A Formal Theory of Justifications

Marc Denecker[1] and Gerhard Brewka[2] and Hannes Strass[2]

[1]Department of Computer Science, K.U. Leuven, 3001 Heverlee, Belgium
[2]Computer Science Institute, Leipzig University, Leipzig, Germany

Abstract. We develop an abstract theory of justifications suitable for describing the semantics of a range of logics in knowledge representation, computational and mathematical logic. A theory or program in one of these logics induces a semantical structure called a justification frame. Such a justification frame defines a class of justifications each of which embodies a potential *reason* why its facts are true. By defining various evaluation functions for these justifications, a range of different semantics are obtained. By allowing nesting of justification frames, various language constructs can be integrated in a seamless way. The theory provides elegant and compact formalisations of existing and new semantics in logics of various areas, showing unexpected commonalities and interrelations, and creating opportunities for new expressive knowledge representation formalisms.

1 Introduction

In this paper we introduce a new semantical framework suitable for describing semantics of a broad class of existing and new (nonmonotonic) logics. These logics are from the area of knowledge representation, nonmonotonic reasoning and mathematical logic. The purpose of the framework is four-fold: (1) By providing a unified semantical account of different semantical principles, it highlights the differences between different semantics within the same formalism (e.g., various semantics of logic programs) and (2) it highlights common semantical principles of different formalisms (e.g. logic programs vs. argumentation frameworks); (3) for existing formalisms it gives rise to new semantics nobody has thought of before; (4) last but not least it provides ways for seamless integration of various expressive language constructs in a single logic.

As for (4), it is a fundamental goal of the field of knowledge representation to build expressive languages, that is, ones that provide a range of different language constructs. It is well-known that (certainly for nonmonotonic logics) extending a logic with new language constructs can be very difficult. An illustration is the saga of extending logic and answer set programming with aggregates [1, 2], a topic on which many effort-years, several PhD theses and countless papers have been devoted. Clearly, it would be of great value to have a clean, modular way to compose (nonmonotonic) logics from existing language constructs. A powerful such principle is presented here, in the form of *nesting*.

The framework we present is based on so-called justification frames. They specify a class of defined and parameter facts and include a set of semantic rules $x \leftarrow S$ (x a defined fact, S a set of facts) that provide potential reasons for defined facts x. Justifications J are graphs obtained by concatenation of such reasons. In the context of an interpretation $\mathfrak{A}$, a justification J may justify a fact x or not, depending on the

branches below x in J. The value of these branches in $\mathfrak{A}$ is specified by a mapping from sequences of facts to $\{t, f\}$, called a *branch evaluation*. A justification system consists of a justification frame together with a branch evaluation. A *supported interpretation* of such a system is one in which true facts x are those with a justification J in which all branches from x evaluate to t. A very useful feature of the framework is the nesting of justification systems. This yields a powerful way for meaning-preserving integration of different language constructs by nesting the justification system of one construct in that of another.

Several semantical frameworks of a seemingly similar kind already exist. In particular, our framework is a (substantial) extension of a justification framework for logic programming proposed in [3]. Another framework that comes to mind is approximation fixpoint theory (AFT) developed by Denecker, Marek and Truszczyński [4]. We believe that the framework presented here has some clear advantages over AFT:

- AFT is a more abstract algebraical operator-based approach, whereas the justification framework rests on a logical notion: *a justification as a reason for a fact to hold*. The advantage of AFT's abstract approach is that it is applicable to a broader range of logics, (e.g., logic programming, AEL, default logic). The advantage of the logical approach here is that it is much more intuitive.
- (Direct) application of AFT induces non-standard *ultimate* variants of stable and well-founded semantics [5], whereas the justification framework formalizes standard versions.
- The justification framework identifies a single source of difference in various semantics, namely the way infinite branches are evaluated in justification graphs.
- The nesting of justification systems is a new technique for seamless integration of complex language constructs (e.g., aggregates, rule sets) in one logic. Nesting is a feature derived from μ-calculus [6] and nested fixpoint logics [7]. This feature is not supported by AFT nor in any other semantic frameworks that we know of.
- Last but not least, justifications are used in the implementation of practical systems such as clasp [8] and IDP [9].

As a consequence, to the best of our knowledge, no other framework (including AFT) is currently capable of formalizing and integrating the logics treated in this paper. These logics are from mathematical logic and formal methods, knowledge representation and nonmonotonic reasoning, and include logic programs and answer set programs under various semantics [10], abstract argumentation [11], inductive definitions, co-inductive logic programming [12] and nested inductive/coinductive definitions [13].

2 Justification Frames

Let $\mathcal{F}$ be a set and $\sim$: $\mathcal{F} \to \mathcal{F}$ an involution, that is, a bijection that is its own inverse (hence for all $x \in \mathcal{F}$, $\sim\sim x = x$). We assume $\mathcal{F}$ has a partition $\{\mathcal{F}^p, \mathcal{F}^n\}$ such that $\sim\mathcal{F}^p = \mathcal{F}^n$ (and vice versa). We call elements of $\mathcal{F}$ facts, of $\mathcal{F}^p$ positive facts, and of $\mathcal{F}^n$ negative facts. We call $\sim x$ the complement of x. The operator $\sim$ is called the complementation operator. A good intuiton for the complementation operator is as (classical) negation. We frequently call $\mathcal{F}$ a *fact space*.

