+1 \leq i \leq n)$ are converted to L_j. $WDC(\Pi)$ and $SDC(\Pi)$ are GEDPs and their semantics are defined by their answer sets. In particular, $SDC(\Pi)$ becomes an EDP if Π is an EDP. The WDC and SDC produce different results in general.

Example 3.5. Let $\Pi = \{\, p \leftarrow not\ q \,\}$. Then, $WDC(\Pi) = \{\, p \leftarrow not\ q,\quad q \leftarrow not\ p \,\}$ and $SDC(\Pi) = \{\, p \leftarrow not\ q,\quad q \leftarrow \neg p \,\}$. $WDC(\Pi)$ has two answer sets $\{p\}$ and $\{q\}$, while $SDC(\Pi)$ has the single answer set $\{p\}$.

Example 3.5 shows that WDC is nonmonotonic as $\Pi \models_s p$ but $WDC(\Pi) \not\models_s p$. SDC is also nonmonotonic (see Example 3.8). The result of DC completion is syntax-dependent in general.

Example 3.6. Let $\Pi_1 = \{\, not\ p \leftarrow q \,\}$ and $\Pi_2 = \{\, \leftarrow p, q \,\}$. Then, $SDC(\Pi_1) = \Pi_1 \cup \{\, \neg q \leftarrow p \,\}$ and $SDC(\Pi_2) = \Pi_2 \cup \{\, \neg p; \neg q \leftarrow \,\}$. As a result, $SDC(\Pi_1)$ has the answer set $\varnothing$ while $SDC(\Pi_2)$ has two answer sets $\{\neg p\}$ and $\{\neg q\}$.

WDC keeps the consistency of the original program.

Proposition 3.3. *If a program* Π *has a consistent answer set* S*, then* S *is an answer set of* $WDC(\Pi)$.

The converse of Proposition 3.3 does not hold in general.

Example 3.7. The program $\Pi = \{\, \leftarrow not\ p \,\}$ has no answer set, while $WDC(\Pi) = \{\, \leftarrow not\ p,\quad p \leftarrow \,\}$ has the answer set $\{p\}$.

Proposition 3.4. *Let* Π *be a consistent program such that every constraint in* Π *is not-free (i.e.,* $head^+(r) = head^-(r) = \varnothing$ *implies* $body^-(r) = \varnothing$ *for any* $r \in \Pi$*). Then* $SDC(\Pi)$ *is not contradictory.*

A program Π satisfying the condition of Proposition 3.4 may produce an incoherent $SDC(\Pi)$.

Example 3.8. The program $\Pi = \{\, p \leftarrow q,\quad p \leftarrow \neg q,\ \neg p \leftarrow \,\}$ has the answer set $\{\neg p\}$, but $SDC(\Pi) = \Pi \cup \{\, \neg q \leftarrow \neg p,\quad q \leftarrow \neg p,\quad \leftarrow p \,\}$ is incoherent.

Proposition 3.5. *If a program* Π *is contradictory, then both* $WDC(\Pi)$ *and* $SDC(\Pi)$ *are contradictory.*

In GEDPs contraposition of a rule does not hold in general, so the program $\Pi = \{\, p \leftarrow q,\quad \neg p \leftarrow \,\}$ does not deduce $\neg q$. SDC completes the program as $SDC(\Pi) = \Pi \cup \{\, \neg q \leftarrow \neg p,\quad \leftarrow p \,\}$ and makes $\neg q$ deducible. In this sense, SDC has the effect of making explicit negation closer to classical negation in GEDP.

3.3. DA Completion

As a third extension, we introduce a framework for reasoning by denying the antecedent (DA) in ASP. As in the case of DC completion, two different ways of completion are considered depending on the choice of negation.

Definition 3.3 (weak DA completion). Let Π be a program and $r \in \Pi$ a rule of the form (1). First, inverse the implication:

$$not\ L_i \leftarrow not\ L_{l+1}\ ;\ \cdots\ ;\ not\ L_m\ ;\ L_{m+1}\ ;\cdots\ ;\ L_n \quad (8)$$
$$(1 \leq i \leq k),$$

$$L_i \leftarrow not\ L_{l+1}\,;\cdots;\,not\ L_m\,;\,L_{m+1}\,;\cdots;\,L_n \quad (9)$$
$$(k+1 \leq i \leq l).$$

In (8) and (9), the disjunction $not\ L_{l+1}\,;\cdots;\,not\ L_m\,;\,L_{m+1}\,;\cdots;\,L_n$ appears on the right of $\leftarrow$. The produced (8) (resp. (9)) is considered an abbreviation of the collection of $(n-l)$ rules: $(not\ L_i \leftarrow not\ L_{l+1}), \ldots, (not\ L_i \leftarrow L_n)$ (resp. $(L_i \leftarrow not\ L_{l+1}), \ldots, (L_i \leftarrow L_n)$), hence we abuse the term 'rule' and call (8) or (9) a rule. In particular, (8) is not produced if $head^+(r) = \varnothing$ or $body^+(r) = body^-(r) = \varnothing$; and (9) is not produced if $head^-(r) = \varnothing$ or $body^+(r) = body^-(r) = \varnothing$. The set of rules (8)–(9) is denoted as $winv(r)$. Next, define

$$wda(\Pi) = \{\, \ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p \mid \ell_i \leftarrow \Gamma_j\ (1 \leq j \leq p)\ \text{ is in } \bigcup_{r\in\Pi} winv(r) \,\}$$

