Human Conditional Reasoning in Answer Set Programming*

Chiaki Sakama

Wakayama University, 930 Sakaedani, Wakayama 640 8510, Japan

Abstract

Given a conditional sentence \( \varphi \Rightarrow \psi \) and respective facts, four different types of inferences are observed in human reasoning. Affirming the antecedent (AA) (or modus ponens) reasons \( \psi \) from \( \varphi \); affirmin the consequent (AC) reasons \( \varphi \) from \( \psi \); 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:

\[ S: \text{If the team wins the first round tournament, then it advances to the final round.} \]

\[ P: \text{The team wins the first round tournament.} \]

\[ C: \text{The team advances to the final round.} \]

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 “\( p \text{ if } q \)” is rephrased into the form “\( p \text{ only if } q \)” 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 \( p \text{ then } q \)” [20]. When a conditional is given as “\( p \text{ only if } q \)” on the other hand, it is rephrased as “\( \text{if not } q \text{ then not } p \)” 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 ‘\( \text{if } q \text{ only if } p \)’ as conveying supposition and introduces a set of pragmatic inference schemas for if-conditions. 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 “\( p \text{ then } q \)” 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...
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.

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:

$$L_1; \ldots; L_k; \not L_k+1; \ldots; \not L_l \leftarrow L_{l+1}, \ldots, L_m, \not L_{m+1}, \ldots, \not L_r \tag{1}$$

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 $\neg L = L$. 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 $\text{head}^+(r) = \{L_1, \ldots, L_k\}$, $\text{head}^-(r) = \{L_{k+1}, \ldots, L_l\}$, $\text{body}^+(r) = \{L_{l+1}, \ldots, L_m\}$, and $\text{body}^-(r) = \{L_{m+1}, \ldots, L_r\}$. A rule (1) is called a fact if $\text{body}^+(r) = \text{body}^-(r) = \emptyset$; and it is called a constraint if $\text{head}^+(r) = \text{body}^-(r) = \emptyset$. A GEDP II is called not-free if $\text{head}^+(r) = \text{body}^-(r) = \emptyset$ for each rule $r$ in $\Pi$.

A GEDP II coincides with an extended disjunctive program (EDP) of [17] if $\text{head}^+(r) = \emptyset$ for any rule $r$ in $\Pi$. An EDP II is called (i) an extended logic program (ELP) if $\text{head}^+(r) \leq 1$ for any $r \in \Pi$; and (ii) a normal disjunctive program (NDP) if $\Pi$ contains no negative literal. An NDP II is called (i) a positive disjunctive program (PDP) if $\Pi$ contains no NAF-literal; and (ii) a normal logic program (NLP) if $\text{head}^+(r) \leq 1$ 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 $S \subseteq \text{Lit}$ satisfies a ground rule $r$ of the form (i) iff $\text{body}^+(r) \subseteq S$ and $\text{body}^-(r) \cap S = \emptyset$ imply either $\text{head}^+(r) \cap S \neq \emptyset$ or $\text{head}^-(r) \subseteq S$. The answer sets of a GEDP are defined by the following two steps. First, let $\Pi$ be a not-free GEDP and $S \subseteq \text{Lit}$. Then, $S$ is an answer set of $\Pi$ iff $S$ is a minimal set satisfying the conditions: (i) $S$ satisfies every rule from $\Pi$, that is, for each ground rule: $L_1; \ldots; L_k \leftarrow L_{k+1}, \ldots, L_m$ from $\Pi$, $\{L_{k+1}, \ldots, L_m\} \subseteq S$ implies $\{L_1, \ldots, L_k\} \cap S \neq \emptyset$. In particular, for each constraint $L_{l+1}, \ldots, L_m$ from $\Pi$, $\{L_{l+1}, \ldots, L_m\} \not\subseteq S$. (ii) If $S$ contains a pair of complementary literals $L$ and $\neg L$, then $S = \text{Lit}$.