As will become clear, it can be useful to have facts $t, u, i \in \mathcal{F}^p$ and their complements $\sim t, \sim u, \sim i \in \mathcal{F}^n$. We call them *logical* facts. For convenience, we denote $\sim t$

as f. They are interpreted facts. They stand respectively for *true, unknown (positive), inconsistent (positive), false, unknown (negative) and inconsistent (negative).*

Definition 1. *An interpretation $\mathfrak{A}$ is a subset of $\mathcal{F}$ such that $t, i, {\sim}i \in \mathfrak{A}$ and $f, u, {\sim}u \notin \mathfrak{A}$ when present in $\mathcal{F}$. The set of interpretations is denoted $\mathbb{I}$. An interpretation $\mathfrak{A}$ is* consistent *if for every non-logical fact $x \in \mathcal{F}$, $\mathfrak{A}$ contains at most one of x and ${\sim}x$. $\mathfrak{A}$ is* anti-consistent *if for every non-logical fact $x \in \mathcal{F}$, $\mathfrak{A}$ contains at least one of x and ${\sim}x$. $\mathfrak{A}$ is* exact *if consistent and anti-consistent.*

Interpretations as defined above can be viewed as 4-valued interpretations, consistent interpretations correspond to 3-valued interpretations, exact interpretations to standard 2-valued interpretations. In particular, x is true in $\mathfrak{A}$ if $x \in \mathfrak{A}$ and ${\sim}x \notin \mathfrak{A}$; false if $x \notin \mathfrak{A}$ and ${\sim}x \in \mathfrak{A}$; unknown if $x, {\sim}x \notin \mathfrak{A}$ and inconsistent if $x, {\sim}x \in \mathfrak{A}$. Thus, interpretations assign each of t, f, u, i its standard truth value (if present in $\mathcal{F}$).

Example 1. Let $\mathcal{F}$ be the set of literals of a propositional vocabulary Σ. The positive facts are the atoms (i.e., $\mathcal{F}^p = \Sigma$), the negative facts are the negative literals (classical negation) (i.e., $\mathcal{F}^n = \{\neg p \mid p \in \Sigma\}$. The complementation operator ${\sim}$ is the obvious one. For $a, b, c, d \in \Sigma$, the set $\mathfrak{A} = \{a, \neg a, b, \neg c\}$ is a 4-valued interpretation where a is inconsistent, b is true, c is false and d is unknown.

Using the vocabulary of facts and their negations, we can formulate rules that can be used to justify the truth of facts.

Definition 2. *A justification frame $\mathcal{JF}$ is a structure $(\mathcal{F}, \mathcal{F}_d, R)$ such that:*

- *$\mathcal{F}_d \subseteq \mathcal{F}$ is closed under ${\sim}$, that is, ${\sim}\mathcal{F}_d = \mathcal{F}_d$;*
- *$t, f, u, {\sim}u, i, {\sim}i \notin \mathcal{F}_d$;*
- *$R \subseteq \mathcal{F}_d \times 2^{\mathcal{F}}$.*

We view a tuple $(x, S) \in R$ as a rule and present rules as $x \leftarrow S$. We call S a case *of x in $\mathcal{JF}$ if $(x, S) \in R$. The set of cases of x in $\mathcal{JF}$ is denoted $\mathcal{JF}(x)$. We define the set of parameter facts of $\mathcal{JF}$ as $\mathcal{F} \setminus \mathcal{F}_d$ and denote it as $\mathcal{F}_o$; we also sometimes write $x \leftarrow S \in \mathcal{JF}$ and mean $(x, S) \in R$.*

A justification frame contains a set of rules. The interpretation varies. We may view $\mathcal{F}_d$ as a set of facts defined by their rules, while $\mathcal{F}_o$ is a set of parameter symbols. Or we might view $\mathcal{F}_d$ as endogenous facts in a causal system, while $\mathcal{F}_o$ are exogenous facts. The idea is that the endogenous facts are governed by causal relationships described by the rules, while exogenous facts are governed by the external environment (e.g., a human agent or external system). More interpretations are possible.

It is possible that for some $x \in \mathcal{F}_d$, there are no rules with x in the head ($\mathcal{JF}(x) = \emptyset$). Then x is never "justified". It is also possible that $x \leftarrow \emptyset \in R$, then x is always "justified". The role of the parameters is illustrated below.

Example 2. We construct a justification frame for defining the transitive closure of a graph. Consider a set V of nodes and define $\mathcal{F}_o = \{E(a, b), {\sim}E(a, b) \mid a, b \in V\}$. Each exact interpretation of $\mathcal{F}_o$ determines a graph $G = (V, E)$. Set the defined facts to $\mathcal{F}_d = \{Path(a, b), {\sim}Path(a, b) \mid a, b \in V\}$ and all facts to $\mathcal{F} = \mathcal{F}_d \cup \mathcal{F}_o$. The

intended interpretation of fact $Path(a, b)$ is that there is a path from a to b in graph G. The rules R correspond to those of the monotone inductive definition that $Path$ is the transitive closure of E:

$$R = \{Path(a, b) \leftarrow \{E(a, b)\}, Path(a, b) \leftarrow \{Path(a, c), Path(c, b)\} \mid a, b, c \in V\}$$

Later we will see how to derive the rules for negative facts $\sim Path(a, b)$ and how to determine the interpretation for the defined facts given an arbitrary interpretation of the parameter facts. Note that the rules define $Path$ for each choice of edges E but do not constrain G in any way. It is in this respect that $E(a, b)$ literals are called parameters.

We next associate an operator on interpretations with each justification frame. This operator is – in essence – like (Fittings 4-valued version of) the immediate consequence operator T_P for logic programs [14, 15]. It takes as input an interpretation and returns an interpretation containing exactly those defined facts that are justified by the input.

Definition 3. *We define the* derivation operator *of $\mathcal{JF}$ as the mapping $T_{\mathcal{JF}} : \mathbb{I}_{\mathcal{F}} \to \mathbb{I}_{\mathcal{F}_d}$ that maps an interpretation $\mathfrak{A}$ of $\mathcal{F}$ to an interpretation $T_{\mathcal{JF}}(\mathfrak{A})$ of $\mathcal{F}_d$ such that for each $x \in \mathcal{F}_d$, we set $x \in T_{\mathcal{JF}}(\mathfrak{A})$ iff there exists $x \leftarrow S \in \mathcal{JF}$ such that $S \subseteq \mathfrak{A}$.*

The framework below will be geared towards systems where each case of fact x provides a sufficient condition for x while the set of cases of x represent a necessary condition for x in the sense that if x is true, then at least one case must apply. Stated more concisely, the framework is geared towards fixpoints of the operator. However, this is still quite vague (an operator may have many sorts of fixpoints) and will be refined later (when we define various "branch evaluations"). This operator can be used to define when two given justification frames are equivalent.