where ℓ_i is either a literal L_i $(k+1 \leq i \leq l)$ or an NAF literal $not\ L_i$ $(1 \leq i \leq k)$, and each Γ_j $(1 \leq j \leq p)$ is a disjunction of literals and NAF literals. The *weak DA completion* of Π is defined as:

$$WDA(\Pi) = \Pi \cup wda(\Pi).$$

(8) and (9) in $winv(r)$ represent inverse implication from the (default) negation of the conjunction in the body of r to the (default) negation of the disjunction in the head of r. $wda(\Pi)$ collects rules $\ell_i \leftarrow \Gamma_j$ $(1 \leq j \leq p)$ having the same (NAF-)literal ℓ_i on the left of $\leftarrow$, and constructs "$\ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p$", which we call an *extended rule*. Introducing $wda(\Pi)$ to Π realizes reasoning by weak DA. An extended rule has a conjunction of disjunctions in its body, while it is transformed to rules of a GEDP as the case of AC completion. That is, the extended rule:

$$\ell_i \leftarrow (\ell_1^1\,;\cdots;\,\ell_{m_1}^1)\,,\ldots,\,(\ell_1^p\,;\cdots;\,\ell_{m_p}^p)$$

is identified with the set of $m_1 \times \cdots \times m_p$ rules of the form: $\ell_i \leftarrow \ell_{j_1}^1, \ldots, \ell_{j_p}^p$ $(1 \leq j_k \leq m_k;\ 1 \leq k \leq p)$. By this fact, $WDA(\Pi)$ is viewed as a GEDP and we do not distinguish extended rules and rules of a GEDP hereafter. The semantics of $WDA(\Pi)$ is defined by its answer sets.

Example 3.9. Let $\Pi = \{\, p\,;\, q \leftarrow r,\ not\ s,\ q\,;\, not\ r \leftarrow t,\ \ s \leftarrow \,\}$. Then. $wda(\Pi) = \{\, not\ p \leftarrow not\ r\,;\ s,\ \ not\ q \leftarrow (not\ r\,;\ s),\ not\ t,\ \ r \leftarrow not\ t\,\}$ where the first rule "$not\ p \leftarrow not\ r\,;\ s$" is identified with "$not\ p \leftarrow not\ r$" and "$not\ p \leftarrow s$"; and the second rule "$not\ q \leftarrow (not\ r\,;\ s),\ not\ t$" is identified with "$not\ q \leftarrow not\ r,\ not\ t$" and "$not\ q \leftarrow s,\ not\ t$". Then, $WDA(\Pi)$ has the answer set $\{s, r\}$.

As in the case of AC completion, if there is more than one rule having the same (NAF-)literal in the heads, they are collected to produce a single inverse rule. For instance, $\Pi = \{\, p \leftarrow q,\ \ p \leftarrow r\,\}$ produces $wda(\Pi) = \{\, not\ p \leftarrow not\ q,\ not\ r\,\}$ but not $\Lambda = \{\, not\ p \leftarrow not\ q,\ \ not\ p \leftarrow not\ r\,\}$. Suppose the new fact $r \leftarrow$ is added to Π. Put $\Pi' = \Pi \cup \{r \leftarrow\}$. Then $WDA(\Pi')$ has the answer set $\{p, r\}$. If Λ is used instead of $wda(\Pi)$, however, $\Pi' \cup \Lambda$ is incoherent because the first rule of Λ is not satisfied. The result is too strong because p is deduced by $p \leftarrow r$ and $r \leftarrow$, and it has no direct connection to DA inference in the first rule of Λ. Hence, we conclude $not\ p$ if both q and r are negated in $wda(\Pi)$.

The strong DA completion is defined in a similar way.

Definition 3.4 (strong DA completion). Let Π be a program and $r \in \Pi$ a rule of the (1). First, inverse the implication:

$$\neg L_i \leftarrow \neg L_{l+1}\,;\cdots;\,\neg L_m\,;\,L_{m+1}\,;\cdots;\,L_n \quad (10)$$
$$(1 \leq i \leq k),$$
$$L_i \leftarrow \neg L_{l+1}\,;\cdots;\,\neg L_m\,;\,L_{m+1}\,;\cdots;\,L_n \quad (11)$$
$$(k+1 \leq i \leq l).$$

As in the case of WDA, the produced (10) (resp. (11)) is considered an abbreviation of the collection of $(n-l)$ rules: $(\neg L_i \leftarrow \neg L_{l+1}), \ldots, (\neg L_i \leftarrow L_n)$ (resp. $(L_i \leftarrow \neg L_{l+1}), \ldots, (L_i \leftarrow L_n)$), hence we call (10) or (11) a rule. In particular, (10)–(11) are not produced when their heads or bodies are empty. The set of rules (10)–(11) is denoted as $sinv(r)$. Next, define

$$sda(\Pi) = \{\, \ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p \mid \ell_i \leftarrow \Gamma_j\ (1 \leq j \leq p)\ \text{ is in } \bigcup_{r\in\Pi} sinv(r) \,\}$$

where ℓ_i is either a literal L_i $(k+1 \leq i \leq l)$ or $\neg L_i$ $(1 \leq i \leq k)$, and each Γ_j $(1 \leq j \leq p)$ is a disjunction of positive/negative literals. The *strong DA completion* of Π is defined as:

$$SDA(\Pi) = \Pi \cup sda(\Pi).$$

As in the case of WDA, extended rules in $sda(\Pi)$ is transformed to rules of a GEDP. Then $SDA(\Pi)$ is viewed as a GEDP and its semantics is defined by its answer sets. In particular, $SDA(\Pi)$ becomes an EDP if Π is an EDP.