Second, let $\Pi$ be any GEDP and $S \subseteq \text{Lit}$. The reduct $\Pi^S$ of $\Pi$ by $S$ is a not-free EDP obtained as follows: a rule $r^S$ of the form: $L_1; \ldots; L_k \leftarrow L_{k+1}, \ldots, L_m$ is
in \( \Pi \)' if there is a ground rule \( r \) of the form (1) from \( \Pi \) such that head\(^+\)(\( r \)) \( \subseteq \) \( S \) and body\(^-\)(\( r \)) \( \cap \) \( S \) = \( \emptyset \). For programs of the form \( \Pi \)'\( ^2 \), their answer sets have already been defined. Then, \( S \) is an answer set of \( \Pi \) if \( S \) is an answer set of \( \Pi \)'\( ^2 \).

When a program \( \Pi \) 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 \( \Pi \) satisfies every rule from \( \Pi \) [20]. An answer set is consistent if it is not \( \text{Lit} \) set. A program \( \Pi \) is consistent if it has a consistent answer set; otherwise, \( \Pi \) is inconsistent. When a program \( \Pi \) is inconsistent, there are two different cases. If \( \Pi \) has the answer set \( \text{Lit} \), \( \Pi \) is called contradictory; else if \( \Pi \) has no answer set, \( \Pi \) is called incoherent.

The difference of two cases is illustrated by the following example.

**Example 2.1.** The program \( \Pi_1 = \{ p \leftarrow \neg q, \neg p \leftarrow \} \) is incoherent, while \( \Pi_2 = \{ p \leftarrow q, q \leftarrow, \neg p \leftarrow \} \) is contradictory. Note that \( \text{Lit} \) is not the answer set of \( \Pi_1 \) because \( \text{Lit} \) is not the answer set of \( \Pi_1' = \{ \neg p \leftarrow \} \).

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

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

**Example 2.2.** Let \( \Pi \) be the program:

\[
p; \neg q \leftarrow, \quad q; \neg p \leftarrow
\]

Then \( \Pi \) has two answer sets \( S \) and \( \{ p, q \} \).

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

\[
\neg L_{k+1}; \cdots; \neg L_1 \leftarrow L_{l+1}, \ldots, L_m, \neg L_{m+1}, \ldots, \neg L_n.
\]

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

\[
\leftarrow L_{k+1}, \ldots, L_1, L_{l+1}, \ldots, L_m,
\]

that is obtained by shifting \( \neg L_{k+1}; \cdots; \neg L_1 \) in head\(^-\)(\( r \)) to \( L_{l+1}, \ldots, L_1 \) in body\(^-\)(\( \eta(r) \)).

Proposition 2.2 (20]). Let \( \Pi \) be a program and \( \Phi = \{ r \mid r \in \Pi \text{ and } \text{head}^+(r) = \emptyset \} \). Also, let \( \Pi' = (\Pi \setminus \Phi) \cup \{ \eta(r) \mid r \in \Phi \} \). Then \( \Pi \) and \( \Pi' \) have the same answer sets.

**Example 2.3.** For any \( \Pi \), \( \Pi \cup \{ \neg p \leftarrow q, \neg p \} \) is equivalent to \( \Pi \cup \{ p, q, \neg p \} \) (Proposition 2.1), which is further simplified to \( \Pi \) (Proposition 2.2).

**Example 2.4.** Let \( \Pi \) be a not-free GEDP. If there is a constraint in \( \Pi \), then \( \Pi \) 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 \( \varphi \Rightarrow \psi \) is represented by the rule \( \varphi \leftarrow \psi \) where \( \psi \) is a disjunction “\( L_1; \ldots; L_{n-1}; \not L_{n+1}; \ldots; \not L_m; \)”. We write \( \Pi \models \varphi \Rightarrow \psi \) if a literal \( \varphi \) is in the answer set of \( \Pi \cup \{ \neg p \leftarrow q, \) if \( \varphi \text{ is incoherent} \), while \( \varphi \text{ is incoherent} \), \( \Pi \) is an answer set. Two programs \( \Pi_1 \) and \( \Pi_2 \) are equivalent if they have the same an answer set.

