Exploiting Game Theory for Analysing Justifications

Marynissen, Simon Erik E; Bogaerts, Bart; Denecker, Marc

Published in:
Theory & Practice of Logic Programming

DOI:
10.1017/S1471068420000186

Publication date:
2020

Document Version:
Accepted author manuscript

Exploiting Game Theory for Analysing Justifications

SIMON MARYNISSEN[1,2], BART BOGAERTS[2], MARC DENECKER[1]
[1]*KU Leuven* [2]*Vrije Universiteit Brussel*

submitted 5 June 2020; revised 23 July 2020; accepted 30 July 2020

Abstract

Justification theory is a unifying semantic framework. While it has its roots in non-monotonic logics, it can be applied to various areas in computer science, especially in explainable reasoning; its most central concept is a justification: an explanation why a property holds (or does not hold) in a model.

In this paper, we continue the study of justification theory by means of three major contributions. The first is studying the relation between justification theory and game theory. We show that justification frameworks can be seen as a special type of games. The established connection provides the theoretical foundations for our next two contributions. The second contribution is studying under which condition two different dialects of justification theory (graphs as explanations vs trees as explanations) coincide. The third contribution is establishing a precise criterion of when a semantics induced by justification theory yields consistent results. In the past proving that such semantics were consistent took cumbersome and elaborate proofs.

We show that these criteria are indeed satisfied for all common semantics of logic programming.

KEYWORDS: justification theory, positional games, logic programming, non-monotonic logic

1 Introduction

Justification theory is an abstract theory to define semantics of non-monotonic logics. It is a powerful and versatile framework: First, it provides a mechanism to define new logics based on well-known principles, and to transfer results between domains. Second, it brings order in the zoo of logics and semantics, on the one hand by allowing a systematic comparison between multiple semantics for a single logic and on the other hand by enabling a comparison between different logics (which semantics of which logic coincides with semantics of another logic?). Third, it enables modular definitions of semantics. Using so-called nesting of justification frames, efforts needed to introduce new language constructs (e.g., aggregates) are reduced.

Next to these theoretic benefits, justifications are also used in implementation of solvers. In the unfounded set algorithm (Gebser et al. 2009), justifications for atoms are stored (the so-called *source-pointer approach* essentially maintains a justification). Bogaerts and Weinzierl (2018) used justifications to learn new clauses to improve search in lazy grounding algorithms. Additionally, justifications were used to improve parity game solvers (Lapauw et al. 2020).

The key semantic concept in justification theory is a *justification*: an explanation why something holds (or does not hold) in a model. Because of this, defining semantics of a logic through justification theory does not just provide benefits on the level of an analysis of its semantics, but also equips the logic with a mechanism of *explanation*, thereby providing at least a partial answer to constantly increasing need for explainable methods in artificial intelligence (Miller 2019), which is especially important in the light of the EU General Data Protection Regulation, article 22 of which requires that all AI with an impact on human lives needs to be accountable.

Historically, justification theory was first defined in the doctoral thesis of Denecker (1993) as a framework for studying semantics of logic programs. In that work, a justification is a tree where the nodes are labelled with literals. Later, Denecker et al. (2015) developed a more general theory, aiming to also capture other knowledge representation formalisms. One notable difference with the early work was that justifications were no longer formalised as a tree, but as a graph. While it is known that for the major semantics of logic programming, these two concepts coincide, in the sense that they induce the same semantics, in general very little is known about the relation between tree-like and graph-like justifications; one of the goals of this paper is to bridge that gap.

One benefit of justification theory is that it provides a lot of freedom to create new semantics by means of so-called *branch evaluations*. However, as we show in this paper, not all branch evaluations lead to a well-defined semantics. Showing well-definedness can be an intricate job (Marynissen et al. 2018) and until now no techniques have been developed to do this in a systematic way. We show that this issue closely relates to the coincidence of graph-like and tree-like justifications and provide relatively easy to verify criteria that guarantee that a finite justification framework is well-behaved. Our results build on existing work in the context of antagonistic games over graphs (Gimbert and Zielonka 2005): we establish a connection between justification theory and game theory, and exploit it to transfer existing results on antagonistic games over graphs to justification theory. The main contributions of this paper are as follows:

- We present graph-based and tree-based justifications in a uniform way, thus enabling an in-depth study of their relationship.
- We show that justifications induce games in the sense of Gimbert and Zielonka (2005), thereby bridging a gap between non-monotonic reasoning and game theory.
- Inspired by Gimbert and Zielonka (2005), we develop two criteria for branch evaluations, namely monotonicity and selectivity, and we show for branch evaluations satisfying these conditions in *finite* justification systems, graph-based justifications are well-behaved. While this means that our results do not apply, e.g., to infinitary semantics, for practical applications finiteness is not a major limitation.
- Additionally, we show that whenever graph-based justifications are well-behaved, then graph-based and tree-based justifications are equally powerful.
- Finally, we show that all branch evaluations corresponding to major semantics of logic programming are indeed selective and monotone.

As a consequence, with our results, proving that a branch evaluation induces well-behaved graph-based justifications is much easier: all that is needed is to show that it is monotone and selective. This is in shrill contrast with the proofs of well-behavedness of Marynissen et al. (2018) and Denecker (1993), which span at least a couple of pages for each semantics. The proof of Proposition 6.6, on the other hand, is much simpler and shorter.

Additional proofs can be found in the `arXiv` version of this paper (Marynissen et al. 2020).

2 Justification systems

In this section we introduce justification theory for three-valued logics. Marynissen et al. (2018) used four-valued logics but the benefit of that still needs to be researched. The formalisation we present here is the first in which graph-like and tree-like justifications are unified in a single theory, thereby facilitating a study of the relationship between them. In the rest of this paper, let $\mathcal{F}$ be a set, referred to as a *fact space*, such that $\mathcal{L} = \{\mathbf{t}, \mathbf{f}, \mathbf{u}\} \subseteq \mathcal{F}$, where $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{u}$ have

the respective meaning *true*, *false*, and *unknown*. The elements of $\mathcal{F}$ are called *facts*. The set $\mathcal{L}$ behaves as the three-valued logic with truth order $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$. We assume that $\mathcal{F}$ is equipped with an involution $\sim : \mathcal{F} \to \mathcal{F}$ (i.e. a bijection that is its own inverse) such that $\sim\mathbf{t} = \mathbf{f}$, $\sim\mathbf{u} = \mathbf{u}$, and $\sim x \neq x$ for all $x \neq \mathbf{u}$. For any fact x, $\sim x$ is called the *complement* of x. An example of a fact space is the set of literals over a propositional vocabulary Σ extended with $\mathcal{L}$ where $\sim$ maps a literal to its negation. For any set A we define $\sim A$ to be the set of elements of the form $\sim a$ for $a \in A$. We distinguish two types of facts: *defined* and *open* facts. The former are accompanied by a set of rules that determine their truth value. The truth value of the latter is not governed by the rule system but comes from an external source or is fixed (as is the case for logical facts).