Definition 4. *Two justification frames $\mathcal{JF}, \mathcal{JF}'$ are equivalent (denoted $\mathcal{JF} \equiv \mathcal{JF}'$) if and only if $T_{\mathcal{JF}} = T_{\mathcal{JF}'}$.*

We call a rule $x \leftarrow S$ redundant in R if there is a more general rule $x \leftarrow S' \in R$ such that $S' \subsetneq S$. Redundant rules may always be deleted from the rule set of a justification frame, as long as the more general rule stays. Formally, let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ and $Re \subseteq R$ be a set of rules each of which is redundant in $R \setminus Re$. Then it is easy to see that $\mathcal{JF}' = \langle \mathcal{F}, \mathcal{F}_d, R \setminus Re \rangle$ is equivalent to $\mathcal{JF}$. However, it is not always possible to remove all redundant rules from a justification frame.

Example 3. Take the (infinite) justification frame $\mathcal{JF}$ with $\mathcal{F} = \{p, \sim p, q_i, \sim q_i \mid i \in \mathbb{N}\}$, $\mathcal{F}_d = \{p, \sim p\}$ and $R = \{p \leftarrow \{q_n, q_{n+1}, \dots\} \mid n \in \mathbb{N}\}$. Every rule of R is redundant in R. Deleting all of them leads to a justification frame that is not equivalent to $\mathcal{JF}$.

We will only be concerned with justification frames where every defined fact has at least one rule, and rule bodies are not empty. We call them *proper* justification frames.

Definition 5. *A justification frame $\mathcal{JF}$ is* proper *if for all $x \in \mathcal{F}_d$, we have $\mathcal{JF}(x) \neq \emptyset$ and $x \leftarrow \emptyset \notin \mathcal{JF}$.*

Each justification frame can be translated to an equivalent proper one. For each $\mathcal{JF}$, we define its proper justification frame as $\mathcal{JF}'$ with identical sets of parameter facts and defined facts and the following rules (for $x \in \mathcal{F}_d$):

- all rules $x \leftarrow S \in \mathcal{JF}$ such that $S \neq \emptyset$;
- rule $x \leftarrow \{t\}$ if $x \leftarrow \emptyset \in \mathcal{JF}$;
- rule $x \leftarrow \{f\}$ if $x \in \mathcal{F}_d$ and $\mathcal{JF}(x) = \emptyset$.

Proposition 1. *$\mathcal{JF} \equiv \mathcal{JF}'$, that is, $T_{\mathcal{JF}}$ and $T_{\mathcal{JF}'}$ are identical on all interpretations.*

We will – for readability – sometimes present justification frames that are not proper and take them to mean their equivalent proper justification frames.

3 Justifications and Branch Evaluations

Next, we define the central concept of the paper, a justification for a given justification frame. This will be our first step in defining the semantics of justification frames.

Definition 6. *Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame. A $\mathcal{JF}$-*justification *J is a subset of R containing at most one rule $x \leftarrow S$ for each $x \in \mathcal{F}_d$. J is called* complete *if for each $x \in \mathcal{F}_d$ there is some $x \leftarrow S$ in J. If $x \leftarrow S \in J$, we denote $J(x) = S$. If J is not complete we call it* partial.

Alternatively, a justification J can be seen as a partial function from $\mathcal{F}_d$ to $2^{\mathcal{F}}$ such that $x \leftarrow J(x) \in R$ if J is defined in $x \in \mathcal{F}_d$. If some $x \in \mathcal{F}_d$ has no rules $x \leftarrow S$ (i.e., if $\mathcal{JF}(x) = \emptyset$), then no complete justification exists for $\mathcal{JF}$.

Proper justification systems $\mathcal{JF}$ have the interesting property that complete justifications exist and that justifications J are equivalent to a special class of directed graphs G on domain $\mathcal{F}$ such that if a fact x has children in G, then $x \in \mathcal{F}_d$ and the set S of its children is a case of x in $\mathcal{JF}$ (that is, $x \leftarrow S$ is a rule of $\mathcal{JF}$). It is easy to see that if $\mathcal{JF}$ is proper, there is a one-to-one correspondence between justifications J and such graphs G. Indeed, given J, we derive $G = \{(x, y) \mid y \in J(x)\}$; vice versa, given G, J is defined in x if x has children, and then $J(x) = \{y \mid (x, y) \in G\}$. Notice that this correspondence is not one-to-one in case $\mathcal{JF}$ is not proper. For example, if $x \leftarrow \emptyset \in \mathcal{JF}$, then if x has no children in G, it is unclear whether J is undefined in x or whether $J(x) = \emptyset$. By Proposition 1, we may impose (w.l.o.g.) the condition that justification frames are proper. In examples we will represent and treat justifications as graphs.

Proposition 2. *For any proper $\mathcal{JF}$, a complete justification corresponds to a graph whose leaves are $\mathcal{F}_o$ and for each non-leaf x with children S exists a rule $x \leftarrow S$ in $\mathcal{JF}$.*

A complete justification J contains for each fact $x \in \mathcal{F}_d$ a potential reason (or a justification, or argument, or cause, etc.) for x to be true: this reason is expressed in the subtree of J below x. Of course, not every such reason is good. It can be flawed for external reasons (e.g., it is based on a parameter fact y that is false in the world) or because of intrinsic reasons (e.g., there is a cyclic argument). In the framework defined here, the support given by J to x in an interpretation $\mathfrak{A}$ is determined by evaluating the branches below x in J. With each branch B we can associate a unique fact, denoted $\mathcal{B}(B)$, so that B evaluates positively in $\mathfrak{A}$ iff $\mathcal{B}(B) \in \mathfrak{A}$. Thus, J justifies x iff $\mathcal{B}(B) \in \mathfrak{A}$, for all branches leaving x. Below, we formalize these concepts.

Definition 7. *An $\mathcal{F}_d$-branch B (briefly, a branch) of $\mathcal{JF}$ for $x_0 \in \mathcal{F}_d$ is an infinite sequence $x_0 \to x_1 \to \ldots$ such that $x_i \in \mathcal{F}_d$ or a finite sequence $x_0 \to \ldots \to x_n$ such that $x_i \in \mathcal{F}_d$ for $i < n$ and $x_n \in \mathcal{F}_o$. An infinite branch (compactly, an ∞-branch) is positive (negative) if it has a tail of positive (negative) facts. It is mixed if neither positive nor negative. A branch evaluation $\mathcal{B}$ is a mapping from branches to facts.*

A branch contains at least two facts. In a mixed ∞-branch, each tail contains infinitely many positive and negative facts.

