KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Advances in Justification Theory

Simon Marynissen

Supervisors:
Prof. dr. M. Denecker (KU Leuven)
Prof. dr. B. Bogaerts (Vrije Universiteit Brussel)

Dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

January 2022

Advances in Justification Theory

Simon MARYNISSEN

Examination committee:
Prof. dr. ir. R. Puers, chair
(KU Leuven)
Prof. dr. M. Denecker, supervisor
(KU Leuven)
Prof. dr. B. Bogaerts, supervisor
(Vrije Universiteit Brussel)
Prof. dr. ir. G. Janssens
(KU Leuven)
Prof. dr. ir. B. Jacobs
(KU Leuven)
Prof. dr. G. A. Wiggins
(Vrije Universiteit Brussel)
Prof. dr. P. Cabalar
(University of Corunna)
Prof. dr. P. Rondogiannis
(National and Kapodistrian University of Athens)

Dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

January 2022

Preface

Today marks the end of a chapter of my life. The past four years and a bit, I have had a wonderful time, filled with tackling challenging research questions, meeting new people, and travelling abroad.[1] Nevertheless, I feel relieved that I can close this chapter and start a new one. During this period, many people contributed to making this thesis possible. Therefore, I would like to take a moment to thank some people.

First of all, I would like to thank my **supervisors**. I started this PhD with just one supervisor, Marc. Over the course of the past four years, he has been a non-stop source of research ideas, too many to work them all out. Therefore, it is not surprising that much of the research in this thesis originated from his ideas. Marc, thank you for turning me into a researcher and for the countless interesting discussions and coffee breaks (chocolate milk in my case)!

A few months after I started my PhD, I got a second supervisor, Bart. In the first year, we worked in the same office[2]. This allowed me to quickly absorb all the new computer science material. When he went to become a professor at the Vrije Universiteit Brussel, my PhD became a joint PhD. While my conversations with Marc were usually about high-level stuff, my discussion with Bart also went into the intricate details of proofs and methods. Without these discussions, most of the results in this thesis would still be left unproven, thank you for that! I would also like to thank you for improving my English language skills by providing numerous feedback on preliminary texts.

Next, my gratitude goes to the members of my **examination committee** for finding the time to read this thesis. Your constructive remarks and suggestions were greatly appreciated.

Over the past four years, I have also had the pleasure to work with fantastic **colleagues**. In no particular order, I would like to thank them as well for

[1] Not as much as I would have wanted. Damn you, coronavirus!

[2] I still remember you messing with my computer. ;)

the wonderful time in the office (and online): Joachim, Jo, Ingmar, Ruben, Matthias, Pierre, Đorđe, and Linde.

I would also like to thank my **friends** for the entertainment during the past four years. I would especially like to thank the members of the **catanclan** for the fun weekends.

At last, I want to thank my **family**, in particular my parents for the unconditional support they provide. It is always a joy to return home.[3] I would also like to thank my sister Marleen and my brothers Thomas, Matthijs, Jonah, and Johannes. It is nice to have a big family.

Last, but foremost, I would like to dedicate this text to one special person. **Joke**, thanks for being there for me! I couldn't have done this without you!

[3] And return with hands full of stuff.

Abstract

To practice knowledge representation, it is paramount that knowledge representation languages have formal semantics. However, since there are numerous different languages all with a formalisation, it is valuable to have unifying frameworks that can capture the semantics of families of languages and logics. One such framework is justification theory, in which the semantics are defined by the use of explanations, called justifications in our terminology. Intuitively, a justification is a graph that explains the truth values of certain facts. However this introduces a potential problem: the justification status of a fact and its negation can be inconsistent. So, for justification semantics to be well-defined, these statuses should be opposite. Such semantics are called consistent.

In the first part of this thesis we prove that the main semantics of justification theory are in fact consistent. Moreover, we prove useful results for justifications, such as the ability to compose together justifications. An other issue with justification semantics is that there are different flavours of justifications, which could result in distinct semantics. We show that these two seemingly unrelated issues are actually deeply connected.

After that, we establish a connection between justification theory and game theory, which allows for justifications to be seen as strategies in a two-player game. This connection provides a resolution of the two issues in justification theory by providing a general condition on the semantics in case the system is finite.

Justification theory is not the only unifying framework for semantics of non-monotonic logics. Another well-known framework is approximation fixpoint theory, which is more algebraic in nature. We establish a connection between justification theory and approximation fixpoint theory. The notion of ultimate semantics of approximation fixpoint theory can be transferred into the realm of justification semantics. This allows for justification semantics to capture more

semantics than previously.

As the final topic of this thesis, we look into the nesting of justifications, which can be used to define modular semantics. In previous definitions of nesting, a significant amount of information is lost because a compression operation is used. We provide an alternative and more general definition without this disadvantage. This allows for more intuitive modular semantics based on justifications. We prove that this is equivalent to the compression for tree-like justifications and in special cases for graph-like justifications. We investigate the consistency of both approaches as well and as an added bonus we solve the consistency for tree-like justifications.

In summary, this thesis gathers a number of advances in justification theory and illustrates them with examples.

Beknopte Samenvatting

Om kennisrepresentatie te beoefenen is het belangrijk dat kennisrepresentatietalen een formele semantiek hebben. Omdat er echter talloze verschillende talen zijn, allemaal met een formalisatie, is het waardevol om overkoepelende kaders te hebben die de semantiek van families van talen en logica's kunnen vastleggen. Een voorbeeld van zo'n unificerend kader is justificatietheorie, waarin de semantiek wordt gedefinieerd door het gebruik van verklaringen, in onze terminologie justificaties genoemd. Intuïtief is een justificatie een grafe die de waarheidswaarden van bepaalde feiten verklaart. Dit introduceert echter een potentieel probleem: de justificatiestatus van een feit en de ontkenning ervan kunnen inconsistent zijn. Dus om de justificatiesemantiek correct te definiëren, moeten deze statussen tegengesteld zijn. Een dergelijke semantiek wordt consistent genoemd.

In het eerste deel van dit proefschrift bewijzen we dat de belangrijkste semantieken van de justificatietheorie in feite consistent zijn. Bovendien bewijzen we bruikbare resultaten voor justificaties, zoals de mogelijkheid om justificaties samen te stellen. Een ander probleem met justificatietheorie is dat er twee soorten van justificaties zijn, wat kan resulteren in verschillende semantieken. We laten zien dat deze twee ogenschijnlijk niet-gerelateerde problemen in feit nauw met elkaar verbonden zijn.