The result of DA completion is syntax-dependent in general.

Example 3.10. Let $\Pi_1 = \{\, not\ p \leftarrow q\,\}$ and $\Pi_2 = \{\, \leftarrow p, q\,\}$. Then, $WDA(\Pi_1) = \Pi_1 \cup \{\, p \leftarrow not\ q\,\}$ and $WDA(\Pi_2) = \Pi_2$. As a result, $WDA(\Pi_1)$ has the answer set $\{p\}$ while $WDA(\Pi_2)$ has the answer set $\varnothing$.

Both WDA and SDA are nonmonotonic in general.

Example 3.11. (1) $\Pi_1 = \{\, p \leftarrow not\, q,\quad not\, q \leftarrow p\,\}$ produces $WDA(\Pi_1) = \Pi_1 \cup \{\, not\, p \leftarrow q,\quad q \leftarrow not\, p\,\}$. Then, $\Pi_1 \models_s p$ but $WDA(\Pi_1) \not\models_s p$. (2) $\Pi_2 = \{\, p \leftarrow not\, \neg r,\quad r \leftarrow not\, q,\quad q \leftarrow\,\}$ produces $SDA(\Pi_2) = \Pi_2 \cup \{\, \neg p \leftarrow \neg r,\quad \neg r \leftarrow q\,\}$. Then, $\Pi_2 \models_c p$ but $SDA(\Pi_2) \not\models_c p$.

When a program is a consistent EDP, the WDA does not introduce a new answer set.

Proposition 3.6. *Let Π be an EDP. If S is a consistent answer set of $WDA(\Pi)$, then S is an answer set of Π.*

Proposition 3.7. *If a program Π is contradictory, then both $WDA(\Pi)$ and $SDA(\Pi)$ are contradictory.*

As in the case of AC, a consistent program Π may produce an inconsistent $WDA(\Pi)$ or $SDA(\Pi)$. In converse, an incoherent Π may produce a consistent $WDA(\Pi)$ or $SDA(\Pi)$.

Example 3.12. (1) $\Pi_1 = \{\, not\, p \leftarrow p\,\}$, which is equivalent to $\{\, \leftarrow\ p\,\}$ (Proposition 2.1), is consistent, but $WDA(\Pi_1) = \Pi_1 \cup \{\, p \leftarrow not\, p\,\}$ is incoherent.

(2) $\Pi_2 = \{\, \neg p \leftarrow p,\quad \neg p \leftarrow\,\}$ is consistent, but $SDA(\Pi_2) = \Pi_2 \cup \{\, p \leftarrow \neg p\,\}$ is contradictory.

(3) $\Pi_3 = \{\, not\, p \leftarrow q,\quad \leftarrow not\, p\,\}$, which is equivalent to $\{\, \leftarrow p, q,\quad \leftarrow not\, p\,\}$ (Proposition 2.1), is incoherent, but $WDA(\Pi_3) = \Pi_3 \cup \{\, p \leftarrow not\, q\,\}$ is consistent (having the answer set $\{p\}$).

(4) $\Pi_4 = \{\, \leftarrow not\, p,\quad \neg p \leftarrow not\, q,\quad q \leftarrow\,\}$ is incoherent, but $SDA(\Pi_4) = \Pi_4 \cup \{\, p \leftarrow q\,\}$ is consistent (having the answer set $\{p, q\}$).

4. AC and DA as Default Reasoning

AC and DA are logically invalid and additional rules for AC and DA often make a program inconsistent. In this section, we relax the effects of the AC or DA completion by introducing additional rules as *default rules* in the sense of [31]. More precisely, we capture AC and DA as the following default inference rules:

$$(\textbf{default AC}) \qquad \frac{(\varphi \Rightarrow \psi) \wedge \psi : \varphi}{\varphi}$$

$$(\textbf{default DA}) \qquad \frac{(\varphi \Rightarrow \psi) \wedge \neg\varphi : \neg\psi}{\neg\psi}$$

The default AC rule says: given the conditional $\varphi \Rightarrow \psi$ and the fact ψ, conclude φ as a default consequence. The default DA rule is read in a similar manner. We encode these rules in ASP.

4.1. Default AC completion

The AC completion is modified for default AC reasoning.

Definition 4.1 (default AC completion). Let Π be a program. For each rule $r \in \Pi$ of the form (1), define $dac(r)$ as the set of rules:

$$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \leftarrow L_i, \Delta \quad (1 \le i \le k), \tag{12}$$

$$L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \leftarrow not\, L_i, \Delta \quad (k+1 \le i \le l) \tag{13}$$

where $\Delta = not \neg L_{l+1}, \ldots, not \neg L_m, not\, L_{m+1}, \ldots, not\, L_n$. As before, (12) is not produced if $head^+(r) = \varnothing$ or $body^+(r) = body^-(r) = \varnothing$; and (13) is not produced if $head^-(r) = \varnothing$ or $body^+(r) = body^-(r) = \varnothing$. The *default AC completion* of Π is defined as:

$$DAC(\Pi) = \Pi \cup dac(\Pi)$$

in which

$$dac(\Pi) = \{\, \Sigma_1 ; \cdots ; \Sigma_p \leftarrow \ell_j, \Delta_i \mid \Sigma_i \leftarrow \ell_j, \Delta_i\ (1 \le i \le p)\ \text{is in}\ \bigcup_{r \in \Pi} dac(r)\,\}$$

where each Σ_i $(1 \le i \le p)$ is a conjunction of literals and NAF-literals, and ℓ_j is either a literal L_j $(1 \le j \le k)$ or an NAF-literal $not\, L_j$ $(k+1 \le j \le l)$.