Proposition 2.3. Let \( \Pi \) be a not-free GEDP. If there is a constraint in \( \Pi \), then \( \Pi \) is not contradictory.

---

2We often use the parenthesis ‘( )’ to improve the readability.
Example 3.1. Let \( \Pi = \{ p \not\rightarrow q, r, s, p \not\rightarrow q \} \). Then, \( \text{ac}(\Pi) = \{ r, s \not\rightarrow q \} \) and \( \Pi' = \{ q \not\rightarrow p \} \). By Proposition 2.1, \( \Pi' \) and \( \Pi'' \) are strongly equivalent, but \( \text{AC}(\Pi') = \Pi' \cup \{ q \not\rightarrow p \} \) and \( \text{AC}(\Pi'') = \Pi'' \). As a result, \( \text{AC}(\Pi') \) has the answer set \( \{ q \} \) while \( \text{AC}(\Pi'') \) has the answer set \( \varnothing \).

In the above example, \( \not\rightarrow p \not\rightarrow q \) is a conditional sentence which is subject to AC inference, while \( \not\rightarrow p, q \) 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 \( \Pi \models_s L \) (or \( \Pi \models_s L \)) does not imply \( \text{AC}(\Pi) \models_s L \) (or \( \text{AC}(\Pi) \models_s L \)) in general.

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

In Example 3.3, reasoning by AC produces \( q \) which blocks deriving \( p \) using the first rule in \( \Pi \). 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 \( p=\"\text{online-meeting is held on time}\" \), \( q=\"\text{network trouble}\" \), \( r=\"\text{the web browser is unconnected}\" \). In this case, one may withdraw the conclusion \( p \) after knowing \( r \). As such, additional rules \( \text{ac}(\Pi) \) may change the results of \( \Pi \). One can see the effect of AC reasoning in a program II by comparing answer sets of \( \Pi \) and \( \text{AC}(\Pi) \).

A consistent program \( \Pi \) may produce an inconsistent \( \text{AC}(\Pi) \). In converse, an inconsistent \( \Pi \) may produce a consistent \( \text{AC}(\Pi) \).

Example 3.4. \( \Pi_1 = \{ p \not\rightarrow p, p \not\rightarrow p \} \) is consistent, but \( \text{AC}(\Pi_1) = \Pi_1 \cup \{ q \not\rightarrow p \} \) is contradictory. \( \Pi_2 = \{ \not\rightarrow p \not\rightarrow p, q \not\rightarrow q \} \) is incoherent, but \( \text{AC}(\Pi_2) = \Pi_2 \cup \{ p \not\rightarrow q \} \) is consistent.
A sufficient condition for the (in)consistency of $AC(\Pi)$ is given below.

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

**Proposition 3.2.** If a program $\Pi$ 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 $\Pi$ be a program. For each rule $r \in \Pi$ of the form (1), define $wdc(r)$ as the rule:

$$
\neg L_{i+1}; \ldots; \neg L_m; L_{m+1}; \ldots; L_n \leftarrow \neg L_1, \ldots, \neg L_k, L_{k+1}, \ldots, L_l.
$$

(6)

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

$$
\neg L_{i+1}; \ldots; \neg L_m; L_{m+1}; \ldots; L_n \leftarrow \neg L_1, \ldots, \neg L_k, L_{k+1}, \ldots, L_l.
$$

(7)

In particular, (6) or (7) becomes a fact if $head^+(r) = head^-(r) = \emptyset$; and it becomes a constraint if $body^+(r) = body^-(r) = \emptyset$. The weak DC completion and the strong DC completion of $\Pi$ 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 $\neg$ and NAF-literals $\neg L_j$ ($k + 1 \leq i \leq l$; $m + 1 \leq i \leq n$) 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 $\neg L_j$ ($k + 1 \leq i \leq l$; $m + 1 \leq i \leq n$) 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 $\Pi$ is an EDP. The WDC and SDC produce different results in general.