Daarna leggen we een verband tussen justificatietheorie en speltheorie, waardoor justificaties kunnen worden gezien als strategieën in een spel met twee spelers. Deze verbinding biedt een oplossing voor de problemen die zich voordoen in justificatietheorie door een algemene voorwaarde op de semantiek te geven in het geval van een eindig systeem.

Justificatietheorie is niet het enige unificerend kader voor de semantiek van niet-monotone logica's. Een ander bekend kader is benaderende vastepuntstheorie, dat een meer algebraïsch karakter heeft. We leggen een verband tussen justificatietheorie en benaderende vastepuntstheorie. De notie van ultieme

semantiek van benaderende vastepuntstheorie kan overgebracht worden naar het domein van justificatietheorie. Hierdoor kan de justificatiesemantiek meer semantieken vastleggen dan voorheen.

Als laatste onderwerp van dit proefschrift kijken we naar het nesten van justificaties, dat gebruikt dan worden om modulaire semantieken te definiëren. In eerdere definities van nesten gaat een aanzienlijke hoeveelheid informatie verloren omdat er een compressiebewerking wordt gebruikt. We geven een alternatieve en meer algemene definitie zonder dit nadeel. Dit zorgt voor een meer intuïtieve modulaire semantiek op basis van justificaties. We bewijzen dat dit equivalent is aan de compressie voor boomachtige justificaties en in speciale gevallen voor grafeachtige justificaties. We onderzoeken ook de consistentie van beide benaderingen en als een toegevoegde bonus lossen we consistentie op voor boomachtige justificaties.

Samengevat, dit proefschrift verzamelt een aantal vorderingen in de justificatietheorie en illustreert deze met voorbeelden.

List of Abbreviations

AF Argumentation Framework.

AFT Approximation Fixpoint Theory.

AI Artificial Intelligence.

ASP Answer Set Programming.

FO first-order logic.

FO(ID) First-order logic extended with Inductive Definitions.

KRR Knowledge Representation and Reasoning.

NMR Non-Monotonic Reasoning.

ZFC Zermelo–Fraenkel set theory with the axiom of Choice.

List of Symbols

$A_{\mathcal{JF}}$ The approximator corresponding to $\mathcal{JF}$

$\mathcal{B}$ A branch evaluation

$\mathbf{b}$ A branch

$B_J(x)$ The set of J-branches starting with x

C($\mathcal{JF}$) The complement of $\mathcal{JF}$

CC($\mathcal{JF}$) complementation of $\mathcal{JF}$

Compress($\mathcal{JS}$) The compression of $\mathcal{JS}$

$\overline{\mathcal{B}}$ The dual branch evaluation of $\mathcal{B}$

$\overline{\mathcal{JF}}$ The dual justification frame of $\mathcal{JF}$

$\ell(\mathbf{b})$ The length of $\mathbf{b}$

Expand(J) The expansion of J

$J \uparrow K$ The extension of justification J with justification K

$\mathcal{F}$ A fact space

$\mathbf{f}$ The truth value *false*

$\mathcal{F}_d$ The set of defined facts

Flat($\mathcal{JS}$) The flattening of $\mathcal{JS}$

$\mathcal{F}_-$ The set of negative facts

$\mathcal{F}_o$ The set of open facts

$\mathcal{F}_+$ The set of positive facts

$\mathcal{G}$ A game

$\mathcal{G}_Y^P$ Gale-Stewart game on Y with payoff set P

$\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ The justification game associated with $\mathcal{JS}$ and $\mathcal{I}$

$G_{\mathcal{JF}}$ The game graph corresponding to $\mathcal{JF}$

$\mathcal{I}$ An interpretation

$\mathcal{JF}$ A justification frame

$\langle \mathcal{F}, \mathcal{F}_d, R \rangle$ A justification frame

$\mathcal{JS}$ A justification system

$\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ A justification system

$\mathfrak{J}(x)$ The set of locally complete justification that have x as internal node

$\mathrm{val}(J, x, \mathcal{I})$ The value of a justification J for a fact x with respect to an interpretation $\mathcal{I}$

$\leq_p$ The precision order

$\leq_t$ The truth order

$\mathcal{L}$ The set of logical facts

$\mathfrak{F}(Y)$ The collection of infinite sequence all of whose finite initial segments belong to Y

$\mathrm{Merge}(\mathcal{JS})$ The merge of $\mathcal{JS}$

$\left\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \left\{ \mathcal{JS}^1, \ldots, \mathcal{JS}^k \right\} \right\rangle$ A nested justification system

$O_{\mathcal{JF}}$ The operator corresponding to $\mathcal{JF}$

Path_G The set of finite paths in G

$\mathbf{F}$ The false player

Play_G The set of plays in G

$p_G(s, \sigma, \tau)$ The play consistent starting with s consistent with σ and τ

$\sqsubseteq_P$ The preference relation for player P

$\mathbf{T}$ The true player

R A set of rules

$\mathrm{sgn}(x)$ The sign of x

$\mathrm{Shrink}(J)$ The shrinking of J

σ A strategy for $\mathbf{T}$

S_P The states owned by a player P

$\mathfrak{S}_g(P)$ The set of general strategies for P

$\mathfrak{S}_p(P)$ The set of positional strategies for P

$\mathcal{S}_{\mathcal{JS}}$ The support operator associated with $\mathcal{JS}$

$\mathrm{SV}(x, \mathcal{I})$ The supported value of x with respect to an interpretation $\mathcal{I}$

τ A strategy for $\mathbf{F}$

$\sim$ The negation operator

$\mathbf{t}$ The truth value *true*

$\mathrm{U}(O)$ The ultimate approximator of O

$\mathrm{U}(\mathcal{JF})$ The ultimate frame of $\mathcal{JF}$

$\mathrm{U}_{\mathcal{JF}}$ The ultimate approximator associated to the justification frame $\mathcal{JF}$

$\mathbf{u}$ The truth value *unknown*

$\mathrm{Unfold}\,(\mathcal{JS})$ The unfolding of $\mathcal{JS}$

$J_\sigma(x)$ The justification corresponding to the strategy σ

L^2 The bilattice on L

M^* The set of paths obtained by finite iterations of loops in M

M^ω The set of paths obtained by infinite iterations of loops in M

$u(x, \sigma, \tau)$ The value of the play starting with x consistent with σ and τ