Like $AC(\Pi)$, rules in $dac(\Pi)$ are converted into the form of a GEDP, then $DAC(\Pi)$ is viewed as a GEDP. Compared with the AC completion, the DAC completion introduces the conjunction Δ of NAF literals to the body of each rule. Then the rules $(\Sigma_1 ; \cdots ; \Sigma_p \leftarrow \ell_j, \Delta_i)$ having the same head with different bodies are constructed for $i = 1, \ldots, p$.

Example 4.1. Let $\Pi = \{\, p \leftarrow q,\quad p \leftarrow r,\quad p \leftarrow,\ \neg r \leftarrow\,\}$. Then $DAC(\Pi) = \Pi \cup dac(\Pi)$ where

$$dac(\Pi) = \{\, q ; r \leftarrow p, not\, \neg q,\quad q ; r \leftarrow p, not\, \neg r\,\}.$$

As a result, $DAC(\Pi)$ has the answer set $\{p, q, \neg r\}$.

Proposition 4.1. *Let Π be a consistent program. If $DAC(\Pi)$ has an answer S, then $S \neq Lit$.*

$DAC(\Pi)$ turns a contradictory $AC(\Pi)$ into a consistent program.

Example 4.2 (cont. Example 3.4). Let $\Pi_1 = \{\, p \leftarrow\ \neg p,\ \ p \leftarrow\,\}$. Then, $DAC(\Pi_1) = \Pi_1 \cup \{\, \neg p \leftarrow p, not\, p\,\}$ has the single answer set $\{\, p\,\}$. So $AC(\Pi_1)$ is contradictory, but $DAC(\Pi_1)$ is consistent.

When $AC(\Pi)$ is incoherent, however, $DAC(\Pi)$ does not resolve incoherency in general.

Example 4.3. Let $\Pi = \{ p \leftarrow q, \quad p \leftarrow, \quad \leftarrow q \}$. Then, $AC(\Pi) = \Pi \cup \{ q \leftarrow p \}$ is incoherent. $DAC(\Pi) = \Pi \cup \{ q \leftarrow p, \mathit{not} \neg q \}$ is still incoherent.

When $AC(\Pi)$ has a consistent answer set, $DAC(\Pi)$ does not change it.

Proposition 4.2. *Let Π be a program. If $AC(\Pi)$ has a consistent answer set S, then S is an answer set of $DAC(\Pi)$.*

4.2. Default DA completion

The DA completion is modified for default DA reasoning.

Definition 4.2 (default DA completion). Let Π be a program. Define

$$wdda(\Pi) = \{ \ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p, \delta_i^w \mid \ell_i \leftarrow \Gamma_j \ (1 \leq j \leq p) \text{ is in } \bigcup_{r \in \Pi} winv(r) \},$$

$$sdda(\Pi) = \{ \ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p, \delta_i^s \mid \ell_i \leftarrow \Gamma_j \ (1 \leq j \leq p) \text{ is in } \bigcup_{r \in \Pi} sinv(r) \}$$

where $\ell_i, \Gamma_j, winv(r)$, and $sinv(r)$ are the same as those in Defs. 3.3 and 3.4. In addition, $\delta_i^w = \mathit{not} \neg L_i$ if $\ell_i = L_i$, and $\delta_i^w = \mathit{not}\, L_i$ if $\ell_i = \mathit{not}\, L_i$; $\delta_i^s = \mathit{not} \neg L_i$ if $\ell_i = L_i$, and $\delta_i^s = \mathit{not}\, L_i$ if $\ell_i = \neg L_i$. The *weak default DA completion* and the *strong default DA completion* of Π are respectively defined as:

$$WDDA(\Pi) = \Pi \cup wdda(\Pi),$$
$$SDDA(\Pi) = \Pi \cup sdda(\Pi).$$

Rules in $wdda(\Pi)$ and $sdda(\Pi)$ are converted into the form of a GEDP, so $WDDA(\Pi)$ and $SDDA(\Pi)$ are viewed as GEDPs. Like the DAC completion, both WDDA and SDDA introduce an additional NAF literal to each rule.

Proposition 4.3. *Let Π be a consistent program. If $WDDA(\Pi)$ (or $SDDA(\Pi)$) has an answer set S, then $S \neq Lit$.*

The WDDA/SDDA eliminates contradiction but does not resolve incoherency.

Example 4.4 (cont. Example 3.12). Let $\Pi_1 = \{ \mathit{not}\, p \leftarrow p \}$ where $WDA(\Pi_1)$ is incoherent. $WDDA(\Pi_1) = \Pi_1 \cup \{ p \leftarrow \mathit{not}\, p, \mathit{not} \neg p \}$ is still incoherent. Let $\Pi_2 = \{ \neg p \leftarrow p, \quad \neg p \leftarrow \}$ where $SDA(\Pi_2)$ is contradictory. $SDDA(\Pi_2) = \Pi_2 \cup \{ p \leftarrow \neg p, \mathit{not} \neg p \}$ has the answer set $\{\neg p\}$.