Definition 2.1
A *justification frame* $\mathcal{JF}$ is a tuple $\langle \mathcal{F}, \mathcal{F}_d, R \rangle$ such that

- $\mathcal{F}_d$ is a subset of $\mathcal{F}$ closed under $\sim$, i.e. $\sim\mathcal{F}_d = \mathcal{F}_d$; facts in $\mathcal{F}_d$ are called *defined*;
- no logical fact is defined: $\mathcal{L} \cap \mathcal{F}_d = \emptyset$;
- $R \subseteq \mathcal{F}_d \times 2^{\mathcal{F}}$;
- for each $x \in \mathcal{F}_d$, $(x, \emptyset) \notin R$ and there is an element $(x, A) \in R$ for $\emptyset \neq A \subseteq \mathcal{F}$.

The set of *open* facts is denoted as $\mathcal{F}_o := \mathcal{F} \setminus \mathcal{F}_d$. An element $(x, A) \in R$ is called a *rule* with *head* x and *body* (or *case*) A. The set of cases of x in $\mathcal{JF}$ is denoted as $\mathcal{JF}(x)$. Rules $(x, A) \in R$ are denoted as $x \leftarrow A$ and if $A = \{y_1, \ldots, y_n\}$, we often write $x \leftarrow y_1, \ldots, y_n$.

Logic programming rules can easily be transfered to rules in a justification frame. However, in logic programming, only rules for positive facts are given; never for negative facts. Hence, in order to apply justification theory to logic programming, a mechanism for deriving rules for negative literals is needed as well. For this, a technique called *complementation* was invented; it is a generic mechanism that allows turning a set of rules for x into a set of rules for $\sim x$.

Before defining complementation, we first define *selection functions* for x. A selection function for x is a mapping $s \colon \mathcal{JF}(x) \to \mathcal{F}$ such that $s(A) \in A$ for all rules of the form $x \leftarrow A$. Intuitively, a selection function chooses an element from the body of each rule of x. For a selection function s, the set $\{s(A) \mid A \in \mathcal{JF}(x)\}$ is denoted by $\mathrm{Im}(s)$.

Definition 2.2
Given a set of rules R. We define R^* to be the set of elements of the form $\sim x \leftarrow \sim Im(s)$ for $x \in \mathcal{F}_d$ that has rules in R and s a selection function for x. The *complementation* of $\mathcal{JF}$ is defined as $\langle \mathcal{F}, \mathcal{F}_d, R \cup R^* \rangle$. A justification frame $\mathcal{JF}$ is *complementary* if it is fixed under complementation.

In case of logic programming where only rules for positive facts are given, we can construct a complementary justification frame that is "equivalent" by first taking the complementation and then adding rules of the form $x \leftarrow A'$ if there is a rule $x \leftarrow A \in R$ with $A \subseteq A' \subseteq \mathcal{F}$. If a logic program is finite, then so is this complementary justification frame. Remark that complementarity is a natural property that is satisfied in all practical applications of justification theory. This is because in all practical applications you start with rules for only of the polarities, and apply complementation to this set of rules.

Definition 2.3
A *directed labelled graph* is a quadruple (N, L, E, ℓ) where N is a set of nodes, L is a set of labels, $E \subseteq N \times N$ is the set of edges, and $\ell \colon N \to L$ is a function called the labelling. An *internal* node is a node without outgoing edges.

Definition 2.4
Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame. A *graph-like justification* J in $\mathcal{JF}$ is a directed labelled graph $(N, \mathcal{F}_d, E, \ell)$ such that

- for every internal node $n \in N$ it holds that $\ell(n) \leftarrow \{\ell(m) \mid (n,m) \in E\} \in R$;
- no two nodes have the same label, i.e., ℓ is injective.

A *tree-like justification* J in $\mathcal{JF}$ is a directed labelled graph $(N, \mathcal{F}_d, E, \ell)$ such that

- the underlying undirected graph is a forest, i.e., is acyclic;
- for every internal node $n \in N$ it holds that $\ell(n) \leftarrow \{\ell(m) \mid (n,m) \in E\} \in R$.

We write $\mathfrak{J}_g(x)$ (resp. $\mathfrak{J}_t(x)$) to denote the set of graph-like (resp. tree-like) justifications that have a node labelled x. Whenever we say "justification" it can be either a graph-like or a tree-like.

Definition 2.5
A justification is *locally complete* if it has no leaves with label in $\mathcal{F}_d$, and it is *connected* if the underlying undirected graph is connected. We call $x \in \mathcal{F}_d$ a *root* of a justification J if there is a node n labelled x such that every node is reachable from n in J.

Definition 2.6
Let $\mathcal{JF}$ be a justification frame. A *$\mathcal{JF}$-branch* is either an infinite sequence in $\mathcal{F}_d$ or a finite non-empty sequence in $\mathcal{F}_d$ followed by an element in $\mathcal{F}_o$. For a justification J in $\mathcal{JF}$, a *J-branch* starting in $x \in \mathcal{F}_d$ is a path in J starting in x that is either infinite or ends in a leaf of J. We write $B_J(x)$ to denote the set of J-branches starting in x.

Not all J-branches are $\mathcal{JF}$-branches since they can end in nodes with a defined fact as label. However, if J is locally complete, any J-branch is also a $\mathcal{JF}$-branch.

We denote a branch $\mathbf{b}$ as $\mathbf{b} : x_0 \to x_1 \to \cdots$ and define $\sim\mathbf{b}$ as $\sim x_0 \to \sim x_1 \to \cdots$.

Definition 2.7
A *branch evaluation* $\mathcal{B}$ is a mapping that maps any $\mathcal{JF}$-branch to an element in $\mathcal{F}$ for all justification frames $\mathcal{JF}$. A branch evaluations $\mathcal{B}$ is *consistent* if $\mathcal{B}(\sim\mathbf{b}) = \sim\mathcal{B}(\mathbf{b})$ for any branch $\mathbf{b}$. A justification frame $\mathcal{JF}$ together with a branch evaluation $\mathcal{B}$ form a *justification system* $\mathcal{JS}$, which is presented as a quadruple $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$.

We now define some branch evaluations that induce semantics (as shown later) corresponding to the equally named semantics of logic programs.

Definition 2.8
The *supported* (completion) branch evaluation $\mathcal{B}_{\text{sp}}$ maps $x_0 \to x_1 \to \cdots$ to x_1. The *Kripke-Kleene* branch evaluation $\mathcal{B}_{\text{KK}}$ maps finite branches to their last element and infinite branches to $\mathbf{u}$.

This definition is similar to the one of Denecker et al. (2015), except that they define $\mathcal{B}_{\text{sp}}$ to map finite branches to their last element. Our definition is closer to the intuitions behind completion semantics, which compares the value of rule bodies with the value of its head. However, the induced semantics are the same and it is not difficult to see that all good properties of $\mathcal{B}_{\text{sp}}$ we prove (monotonicity and selectivity) also hold for the definition of Denecker et al. (2015).