$x \leftarrow A$ A rule with head x and body A

Contents

Chapter 1

Introduction

1.1 Background

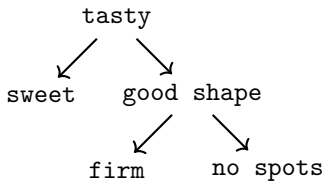
This thesis is situated in the field of Artificial Intelligence (AI), in particular the Knowledge Representation and Reasoning (KRR) subfield. The primary concern of KRR is to define languages and logics to represent knowledge, and conceive tools and techniques to reason with this knowledge. Of course, this is an oversimplified view, but it allows us to look at the bigger picture. To have a perfect reasoner, one should first capture the knowledge in the problem domain and then unleash the reasoning power at hand. We should contrast this with function-fitting methods in other parts of artificial intelligence, such as deep learning. These methods, although very useful for other problems (e.g., speech recognition), are not always able to reason 100% logically correct due to their approximating nature. Darwiche (2018) defines this dichotomy of AI as model-based vs. function-based AI. Model-based AI has a distinct advantage over function-based techniques: the internal mechanisms are usually captured as well, in contrast to function-based techniques which behave like black boxes. Therefore, these methods lean better towards explainable systems as opposed to more opaque function-based methods.[1] Moreover, function-based AI is usually tailored for one specific goal. A major part of model-based AI is Knowledge Representation and Reasoning (KRR). Represented knowledge can be used to solve multiple problems; hence it is multi-goal oriented. This idea is central in the knowledge base paradigm (Cat et al., 2014; Denecker and Vennekens, 2008).

[1] There are efforts to add explanations to function-based methods, but these are often of a rudimentary form, e.g., highlighting specific parts of an image that led to the recognising of an object (Darwiche, 2018).

As such, these techniques are better to handle novel scenarios, for example in causal reasoning. The higher degree of explainability of the model-based approach is ever more important due to AI regulations and policies, for example the *right to explanation*[2]. Embracing explanations would be a perfect starting point for practising contemporary Knowledge Representation and Reasoning (KRR).

In KRR, many languages and logics are invented for varying purposes ranging from semantic web ontology languages (Antoniou and van Harmelen, 2009; Baader et al., 2005) to robotics (Paulius and Sun, 2019), legislation and law (Jones, 1990). Often, these logics are based on the same principles, but are worked out in slightly different ways, or they are addressed from a different angle. To see the bigger picture, it is paramount that we can establish relations between the different languages and logics: are they distinct dialects with the same underlying principles or are they inherently different. One technique available to study these relationships is to define transformations between languages. In particular, one can find an embedding of one language into another. However, this is impractical due the sheer amount of knowledge representation languages available. Another, more feasible, way is to conceive frameworks unifying the semantics of a large group of logics.

Luckily, justification theory tackles both the issue of explainability and that of unification. The semantics (meaning) of justification theory are based on the notion of a justification, which is a graph[3] explaining why something holds. For example, the justification (it is not important to completely understand these pictures at this point)



serves as an explanation why an apple is tasty (your definition of tasty can vary of course).

Justification semantics work by giving a rating to each justification indicating if it is good or not. By varying the way we rate the justifications we can vary the semantics of the same domain. To change domains, justification theory

[2] As manifested for example in the European Union General Data Protection Regulation.

[3] Not a graph of a function, but the mathematical structure representing relations between objects.

uses a set of rules as a basis to construct explanations. Therefore, it is not difficult to see that justification theory acts as a unifying framework. Combined with the fact that justifications (explanations) are the major constituent of the semantics, justification theory is an excellent study object for KRR.

1.2 Justifications in KRR

Justification theory as detailed in this thesis originated in the PhD thesis of Denecker (1993) and was further formalised by Denecker, Brewka, and Strass (2015). Justifications – albeit not always in the exact formal form as described by Denecker (1993); Denecker et al. (2015) – have appeared in different ways in different areas in KRR. Justification theory has its origin in logic programming, where the concept of justifications has been around for quite a while. According to Fandinno and Schulz (2019), the concept of a justification originates from work on debugging by Shapiro (1983); Sterling and Lalee (1986). Building on that work, Lloyd (1987) introduced the notions of incorrect rules and uncovered atoms for a declarative error diagnoser. These notions indicate when a structure is not a model of the given logic program. The next year, Sterling and Yalçinalp (1989) explained Prolog expert systems with a meta-interpreter. In 1990, Fages (1990) defined the stable semantics for logic programs in terms of justifications, which in this case are sets of atoms. A lot of justification approaches use the notion of a supported set/value, which was first introduced by Pereira et al. (1992, 1993). This was around the same time Denecker and Schreye (1993) introduced the early version of justification theory. In 2000, Roychoudhury et al. (2000) justified tableau-based proofs by providing justifications for logic programs.

Fandinno and Schulz (2019) listed several prominent justification approaches for Answer Set Programming (ASP) besides justification theory:

- off-line justifications (Pontelli and Son, 2006)
- labelled assumption-based argumentation based answer set (LABAS) justifications (Schulz and Toni, 2016)
- causal justifications (Cabalar and Fandinno, 2016; Cabalar et al., 2014)
- why-not provenance (Damásio et al., 2013)
- rule-based justifications (Béatrix et al., 2016)

The first three approaches are similar to justification theory since justifications are also dependency graphs between literals or rules.

Off-line justifications (Pontelli and Son, 2006) are graph structures that describe the truth value of an atom with respect to a given answer set. Because off-line justifications only use atoms as nodes, nodes have to be labelled with + or − indicating if it is true or false. Similarly, edges have to be labelled with + and − as well to indicate positive or negative dependence. The truth of negative literals are assumed to be true and are not further explained.

LABAS justifications (Schulz and Toni, 2016) are very similar to off-line justification, but they abstract away from intermediate rule applications. So they only point out the literal in question and the facts occurring in rules used in the derivation. Additionally, the truth of negative literals is not assumed, but is further explained in terms of the truth value of the underlying atom.

Causal graph justifications (Cabalar and Fandinno, 2016, 2017; Cabalar et al., 2014), contrary to off-line and LABAS justifications, are used to formalise and reason with causal knowledge, which is its main focus. They can also be used to explain why a literal is contained in an answer set. Additionally, an algebra for combining justifications was defined.