Definition 8. *A branch of a complete justification J from defined fact x_0 is a maximally long path $x_0 \to x_1 \to \ldots$ in J. (Hence, $x_{i+1} \in J(x_i)$ for all $i \geq 0$.)*

It is obvious that a branch of a complete J from x is a branch in the sense of Definition 7. (This property holds for proper justification frames but not in general).

Example 4. We define four branch evaluations that later will be shown to induce well-known logic programming semantics. For every branch $B = x_0 \to x_1 \to \ldots$, we define $\mathcal{B}_{sp}(B) = x_1$. (The subscript refers to "supported" semantics.) Next, we define three more branch evaluations all of which map B to its leaf x_n if B is a finite branch $x_0 \to \ldots \to x_n$. If B is an ∞-branch, we have:

- (Kripke-Kleene) $\mathcal{B}_{KK}(B) = u$ if $x_0 \in \mathcal{F}^p$ and $\mathcal{B}_{KK}(B) = {\sim}u$ if $x_0 \in \mathcal{F}^n$.
- (stable) $\mathcal{B}_{st}(B) = t$ if B consists of negative facts only; $\mathcal{B}_{st}(B) = f$ if B consists of positive facts only; otherwise, $\mathcal{B}_{st}(B) = x_i$ if x_i is the first fact in B with another sign than x_0.
- (well-founded) $\mathcal{B}_{wf}(B) = t$ if B is a negative ∞-branch; $\mathcal{B}_{wf}(B) = f$ if B is a positive ∞-branch; $\mathcal{B}_{wf}(B) = u$ if B is mixed and $x_0 \in \mathcal{F}^p$; $\mathcal{B}_{wf}(B) = {\sim}u$ if B is mixed and $x_0 \in \mathcal{F}^n$.

The names suggest a connection to different semantics of logic programs. This will be explored below.

Definition 9. *Given some branch evaluation $\mathcal{B}$, we say that $x \in \mathcal{F}_d$ is supported by J in $\mathfrak{A}$ (under $\mathcal{B}$) if for all branches $B = x \to \ldots$ in J, we find $\mathcal{B}(B) \in \mathfrak{A}$. We say that x is supported by $\mathcal{JF}$ in $\mathfrak{A}$ (under $\mathcal{B}$) if there exists a complete justification J of $\mathcal{JF}$ such that x is supported by J in $\mathfrak{A}$. We denote the set of supported facts by $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\mathfrak{A})$.*

Using the specific branch evaluation $\mathcal{B}_{sp}$ allows to express the derivation operator associated to a justification frame.

Proposition 3. *For $\mathcal{B}_{sp}$ we have $\mathcal{S}^{\mathcal{B}_{sp}}_{\mathcal{JF}}(\mathfrak{A}) = T_{\mathcal{JF}}(\mathfrak{A})$.*

This property does not hold for other branch evaluations. Each combination of justification frame $\mathcal{JF}$ and branch evaluation $\mathcal{B}$ induces an operator $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\cdot)$ from interpretations of $\mathcal{F}$ to interpretations of $\mathcal{F}_d$.

Proposition 4. *If $\mathcal{JF}, \mathcal{JF}'$ are equivalent (i.e., they induce the same operator) then for each branch evaluation $\mathcal{B}$, the operators $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\cdot)$ and $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}'}(\cdot)$ are identical.*

The meaning of a justification frame is not only specified by the set of its rules but also by the selected branch evaluation. The same set of rules may have a different meaning for a different branch evaluation. This is captured in the next definition, another central concept of the paper: a *justification system* is a justification frame extended with a branch evaluation.

Definition 10. *A justification system is a structure $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ with $\langle \mathcal{F}, \mathcal{F}_d, R \rangle$ a justification frame and $\mathcal{B}$ a branch evaluation on that frame.*

Again, we can associate an operator with a justification system just like we did for justification frames. We only have to additionally take into account the specific branch evaluation at hand.

Definition 11. *Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system and let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be its included justification frame. With $\mathcal{JS}$ we associate the operator $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\cdot) : \mathbb{I}_{\mathcal{F}} \to \mathbb{I}_{\mathcal{F}_d}$ and denote the mapping of an interpretation $\mathfrak{A}$ under this operator as $\mathcal{JS}(\mathfrak{A})$. The justified interpretations of a justification system $\mathcal{JS}$ are the fixpoints of $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\cdot)$.*

The operator $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\cdot) : \mathbb{I}_{\mathcal{F}} \to \mathbb{I}_{\mathcal{F}_d}$ can only be iterated if its domain and co-domain are identical, that is, if $\mathcal{F}_o = \emptyset$. There is a simple way to fix this: each operator O from $\mathbb{I}_{\mathcal{F}}$ to $\mathbb{I}_{\mathcal{F}_d}$ has a canonical extension $O' : \mathbb{I}_{\mathcal{F}} \to \mathbb{I}_{\mathcal{F}}$ defined as $O'(\mathfrak{A}) = O(\mathfrak{A}) \cup (\mathfrak{A} \cap \mathcal{F}_o)$. The extended operator copies the interpretation of the parameters and can be iterated.

4 Reconstructions

This section shows how several established knowledge representation formalisms can be reconstructed within our theory of justifications. Often, formalisms make implicit semantic assumptions – e.g. in logic programs any atom without a rule is considered to be false. The next definitions show how our theory makes such assumptions explicit.