Definition 2.9
A *(three-valued) interpretation* of $\mathcal{F}$ is a function $\mathcal{I} : \mathcal{F} \to \mathcal{L}$ such that $\mathcal{I}(\sim x) = \sim\mathcal{I}(x)$ and $\mathcal{I}(\ell) = \ell$ for all $\ell \in \mathcal{L}$. The set of interpretations of $\mathcal{F}$ is denoted by $\mathfrak{I}_{\mathcal{F}}$.

Definition 2.10
Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system, $\mathcal{I}$ an interpretation of $\mathcal{F}$, and J a locally complete graph-like (respectively tree-like) justification in $\mathcal{JS}$. Let $x \in \mathcal{F}_d$ be a label of a node in J. The *value* of $x \in \mathcal{F}_d$ by J under I is defined as $\text{val}(J,x,\mathcal{I}) = \bigwedge_{\mathbf{b} \in B_J(x)} \mathcal{I}(\mathcal{B}(\mathbf{b}))$, where $\bigwedge$ is the greatest lower bound with respect to $\leq_t$.

The *graph-like supported value* of $x \in \mathcal{F}$ in $\mathcal{JS}$ under $\mathcal{I}$ is defined as

$$\text{SV}_g(x,\mathcal{I}) = \bigvee_{J \in \mathfrak{J}_g(x)} \text{val}(J,x,\mathcal{I}) \text{ for } x \in \mathcal{F}_d \qquad \text{SV}_g(x,\mathcal{I}) = \mathcal{I}(x) \text{ for } x \in \mathcal{F}_o.$$

Likewise, the *tree-like supported value* of $x \in \mathcal{F}$ in $\mathcal{JS}$ under $\mathcal{I}$ is defined as

$$\text{SV}_t(x,\mathcal{I}) = \bigvee_{J \in \mathfrak{J}_t(x)} \text{val}(J,x,\mathcal{I}) \text{ for } x \in \mathcal{F}_d \qquad \text{SV}_g(x,\mathcal{I}) = \mathcal{I}(x) \text{ for } x \in \mathcal{F}_o.$$

The following proposition shows that for determining the supported value of x it suffices to look at justifications that are connected and that have x as a root. This is also the reason why tree-like justifications are not called forest-like justifications.

Proposition 2.11
For any locally complete graph-like (respectively tree-like) justification J and $x \in \mathcal{F}_d$ a label of an internal node, there is a connected and locally complete graph-like (respectively tree-like) justification J' such that x is a root of J' and $\text{val}(J,x,\mathcal{I}) \leq_t \text{val}(J',x,\mathcal{I})$ for all interpretations $\mathcal{I}$.

In many logical frameworks, there is an asymmetry between positive and negative literals. Consider for instance stable semantics of logic programs (Gelfond and Lifschitz 1988). There, the default value for atoms is false; as such, the default value for its negation is true. Thus, the set of defined literals is divided into two parts, those that are default and those that are not (sometimes called *deviant*).This idea is generalised to justification theory in signed justification frames.

Definition 2.12
Let $\mathcal{JF}$ be a justification frame. A *sign function* on $\mathcal{JF}$ is a map $\text{sgn} : \mathcal{F}_d \to \{-,+\}$ such that $\text{sgn}(x) \neq \text{sgn}(\sim x)$ for all $x \in \mathcal{F}_d$. We denote $\mathcal{F}_- := \text{sgn}^{-1}(\{-\})$ and $\mathcal{F}_+ := \text{sgn}^{-1}(\{+\})$.

From now on, we fix a sign function on $\mathcal{JF}$. We say that an infinite branch has a positive (respectively negative) tail if from some point onwards all elements are in $\mathcal{F}_+$ (respectively $\mathcal{F}_-$).

Definition 2.13
The *well-founded* branch evaluation $\mathcal{B}_{\text{wf}}$ maps finite branches to their last element. It maps infinite branches to **t** if they have a negative tail, to **f** if they have a positive tail and to **u** otherwise.

The *stable* (answer set) branch evaluation $\mathcal{B}_{\text{st}}$ maps a branch $x_0 \to x_1 \to \cdots$ to the first element that has a different sign than x_0 if it exists; otherwise **b** is mapped to $\mathcal{B}_{\text{wf}}(\mathbf{b})$.

In the next section we discuss how justification systems induce semantics. For the case of logic programs, the semantics induced by the well-founded branch evaluation coincides with the well-founded semantics (Van Gelder et al. 1991) and the one induced by the stable branch evaluation coincides with the stable semantics (Gelfond and Lifschitz 1988).

3 Research questions

A semantics using justification theory is defined as follows: an interpretation $\mathcal{I}$ is a "model" according a justification system if the supported value of each fact is equal to its value in $\mathcal{I}$, i.e.,

if $\mathrm{SV}_g(x,\mathcal{I}) = \mathcal{I}(x)$ for each $x \in \mathcal{F}_d$. Given such a model, a justification with the best (largest with respect to $\leq_t$) value of x constitutes an explanation why x has its particular value.

In many logics, including logic programming, there is an asymmetry between the defined facts by focusing either only on the positive or only on the negative facts. At the level of the justification frame, this asymmetry is resolved by *complementation*. At the semantic level, such logics also only focus on the value assigned to facts of certain polarity (most often, positive facts, i.e., atoms). This induces the following question: "a justification for x with value **t** can be seen as an explanation of why x is true; but which semantic structure can explain why x is false?" From the definition of supported value, it can be seen that x is false if *there are no justifications* with a better value for x. The question that then remains is: how to show that there are no better justifications for x. The most obvious solution is considering a justification of $\sim x$. Indeed: intuitively, an explanation why the negation of x is true should explain why x is false. However, this method implicitly assumes that $\mathrm{SV}(\sim x,\mathcal{I}) = \sim \mathrm{SV}(x,\mathcal{I})$, a property which was called the *consistency property* by Marynissen et al. (2018). Consistency is a reasonable assumption that, unfortunately, is not always satisfied, not even for complementary justification systems.

For the most common branch evaluations in logic programming (completion, Kripke-Kleene, well-founded, and stable), consistency has been shown in the context of graph-like justifications by Marynissen et al. (2018) and in the context of tree-like justifications by Denecker (1993). However, these proofs are often very intricate, consist of many pages and there are currently no insights about *under which conditions on a branch evaluation* consistency is guaranteed.

Research question 1: under which conditions on branch evaluations is the consistency property satisfied (i.e., is $\mathrm{SV}(\sim x,\mathcal{I}) = \sim \mathrm{SV}(x,\mathcal{I})$ for each x and $\mathcal{I}$)?