Proposition 4.4. *Let Π be a program. If $WDA(\Pi)$ (resp. $SDA(\Pi)$) has a consistent answer set S, then S is an answer set of $WDDA(\Pi)$ (resp. $SDDA(\Pi)$).*

Thus, default AC/DA completion is used for avoiding contradiction in programs containing explicit negation.

5. Related Work

There is a number of studies on human conditional reasoning in psychology and cognitive science. In this section, we focus on related work based on logic programming.

5.1. Completion

The idea of interpreting if-then rules in logic programs as bi-conditional dates back to [7]. Clark introduces *predicate completion* in normal logic programs (NLPs), which introduces the only-if part of each rule to a program. Given a propositional program Π, Clark completion $Comp(\Pi)$ is obtained by two steps: (i) all rules $p \leftarrow B_1, \ldots, p \leftarrow B_k$ in Π having the same head p are replaced by $p \leftrightarrow B_1 \vee \cdots \vee B_k$, where B_i $(1 \leq i \leq k)$ is a conjunction of literals; and (ii) for any atom p appearing in the head of no rule in Π, add $p \leftrightarrow \mathit{false}$. The AC completion introduced in this paper extends the technique to the class of GEDP, while the result is generally different from Clark completion in NLPs. For instance, given the program: $\Pi_1 = \{ p \leftarrow q, \quad p \leftarrow \}$, Clark completion becomes $Comp(\Pi_1) = \{ p \leftrightarrow q \vee \top, \quad q \leftrightarrow \bot \}$ where $\top$ and $\bot$ represent *true* and *false*, respectively. $Comp(\Pi_1)$ has the single completion model $\{p\}$ called a *supported model* [2]. In contrast, $AC(\Pi_1) = \Pi_1 \cup \{ q \leftarrow p \}$ has the answer set $\{p, q\}$. The difference comes from the fact that in $Comp(\Pi_1)$, q is identified with *false* but this is not the case in $AC(\Pi_1)$. In Clark completion undefined atoms (i.e., atoms appearing in the head of no rule) are interpreted *false*. We do not use this type of completion because it disturbs the basic type of AC reasoning that infers q from p and $p \leftarrow q$. Clark completion is extended to normal disjunctive programs by several researchers [27, 1, 28]. Those extensions reduce to Clark completion in NLPs, so that they are different from the AC completion. We also introduce the DC completion and the DA completion. When $\Pi_2 = \{ p \leftarrow \mathit{not}\, q \}$, $Comp(\Pi_2) = \{ p \leftrightarrow \neg q, \quad q \leftrightarrow \bot \}$ has the supported model $\{p\}$ while $WDC(\Pi_2) = \Pi_2 \cup \{ q \leftarrow \mathit{not}\, p \}$ has two answer sets $\{p\}$ and $\{q\}$. When $\Pi_3 = \{ p \leftarrow \mathit{not}\, q, \quad p \leftarrow q, \quad q \leftarrow p \}$, $Comp(\Pi_3) = \{ p \leftrightarrow q \vee \neg q, \quad q \leftrightarrow p \}$ has the supported model $\{p, q\}$ while $WDA(\Pi_3) = \Pi_3 \cup \{ \mathit{not}\, p \leftarrow q, \mathit{not}\, q, \quad \mathit{not}\, q \leftarrow \mathit{not}\, p \}$ has no answer set. Thus, completion introduced in this paper is generally different from Clark completion in NLPs.

The *weak completion* [18] leaves undefined atoms unknown under 3-valued logic. In $\Pi_1 = \{ p \leftarrow q, \quad p \leftarrow \}$,

the weak completion becomes $wcomp(\Pi_1) = \{\, p \leftrightarrow \top \,\}$. Then p is true but q is unknown in $wcomp(\Pi_1)$, which is again different from the result of $AC(\Pi_1)$ that has the answer set $\{p, q\}$. In $\Pi_2 = \{\, p \leftarrow not\, q \,\}$, $wcomp(\Pi_2) = \{\, p \leftrightarrow \neg q \,\}$ then both p and q are unknown. In contrast, $WDC(\Pi_2)$ has two answer sets $\{p\}$ and $\{q\}$, and $WDA(\Pi_2)$ has the single answer set $\{p\}$.