Take a justification frame $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ and a fact $x \in \mathcal{F}_d$. We can view the body of a rule $x \leftarrow S \in R$ as a logical conjunction of literals, $\delta_S = \bigwedge_{y \in S} y$. The set of all cases for x then can be thought of as a (possibly infinite) disjunction of such conjunctions, $\gamma_x = \bigvee_{x \leftarrow S \in R} \delta_S$. In a sense, γ_x characterizes the truth value of x in any given interpretation. Intuitively, our definition of complement closure below aims to construct the rules that are needed to characterize the negation of x, the fact $\sim x$. To obtain these rules we use the negation of γ_x, that is, $\neg\gamma_x = \neg \bigvee_{x \leftarrow S \in R} \delta_S \equiv \bigwedge_{x \leftarrow S \in R} \neg\delta_S$. To get actual rules according to the definition of a justification system, we consider the DNF of $\neg\gamma_x$, that is, all possible ways of making all possible cases for x inapplicable.

Definition 12. *A selection function for $x \in \mathcal{F}_d$ is a function $\mathbf{S}$ from the set $\mathcal{JF}(x)$ of cases of x to $\mathcal{F}$ such that $\mathbf{S}(S) \in S$ for each $S \in \mathcal{JF}(x)$. A complement selection of $x \in \mathcal{F}_d$ in $\mathcal{JF}$ is a set $\{\sim \mathbf{S}(S) \mid S \in \mathcal{JF}(x)\}$ for some selection function $\mathbf{S}$ for x.*

The complement selections of x correspond to the disjuncts in the DNF of $\neg\gamma_x$. Each selects at least one element from each case of x and adds the negations of all selected elements in one set. It can be seen that if all elements of this set are true, then the bodies of all rules of x are false. Vice versa, if the bodies of all rules of x are false then all elements of at least one complement selection are true.

Definition 13. *Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame such that either one of: (1) $x \leftarrow S \in R$ implies that $x \in \mathcal{F}^p$ (there are no rules for negative facts); or (2) $x \leftarrow S \in R$ implies that $x \in \mathcal{F}^n$ (there are no rules for positive facts). The complement closure of $\mathcal{JF}$ is the justification frame $\langle \mathcal{F}, \mathcal{F}_d, R \cup R^c \rangle$ where R^c consists of all rules $\sim x \leftarrow S$ with $x \in \mathcal{F}_d$ and S a complement selection of x in $\mathcal{JF}$.*

Example 5 (Continuation of Example 2). The complement closure of the justification frame for transitive closure contains all possible rules of the form $\sim Path(a, b) \leftarrow S$, where S is any subset of $\mathcal{F}$ that contains at least $\sim E(a, b)$ and for every $c \in V$ either the literal $\sim Path(a, c)$ or the literal $\sim Path(c, b)$.

We now look at how existing semantics of existing formalisms can be reconstructed using justification systems.

Argumentation Our first reconstruction shows how Dung-style argumentation frameworks (AFs) [11] give rise to justification frames. Argumentation frameworks are a simple, popular formalism for representing arguments and attacks between these arguments. More precisely, an AF is a pair $F = (A, X)$ where A is a set of (atomic) arguments and $X \subseteq A \times A$ is a binary relation ("attack") on arguments. Intuitively, if $(a, b) \in X$, then argument a attacks argument b.

Definition 14. *Given an AF $F = (A, X)$, the justification frame associated with F is $\mathcal{JF}_F = (\mathcal{F}, \mathcal{F}_d, R)$ such that $\mathcal{F} = \mathcal{F}_d = A \cup \sim A$ and $R = \{\sim a \leftarrow \{b\} \mid (b, a) \in X\}$.*

Thus in the resulting fact space $\mathcal{F}$ there is a fact a for each argument $a \in A$, and a fact $\sim a$ for the opposite of each argument $a \in A$. All of these facts are (going to be) defined. The rules of $\mathcal{JF}_F$ for negative $\sim a \in \mathcal{F}$ encode the meaning of "attack": an argument a is rejected (that is, its opposite $\sim a$ is true) if one of its attackers b is accepted (b is true). The complement closure $\mathcal{JF}^c_F$ will additionally contain the rules $R^c = \{a \leftarrow \{\sim b \mid (b, a) \in X\} \mid a \in A\}$. Intuitively, these derived rules express that for an argument a to be accepted, all its attackers b must be rejected (that is, their opposites $\sim b$ must be true). Using one and the same branch evaluation, namely $\mathcal{B}_{sp}$, we can show that justification systems allow us to reconstruct the major semantics of abstract argumentation frameworks.

Proposition 5. *Let $F = (A, X)$ be an argumentation framework and $\mathcal{JF}^c_F$ be the complement closure of its associated justification frame $\mathcal{JF}_F$. A consistent interpretation $\mathfrak{A}$*

- *is an exact fixpoint of $T_{\mathcal{JF}^c_F}$ iff it is stable for F;*
- *is a fixpoint of $T_{\mathcal{JF}^c_F}$ iff it is complete for F;*
- *is a $\subseteq$-maximal fixpoint of $T_{\mathcal{JF}^c_F}$ iff it is preferred for F;*
- *is the $\subseteq$-least fixpoint of $T_{\mathcal{JF}^c_F}$ iff it is grounded for F;*
- *satisfies $\mathfrak{A} \subseteq T_{\mathcal{JF}^c_F}(\mathfrak{A})$ iff it is admissible for F.*

Proof (sketch). Consistent interpretations $\mathfrak{A}$ can be seen as three-valued interpretations on the set A. It can be shown that $T_{\mathcal{JF}_F}$ is (isomorphic to) the three-valued characteristic operator of the AF F. The claims then follow from Propositions 4.4 to 4.9 in [16]. □

Four out of five of these types of fact sets correspond to specific sorts of fixpoints of $T_{\mathcal{JF}^c_F}$. Thus an argument (or its opposite) belongs to such a set iff it is justified by one of its cases. The exceptions are the admissible sets, which are only postfixpoints: facts in an admissible set have to be justified by it but not all facts justified by such a set must belong to the set. Although we technically use sets, these semantics are three-valued and thus closer to the notion of a labelling than to that of a set-based extension.