Why-not provenance (Damásio et al., 2013) is based on database literature. Justifications here are not graph-based and are used to explain the truth value of an atom.

Rule-based justifications (Béatrix et al., 2016) are defined in the context of rule-based answer set computation (Lefèvre et al., 2017): the search algorithm guesses on the application or non-application of a rule instead of the truth value of atoms. The ASP-solver ASPeRiX uses these justifications for backjumping.

Readers interested in the intricate differences between the different justification approaches should take a look at the excellent work from Fandinno and Schulz (2019). Besides these approaches, there are a number of applications where justifications are used. The work on inductive definitions by Mariën (2009); Mariën et al. (2005, 2007, 2008) uses justifications as a semantical basis for inference techniques in inductive definitions. Justifications are used to perform non-clausal local search (Järvisalo et al., 2008). Data structures similar to justifications are also used in implementations of answer set solvers (they form the basis of the so-called source-pointer approach in the unfounded set algorithm (Gebser et al., 2009)). Moreover, justifications turned out to be key in analysing conflicts in the context of lazy grounding (Bogaerts and Weinzierl, 2018). Very recently, Lapauw et al. (2020) used justifications to improve parity game solvers.

1.3 Research Aims and Motivation

In broad terms, our aim is to advance justification theory. In particular, two properties with associated research questions are extensively studied throughout this text.

The first property is concerned with the soundness of justification semantics. A benefit of justification theory is that it provides a lot of freedom to create new semantics by means of *branch evaluations*. These evaluations give values to paths in an explanation graph and thus determine whether a justification is considered good or not. In justification theory, there is no real distinction between a fact and its negation. This means that both a fact and its negation can have good justifications, which in normal circumstances[4] is considered a contradiction. For example, if you have a good reason why x holds and a good reason why x does not hold, then we obtain a contradiction. If such contradictions do not occur, we say that its justification semantics is consistent.

Justification theory was originally developed to capture the semantics of inductive definitions. Inductive definitions are extremely important in mathematics as they are used to generate mathematical objects such as the subgroup generated by a subset, Borel sets, the natural numbers, recursive formulae, truth relations in logics, etc (Buchholz et al., 1981). To quote Gandy (1974, page 265)

> Mathematical logic is certainly permeated with inductive definitions.

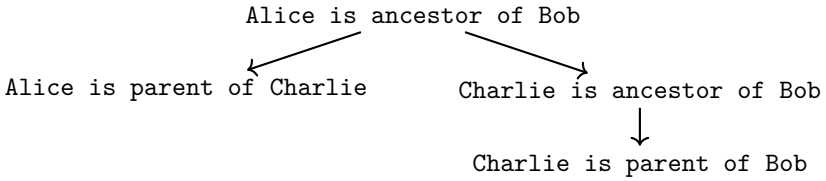
For example, we can inductively define the ancestor relation given the parent relation with the following rules.

- A person p is an ancestor of q if p is a parent of q.
- A person p is an ancestor of q if there is a person r so that p is a parent of r and r is an ancestor of q.

Intuitively, it is clear what this should mean. However, the formal semantics of a set of inductive rules is not immediately clear. Generally, there are two approaches to provide formal semantics to inductive definitions. The first approach is constructive and builds the relation from the bottom up: you start with the empty relation and iteratively apply rules in the inductive definition until the relation stops increasing. The second approach is defining the relation

[4] Paraconsistent logics handle contradictions in a tolerant way. We are not concerned with paraconsistent logics in this text.

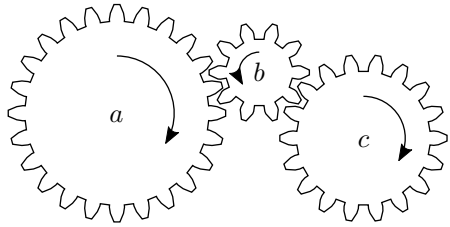
as the smallest relation that satisfies the induction rules. Both approaches provide the same result if negation does not occur in the induction rules, however this breaks down when negation is involved. Justification theory follows the constructive approach, but extends it to a broader class of constructions.[5] In that sense, we can see justifications as a construction of why a fact belongs to a set/relation. Going back to the ancestor relation, the following justification respresents the construction process of why Alice is an ancestor of Bob.



The justification above describes the process by which Alice is an ancestor of Bob: which induction rules are needed.

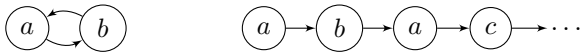
It still remains the question what a justification for a fact such as `Bob is not an ancestor of Alice` means. Such a justification provides a witness of why `Bob is an ancestor of Alice` is not constructible. The notion of constructability is of course classical: something cannot be both in a relation and not in a relation. This means that investigating the consistency of justification semantics is paramount.

The second property is more concerned about what shape our explanation can be. In more technical terms, justifications are directed graphs. However, we allow two forms, one in which always the same 'choice' is made, and one where multiple choices can be made. Let us take a look at an interlocking gear example.



In this situation, we have the following two justifications:

[5] Approximation Fixpoint Theory (AFT) follows and extends the second approach.



On the left side, you see a circular reasoning[6] where the clockwise turning of the first cog is justified by the counter-clockwise turning of the second cog and vice versa. While, on the right side, the clockwise turning of the first cog is first justified by the counter-clockwise turning of the second cog, which is later on justified by the clockwise turning of the third cog. Since writing down the right type of graph, leaves us with a tree-like structure (or root-like structure depending on the orientation), we say that the right type are *tree-like* justifications. To contrast between tree-like justifications, the first type are called *graph-like* justifications. Initially, Denecker and Schreye (1993) invented tree-like justifications, but later on Denecker et al. (2015) invented graph-like justifications.

Now we have enough information to state the second central question of this thesis: when are the two types of justification equivalent; meaning a fact has a good graph-like justification if and only if it has a good tree-like justification. This property is called *graph-reducibility*, the reason why it is called that way is explained in Chapter 2.

Investigating both properties is the central aim of this thesis. In particular, we aim to find restrictions on branch evaluations such that these properties hold. Additionally, these properties are future guidelines for new justification semantics. If new justification semantics are devised, it should satisfy both properties to be of practical use as the first property ensures that the semantics are not contradictory, while the second property allows for the compression of tree-like justification into graph-like justifications. This has important ramifications in real computer applications, as most of the time, a tree-like justification is an infinite structure and consequentially does not fit in memory. So if there is an equally good graph-like justification, this can then be used instead. Justifications are used for example in parity game solvers, see the work by Lapauw et al. (2020).