5.2. Abductive Logic Programming

An *abductive logic program* [23] is defined as a pair $\langle\, \Pi, \Gamma \,\rangle$ where Π is a program and $\Gamma\, (\subseteq\ Lit)$ is a set of literals called *abducibles*. It is assumed that abducibles appear in the head of no rule in Π. Given an *observation* O as a ground literal, the abduction problem is to find an *explanation* $E\, (\subseteq\ \Gamma)$ satisfying (i) $P \cup E \models_x O$ and (ii) $P \cup E$ is consistent, where $\models_x$ is either $\models_c$ or $\models_s$ depending on the problem. Here we consider $\models_c$ that realizes *credulous* abduction. Consider (Π_1, Γ_1) where $\Pi_1 = \{\, arrive\_on\_time \leftarrow not\ accident,\quad \neg\, arrive\_on\_time \leftarrow accident \,\}$ and $\Gamma_1 = \{accident\}$. Π_1 represents that a train arrives on time unless there is an accident. Then the observation $O = \neg\, arrive\_on\_time$ has the explanation $E = \{accident\}$. Since abduction reasons backward from an observation, it is computed using the AC completion. Let $\langle\, \Pi, \Gamma \,\rangle$ be an abductive program and O an observation. Then, a set $E \subseteq \Gamma$ is an explanation of O if $O \in head^+(r)$ for some $r \in \Pi$ and $AC(\Pi) \cup \{O\}$ has a consistent answer set S such that $S \cap \Gamma = E$. In the above example, $AC(\Pi_1) \cup \{O\}$ is $\Pi_1 \cup ac(\Pi_1) \cup \{O\}$ where $ac(\Pi_1) = \{\, not\ accident \leftarrow arrive\_on\_time,\quad accident \leftarrow \neg\, arrive\_on\_time \,\}$. $AC(\Pi_1) \cup \{O\}$ has the answer set $S = \{\, \neg\, arrive\_on\_time, accident \,\}$. Then, $S \cap \Gamma_1 = \{\, accident \,\}$ is the explanation. Note that $AC(\Pi)$ introduces converse of every rule, while explanations are computed using the AC completion of a subset $\Pi' \subseteq \Pi$ in general. For instance, consider $\Pi_2 = \{\, p \leftarrow a,\ \ q \leftarrow \neg a,\ \ q \leftarrow \,\}$ and $\Gamma_2 = \{a, \neg a\}$. Then $O = p$ has the explanation $E = \{a\}$ in $\langle\, \Pi_2, \Gamma_2 \,\rangle$, while $AC(\Pi_2) \cup \{O\} = \Pi_2 \cup \{\, a \leftarrow p,\ \ \neg a \leftarrow q,\ \ p \leftarrow \,\}$ is contradictory. By putting $\Pi_2' = \{\, p \leftarrow a \,\}$, $AC(\Pi_2') \cup \{O\}$ has the consistent answer set $S = \{p, a\}$ where $S \cap \Gamma = \{a\}$. As such, abduction and AC completion produce different results in general.

Abductive logic programs of [23] cannot compute explanations when contrary to the consequent is observed. For instance, consider $\langle\, \Pi_3, \Gamma_1 \,\rangle$ where $\Pi_3 = \{\, arrive\_on\_time \leftarrow not\ accident \,\}$. Given the observation $O = \neg\, arrive\_on\_time$, no explanation is obtained from $\langle\, \Pi_3, \Gamma_1 \,\rangle$. Generally, a program Π does not necessarily contain a pair of rules r and r' that define L and $\neg L$, respectively. When there is a rule defining L but no rule defining $\neg L$, abduction computes no explanation for the observation $O = \neg L$. The problem is resolved by reasoning by DC. For the rule r in Π_3, $sdc(r) = \{\, accident \leftarrow \neg\, arrive\_on\_time \,\}$. Then $SDC(\Pi_3) \cup \{O\}$ computes the explanation $\{\, accident \,\}$. In contrast to SDC, WDC is used for abduction from *negative* observations. A negative observation represents that some evidence G is not observed and it is represented as $O = not\, G$, which should be distinguished from the (positive) observation $O = \neg\, G$ meaning that $\neg\, G$ is observed. In the abductive program $\langle\, \Pi_3, \Gamma_1 \,\rangle$, the negative observation $O = not\, arrive\_on\_time$ is explained using $wdc(r) = \{\, accident \leftarrow not\ \ arrive\_on\_time \,\}$. Then $WDC(\Pi_3) \cup \{O\}$ has the answer set $\{\, accident \,\}$.

In this way, both AC and DC are used for computing explanations deductively, and DC is used for computing explanations that are not obtained using the framework of [23]. Console *et al*. [8] and Fung *et al*. [15] compute abduction by deduction using Clark completion. Abduction using AC/DC completion is close to those approaches, while the approach based on Clark completion is restricted to normal logic programs (NLPs). As argued above, a (positive) observation $O = \neg G$ is distinguished from a negative observation $O = not\, G$, but such a distinction is not considered in NLPs handling only default negation. Inoue *et al*. [21] introduce *transaction programs* for computing *extended abduction*, which computes explanations for both positive/negative observations. A transaction program is a meta-level specification for computing the converse of conditionals, and is defined for NLPs only.

5.3. Human Conditional Reasoning

Stenning *et al*. [33] formulate human conditional reasoning using Clark's program completion under the three-valued logic of [14]. They represent a conditional sentence "if p then q" as a logic programming rule:"$q \leftarrow p \wedge \neg ab$" where ab represents an abnormal atom. In this setting, DA is represented as $\Pi_1 = \{\, p \leftarrow \bot,\quad q \leftarrow p \wedge \neg ab,\quad ab \leftarrow \bot \,\}$. The rule "$A \leftarrow \bot$" means that A is a proposition to which the *closed world assumption* [30] is applied. If a program does not contain $A \leftarrow \bot$, nor any other rule in which A occurs in its head, then A is interpreted *unknown*. Then its completion $Comp(\Pi_1) = \{\, p \leftrightarrow \bot,\quad q \leftrightarrow p \wedge \neg ab,\quad ab \leftrightarrow \bot \,\}$ derives $q \leftrightarrow \bot$. On the other hand, completion does not realize AC or DC inference by itself. In their framework, AC is represented as $\Pi_2 = \{\, q \leftarrow \top,\quad q \leftarrow p \wedge \neg ab,\quad ab \leftarrow \bot \,\}$, while $Comp(\Pi_2) = \{\, q \leftrightarrow \top \vee (p \wedge \neg ab),\quad ab \leftrightarrow \bot \,\}$ does not derive p. Likewise, DC is represented as $\Pi_3 = \{\, q \leftarrow \bot,\quad q \leftarrow p \wedge \neg ab,\quad ab \leftarrow \bot \,\}$, while $Comp(\Pi_3) = \{\, q \leftrightarrow \bot \vee (p \wedge \neg ab),\quad ab \leftrightarrow \bot \,\}$ does not derive $p \leftrightarrow \bot$. They then interpret $q \leftarrow p \wedge \neg ab$ as an *integrity constraint* meaning that "if q succeeds (resp. fails) then $p \wedge \neg ab$ succeeds (resp. fails)" to get the AC consequence p (resp. DC consequence $\neg p$).