Next to guaranteeing the ability to explain falsity of atoms similar to truth, consistency brings another advantage: in case of consistency of graph-like (respectively tree-like) justifications, a justification system induces an operator (the support operator)

$$\mathcal{S}^g_{\mathcal{JS}} : \mathfrak{I}_{\mathcal{F}} \to \mathfrak{I}_{\mathcal{F}} : \mathcal{I} \mapsto \mathrm{SV}_g(\cdot,\mathcal{I}), \text{ respectively}$$
$$\mathcal{S}^t_{\mathcal{JS}} : \mathfrak{I}_{\mathcal{F}} \to \mathfrak{I}_{\mathcal{F}} : \mathcal{I} \mapsto \mathrm{SV}_t(\cdot,\mathcal{I}).$$

In case of non-consistency, the function $\mathrm{SV}_g(\cdot,\mathcal{I})$ does not necessarily define an interpretation. Having such an operator defined allows bridging a gap towards operator-based studies of semantics, such as done, e.g., in approximation fixpoint theory (Denecker et al. 2000).

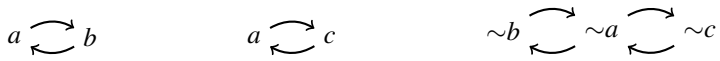
We now define a branch evaluation that illustrates that consistency is not always guaranteed.

Definition 3.1
For a signed justification frame $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$, we define the branch evaluation $\mathcal{B}_{\mathrm{ex}}$ as follows:

- $\mathcal{B}_{\mathrm{ex}}(x_0 \to x_1 \to \cdots \to x_n) = x_n$;
- $\mathcal{B}_{\mathrm{ex}}(x_0 \to x_1 \to \cdots) = \mathbf{f}$ if $\exists i_0 \in \mathbb{N} : \forall i,j > i_0 : x_i \in \mathcal{F}_+$ and if $x_i = x_j$, then $x_{i+1} = x_{j+1}$;
- $\mathcal{B}_{\mathrm{ex}}(x_0 \to x_1 \to \cdots) = \mathbf{t}$ if $\exists i_0 \in \mathbb{N} : \forall i,j > i_0 : x_i \in \mathcal{F}_-$ and if $x_i = x_j$, then $x_{i+1} = x_{j+1}$;
- $\mathcal{B}_{\mathrm{ex}}(x_0 \to x_1 \to \cdots) = \mathbf{u}$ otherwise.

Example 3.2
Take $\mathcal{F}_d = \{a, \sim a, b, \sim b, c, \sim c\}$ and $\mathcal{F}_o = \mathcal{L}$, $\mathcal{F}_+ = \{a,b,c\}$, and $\mathcal{F}_- = \{\sim a, \sim b, \sim c\}$. The set of rules is the complementation of $\{a \leftarrow b; a \leftarrow c; b \leftarrow a; c \leftarrow a\}$. Under the branch evaluation $\mathcal{B}_{\mathrm{ex}}$, we have that $\mathrm{SV}_g(a,\mathcal{I}) = \mathbf{f}$, while $\mathrm{SV}_g(\sim a,\mathcal{I}) = \mathbf{u}$ for any interpretation $\mathcal{I}$ of $\mathcal{F}$, i.e., the

consistency property is not satisfied here. This can easily be checked by noting that the only connected graph-like justifications with a or $\sim a$ as root are the following



Additionally, it can also be seen that $\mathrm{SV}_t(a,\mathcal{I}) = \mathbf{u}$ and $\mathrm{SV}_t(\sim a,\mathcal{I}) = \mathbf{u}$ for each interpretation $\mathcal{I}$, and thus that the graph-like supported value can differ from the tree-like supported value. A tree-like justification J with $\mathrm{val}(J,a,\mathcal{I}) = \mathbf{u}$ is given below:

$$a \longrightarrow b \longrightarrow a \longrightarrow c \longrightarrow a \longrightarrow b \longrightarrow a \longrightarrow c \longrightarrow \cdots$$

A tree-like justification J with $\mathrm{val}(J,\sim a,\mathcal{I}) = \mathbf{u}$ is given by the tree unfolding of the graph-like justification for $\sim a$ with value $\mathbf{u}$.

This example immediately brings us to the second research question, namely the relation between graph-like and tree-like justifications.

Research question 2: under which conditions on branch evaluations do the graph-like and the tree-like supported value coincide (i.e., is $\mathrm{SV}_t(x,\mathcal{I}) = \mathrm{SV}_g(x,\mathcal{I})$ for each x and $\mathcal{I}$)?

Resolving this question is important from a practical perspective, e.g., for justification-based algorithms. Indeed, tree-like justifications tend to be infinitely large when recursion is present. Graph-like justifications on the other hand, can easily be maintained. It is easy to see that tree-like justifications are at least as powerful as graph-like justifications.

Proposition 3.3
For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, it holds that $\mathrm{SV}_g(x,\mathcal{I}) \leq_t \mathrm{SV}_t(x,\mathcal{I})$.

The fact that the branch evaluation from Definition 3.1 serves as a counterexample for both research questions is not a coincidence: these questions are closely related. We will show (Theorem 5.14) that under very unrestrictive conditions, if graph-like justifications are consistent, then tree-like justifications give the same result as graph-like justifications, and thus are also consistent.

As a technical means to answer these questions, we use game theory: we will show that justification systems induce a game in the sense of (Gimbert and Zielonka 2005) and that coincidence of graph-like and tree-like justifications is inherently tied to the existence of optimal positional strategies in the induced game. Before doing so, we introduce some preliminaries on games.

4 Games

We now define antagonistic games over graphs (Gimbert and Zielonka 2005). Our formalisation slightly differs from the one of Gimbert and Zielonka, but the differences are minor. Intuitively a game unfolds as follows: There are two players **T** and **F**, with opposite interests. Let G be a graph in which each vertex is owned either by **T**, or by **F**. Initially, a stone is placed on some vertex in G. At each step, the player owning the vertex with the stone moves the stone along an outgoing edge. The players interact in this way ad infinitum or until the stone is on a vertex without outgoing edges.

Definition 4.1
A *game graph* is a quadruple $G = (S, S_{\mathbf{T}}, S_{\mathbf{F}}, E)$ where $S_{\mathbf{T}}$ and $S_{\mathbf{F}}$ form a partition of the set of states S, and $E \subseteq S \times S$. For a transition $e = (s,t) \in E$, the states s and t are respectively called the

source and target of e (denoted source(e) and target(e)). For a state $s \in S$, sE is the set of outgoing edges from s. A *path* in G is a non-empty finite or infinite sequence of states $p = s_0 s_1 s_2 \dots$ such that for all $i \geq 0$, $(s_i, s_{i+1}) \in E$. We define source(p) to be equal to s_0. If p is finite, then target(p) is the last state of p. If a path consists of a single state s, we call it *empty*. We denote the empty path in s by λ_s. The set of finite paths in G (including the empty paths) is denoted by Path_G.

Definition 4.2
Let G be a game graph. A *play* in G is either an infinite path in G or a finite path in G that ends in a state without outgoing edges. The set of plays in G is denoted by Play_G.

In its most general form, games of this form do not have a winner, but instead, each player has a preference relation indicating which plays they prefer over others.