Somewhat unexpectedly, these properties are intrinsically related to each other: if graph-like justifications are consistent, then graph-reducibility holds and tree-like justifications are also consistent.

Apart from these two properties, we have two additional lines of research. As mentioned before, justification theory aims to be a unifying framework. Another framework with origins in logic programming is AFT. The foundations of AFT lie in Tarksi's fixpoint theory (Tarski, 1955); hence the fundamental objects

[6] In some semantics, such as co-well-founded semantics, as we see later, this is actually a valid argument.

in AFT are operators and their fixpoints. Since there is an overlap between the logics justification and AFT capture, it begs the question of what is the relation between the two formalisms.

Our last line of research is about modularity in knowledge representation. In order to have efficient knowledge representation, it is paramount that if new logical constructs are added, that the semantics does not have to be completely overhauled to get it working. That is, knowledge representations should be composable. Denecker et al. (2015) provided a way to compose different justification semantics by nesting of justification frames. However, to give meaning to the nesting, it is compressed to regular justification semantics. By doing this, a lot of information is lost in the justifications, which has the effect that justifications do not have a proper explaining characteristics. Therefore, it begs the question whether we can improve on this by providing an alternative way to look at nestings without losing information in the justifications.

1.4 Structure of Thesis

In the next chapter, we establish the preliminaries for justification theory, which includes detailed definitions for justification theory as well as the four main branch evaluations. We do some ground work for the next chapters such as results concerning pasting-together of justifications as well as formally defining our two research questions and goals. This chapter finishes by showing that justification theory can capture logic programming semantics and abstract argumentation frameworks.

In the third chapter, we take a deeper look into the consistency of the four main justification semantics and prove some relations between the models of these main justification semantics.

The fourth chapter further investigates the consistency problem, but does so from a different angle by embedding justification theory into game theory. This embedding effectively shows that justifications can be seen as strategies in some two-player game. Using game theoretic results, we find two additional properties for branch evaluations that imply its consistency in a finite setting.

Previously, we talked about justification theory being a unifying framework. Another unifying framework that tries to capture similar semantics is AFT. In Chapter 5, we explore the relation between both unifying frameworks. As an added bonus, we transfer ultimate semantics, a concept from AFT, to justification theory; further expanding the number of semantics being able to be captured by justification theory.

One benefit of justification theory that we have not discussed yet is its modularity. Justification theory achieves this by nesting. Intuitively, this can be viewed as justifications inside justifications. This is further explored in Chapter 6. As an unexpected consequence we find a property that implies the satisfaction of graph-reducibility. Moreover, we settle the consistency for tree-like justifications. We end this chapter with a number of applications that use nested justification systems.

The seventh and final chapter provides a conclusion and possible future directions for this line of research.

Most chapters are a stand-alone topic, except for Chapter 2 as it lays the ground work for the subsequent chapters. Each chapters has a conclusion in which future directions are laid out for that line of research.

Chapter 2

Justification Theory

2.1 Introduction

This chapter first introduces the basic notions of justification theory. After that, we discuss the consistency problem in detail. Equivalence of justification frames and complementarity of rules are also discussed. We give an important counterexample that does not have the consistency property. Incidentally, it is also a counterexample for graph-reducibility, which is then discussed next. The relation between consistency and graph-reducibility is then explained and elaborated. Next, a number of branch evaluation types are given, as well as dual justification semantics. One interesting set of branch evaluations are splittable and they allow us to paste justifications together to obtain an explanation for everything at once. The chapter ends by showing how logic programming semantics and abstract argumentation framework semantics can be captured by justification theory.

The content of this chapter is an amalgamation of four papers (Marynissen et al., 2018a,b, 2020, 2021) with myself as first author. Most proofs and examples are unpublished.

2.2 Definitions

In the introduction we saw a few simple examples of justifications. The nodes in justifications are the basic elements for forming justifications. These are called facts. Each fact has its corresponding negative fact. For instance, if `fly`

means a certain object can fly, then $\sim$`fly` means that that object cannot fly. Here we take the classical convention that $\sim\sim x = x$ for any x.[1] Usually we make an asymmetric distinction between x and $\sim x$. Apart from regular facts, there are a few special facts corresponding to the logical values true, false, and unknown. They are denoted by respectively $\mathbf{t}$, $\mathbf{f}$, and $\mathbf{u}$. Following classical three-valued logic (Belnap, 1977) we get that $\sim\mathbf{t} = \mathbf{f}$, $\sim\mathbf{f} = \mathbf{t}$ and $\sim\mathbf{u} = \mathbf{u}$. The set $\mathcal{L}$ denotes the set $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. We have two orders on $\mathcal{L}$: the truth order $\leq_t$ for which $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$ and the precision order $\leq_p$ for which $\mathbf{u} \leq_p \mathbf{f}$ and $\mathbf{u} \leq_p \mathbf{t}$. This captures that when something is $\mathbf{u}$, we do not have information on its value, while $\mathbf{f}$ and $\mathbf{t}$ are as precise as can be. All of this is summarised in the definition of a fact space.

Definition 2.2.1. A *fact space* is any set $\mathcal{F}$ containing $\mathcal{L}$ together with a function $\sim\colon \mathcal{F} \to \mathcal{F}$ and a partitioning $\mathcal{F}_+ \cup \mathcal{F}_-$ of $\mathcal{F} \setminus \mathcal{L}$ such that

- $\sim$ is an involution, i.e. a bijection that is its own inverse ($\sim\sim x = x$ for all $x \in \mathcal{F}$);
- $\sim\mathbf{t} = \mathbf{f}$, $\sim\mathbf{f} = \mathbf{t}$, and $\sim\mathbf{u} = \mathbf{u}$;
- for all $x \in \mathcal{F} \setminus \{\mathbf{u}\}$ the facts x and $\sim x$ are different;
- $x \in \mathcal{F}_+$ if and only if $\sim x \in \mathcal{F}_-$ for all $x \in \mathcal{F} \setminus \mathcal{L}$.