**Example 3.5.** Let $\Pi = \{p \leftarrow \neg q\}$. Then, $WDC(\Pi) = \{p \leftarrow \neg q, q \leftarrow \neg p\}$ and $SDC(\Pi) = \{p \leftarrow \neg q, q \leftarrow \neg p\}$. $WDC(\Pi)$ 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 $\Pi \models p$ but $WDC(\Pi) \not\models p$. SDC is also nonmonotonic (see Example 3.8). The result of DC completion is syntax-dependent in general.

**Example 3.6.** Let $\Pi_1 = \{\neg p \leftarrow q\}$ and $\Pi_2 = \{p, q\}$. Then, $SDC(\Pi_1) = \Pi_1 \cup \{\neg q \leftarrow p\}$ and $SDC(\Pi_2) = \Pi_2 \cup \{\neg p, \neg q\}$. As a result, $SDC(\Pi_1)$ has the answer set $\emptyset$ while $SDC(\Pi_2)$ has two answer sets $\{\neg p\}$. $\{\neg q\}$.

WDC keeps the consistency of the original program.

**Proposition 3.3.** If a program $\Pi$ 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 $\Pi = \{\neg not p\}$ has no answer set, while $WDC(\Pi) = \{\langle \neg not p\rangle\}$ has the answer set $\{p\}$.

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

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

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

**Proposition 3.5.** If a program $\Pi$ 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 $\Pi = \{p \leftarrow q, \neg p \leftarrow \emptyset\}$ does not deduce $\neg q$. SDC completes the program as $SDC(\Pi) = \Pi \cup \{\langle \neg q \leftarrow \neg p\rangle, \langle \neg p\rangle\}$ 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 $\Pi$ be a program and $r \in \Pi$ a rule of the form (1). First, inverse the implication:

$$
not L_i \leftarrow not L_{i+1}; \ldots; not L_m; L_{m+1}; \ldots; L_n.
$$

(8)

(1 $\leq i \leq k$),
\( L_i \leftarrow \neg L_{i+1}; \cdots ; \neg L_m ; L_{m+1}; \cdots ; L_n \) (9)

\( k + 1 \leq i \leq l \).

In (8) and (9), the disjunction not \( L_{i+1}; \cdots ; \neg L_m ; L_{m+1}; \cdots ; L_n \) appears on the right of \( \leftarrow \). The produced (8) (resp. (9)) is considered an abbreviation of the collection of \((n - l)\) rules: \((\neg L_i \leftarrow \neg L_{i+1}; \cdots ; \neg L_m , \cdots , L_{m+1}; \cdots ; L_n)\) (resp. \((\neg L_i \leftarrow \neg L_{i+1}; \cdots ; L_{i-1} \leftarrow L_n)\)), hence we abuse the term ‘rule’ and call (8) or (9) a rule. In particular, (8) is not produced if head\(^+\)(r) = \( \emptyset \) or body\(^+\)(r) = body\(^-\)(r) = \( \emptyset \); and (9) is not produced if head\(^-\)(r) = \( \emptyset \) or body\(^-\)(r) = body\(^+\)(r) = \( \emptyset \).

Example 3.9. Let \( \Pi = \{ p \leftarrow q, p \leftarrow r \} \) produces \( \text{uda}(\Pi) = \{ \neg p \leftarrow \neg q, \neg p \leftarrow \neg r \} \) but not \( \Lambda = \{ \neg p \leftarrow \neg q, \neg p \leftarrow \neg r \} \). Suppose the new fact \( r \) \( \leftarrow \) is added to \( \Pi \). Put \( \Pi' = \Pi \cup \{ r \leftarrow \} \). Then \( \text{WDA}(\Pi') \) has the answer set \( \{ p, r \} \). If \( \Lambda \) is used instead of \( \text{uda}(\Pi) \), however, \( \Pi' \cup \Lambda \) is incoherent because the first rule of \( \Lambda \) 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 \( \Lambda \). Hence, we conclude \( \neg p \) if both \( q \) and \( r \) are negated in \( \text{uda}(\Pi) \).

The strong DA completion is defined in a similar way.