Dietz *et al*. [11] point out a technical flaw in the formulation by [33]. Suppose a conditional sentence $p \leftarrow q$ where p and q are unknown U. Under the Fitting semantics, however, the truth value of the rule $\mathsf{U} \leftarrow \mathsf{U}$ is U, then it does not represent the truth of the sentence. To remedy the problem, they employ Łukasiewicz's 3-valued logic which maps $\mathsf{U} \leftarrow \mathsf{U}$ to $\top$.

Comparing the above mentioned two studies with our approach, there are several differences. First, they translate a conditional sentence "if p then q" into the rule $q \leftarrow p \wedge \neg ab$. However, it is unlikely that people who commit logical fallacies, especially younger children [32], translate the conditional sentence into the rule of the above complex form in their mind. We represent the conditional sentence directly as $q \leftarrow p$, and assume that people would interpret it as bi-conditional depending on the context it is used. Second, in order to characterize AC or DC inference, [33] interpret a conditional sentence as an integrity constraint, while [11] uses abductive logic programs. Our framework does not need a specific interpretation of rules (such as integrity constraints) nor need an extra mechanism of abductive logic programs. Third, they use a single (weak) completion for all AC/DA/DC inferences, while we introduce different types of completions for each inference. By separating respective completions, individual inferences are realized in a modular way and freely combined depending on their application context. Fourth, they handle normal logic programs, while our framework can handle a more general class of logic programs as GEDPs.

Cramer *et al*. [10] represent conditionals as in [33] and use the weak completion and abductive logic programs as in [11]. They formulate different types of conditionals based on their *contexts* and argue in which case AC or DC is more likely to happen. More precisely, a conditional sentence whose consequent appears to be obligatory given the antecedent is called an *obligation conditional*. An example of an obligation conditional is that "*if Paul rides a motorbike, then he must wear a helmet*". If the consequence of a conditional is not obligatory, then it is called a *factual conditional*. The antecedent A of a conditional sentence is said to be *necessary* iff its consequent C cannot be true unless A is true. For example, the *library being open* is a necessary antecedent for *studying in the library*. Cramer *et al*. argue that AA and DA occur independently of the type of a conditional. On the other hand, in AC most people will conclude A from $A \Rightarrow C$ and C, while the number of people who conclude nothing will increase if A is a non-necessary antecedent. In DC, most people will conclude $\neg A$ from $A \Rightarrow C$ and $\neg C$, while the number of people who conclude nothing will increase if the conditional is factual. Those assumptions are verified by questioning participants who do not receive any education in logic beyond high school training. They then formulate the situation by introducing the abducible $C \leftarrow \top$ if the antecedent is non-necessary, and $ab \leftarrow \top$ if the conditional is factual. In the former case, the observation C does not imply A because $C \leftarrow \top$ can make C explainable by itself. As a result, A is not a skeptical explanation of C. In the latter case, the observation $\neg C$ does not imply $\neg A$ because if one employs the explanation $ab \leftarrow \top$, $C \leftarrow A \wedge \neg ab$ does not produce $C \leftrightarrow A$.

Dietz *et al*. [12] use logic programming rules to represent different types of conditionals. For instance, the rule: "$\mathsf{concl} \leftarrow \mathsf{prem(x)}, \mathsf{sufficient(x)}$" represents MP that concl follows if a sufficient premise is asserted to be true. By contrast, "$\mathsf{not\_concl} \leftarrow \mathsf{not\_prem(x)}, \mathsf{necessary(x)}$" represents DA that concl does not follow if a necessary premise is asserted to be false. In the current study, we do not distinguish different types of conditionals as in [10, 12]. However, completion is done for individual rules, so we could realize *partial* completion by selecting rules $\Pi' \subseteq \Pi$ that are subject to be completed in practice. More precisely, if a program Π consists of rules R_1 having necessary antecedents and R_2 having non-necessary antecedents, apply AC completion to R_1 while keep R_2 as they are. The resulting program then realizes AC inference using R_1 only. Likewise, if a program Π consists of rules R_3 having obligatory consequents and R_4 having factual consequents, apply DC completion to R_3 while keep R_4 as they are. The resulting program then realizes DC inference using R_3 only.

6. Conclusion

This paper studies a method of realizing human conditional reasoning in ASP. Different types of completion are introduced to realize logically invalid inferences AC and DA as well as a logically valid inference DC. In psychology and cognitive science, empirical studies show that people perform AC, DA or DC inference depending on the context in which a conditional sentence is used. We could import the results of those studies and encode knowledge in a way that people are likely to use it. The proposed theory is used for such a purpose to realize pragmatic inferences in ASP and produce results that are close to human reasoning in practice.