Elements in $\mathcal{F}_+$ (respectively $\mathcal{F}_-$) are called positive (respectively negative) facts. Sometimes, the partition in positive and negative facts is given by a *sign function*, which is a function $\operatorname{sgn}\colon \mathcal{F} \setminus \mathcal{L} \to \{+, -\}$ such that $\operatorname{sgn}(x) \neq \operatorname{sgn}(\sim x)$ for all $x \in \mathcal{F} \setminus \mathcal{L}$. This corresponds to the partition as follows: $\mathcal{F}_+ = \operatorname{sgn}^{-1}(\{+\})$ and $\mathcal{F}_- = \operatorname{sgn}^{-1}(\{-\})$.

When a fact space $\mathcal{F}$ is mentioned, the function $\sim$ and the partition $\mathcal{F}_+$ and $\mathcal{F}_-$ are often implicitly given. If A is a subset of $\mathcal{F}$, then by $\sim A$ the set $\{\sim a \mid a \in A\}$ is meant.

When speaking about facts, usually we say that some facts are true and others are false, or even unknown. In a consistent world, if x is true, then of course $\sim x$ is false because our negation $\sim$ is classical. This is represented by an interpretation.

Definition 2.2.2. For a fact space $\mathcal{F}$, a (three-valued) *interpretation* of $\mathcal{F}$ is a function $\mathcal{I}\colon \mathcal{F} \to \mathcal{L}$ that

- commutes with $\sim$, i.e. $\mathcal{I}(\sim x) = \sim\mathcal{I}(x)$ for all $x \in \mathcal{F}$;

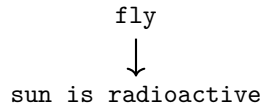
[1] People familiar with Answer Set Programming (ASP) tend to work in different logics such as Gödels G3 logic in which $\sim\sim x$ is not equivalent with x.

- is the identity on $\mathcal{L}$, i.e. $\mathcal{I}(\ell) = \ell$ for $\ell \in \mathcal{L}$.

Interpretations assign logical values to facts so that the value for a fact and its negation are opposite. They represent a possible state of affair of the fact space. Sometimes, interpretations are called structures, especially in the context of first-order logic. The order $\leq_p$ extends to interpretations by piecewise comparison. That is, $\mathcal{I}_1 \leq_p \mathcal{I}_2$ if for all $x \in \mathcal{F}$, $\mathcal{I}_1(x) \leq_p \mathcal{I}_2(x)$. The order $\leq_t$ extends to interpretations by piecewise comparison of positive facts, i.e. $\mathcal{I}_1 \leq_t \mathcal{I}_2$ if for all $x \in \mathcal{F}_+$, $\mathcal{I}_1(x) \leq_t \mathcal{I}_2(x)$.

Example 2.2.3. Let $\mathcal{F} = \mathcal{L} \cup \{\texttt{bird}, {\sim}\texttt{bird}, \texttt{wounded}, {\sim}\texttt{wounded}, \texttt{bat}, {\sim}\texttt{bat}\}$ be a fact space with $\mathcal{F}_+ = \{\texttt{bird}, \texttt{wounded}, \texttt{bat}\}$. If we have a certain animal a, then the informal meaning of the facts above should be clear. An interpretation $\mathcal{I}$ with $\mathcal{I}(\texttt{bird}) = \mathbf{t}$ represents a state of affair in which a is a bird. ▲

A fact space is the main ingredient for constructing justifications. The following justification however shows we need some extra structure:



We know that the sun is radioactive, however, that is not a good explanation of why something can fly and these facts are not even remotely related to each other. What we need is some way to describe the relations between facts. This is done by providing rules for certain facts. For example, one rule could be

$$\texttt{fly} \leftarrow \texttt{bird}, {\sim}\texttt{wounded}$$

where the comma between the facts can be read as 'and'. This rule describes that if the animal is a bird and not wounded, then the animal can fly. Another rule could for example be

$$\texttt{fly} \leftarrow \texttt{bat}, {\sim}\texttt{wounded}$$

However, not all facts have such rules. It could be that the value of a fact is determined from the outside, not described by our system. For example, we could have established with measurements that the sun is radioactive, but there is no good rule for this (in the fact space we are working with). Facts that do have rules are called *defined* and facts that do not have rules are called *open*. Sometimes, open facts are called parameters. It should be obvious that the logical facts $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{u}$ are open. We assume that if a fact x is defined,

that $\sim x$ is also defined. This is evident by noting that if you can describe why something holds, then you can also describe why something does not hold. That is why, usually, there is a relation between the rules for x and $\sim x$. The exact correspondence is postponed to Section 2.3.2. The discussion on justification frames is summarised in the definition below.

Definition 2.2.4. A *justification frame* $\mathcal{JF}$ is a tuple $\langle \mathcal{F}, \mathcal{F}_d, R \rangle$ such that

- $\mathcal{F}$ is a fact space and $\mathcal{F}_d$ is a subset of $\mathcal{F}$;
- no logical fact is defined: $\mathcal{L} \cap \mathcal{F}_d = \emptyset$;
- $R \subseteq \mathcal{F}_d \times \left(2^{\mathcal{F}} \setminus \{\emptyset\}\right)$;
- for each $x \in \mathcal{F}_d$ there is at least one element $(x, A) \in R$.

The third point says that we cannot have a rule $(x, \emptyset)$. This means that in every case x should hold. However, such a situation can be described by a rule $(x, \{\mathbf{t}\})$. The last point of the definition says that every defined fact x has at least one rule (x, A). This makes sense, otherwise x is not described by the justification frame, and thus would have been an open fact instead. We say (x, A) is a rule for x. The set A is called the body of the rule and a case of x. We write $\mathcal{JF}(x)$ for the set of cases of x in $\mathcal{JF}$. The set $\mathcal{F}_o$ denotes $\mathcal{F} \setminus \mathcal{F}_d$ and naturally represents the set of open facts. Rules in a justification frame are technically a tuple (x, A) but we usually denote them by $x \leftarrow A$. If A is an enumerable set $\{y_1, \ldots, y_n\}$, then we write $x \leftarrow y_1, \ldots, y_n$ instead of the more formal $x \leftarrow \{y_1, \ldots, y_n\}$.

We are now ready to formally define justifications. The underlying structure of a justification is a directed graph. For the sake of completion we give the definition of a directed graph we use.