Definition 4.3
Let G be a game graph. A *preference relation* for a player P is a total preorder (i.e., a reflexive and transitive relation such that for all plays p and q, $p \sqsubseteq_P q$ or $q \sqsubseteq_P p$ holds) over Play_G.
A *game* is a tuple $\mathcal{G} = (G, \sqsubseteq_{\mathbf{T}})$, where G is a game graph and $\sqsubseteq_{\mathbf{T}}$ is a preference relation for $\mathbf{T}$. We define $\sqsubseteq_{\mathbf{F}}$ to be the inverse relation of $\sqsubseteq_{\mathbf{T}}$.

Intuitively, if $p \sqsubseteq_P q$, then the player P prefers the play q at least as much as p. If both $p \sqsubseteq_P q$ and $q \sqsubseteq_P p$ hold, then we say that p and q are equivalent with respect to $\sqsubseteq_P$.

When playing, players usually follow some strategy. In this paper we are interested in two types of strategies: *general* and *positional* strategies. Intuitively, the former can take into account the entire play history to determine a move, while the latter only depends on the current state.

Definition 4.4
Let $G = (S, S_{\mathbf{T}}, S_{\mathbf{F}}, E)$ be a game graph. A *general strategy* for a player P in G is a function $\sigma_P : \{p \in \text{Path}_G \mid \text{target}(p) \in S_P\} \to E$ such that $\sigma_P(p) \in \text{target}(p)E$. The set of general strategies for player P is denoted by $\mathfrak{S}_g(P)$. A *positional (or memoryless) strategy* for a player P in G is a mapping $\sigma_P : S_P \to E$ such that $\sigma_P(s) \in sE$ for all $s \in S_P$. The set of positional strategies for P is denoted by $\mathfrak{S}_p(P)$.

We can embed $\mathfrak{S}_p(P)$ into $\mathfrak{S}_g(P)$ by mapping a positional strategy σ to a general strategy that maps $s_0 \dots s_n$ to $\sigma(s_n)$. Slightly abusing notation, we thus have that $\mathfrak{S}_p(P) \subseteq \mathfrak{S}_g(P)$.

Definition 4.5
Let $G = (S, S_{\mathbf{T}}, S_{\mathbf{F}}, E)$ be a game graph. A finite or infinite path $p = s_0 s_1 s_2 \dots$ in G is *consistent* with a general strategy σ_P for a player P if $\sigma_P(s_0 \dots s_i) = (s_i, s_{i+1})$ whenever $s_i \in S_P$. It is *consistent* with a positional strategy σ_P for player P if $\sigma_P(s_i) = (s_i, s_{i+1})$ whenever $s_i \in S_P$.

Given a state s and strategies σ and τ for players $\mathbf{T}$ and $\mathbf{F}$, there exists a unique play in G starting in s consistent with both σ and τ. We denote this play by $p_G(s, \sigma, \tau)$.

Definition 4.6
Let σ be a general strategy for a player P. The *play tree* for σ in x is the tree with nodes labelled by states whose maximally long branches correspond to the plays starting in x consistent with σ.
Let σ_p be a positional strategy for a player P. The *play graph* for σ_p in x is the subgraph of the game graph consisting of all edges occurring in plays starting in x consistent with σ_p.

Under the assumption that **T** and **F** are rational agents, trying to maximize the value of the resulting play in their preference relation ($\sqsubseteq_{\mathbf{T}}$ respectively $\sqsubseteq_{\mathbf{F}}$), some strategies will not be followed. So-called *Nash equilibria* are often used as "solutions" to games; intuitively, they are pairs of strategies from which neither player wants to unilaterally deviate. In antagonistic games over graphs this is defined as follows.

Definition 4.7
Let $G = (S, S_{\mathbf{T}}, S_{\mathbf{F}}, E)$ be a game graph and let σ^* and τ^* be general strategies for respectively **T** and **F**. The pair (σ^*, τ^*) is *optimal* if for all states $s \in S$ and all general strategies σ and τ for respectively **T** and **F** it holds that $p_G(s, \sigma, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(s, \sigma^*, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(s, \sigma^*, \tau)$.

Intuitively, in an optimal pair of strategies, no player can benefit by changing only their strategy. In this paper, we will be particularly interested in the question whether an optimal pair of *positional* strategies (i.e., an optimal pair of strategies such that σ^* and τ^* are positional) exists.

5 Embedding justification theory in game theory

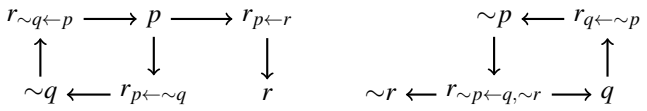
We now show how to derive an antagonistic game from a justification system. We first introduce a construction and then show how strategies in the constructed game correspond to justifications of the original system, which will serve as the basis for proving the consistency property. The following serves as a running example for this section.

Example 5.1
Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ with $\mathcal{F}_d = \{p, q, \sim p, \sim q\}$, $\mathcal{F}_o = \mathcal{L} \cup \{r, \sim r\}$, and R the complementation of $\{p \leftarrow \sim q; q \leftarrow \sim p; p \leftarrow r\}$.

Games associated to justifications Let $\mathcal{JF}$ be a justification frame. For any $x \in \mathcal{F}_d$ and rule $x \leftarrow A \in R$, we introduce new symbols $r_{x \leftarrow A}$, which we call rule symbols. In the game associated with a justification system, **T** owns the defined facts and **F** owns the rule symbols.

Definition 5.2
The game graph $G_{\mathcal{JF}} = (S, S_{\mathbf{T}}, S_{\mathbf{F}}, E)$ corresponding to the justification frame $\mathcal{JF}$ is defined by $S_{\mathbf{T}} = \mathcal{F}$, $S_{\mathbf{F}} = \{r_{x \leftarrow A} \mid x \in \mathcal{F}_d, x \leftarrow A \in R\}$, $S = S_{\mathbf{T}} \cup S_{\mathbf{F}}$, and for any $x \in \mathcal{F}_d$, $x \leftarrow A \in R$, and $y \in A$ we have edges $(x, r_{x \leftarrow A})$ and $(r_{x \leftarrow A}, y)$.

Example 5.3
For the justification frame from Example 5.1, the game graph (without isolated nodes) is visualized below.



In a game graph corresponding to a justification frame, a strategy for **T** chooses a rule for every defined fact and a strategy for **F** chooses for each rule an element from its body. Intuitively, this means that the true player will be the one choosing a justification, while the false player chooses a branch in that justification.

Any play in this graph corresponds to a $\mathcal{JF}$-branch as follows: Let $s_0 s_1 \ldots$ be a play in $G_{\mathcal{JF}}$. By removing the rule symbols, we get a sequence in $\mathcal{F}$, which is a $\mathcal{JF}$-branch. Therefore, any play $p \in \text{Play}_{G_{\mathcal{JF}}}$ corresponds to a $\mathcal{JF}$-branch $\mathbf{b}_p$. These branches are used to define a preference relation for the player **T** to obtain a game.