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

\(-L_i \leftarrow \neg L_{i+1}; \cdots ; \neg L_m; L_{m+1}; \cdots ; L_n \) (10)

\(1 \leq i \leq k \),

\(-L_i \leftarrow \neg L_{i+1}; \cdots ; \neg L_m; L_{m+1}; \cdots ; L_n \) (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: \((\neg L_i \leftarrow \neg L_{i+1}; \cdots , \neg L_m , \cdots , L_{m+1}; \cdots , L_n)\) (resp. \((\neg L_i \leftarrow \neg L_{i+1}; \cdots , (L_i \leftarrow L_m)\)), 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 \( \text{sinv}(\Pi) \). Next, define

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

where \( \ell_i \) is either a literal \( L_i \) \((k + 1 \leq i \leq l)\) or an NAF literal \( \neg L_i \) \((1 \leq i \leq k)\), and each \( \Gamma_j \) \((1 \leq j \leq p)\) is a disjunction of literals and NAF literals. The weak DA completion of \( \Pi \) is defined as:

\[ \text{WDA}(\Pi) = \Pi \cup \text{uda}(\Pi) \].

(8) and (9) in \( \text{winv}(r) \) represent inverse implication from the (default) negation of the conjunct in the body of \( r \) to the (default) negation of the disjunction in the head of \( r \). \( \text{uda}(\Pi) \) collects rules \( \ell_i \leftarrow \Gamma_j \) \((1 \leq j \leq p)\) having the same (NAF)-literal \( \ell_i \) on the left of \( \leftarrow \), and constructs \( \ell_i \leftarrow \Gamma_1, \ldots , \Gamma_p \), which we call an extended rule. Introducing \( \text{uda}(\Pi) \) to \( \Pi \) 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:

\[ \ell_i \leftarrow (l_1 \cdots , l_{m_1}) , \ldots , (r_p \cdots , r_{m_p}) \]

is identified with the set of \( m_1 \times \cdots \times m_p \) rules of the form: \( \ell_i \leftarrow l_{j_1} \cdots l_{j_{m_1}} , \ldots , r_{p_{k_p}} \leftarrow \ell_1, \ldots , \ell_{k_p} \leftarrow 1 \leq j_k \leq m_k; 1 \leq k \leq p \).

By this fact, \( \text{WDA}(\Pi) \) is viewed as a GEDP and we do not distinguish extended rules and rules of a GEDP hereafter. The semantics of \( \text{WDA}(\Pi) \) is defined by its answer sets.

Example 3.9. Let \( \Pi = \{ p \leftarrow q, p \leftarrow r, \neg s, q; \neg r \leftarrow t, s \leftarrow \} \). Then, \( \text{uda}(\Pi) = \{ \neg p \leftarrow \neg q, \neg p \leftarrow \neg r; s, \neg q \leftarrow \neg r; s, \neg t, r \leftarrow \neg t \} \) where the first rule “\( \neg p \leftarrow \neg q; s \)” is identified with “\( \neg p \leftarrow \neg r; s \)” and “\( \neg p \leftarrow \neg s; t \)” and the second rule “\( \neg q \leftarrow \neg r; s \)” is identified with “\( \neg q \leftarrow \neg r; t \)” and “\( \neg q \leftarrow \neg s, t \)”.

Then, \( \text{WDA}(\Pi) \) 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 \( \text{uda}(\Pi) = \{ \neg p \leftarrow \neg q, \neg p \leftarrow \neg r \} \) but not \( \Lambda = \{ \neg p \leftarrow \neg q, \neg p \leftarrow \neg r \} \). Suppose the new fact \( r \) \( \leftarrow \) is added to \( \Pi \). Put \( \Pi' = \Pi \cup \{ r \leftarrow \} \). Then \( \text{WDA}(\Pi') \) has the answer set \( \{ p, r \} \). If \( \Lambda \) is used instead of \( \text{uda}(\Pi) \), however, \( \Pi' \cup \Lambda \) is incoherent because the first rule of \( \Lambda \) 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 \( \Lambda \). Hence, we conclude \( \neg p \) if both \( q \) and \( r \) are negated in \( \text{uda}(\Pi) \).