Definition 2.2.5. A *directed (unlabelled) graph* is a tuple $\langle N, E \rangle$ where N is a set of nodes and $E \subseteq N \times N$ is a set of edges. A *directed labelled graph* is a quadruple $\langle N, L, E, \ell \rangle$ where N is a set of nodes, L a set of labels, $E \subseteq N \times N$ is a set of edges, and $\ell \colon N \to L$ is a function called the labelling.

A node with outgoing edges is called an *internal* node. If a node does not have outgoing edges, we call it a *leaf*[2]. When there is an edge from x to y, then x is a *parent* of y and y is a *child* of x. In previous examples of justifications, we observed that the children of a node should be a case of that node. Or in other words, the justification should reflect the structure of the justification frame.

[2]The more common terminology for graphs is a *sink*. However, the work by Denecker (1993) only considered justifications which were trees, in which this terminology is customary. We tried not to break too much with existing terminology of early justification theory.

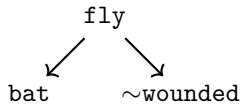
Definition 2.2.6. Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame. A *graph-like justification* is a directed graph $\langle N, E \rangle$ with $N \subseteq \mathcal{F}$ such that for all internal nodes $x \in N$ we have that $\{y \in N \mid (x, y) \in E\}$ is a case of x in $\mathcal{JF}$, i.e. $x \leftarrow \{y \in N \mid (x, y) \in E\} \in R$.

A *tree-like justification* is a directed labelled graph $\langle N, \mathcal{F}, E, \ell \rangle$ that is acyclic (considered as an undirected graph) and for all internal nodes $x \in N$ we have that $\{\ell(y) \mid (x, y) \in E\}$ is a case of $\ell(x)$ in $\mathcal{JF}$, i.e. $\ell(x) \leftarrow \{\ell(y) \mid (x, y) \in E\} \in R$.

Slightly abusing notation, for a node n with label x in a tree-like justification, we usually refer to x instead of n.

We have gone through enough preliminaries to explain how justifications are rated, i.e. when a justification is considered a good explanation.

Example 2.2.7. Take a look at the following justification:



If the underlying justification frame also has a rule $\sim$`wounded` $\leftarrow$ $\sim$`scratched`, then the justification is not a complete explanation of why the animal can fly. ▲

In the previous example, we see that there is a leaf that is in $\mathcal{F}_d$. This means that the justification is only a partial explanation. Therefore, to complete it, this fact should also be explained. So we tend to only consider justifications that do not have defined facts in their leaves.[3]

Definition 2.2.8. A justification is *locally complete* if it has no defined leaves. For $x \in \mathcal{F}$ the set $\mathfrak{J}(x)$ is the set of locally complete justifications that have x as an internal node.

Since open facts are determined from the outside, we only look at justifications as explanations for defined facts. So take a defined fact x and let J be a locally complete graph-like justification. The only 'relevant' part of the justification is the part of J reachable from x. One way to do this is by looking at paths in J starting with x. In particular, justification theory considers all maximal paths of J starting with x, called J-branches.

[3] Non locally complete justifications can also be evaluated, see Definition 2.6.3.

Definition 2.2.9. For a justification J, a *J-branch* is a path of facts $x_0 \to x_1 \to \ldots$ in J that is either infinite or finite and ending in a leaf of J. The set of J-branches starting with x is denoted as $B_J(x)$.

To provide meaning to a justification, we need to evaluate these branches. We could have a different evaluation for each justification, but we think of the meaning as something universal; being applicable to any justification frame. Therefore, it is useful to define what a branch is with respect to a justification frame $\mathcal{JF}$.

Definition 2.2.10. 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 $\mathcal{F}_o$.

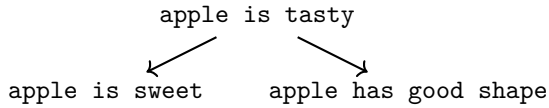
It is straightforward that J-branches are $\mathcal{JF}$-branches if J is locally complete. A branch $\mathbf{b}$ is almost always denoted as $\mathbf{b}\colon x_0 \to x_1 \to \cdots$. The negation $\sim\mathbf{b}$ of $\mathbf{b}$ is defined as $\sim x_0 \to \sim x_1 \to \cdots$. Despite of the '$\cdots$', it can also denote a finite branch. A tail of $\mathbf{b}$ is any branch $x_i \to x_{i+1} \to \cdots$ for $i \geq 0$. When saying a positive branch, we mean a branch having only positive elements.

To evaluate branches we need a mapping from branches to facts. This is exactly what a branch evaluation is.

Definition 2.2.11. 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 *justification system* $\mathcal{JS}$ is a tuple $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ such that $\langle \mathcal{F}, \mathcal{F}_d, R \rangle$ is a justification frame and $\mathcal{B}$ is a branch evaluation.

The condition that a branch evaluation should be simultaneously defined for all justification frames is important. This condition ensures that branch evaluations can only use the structure of the underlying fact space. For example, an evaluation that evaluates a branch to $\mathbf{f}$ if it contains the fact `secret` is not allowed since `secret` is not an element in all fact spaces. Therefore, if you want to define a new branch evaluation, then this should not take the specific justification frame into account and should be applicable to all justification frames, and not only a few specific ones. Even though, leaving out this restriction would theoretically still work and all results in thesis will still hold. However, if the semantics belonging to a new branch evaluation adheres to proper knowledge representation principles, it should be broadly applicable. Therefore, branch evaluations are only allowed to use the structure in the given fact space. However, if the new branch evaluation only works on specific fact spaces, then it is advised to put additional structure onto the fact space and allow the branch evaluation only to be used on such fact spaces. We will see an example of that in Chapter 6, when we see nestings of justification systems.

We will provide plenty of examples of branch evaluations later on, but let us first show how branch evaluations are used to rate justifications. Consider the following justification:



This might seem like a good explanation why an apple is tasty, but if `apple is sweet` is not true, then the whole explanation collapses. This shows that the value of a justification depends on the interpretation of its facts. So let us take an interpretation $\mathcal{I}$. This justification has two very simple branches `apple is tasty` $\to$ `apple has good shape` and `apple is tasty` $\to$ `apple is sweet`. Assume, for the sake of the argument, that our branch evaluation maps finite branches to their last element. Hence, we get the facts `apple is sweet` and `apple has good shape`. By interpreting them with $\mathcal{I}$ we get two logical facts. Intuitively, we say that the worst value of these is then our value of the justification. If one of them is false, then our explanation has to be false. If they both hold, then it is a good explanation. This underlies the intuition of the value of a justification.