Completions introduced in this paper are defined in a modular way, so one can apply respective completion to specific rules of a program according to their contexts. They are combined freely and can be mixed in the same program. Those completions are general in the sense that they are applied to logic programs containing disjunction, explicit and default negation. Since a completed program is still in the class of GEDPs and a GEDP is transformed to a semantically equivalent EDP [20], answer sets of completed programs are computed using existing answer set solvers. In the full paper, the proposed theory is applied to representing human reasoning tasks in the literature, and is used for computing common sense reasoning in AI.

References

[1] Alviano, M. & Dodaro, C. 2016. Completion of disjunctive logic programs. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pp. 886–892.

[2] Apt, K. R., Blair, H. A. & Walker, A. 1988. Towards a theory of declarative knowledge. In: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, pp. 89–148.

[3] Braine, M. D. S. 1978. On the relation between the natural logic of reasoning and standard logic. *Psychological Review* 85:1–21.

[4] Braine, M. D. S. & O'Brien, D. P. (eds.) 1998. *Mental Logic*. Mahwah, NJ: Erlbaum.

[5] Byrne, R. M. J. 2005. *The Rational Imagination: How People Create Alternatives to Reality*. Cambridge, MA: MIT Press.

[6] Cheng, P. W. & Holyoak, H. J. 1985. Pragmatic reasoning schemas. *Cognitive Psychology* 17:391–416.

[7] Clark, K. L. 1978. Negation as failure. In: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, pp. 293–322.

[8] Console, L., Dupré, D. T. & Torasso, P. 1991. On the relationship between abduction and deduction. *Journal of Logic and Computation* 1, pp. 661–690.

[9] Cosmides, L. & Tooby, J. 1992. Cognitive adaptions for social exchange. In: Barkow, J., Cosmides, L., Tooby, J. (eds.). *The adapted mind: Evolutionary psychology and the generation of culture*. Oxford University Press. pp. 163–228.

[10] Cramer, M., Hölldobler, S. & Ragnl, M. 2021. Modeling human reasoning about conditionals. In: *Proceedings of the 19th International Workshop on Non-Monotonic Reasoning (NMR-21)*, pp. 223–232.

[11] Dietz, E., Hölldobler, S. & Ragni, M. 2012. A computational approach to the suppression task. In: *Proceedings of the 34th Annual Conference of the Cognitive Science Society*, pp. 1500–1505.

[12] Dietz, E., Fichte, J. K. & Hamiti, F. 2022. A quantitative symbolic approach to individual human reasoning. In: *Proceedings of the 44th Annual Conference of the Cognitive Science Society*, pp. 2838–2846.

[13] Eichhorn, C., Kern-Isberner, G. & Ragni, M. 2018. Rational inference patterns based on conditional logic. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 1827–1834.

[14] Fitting, M. 1985. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming* 2:295–312.

[15] Fung, T. H. & Kowalski, R. 1997. The iff procedure for abductive logic programming. *Journal of Logic Programming* 33, pp. 151–165.

[16] Geis, M. L. & Zwicky, A. 1971. On invited inferences. *Linguistic Inquiry* 2:561–566.

[17] Gelfond, M. & Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

[18] Hölldobler, S. & Kencana Ramli, C. D. 2009. Logic programs under three-valued Lukasiewicz's semantics. In: *Proc. 25th International Conference on Logic Programming*, LNCS vol. 5649, Springer, pp. 464–478.

[19] Horn, L. R. 2000. From if to iff: conditional perfection as pragmatic strengthening. *Journal of Pragmatics* 32:289–326.

[20] Inoue, K. & Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming* 35:39–78.

[21] Inoue, K. & Sakama, C. 1999. Computing extended abduction through transaction programs. *Annals of Mathematics and Artificial Intelligence* 25, pp. 339–367.

[22] Johnson-Laird, P. N. 1983. *Mental models*. Cambridge, MA: Harvard Univ. Press.

[23] Kakas, A. C., Kowalski, R. A. & Toni, F. 1992. Abductive logic programming. *Journal of Logic and Computation* 2:719–770.

[24] Kowalski, R. A. 2011. *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge University Press.

[25] Lifschitz, V. & Woo, T. Y. C. 1992. Answer sets in general nonmonotonic reasoning (preliminary report). In: *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, pp. 603–614.

[26] Lifschitz, V., Pearce, D. & Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2:526–541.

[27] Lobo, J., Minker, J. & Rajasekar, A. 1988. Weak completion theory for non-Horn programs, In: *Proceedings of the 5th lnternational Conference& Symposium on Logic Programming*, MIT Press, pp. 828–842.

[28] Nieves, J. C. & Osorio, M. 2018. Extending well-founded semantics with Clark's completion for disjunctive logic programs. Hindawi Scientific Programming, Article ID 4157030.

[29] Oaksford, M. & Chater, N. 2001. The probabilistic approach to human reasoning. *Trends in Cognitive Science* 5:349–357.

[30] Reiter, R. 1978. On closed world data bases. In: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, pp. 119–140.

[31] Reiter, R. 1980. A logic for default reasoning, *Artificial Intelligence* 13:81–132.

[32] Rumain, B., Connell, J. & Braine, M. D. S. 1983. Conversational comprehension processes are responsible for reasoning fallacies in children as well as adults: IF is not the biconditional. *Developmental Psychology*, 19:471–481.

[33] Stenning, K. & van Lambalgen, M. 2008. *Human Reasoning and Cognitive Science*, MIT Press.