Definition 5.4
The *justification game* $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ corresponding to a justification system $\mathcal{JS}$ and an interpretation $\mathcal{I}$ of $\mathcal{F}$ is the antagonistic game $(G_{\mathcal{JF}}, \sqsubseteq)$ such that for all $p,q \in \text{Play}_G$, $p \sqsubseteq q$ if and only if $\mathcal{I}(\mathcal{B}(\mathbf{b}_p)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}_q))$.

We now formally show how a justification system relates to the induced justification game on the semantic level. Intuitively: strategies correspond to justifications and play values dictate the supported value. We start by the correspondence of strategies for **T** and justifications.

Strategies for **T** *are justifications* Intuitively, a strategy for **T** chooses a rule for every defined fact, which is exactly what a justification does as well. The following propositions make the relation between strategies for **T** and justifications precise.

Proposition 5.5
Let σ be a positional (respectively general) strategy for **T** in the game graph $G_{\mathcal{JF}}$. Take $x \in \mathcal{F}_d$. The play graph (respectively play tree) of σ in x where all the rule symbols are filtered out[1] is a connected, locally complete graph-like (respectively tree-like) justification with x as root in the justification frame $\mathcal{JF}$. We use the symbol $J_\sigma(x)$ for this justification.

Example 5.6
Let $\mathcal{JF}$ be the justification frame from Example 5.1 and let σ be a positional strategy for **T** that maps p to $r_{p\leftarrow\sim q}$. The justifications $J_\sigma(p)$ and $J_\sigma(\sim p)$ are depicted below from left to right.

Proposition 5.7
Any connected, locally complete graph-like (respectively tree-like) justification J with root x is equal to some $J_\sigma(x)$ for a positional (respectively general) strategy σ for **T**.

The correspondence of justifications and strategies for **T** raises the question what the role of **F** is. Intuitively, **T** chooses for each node in the justification a set of children corresponding to a rule and **F** then chooses for each such node a single child. This means that **F** actually chooses a branch in the justification determined by a strategy of **T**. An important remark is that branches in graph-like justifications need not be positional (they can contain both $y \to u$ and $y \to v$ for $u \neq v$; in this case, u and v are elements of the same case). Taking these considerations into account, we can rephrase the supported value of a node in terms of strategies, where **T** has to play positionally to obtain graph-like justifications but **F** is always allowed to use general strategies. To simplify notation, we introduce a utility function defined as follows: $u(x,\sigma,\tau) \coloneqq \mathcal{I}(\mathcal{B}(\mathbf{b}_{p_G(x,\sigma,\tau)}))$.

Theorem 5.8
For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, the following holds

$$\text{SV}_t(x,\mathcal{I}) = \bigvee_{\sigma\in\mathfrak{S}_g(\mathbf{T})} \bigwedge_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau) \qquad \text{and} \qquad \text{SV}_g(x,\mathcal{I}) = \bigvee_{\sigma\in\mathfrak{S}_p(\mathbf{T})} \bigwedge_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau)$$

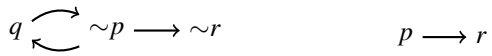
[1] By construction, there are no edges between rule symbols. Therefore, filtering, is removing all edges to and from rule symbols and adding edges (y,z) if in the original graph there are edges (y,r) and (r,z) for some rule symbol r.

Strategies for **F** *are also justifications* We have seen how strategies for **T** correspond to justifications and that strategies for **F** correspond to branches in those justifications. We used this information to calculate the supported value using play values. However, in this section we show that also strategies for **F** correspond to justifications, but for negation of facts. This is a crucial step towards proving the consistency property, because we can then relate the supported values of x and $\sim x$. In order to do so, our justification frames needs to be complementary, which, as mentioned above, is a condition satisfied in all practical applications of justification theory.

Let τ be a strategy for **F**. This means that τ chooses for every rule symbol $y \leftarrow A$ an element from A. This corresponds to a selection function s for y. Using complementarity, this selection function provides a rule for $\sim y$: $\sim y \leftarrow \sim \mathrm{Im}(s)$. Therefore, the strategy τ indirectly chooses a rule for every $\sim y$, which thus again induces a justification.

Proposition 5.9
Let $\mathcal{JF}$ be a complementary justification frame and τ a positional (respectively general) strategy for **F** in the game graph $G_{\mathcal{JF}}$ and take $x \in \mathcal{F}_d$. The play graph (respectively play tree) of τ in x where all the rule symbols are filtered out and all node's labels are inverted (y replaced by $\sim y$, remark $\sim(\sim y) = y$), is a connected, locally complete graph-like (respectively tree-like) justification with $\sim x$ as root in the justification frame $\mathcal{JF}$. We write $J_\tau(x)$ for this justification.

Example 5.10
Let $\mathcal{JF}$ be the justification frame from Example 5.1. Let τ be the positional strategy for **F** that maps $r_{\sim p \leftarrow q, \sim r}$ to $\sim r$. The justifications $J_\tau(p)$ and $J_\tau(\sim p)$ are given below from left to right.



Proposition 5.11
Let $\mathcal{JF}$ be a complementary justification frame. Any connected, locally complete graph-like (respectively tree-like) justification with $\sim x$ as root is equal to $J_\tau(x)$ for some positional (respectively general) strategy τ for **F**.

Completely analogously to Theorem 5.8, we thus obtain a method to compute the supported value of the negation of a fact in terms of strategies.

Theorem 5.12
Let $\mathcal{JF}$ be a complementary justification frame and $\mathcal{B}$ a consistent branch evaluation. For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, the following holds

$$\mathrm{SV}_t(\sim x, \mathcal{I}) = \sim \bigwedge_{\tau \in \mathfrak{S}_g(\mathbf{F})} \bigvee_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau) \quad \text{and} \quad \mathrm{SV}_g(\sim x, \mathcal{I}) = \sim \bigwedge_{\tau \in \mathfrak{S}_p(\mathbf{F})} \bigvee_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$$

Consequences for consistency of justification systems Theorem 5.8 and Theorem 5.12 relate supported values of x and $\sim x$ to strategies. This provides a partial answer to both research questions as shown in the following two results.

Proposition 5.13
Let $\mathcal{JF}$ be a complementary justification frame. If $\mathcal{B}$ is consistent, then for any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$ the following holds:

$$\mathrm{SV}_t(x, \mathcal{I}) \leq_t \sim \mathrm{SV}_t(\sim x, \mathcal{I}) \qquad \text{and} \qquad \mathrm{SV}_g(x, \mathcal{I}) \leq_t \sim \mathrm{SV}_g(\sim x, \mathcal{I})$$

This is one inequality of the consistency. In the next section, the other direction is researched.

Theorem 5.14
Let $\mathcal{JF}$ be a complementary justification frame. If $\mathcal{B}$ is consistent and graph-like justifications are consistent, then tree-like and graph-like justifications are equally powerful.