Both WDA and SDA are nonmonotonic in general.
Example 3.11. (1) $\Pi_1 = \{ p \leftarrow \neg q, \ \neg q \leftarrow p \}$ produces $WDA(\Pi_1) = \Pi_1 \cup \{ \neg p \leftarrow q, \ q \leftarrow \not p \}$. Then, $\Pi_1 \models \not p$ but $WDA(\Pi_1) \not\models p$. (2) $\Pi_2 = \{ p \leftarrow \not r, \ r \leftarrow q, \ q \leftarrow \not r \}$ produces $SDA(\Pi_2) = \Pi_2 \cup \{ \not p \leftarrow \not r, \ \not r \leftarrow q \}$. Then, $\Pi_2 \models \not p$ but $SDA(\Pi_2) \not\models p$.

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

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

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

As in the case of AC, a consistent program $\Pi$ may produce an inconsistent $WDA(\Pi)$ or $SDA(\Pi)$. In converse, an incoherent $\Pi$ 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) $\Pi_2 = \{ -p \leftarrow p, \ -p \leftarrow \}$ is consistent, but $WDA(\Pi_2) = \Pi_2 \cup \{ p \leftarrow -p \}$ is incoherent.

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

(4) $\Pi_4 = \{ \leftarrow \not p, \ -p \leftarrow n, \ q \leftarrow \}$ is incoherent, but $SDA(\Pi_4) = \Pi_4 \cup \{ \not p \leftarrow \not q \}$ 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:

- (default AC) $\phi \Rightarrow \psi \land \psi : \phi$
- (default DA) $\phi \Rightarrow \psi \land \neg \phi : \neg \psi$

The default AC rule says: given the conditional $\phi \Rightarrow \psi$ and the fact $\psi$, conclude $\phi$ 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 $\Pi$ be a program. For each rule $r \in \Pi$ of the form (1), define $dac(r)$ as the set of rules:

$$L_{i+1}, \ldots, L_m, \not L_{m+1}, \ldots, not L_n \leftarrow L_i$$

(12)

$$L_{i+1}, \ldots, L_m, \not L_{m+1}, \ldots, not L_n \leftarrow \not L_i$$

(13)

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

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

in which

$$dac(\Pi) = \{ \Sigma_1 : \cdots : \Sigma_p \leftarrow \ell_j, \Delta_i \ | \Sigma_i \leftarrow \ell_j, \Delta_i \ (1 \leq i \leq p) \}$$

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

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

$$dac(\Pi) = \{ q \leftarrow p, \not q, \ q \leftarrow p, \not r \}$$

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

Proposition 4.1. Let $\Pi$ be a consistent program. If $DAC(\Pi)$ has an answer set $S$, then $S \not\models Lis$.

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

Example 4.2 (cont. Example 3.4). Let $\Pi_1 = \{ p \leftarrow \not p, \ p \leftarrow \}$. Then, $DAC(\Pi_1)$ is $\Pi_1 \cup \{ \not p \leftarrow p, \ not p \}$ 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 \( \Pi = \{ p \leftarrow q, \ p \leftarrow, \ \leftarrow q \} \). Then, \( AC(\Pi) = \Pi \cup \{ q \leftarrow p \} \) is incoherent. \( DAC(\Pi) = \Pi \cup \{ q \leftarrow p, \not\neg q \} \) is still incoherent.

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

**Proposition 4.2.** Let \( \Pi \) 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 \( \Pi \) be a program. Define

\[
\begin{align*}
WDDA(\Pi) &= \{ \ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p, \delta_p^w | \\
& \text{ } \ell_i \leftarrow \Gamma_j, (1 \leq j \leq p) \text{ is in } \bigcup_{r \in \Pi} winv(r) \}, \\
SDDA(\Pi) &= \{ \ell_i \leftarrow \Gamma_1, \ldots, \Gamma_p, \delta_p^s | \\
& \text{ } \ell_i \leftarrow \Gamma_j, (1 \leq j \leq p) \text{ is in } \bigcup_{r \in \Pi} sinv(r) \}
\end{align*}
\]

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

\[
\begin{align*}
WDDA(\Pi) &= \Pi \cup \text{WDDA}(\Pi), \\
SDDA(\Pi) &= \Pi \cup \text{SDDA}(\Pi).
\end{align*}
\]

Rules in \( \text{wdda}(\Pi) \) and \( \text{sdda}(\Pi) \) 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 \( \Pi \) be a consistent program. If \( WDDA(\Pi) \) (resp. \( SDDA(\Pi) \)) has an answer set \( S \), then \( S \neq \text{Lit} \).