Logic programming Justification frames differ from propositional logic programs in two ways: (a) the presence of a set $\mathcal{F}_o$ of parameter literals (whose interpretation is not defined by the rules), and (b) the presence of rules with negation in the head (which via complement closure can be derived from those for positive literals).[1]

Definition 15. *Let Π be a (propositional) logic program over atoms Σ. The justification frame associated with Π is the structure $\mathcal{JF}_\Pi = (\mathcal{F}, \mathcal{F}_d, \Pi)$ where $\mathcal{F}_d$ is the set of all literals over Σ, and $\mathcal{F} \setminus \mathcal{F}_d = \mathcal{F}_o$ is the set of logical facts.*

We assume without loss of generality that $\mathcal{JF}_\Pi$ is proper, i.e. it contains at least one rule per non-logical fact (possibly $p \leftarrow \{f\}$) and rule bodies are non-empty. While the above definition is about propositional logic programs, the approach easily generalizes to the predicate case by simply instantiating the rules using first-order interpretations. We now establish the connection between branch evaluations and various semantics of logic programs. Since Π contains only rules for atoms, we apply complement closure.

Theorem 1. *Let Π be a logic program and $\mathcal{JF}$ be the complement closure of $\mathcal{JF}_\Pi$.*

- *An exact interpretation $\mathfrak{A}$ is an exact fixpoint of $\mathcal{S}^{\mathcal{B}_{sp}}_{\mathcal{JF}}(\cdot)$ iff $\mathfrak{A}$ is a supported model of Π.*
- *An interpretation $\mathfrak{A}$ is a fixpoint of $\mathcal{S}^{\mathcal{B}_{KK}}_{\mathcal{JF}}(\cdot)$ iff $\mathfrak{A}$ is the Kripke Kleene model of Π.*
- *An interpretation $\mathfrak{A}$ is an exact fixpoint of $\mathcal{S}^{\mathcal{B}_{st}}_{\mathcal{JF}}(\cdot)$ iff $\mathfrak{A}$ is a stable model of Π.*
- *An interpretation $\mathfrak{A}$ is a fixpoint of $\mathcal{S}^{\mathcal{B}_{wf}}_{\mathcal{JF}}(\cdot)$ iff $\mathfrak{A}$ is the well-founded model of Π.*

For some branch evaluations $\mathcal{B}$, the value of $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\mathfrak{A})$ depends entirely on the value of parameter facts in $\mathfrak{A}$. This is the case if the branch evaluation maps every branch to a parameter fact.

Definition 16. *A branch evaluation $\mathcal{B}$ is parametric if for every branch B, $\mathcal{B}(B)$ is a parameter fact.*

Proposition 6. *If $\mathcal{B}$ is parametric, then for every parameter interpretation $\mathfrak{A}_p \subseteq \mathcal{F}_o$, $\mathfrak{A} = \mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\mathfrak{A}_p)$ is the unique fixpoint of $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}(\cdot)$ such that $\mathfrak{A} \cap \mathcal{F}_o = \mathfrak{A}_p$.*

The branch evaluations $\mathcal{B}_{KK}$ and $\mathcal{B}_{wf}$ are parametric. They are used in logic programming semantics that have a unique model. $\mathcal{B}_{sp}$ and $\mathcal{B}_{st}$ are not parametric, and thus, supported and stable semantics admit for any number of models.[2]

[1] Negation in the head of (extended) answer set programs is different from the negation studied here, and the justification semantics defined below is not directly suitable to compute answer sets of programs with explicit negation. We focus on systems where the rules for facts and their negations are complementary, hence negation is classical. In contrast, rules of ASP for negative literals are independent of those for positive literals.

[2] By dropping the constraint that $\mathcal{F}_o$ consists of logical facts only, we obtain extensions of all main semantics for a parameterized variant of logic programming.

Example 6. Consider the branch evaluation $\mathcal{B}_{cowf}(B)$ defined like $\mathcal{B}_{wf}$ except that positive ∞-branches are evaluted to t and negative ones to f. The semantics induced by $\mathcal{B}_{cowf}$ is a sort of well-founded semantics that "prefers" maximal models. This induces a coinductive semantics as in, for example, μ-calculus [6] and coinductive logic programming [12]. For an illustration, consider the set D of finite and infinite lists over $\{A, B\}$ and $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R\rangle$ where $\mathcal{F}_d = \{P(s) \mid s \in D\}$ and $\mathcal{F}$ the extension of $\mathcal{F}_d$ with logical facts and, using Prolog notation $[H|T]$ for lists with head H and tail T, the rule set $R = \left\{ P([A, B|s]) \leftarrow \{P(s)\} \mid s \in D \right\}$. After taking the complement closure, an interpretation $\mathfrak{A}$ is a fixpoint of $\mathcal{S}^{\mathcal{B}_{cowf}}_{\mathcal{JF}}(\cdot)$ iff $\mathfrak{A} = \{P([A, B, A, B, A, B, \dots])\}$. Had we used $\mathcal{B}_{wf}$ ("preferring" minimal models), the fixpoint would have been $\emptyset$.

5 Nested Justification Systems

Modularity and composition are key properties of knowledge representation languages. We compose (parametric) justification systems by *nesting* them. It is important to note that nesting as presented in this section is restricted to parametric justification systems for reasons that will become clear soon.

Definition 17. *Let $\mathcal{F}$ be a set of facts. A nested justification system on $\mathcal{F}$ is a tuple $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_{dg}, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \dots, \mathcal{JS}^k\}\rangle$ where:*

- *$\langle \mathcal{F}, \mathcal{F}_{dl}, R, \mathcal{B}\rangle$ is a parametric justification system.*
- *Each $\mathcal{JS}^i$ is a nested justification system on fact space $\mathcal{F}^i = (\mathcal{F} \setminus \mathcal{F}_{dg}) \cup \mathcal{F}^i_{dg}$ with (globally) defined facts $\mathcal{F}^i_{dg}$.*
- *$\mathcal{F}_{dg}$ is the disjoint union of $\mathcal{F}_{dl}$ and $\mathcal{F}^1_{dg}, \dots, \mathcal{F}^k_{dg}$.*

A nested justification system is a tree-like definition that defines the set $\mathcal{F}_{dg}$ of globally defined facts. This set is partitioned into $k+1$ subsets. One subset, $\mathcal{F}_{dl}$, consists of facts that are *locally* defined in the root of the tree by R. The rest of the facts are defined in one of the k nested subdefinitions $\mathcal{JS}^i$ of $\mathcal{JS}$. The branch evaluation of $\mathcal{JS}$ is defined for branches of locally defined facts only. The parameters of subdefinitions $\mathcal{JS}^i$ are those of $\mathcal{JS}$ augmented with the locally defined facts. In particular, for each $\mathcal{JS}^i$, facts defined in siblings $\mathcal{JS}^j$ with $j \neq i$ are not to appear as parameters of $\mathcal{JS}^i$. In leaves of the tree, we have $k = 0$ and $\mathcal{F}_{dg} = \mathcal{F}_{dl}$.