Proof
By using Propositions 3.3 and 5.13 we get the following chain of inequalities:

$$\mathrm{SV}_g(x,\mathcal{I}) \leq_t \mathrm{SV}_t(x,\mathcal{I}) \leq_t \sim\mathrm{SV}_t(\sim x,\mathcal{I}) \leq_t \sim\mathrm{SV}_g(\sim x,\mathcal{I}).$$

By assumption, this chain collapses to only equalities; hence tree-like and graph-like justifications are equally powerful. □

Marynissen et al. (2018) showed that graph-like justifications are consistent for completion, Kripke-Kleene, stable, and well-founded semantics. Therefore, we automatically have that graph-like and tree-like justification are equally powerful for those branch evaluations.

6 Consistency of justification systems

In the previous section we related the supported value under graph-like and tree-like to play values. In this section we first show that whenever an optimal pair of strategies exists, the supported value equals the optimal play value. Next we provide sufficient conditions for the existence of optimal pairs of positional strategies.

Relating supported values to optimal strategies We first focus our attention on the first claim: the relation between supported values of facts (and of their negation) and the value of optimal strategies is formalised in the following two propositions.

Proposition 6.1
Let $x \in \mathcal{F}_d$ and $\mathcal{I}$ be an interpretation. If (σ^*,τ^*) is optimal, then $\mathrm{SV}_t(x,\mathcal{I}) = u(x,\sigma^*,\tau^*)$. If (σ^*,τ^*) is also positional, then $\mathrm{SV}_g(x,\mathcal{I}) = u(x,\sigma^*,\tau^*)$.

Proposition 6.2
Let $\mathcal{JF}$ be a complementary justification frame and $\mathcal{B}$ a consistent branch evaluation. Let $x \in \mathcal{F}_d$ and $\mathcal{I}$ an interpretation. If (σ^*,τ^*) is optimal, then $\mathrm{SV}_t(\sim x,\mathcal{I}) = \sim u(x,\sigma^*,\tau^*)$. If (σ^*,τ^*) is also positional, then $\mathrm{SV}_g(\sim x,\mathcal{I}) = \sim u(x,\sigma^*,\tau^*)$.

These two propositions relate the play value of optimal pairs of (positional) strategies to the supported value of both x and its negation. As such the existence of optimal strategies is a sufficient condition for the consistency property to hold, which is formalised in the next theorem.

Theorem 6.3
Let $\mathcal{JF}$ be a complementary justification frame and $\mathcal{B}$ a consistent branch evaluation. If there exists an optimal pair (σ^*,τ^*) of strategies, then for all $x \in \mathcal{F}_d$ and interpretations $\mathcal{I}$ it holds that $\mathrm{SV}_t(x,\mathcal{I}) = \sim\mathrm{SV}_t(\sim x,\mathcal{I}) = u(x,\sigma^*,\tau^*)$. If the optimal pair (σ^*,τ^*) is also positional, then for all $x \in \mathcal{F}_d$ and interpretations $\mathcal{I}$ it holds that $\mathrm{SV}_g(x,\mathcal{I}) = \sim\mathrm{SV}_g(\sim x,\mathcal{I}) = u(x,\sigma^*,\tau^*)$.

Existence of optimal pairs of positional strategies We now turn our attention to the last piece of the puzzle, namely developing conditions under which optimal pairs of positional strategies exist. The results of the previous section already guarantee that whenever such pairs exist, the consistency property holds, and graph-like and tree-like justifications are equally powerful. Therefore, having results for the existence of optimal pairs of positional strategies is paramount.

We want conditions on our branch evaluation that guarantee the existence of positional optimal pairs. Inspired by Gimbert and Zielonka (2004; 2005), we have the following two definitions.

Definition 6.4
$\mathcal{B}$ is monotone if for every two branches $\mathbf{b}_1$ and $\mathbf{b}_2$ starting in the same fact x and all finite paths p such that $p \to x$ is a path the following holds:

$$\mathcal{I}(\mathcal{B}(\mathbf{b}_1)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}_2)) \quad \Rightarrow \quad \mathcal{I}(\mathcal{B}(p \to \mathbf{b}_1)) \leq_t \mathcal{I}(\mathcal{B}(p \to \mathbf{b}_2)).$$

Monotonicity means that whenever one branch is better than another branch, it does not matter how one arrived there: by adding any single path leading to the start of these two branches in front of the two branches, the inequality is respected. Intuitively, this is a form of a locality principle: the branch evaluation can decide the value of the two extended branches ($p \to \mathbf{b}_1$, $p \to \mathbf{b}_2$) based on their joint start (p; in which case they have the same value) or based on their tail ($\mathbf{b}_1$, $\mathbf{b}_2$; in which case the value of the tail indicates how the value of the extended branches relates).

In the following definition, we use some more notation and terminology. A finite loop starting in x is a finite path p starting in x such that $p \to x$ is a path. If M is a set of loops, M^* denotes the set of (finite) paths obtained by a finite iteration of loops in M; M^ω denotes the set of infinite paths obtained by infinite iterations of loops in M.

Definition 6.5
A branch evaluation $\mathcal{B}$ is selective with respect to $\mathcal{I}$ if for all finite paths p and facts $x \in \mathcal{F}_d$ such that $p \to x$ is a path, for all sets M, N of finite loops starting in x, for all sets K of branches starting in x we have that for all branches $\mathbf{b}$ of the form $p(M \cup N)^* K$ or $p(M \cup N)^\omega$ there exist branches $\mathbf{b}_*, \mathbf{b}^* \in pM^\omega \cup pN^\omega \cup pK$ such that $\mathcal{I}(\mathcal{B}(\mathbf{b}_*)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b})) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}^*))$.

Intuitively, selectivity means that choices can be made consistently. This is most easily seen in case M, N and K are singletons in which x only occurs as the start. In that case the branch $\mathbf{b}$ makes (at most) three different choices for x: the one leading to the loop in M, the one leading to the loop in N and the one leading to K. Selectivity then states that the choice for x can be made consistenly, to obtain a branch that is at least as good ($\mathbf{b}^*$) and one that is at least as bad ($\mathbf{b}_*$).

The branch evaluation $\mathcal{B}_{\text{ex}}$ from Definition 3.1 is not selective. In Example 3.2, if we take $M = \{a \to b\}$, $N = \{a \to c\}$ and $K = \emptyset$, then every branch in N^ω and M^ω is mapped to $\mathbf{f}$, while there are branches in $(M \cup N)^\omega$ that are mapped to $\mathbf{u}$, e.g., $(a \to b \to a \to c \to)^\omega$.

On the other hand, the branch evaluations corresponding to the major logic programming semantics satisfy the two criteria proposed above.

Proposition 6.6
The branch evaluations $\mathcal{B}_{\text{sp}}$, $\mathcal{B}_{\text{KK}}$, $\mathcal{B}_{\text{st}}$, and $\mathcal{B}_{\text{wf}}$ are monotone and selective.