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

**Example 4.4 (cont. Example 3.12).** Let \( \Pi_1 = \{ \not p \leftarrow p \} \) where \( WDDA(\Pi_1) \) is incoherent.

\( WDDA(\Pi_1) = \Pi_1 \cup \{ p \leftarrow \not p, \not p \} \) is still incoherent. Let \( \Pi_2 = \{ \not p \leftarrow p, \neg p \leftarrow \} \) where \( SDA(\Pi_2) \) is contradictory. \( SDDA(\Pi_2) = \Pi_2 \cup \{ p \leftarrow \neg p, \not p \} \) has the answer set \( \{ \neg p \} \).

**Proposition 4.4.** Let \( \Pi \) 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 \( \Pi \), Clark completion \( Comp(\Pi) \) is obtained by two steps: (i) all rules \( p \leftarrow B_1, \ldots, p \leftarrow B_k \) in \( \Pi \) having the same head \( p \) are replaced by \( p \leftarrow B_1 \lor \cdots \lor B_k \), 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 \( \Pi \), add \( p \leftarrow false \). 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: \( \Pi_1 = \{ p \leftarrow q, \ p \leftarrow \} \), Clark completion becomes \( Comp(\Pi_1) = \{ p \leftarrow q \lor \top, \ q \leftarrow \bot \} \) 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 they are different from the AC completion. We also introduce the DC completion and the DA completion. When \( \Pi_2 = \{ p \leftarrow \not q, \ \not q \leftarrow \bot \} \) has the supported model \( \{ p \} \) while \( WDC(\Pi_2) = \Pi_2 \cup \{ q \leftarrow \not p \} \) has two answer sets \( \{ p \} \) and \( \{ q \} \). When \( \Pi_3 = \{ p \leftarrow not q, \ p \leftarrow q, \ p \leftarrow \} \), \( Comp(\Pi_3) = \{ p \leftarrow q \lor \neg q, \ q \leftarrow \bot \} \) has the supported model \( \{ p, q \} \) while \( WDA(\Pi_3) = \Pi_3 \cup \{ not p \leftarrow q, \not q, \not q \leftarrow \not p \} \) 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 \( \Pi_4 = \{ p \leftarrow q, \ p \leftarrow \} \),
the weak completion becomes \( w\text{comp}(\Pi_1) = \{ p \leftrightarrow \top \} \). Then \( p \) is true but \( q \) is unknown in \( w\text{comp}(\Pi_1) \), which is again different from the result of \( AC(\Pi_1) \) that has the answer set \( \{ p, q \} \). In \( \Pi_2 = \{ p \leftarrow \neg q \} \), \( w\text{comp}(\Pi_2) = \{ p \leftarrow \neg q \} \), 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 \((\Pi, \Gamma)\) where \( \Pi \) is a program and \( \Gamma \subseteq \text{Lit} \) is a set of literals called abducibles. It is assumed that abducibles appear in the head of no rule in \( \Pi \). 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 \( \models \) or \( \models_x \) depending on the problem. Here we consider \( \models \), that realizes credulous abduction. Consider \((\Pi_1, \Gamma)\) where \( \Pi_1 = \{ \text{arrive on time} \leftarrow \neg \text{accident} \}, \neg \text{arrive on time} \leftarrow \text{accident} \} \) and \( \Gamma_1 = \{ \text{accident} \} \). \( \Pi_1 \) represents that a train arrives on time unless there is an accident. Then the observation \( O = \neg \text{arrive on time} \) has the explanation \( E = \{ \text{accident} \} \). Since abduction reasons backward from an observation, it is computed using the AC completion. Let \((\Pi, \Gamma)\) be an abductive program and \( O \) an observation. Then, a set \( S \subseteq \Gamma \) is an explanation of \( O \) if \( O \in \text{head}(r) \) 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, \( AC(\Pi_1) \cup \{ O \} = \Pi_1 \cup ac(\Pi_1) \cup \{ O \} \) where \( ac(\Pi_1) = \{ \text{not accident} \leftarrow \text{arrive on time} \}, \neg \text{arrive on time} \leftarrow \text{accident} \} \), \( \Pi_1 \cup \{ O \} \) has the answer set \( S = \{ \neg \text{arrive on time}, \text{accident} \} \). Then, \( S \cap \Gamma_1 = \{ \text{accident} \} \) 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 \( (\Pi_2, \Gamma_2) \), while \( AC(\Pi_2) \cup \{ O \} = \Pi_2 \cup \{ a \leftarrow p, \neg a \leftarrow q, \pi \leftarrow \} \) 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 \((\Pi_3, \Gamma_1)\) where \( \Pi_3 = \{ \text{arrive on time} \leftarrow \neg \text{accident} \} \). Given the observation \( O = \neg \text{arrive on time} \), no explanation is obtained from \((\Pi_3, \Gamma_1) \). Generally, a program \( \Pi \) 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 \( \Pi_4 \), \( sdc(r) = \{ \text{accident} \leftarrow \neg \text{arrive on time} \} \). Then \( SDC(\Pi_4) \cup \{ O \} \) 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 \( O = \neg G \), which should be distinguished from the (positive) observation \( O = \neg G \) meaning that \( G \) is observed. In the abductive program \((\Pi_3, \Gamma_1) \), the negative observation \( O = \neg \text{arrive on time} \) is explained using \( wdc(r) = \{ \text{accident} \leftarrow \neg \text{arrive on time} \} \). 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 \( O = \neg G \), 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 “\( q \leftarrow p \land \neg ab \)” where \( ab \) represents an abnormal atom. In this setting, DA is represented as \( \Pi_1 = \{ p \leftarrow \bot, q \leftarrow p \land \neg ab, ab \leftarrow \bot \} \). 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 \( Comp(\Pi_1) = \{ p \leftarrow \bot, q \leftarrow p \land \neg ab, ab \leftarrow \bot \} \) derives \( q \leftarrow \bot \). On the other hand, completion does not realize AC or DC inference by itself. In their framework, AC is represented as \( \Pi_2 = \{ q \leftarrow T, q \leftarrow p \land \neg ab, \bot \leftarrow \bot \} \), while \( Comp(\Pi_2) = \{ q \leftarrow T \lor (p \land \neg ab), \bot \leftarrow \bot \} \) does not derive \( p \). Likewise, DC is represented as \( \Pi_3 = \{ q \leftarrow \bot, q \leftarrow p \land \neg ab, \bot \leftarrow \bot \} \), while \( Comp(\Pi_3) = \{ q \leftarrow \bot \lor (p \land \neg ab), \bot \leftarrow \bot \} \) does not derive \( p \). They then interpret \( q \leftarrow p \land \neg ab \) as an integrity constraint meaning that “if \( q \) succeeds (resp. fails) then \( p \land \neg ab \) 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 \( U \leftarrow U \) 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 \( U \leftarrow U \) 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 \( q \leftarrow p \land \neg ab. \) 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 admissible \( C \leftarrow \top \) 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 \( ab \leftarrow \top \), \( C \leftarrow A \land \neg ab \) does not produce \( C \leftarrow A \).

Dietz et al. [12] use logic programming rules to represent different types of conditionals. For instance, the rule: \("\text{concl} \leftarrow \text{prem}(x), \text{sufficient}(x)\)" represents MP that concl follows if a sufficient premise is asserted to be true. By contrast, \("\text{not\_concl} \leftarrow \text{not\_prem}(x), \text{necessary}(x)\)" 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 \( \Pi \) 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 \( \Pi \) 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