The semantics of nested justification systems is based on two notions: compression and unfolding. We start explaining the latter. Let R_1 be a set of rules defining facts $\mathcal{F}_{d1} \subseteq \mathcal{F}$, R a second set of rules in fact space $\mathcal{F}$. The unfolding of R_1 on $\mathcal{F}_{d1}$ in R, denoted $UNF_{(\mathcal{F}_{d1}, R_1)}(R)$, is the set of rules that can be obtained from any $x \leftarrow S \in R$ by replacing each fact $y \in S$ defined in R_1, in an arbitrary way, by the body facts of a rule $y \leftarrow S' \in R_1$. E.g., suppose R contains rule $a \leftarrow \{g, b\}$ and R_1 contains the rules $b \leftarrow \{c, d\}$, $b \leftarrow \{f\}$ for b. Then unfolding R_1 on $\{b\}$ in R replaces that rule of R by two rules, $a \leftarrow \{g, c, d\}$ and $a \leftarrow \{g, f\}$. Compression turns a nested definition into an (equivalent) unnested one.

Definition 18. *Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_{dg}, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \dots, \mathcal{JS}^k\}\rangle$ be a justification system. Its compression $\mathcal{C}(\mathcal{JS})$ is defined inductively: $\mathcal{C}(\mathcal{JS}) = \langle \mathcal{F}, \mathcal{F}_{dg}, R_c, \mathcal{B}\rangle$ where*

$$R_c = R^s \cup UNF_{(\mathcal{F}_{dg} \setminus \mathcal{F}_{dl}, R^s)}(R)$$

with $R^s = R_c^1 \cup \cdots \cup R_c^k$ *and* R_c^i *is the set of rules* $x \leftarrow S$ *such that* $x \in \mathcal{F}_{dg}^i$ *and* $S = \left\{\mathcal{B}^i(B) \;\middle|\; B \text{ is a branch of } x \text{ in } J\right\}$ *for some complete justification* J *of* $\mathcal{JS}^i$.

Notice that in the base case $k = 0$, justification system $\mathcal{JS}$ and its compression $\mathcal{C}(\mathcal{JS})$ are essentially the same. Now we see why all branch evaluations $\mathcal{B}$ used in different nodes must be parametric, and why definitions cannot use facts defined in nodes not on the path from the root to the current node. Under these conditions, subdefinitions $\mathcal{JS}^i$ are translated in an equivalence preserving way in a set of flat rules $x \leftarrow S$ where S contains only parameter facts and locally defined facts of $\mathcal{JS}$. With the set R^s of all these rules, we eliminate non-locally defined facts in bodies of the local definition R by unfolding R^s on $\mathcal{F}_{dg} \setminus \mathcal{F}_{dl}$ in R, thus producing rules that contain only parameter and locally defined bodies of $\mathcal{JS}$. For the resulting definition R_c, we use $\mathcal{B}_{sp}$ for branches of facts of $\mathcal{F}_{dg} \setminus \mathcal{F}_{dl}$ and the locally given $\mathcal{B}$ for branches of locally defined facts. Hence, the operator $\mathcal{S}^{\mathcal{B}}_{\mathcal{C}(\mathcal{JS})}(\cdot)$ is well-defined and its fixpoints define the semantics of $\mathcal{JS}$.

Example 7. We define a nested justification system with $\mathcal{JS}^2$ nested in $\mathcal{JS}^1$. Take the list domain D as in Example 6, $\mathcal{F}_{dl}^1 = \{P(s) \mid s \in D\}$ and $\mathcal{F}_{dl}^2 = \{Q(s) \mid s \in D\}$, $\mathcal{B}^1 = \mathcal{B}_{cowf}$ and $\mathcal{B}^2 = \mathcal{B}_{wf}$ and finally,

$$R_1 = \left\{\, P(s) \leftarrow \{Q(s)\} \mid s \in D \,\right\}$$
$$R_2 = \left\{\, Q([A|s]) \leftarrow \{P(s)\} \mid s \in D \,\right\} \cup \left\{\, Q([B|s]) \leftarrow \{Q(s)\} \mid s \in D \,\right\}$$

After taking the complement closure of $\mathcal{JS}^1$ and its compression, the rules defining positive facts are, for each $s \in D$:

$$P([B, \ldots, B, A|s]) \leftarrow \{P(s)\} \quad \text{and} \quad Q([B, \ldots, B, A|s]) \leftarrow \{P(s)\}$$

In the unique supported interpretation of the compression, both P and Q are the set of all lists with infinitely many occurrences of A.

In our final example, we show how our justification framework can treat logic programs with aggregates in rule bodies. In particular, this illustrates the power of nesting.

Example 8. Consider a logic program rule with a weight constraint, that is,

$$p \leftarrow i \leq \{l_1, \ldots, l_n\} \leq j \tag{1}$$

with $1 \leq i \leq j \leq n$, meaning that p is true if at least i and at most j literals from the set $L = \{l_1, \ldots, l_n\}$ are true. An LP rule (1) is translated into the set of $\mathcal{JF}$ rules

$$R_p = \left\{p \leftarrow L^+ \cup {\sim}L^- \;\middle|\; L^+, L^- \subseteq L \text{ and } \left|L^+\right| = i \text{ and } \left|L^-\right| = n - j\right\}$$

Alternatively, we can use nested definitions. A weight-constraint rule (1) appears in the top-level definition with $q_{i \leq \{l_1,\ldots,l_n\} \leq j}$ a single new atom in the body. Such new atoms are then defined by a nested definition $q_{i \leq \{l_1,\ldots,l_n\} \leq j} \leftarrow L^+ \cup {\sim}L^-$ with L^+ and L^- as in R_p above. It is not difficult to see that the compression of the second approach yields the first. This application further clarifies aggregates, nesting and compression.