If every state owned by $\mathbf{T}$ has at most one outgoing transition, then there is only one strategy σ for $\mathbf{T}$. This strategy is positional and there is a general strategy τ for $\mathbf{F}$ such that (σ, τ) is optimal. So we can assume there is a state $x \in S_{\mathbf{T}}$ with more than one outgoing transition. If we take a partition $\{A_1, A_2\}$ of non-empty sets of xE, then we can form the game graphs G_1 and G_2 with

the states of the original graph and $E \setminus A_2$ respectively $E \setminus A_1$ for the edges. Using the preference relation of the original game, we get two smaller games $\mathcal{G}_1$ and $\mathcal{G}_2$. Assume by induction that $\mathcal{G}_i$ has an optimal pair (σ_i^*, τ_i^*) of positional strategies for each $i \in \{1,2\}$. Without loss of generality we have that $u(x, \sigma_2^*, \tau_2^*) \leq_t u(x, \sigma_1^*, \tau_1^*)$. It turns out that if $\mathcal{B}$ is monotone and selective, then σ_1^* is also an optimal strategy for the original game, meaning that there is a general strategy τ for **F** such that (σ_1^*, τ) is an optimal pair of strategies. Therefore, if the game is finite, we can perform finite induction to obtain that there exists a positional strategy σ^* for **T** and a general strategy τ for **F** such that (σ^*, τ) is optimal. By a similar reasoning, there is a positional strategy τ^* for **F** and a general strategy σ for **T** such that (σ, τ^*) is optimal. This means that (σ^*, τ^*) is also an optimal pair, and both strategies are positional. This paragraph essentially sketched the proof of the following theorem.

Theorem 6.7
If $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ is finite and $\mathcal{B}$ is monotone and selective, then there is a positional optimal pair.

Corollary 6.8
If $\mathcal{JS}$ is finite and $\mathcal{B}$ is monotone and selective, then $\mathrm{SV}_g(\sim x, \mathcal{I}) = \sim \mathrm{SV}_g(x, \mathcal{I})$ for all $x \in \mathcal{F}_d$ and interpretations $\mathcal{I}$.

For the major logic programming semantics, this means that for finite logic programs, the supported value is consistent and that graph-like and tree-like justifications are equally powerful.

7 Conclusion

In this paper, we studied the relation between justification theory and game theory to answer two concrete questions about justification systems: when are graph-like and tree-like justifications equally powerful, and when does the supported value commute with negation? All of our work was developed in the general setting except for one (crucial) result: in order to prove the consistency property for graph-like justification systems in case of monotonocity and selectivity, we assumed, following game theory tradition, finiteness. This means for instance that while our results can be applied directly to finite ground logic programs, they are not directly applicable to non-ground programs with an infinite grounding.

Hence, the most obvious direction for future work is generalizing this theorem to the infinite case, thereby possibly extending the definition of selectivity. For inspiration to achieve such result, we think that game theory can again provide valuable directions: there have been result in a different but similar type of games, the Gale-Stewart games (Soare 2016). In these games, two player alternates picking an element from a fixed set A. A play is a countable infinite sequence of elements in A. A Gale-Stewart game is won by the first player if the infinite sequence is in some fixed payoff set, otherwise the second player wins. A famous result by Martin (1975) states that if the pay-off set of a Gale-Stewart game is Borel, then the game is determined, meaning that one of the two players has a winning strategy. In contrast with our justification games, Gale-Stewart games are only two-valued and every play is infinite. Three-valued antagonistic games over graphs can be split into two two-valued antagonistic games over graphs. If these two-valued games can be mapped to Gale-Stewart games such that winning strategies match, Martin's Borel determinacy theorem could be used to prove the existence of optimal pairs of general strategies.

Several earlier papers have already established connections between logic programming and games to study properties of logic programs (van Emden 1986; Blair 1995; Loddo and Cosmo

2000; Galanaki et al. 2008). By establishing a bridge between game theory and justification theory, we developed a mechanism to transfer game-theoretic results to all application domains of justification theory, e.g., to logic programming, abstract argumentation (Dung 1995), and to nested fixpoint definitions (Denecker et al. 2015). Furthermore, by using justification theory, the translation immediately works for all common semantics of logic programs. Indeed, the branch evaluation, which determines the semantics, is only used for determining the resulting preference relation. While we used our results here to get results on consistency and coincidence of graph-like and tree-like justifications, the full impact of this connection still remains to be explored.

References

BLAIR, H. A. 1995. Game characterizations of logic program properties. In *Proceedings of LPNMR*. 99–112.

BOGAERTS, B. AND WEINZIERL, A. 2018. Exploiting justifications for lazy grounding of answer set programs. In *Proceedings of IJCAI*. 1737–1745.

DENECKER, M. 1993. Knowledge representation and reasoning in incomplete logic programming. Ph.D. thesis, K.U.Leuven, Leuven, Belgium.

DENECKER, M., BREWKA, G., AND STRASS, H. 2015. A formal theory of justifications. In *Proceedings of LPNMR*. 250–264.

DENECKER, M., MAREK, V., AND TRUSZCZYŃSKI, M. 2000. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In *Logic-Based Artificial Intelligence*. 127–144.

DUNG, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *AIJ 77,* 2, 321 – 357.

GALANAKI, C., RONDOGIANNIS, P., AND WADGE, W. W. 2008. An infinite-game semantics for well-founded negation in logic programming. *Ann. Pure Appl. Log. 151,* 2-3, 70–88.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., AND SCHAUB, T. 2009. On the implementation of weight constraint rules in conflict-driven ASP solvers. In *ICLP*. 250–264.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP*. 1070–1080.

GIMBERT, H. AND ZIELONKA, W. 2004. When can you play positionnaly? In *Proceedings of MFCS*. 686–697.

GIMBERT, H. AND ZIELONKA, W. 2005. Games where you can play optimally without any memory. In *Proceedings of CONCUR*. 428–442.

LAPAUW, R., BRUYNOOGHE, M., AND DENECKER, M. 2020. Improving parity game solvers with justifications. In *Proceedings of VMCAI*. 449–470.

LODDO, J. AND COSMO, R. D. 2000. Playing logic programs with the alpha-beta algorithm. In *Proceedings of LPAR*. 207–224.

MARTIN, D. A. 1975. Borel determinacy. *Annals of Mathematics 102,* 2, 363–371.

MARYNISSEN, S., BOGAERTS, B., AND DENECKER, M. 2020. Exploiting Game Theory for Analysing Justifications. *arXiv e-prints*, arXiv:2008.01609.

MARYNISSEN, S., PASSCHYN, N., BOGAERTS, B., AND DENECKER, M. 2018. Consistency in justification theory. In *Proceedings of NMR*. 41–52.

MILLER, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *AIJ 267*, 1–38.

SOARE, R. I. 2016. *Gale-Stewart Games*. Springer Berlin Heidelberg, Berlin, Heidelberg, 217–219.

VAN EMDEN, M. H. 1986. Quantitative deduction and its fixpoint theory. *J. Log. Program. 3,* 1, 37–53.

VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM 38,* 3, 620–650.