6 Discussion

Justifications as mathematical semantical constructs have appeared in different ways in different areas. In [17, 18], stable and answer set semantics are defined for programs using justifications similar to ours. Phrased in the terms of our paper, atoms x are justified by sets $S = \{\mathcal{B}_{st}(B) \mid B \text{ is a branch of } x \text{ in } J\}$ for some complete justification J of the program. Tree-shaped justifications were used in [3] to build a semantical framework for (abductive) logic programming. [20] present an algebra of tree-shaped justifications for atomic positive facts for logic programming. [21] propose justification graphs for justifying the truth value of atoms in answer sets of logic programs. Our study differs on the technical level and generalizes these works in several dimensions, e.g. by considering parameters, alternative branch evaluations (e.g. coinduction), nesting and novel applications (e.g. to argumentation frameworks). Justifications as datastructures are used in the ASP solver clasp [8] and in the FO(ID) model expander IDP3 [9]. Justifications are underlying provenance systems in the context of databases [22].

The justification framework defined above is of great amplitude and much uncharted territory lies in front. It covers a remarkable amount of existing semantics in different areas. Here we showed this for argumentation frameworks and logic programming. The framework also induces new and more general versions of these formalisms. For example it comprises nested logic programs with negation and feedback, a new formalism that remains to be studied.[3] Alternative branch evaluations can be introduced. For example, some have argued that in the logic program $\{P \leftarrow \neg P\}$, P should be inconsistent while in $\{P \leftarrow \neg Q, Q \leftarrow \neg P\}$, P and Q should be undefined [24]. A refinement of $\mathcal{B}_{wf}$ that would distinguish between these cases would be the one that assigns i to branches B with complementary facts $x, \sim x$. This remains to be explored. The justification framework may be applicable to many other logics as well. We already mentioned coinductive logic programming [12] (as illustrated by Example 6); we expect the framework to cover other mathematical and knowledge representation logics of nested induction/coinduction such as μ-calculus [6], FO(LFP) with nesting [7] and FO(FD) [13]. We believe that our justification theory can also be applied to assumption-based argumentation [19] and abstract dialectical frameworks [25]. The approach is promising as well for logics of causality such as FO(C) [26]. All these connections still need to be investigated. As mentioned in Section 1, we know of only one other approach with a comparable coverage: Approximation Fixpoint Theory (AFT) [4]. While AFT is defined in a different way (as an algebraical fixpoint theory of lattice operators), it was used to characterize about the same semantics for the same logics and the question is if there is a relationship between both frameworks. Such a relationship would further broaden the application of our justification framework, for example to autoepistemic logic and default logic.

References

1. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. TPLP (3) (2007) 301–353

[3] This should not be confused with the nested logic programs of [23], where nesting refers to the expressions inside a logic program rule, and not sets of rules being nested altogether.

2. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. TPLP (3) (2007) 355–375
3. Denecker, M., De Schreye, D.: Justification semantics: A unifying framework for the semantics of logic programs. In: LPNMR, MIT Press (1993) 365–379
4. Denecker, M., Marek, V., Truszczyński, M.: Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In: Logic-Based Artificial Intelligence. Springer US (2000) 127–144
5. Denecker, M., Marek, V., Truszczyński, M.: Ultimate approximation and its application in nonmonotonic knowledge representation systems. Information and Computation (1) (July 2004) 84–121
6. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science (1) (1983) pp.333–354
7. Park, D.: Fixpoint induction and proofs of program properties. Machine Intelligence (1969) 59–78
8. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence (2012) 52–89
9. Mariën, M., Wittocx, J., Denecker, M., Bruynooghe, M.: SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In: SAT, Springer (2008) 211–224
10. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing (1991) 365–385
11. Dung, P.M.: On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artificial Intelligence (1995) 321–358
12. Gupta, G., Bansal, A., Min, R., Simon, L., Mallya, A.: Coinductive logic programming and its applications. In: ICLP, Springer (2007) 27–44
13. Hou, P., De Cat, B., Denecker, M.: FO(FD): Extending classical logic with rule-based fixpoint definitions. Theory and Practice of Logic Programming (4-6) (2010) 581–596
14. van Emden, M.H., Kowalski, R.A.: The Semantics of Predicate Logic as a Programming Language. Journal of the ACM (4) (1976) 733–742
15. Fitting, M.: Fixpoint Semantics for Logic Programming: A Survey. Theoretical Computer Science (1–2) (2002) 25–51
16. Strass, H.: Approximating operators and semantics for abstract dialectical frameworks. Artificial Intelligence (December 2013) 39–70
17. Fages, F.: A New Fixpoint Semantis for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics. In: ICLP, MIT Press (1990) 443
18. Schulz, C., Toni, F.: ABA-based answer set justification. Theory and Practice of Logic Programming (4-5-Online-Supplement) (2013)
19. Bondarenko, A., Dung, P.M., Kowalski, R.A., Toni, F.: An abstract, argumentation-theoretic approach to default reasoning. Artificial Intelligence (1997) 63–101
20. Cabalar, P., Fandinno, J., Fink, M.: Causal graph justifications of logic programs. Theory and Practice of Logic Programming (4-5) (2014) 603–618
21. Pontelli, E., Son, T.C., Elkhatib, O.: Justifications for logic programs under answer set semantics. Theory and Practice of Logic Programming (01) (2009) 1–56
22. Damásio, C.V., Analyti, A., Antoniou, G.: Justifications for logic programming. In: LPNMR. Springer (2013) 530–542
23. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence (3–4) (1999) 369–389
24. Hallnäs, L.: Partial inductive definitions. Theor. Comp. Sci. (1) (1991) 115–142
25. Brewka, G., Woltran, S.: Abstract Dialectical Frameworks. In: KR. (2010) 102–111
26. Bogaerts, B., Vennekens, J., Denecker, M., Van den Bussche, J.: FO(C): A knowledge representation language of causality. TPLP (4-5-Online-Supplement) (2014) 60–69