Definition 2.2.12. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system, $\mathcal{I}$ an $\mathcal{F}$-interpretation, J a locally complete justification in $\mathcal{JS}$, and $x \in \mathcal{F}_d$ an internal node of J. The *value* of J in x under $\mathcal{I}$ is defined as:

$$\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \min_{\mathbf{b} \in B_J(x)} \mathcal{I}(\mathcal{B}(\mathbf{b})),$$

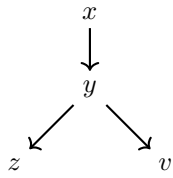
where min is with respect to $\leq_t$.

One of the simplest branch evaluations one can think of is just taking the second element of the branch as its evaluation.

Definition 2.2.13. The *supported (completion)* branch evaluation $\mathcal{B}_{\mathrm{sp}}$ maps $x_0 \to x_1 \to \cdots$ to x_1.

By using this branch evaluation, every justification corresponds to a single rule. Indeed, the children of x in J form a case A of x by the definition of justifications and all J-branches starting with x are mapped to their second element, which is inside A. Therefore, the value of the justification depends solely on the interpretation of the elements in A. Everything else in the justification is ignored to compute the value of J in x.

Example 2.2.14. Let $\mathcal{F}_d \cap \mathcal{F}_+ = \{x, y, z, v\}$ and let R contain the rules $x \leftarrow y$ and $y \leftarrow z, v$. The only locally complete justification for x is

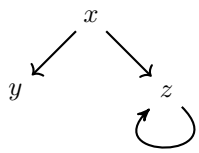


The branches $x \to y \to z$ and $x \to y \to v$ are mapped to y under $\mathcal{B}_{\text{sp}}$. Therefore, the branch evaluation $\mathcal{B}_{\text{sp}}$ only looks at the part $x \to y$; and thus only the rule for x is important to evaluate the justification under $\mathcal{B}_{\text{sp}}$. ▲

Sometimes, you do not want to allow infinite sequences of reasoning.

Definition 2.2.15. The *Kripke-Kleene* branch evaluation $\mathcal{B}_{\text{KK}}$ maps finite branches to their last element and infinite branches to $\mathbf{u}$.

Example 2.2.16. Let $\mathcal{F}_d \cap \mathcal{F}_+ = \{x, y, z\}$ and let R contain the rules $x \leftarrow y, z$ and $z \leftarrow z$. Suppose $\mathcal{I}$ is an interpretation with $\mathcal{I}(y) = \mathbf{t}$. The only locally complete justification for x is

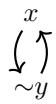


This justifications has two branches: $x \to y$ and $x \to z \to z \to \ldots$. The former is mapped to y under $\mathcal{B}_{\text{KK}}$ and the latter is mapped to $\mathbf{u}$; hence the value of the justification for x is $\min_{\leq_t} \{\mathbf{t}, \mathbf{u}\} = \mathbf{u}$. ▲

The previous branch evaluations did not make use of the partition of $\mathcal{F}_d$ into positive and negative facts. We can use this partition to introduce an asymmetry between x and $\sim x$.

Definition 2.2.17. The *well-founded* branch evaluation $\mathcal{B}_{\text{wf}}$ maps finite branches to their last element. Under $\mathcal{B}_{\text{wf}}$, infinite branches are mapped to $\mathbf{t}$ if they have a negative tail, to $\mathbf{f}$ if they have a positive tail and to $\mathbf{u}$ otherwise. The *stable (answer set)* branch evaluation $\mathcal{B}_{\text{st}}$ maps a finite branch $x_0 \to x_1 \to \cdots \to x_n$ with all of $x_0, \ldots, x_{n-1}$ having the same sign, to x_n; 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 to $\mathbf{t}$ if all x_i are negative and $\mathbf{f}$ if all x_i are positive.

Example 2.2.18. Let $\mathcal{F}_d \cap \mathcal{F}_+ = \{x, y\}$ and let R contain the rules $x \leftarrow \sim y$ and $y \leftarrow \sim x$ and their complementary rules $\sim x \leftarrow y$ and $\sim y \leftarrow x$ (see Definition 2.3.9). The only locally complete justification containing x is



Under $\mathcal{B}_{\text{st}}$, this represents a choice rule between x and y. If $\mathcal{I}(y) = \mathbf{f}$, then the value of this justification is $\mathbf{t}$ and vice versa. ▲

The names of these branch evaluations are not random, since they can be used to capture similarly-named semantics in logic programming, see Section 2.7.1.

Consider the following two justifications:

Suppose we know the animal is not wounded and that it is not a bird. However, we have no information about whether the animal is a bat in the interpretation ($\mathcal{I}(\texttt{bat}) = \mathbf{u}$). For simplicity sake, assume we use $\mathcal{B}_{\text{sp}}$, then the left justification evaluates to $\mathbf{f}$ and the right one to $\mathbf{u}$. Therefore we could say that we do not know `fly` and we do not know that `fly` is not false. This would mean that $\mathcal{I}(\texttt{fly}) = \mathbf{u}$. So if there are multiple justifications, we tend to consider only the best one. Such a best justification (of which there could be more than one) supports the fact up to the value of that justification, just as we have seen that `fly` is supported to be $\mathbf{u}$ by the right justification above. This idea is called the supported value.

Definition 2.2.19. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system, $\mathcal{I}$ an $\mathcal{F}$-interpretation, and $x \in \mathcal{F}_d$. The *supported value* of x in $\mathcal{JS}$ under $\mathcal{I}$ is defined as:

$$\text{SV}_{\mathcal{JS}}(x, \mathcal{I}) = \max_{J \in \mathfrak{J}(x)} \text{val}(J, x, \mathcal{I}),$$

where max is with respect to $\leq_t$. By abuse of notation, if $\mathcal{JS}$ is clear from context, we write $\text{SV}_{\mathcal{B}}(x, \mathcal{I})$ or $\text{SV}(x, \mathcal{I})$.

In principle you do not have to take into account all locally complete justifications with x as internal node. The value of a justification J in x is determined solely

by the set $B_J(x)$. Therefore, the only part of J that is of interest is the set of nodes in J reachable from x.

Definition 2.2.20. A node x is a *root* of J if every node of J is reachable from x in J.

Proposition 2.2.21. *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 locally complete graph-like (respectively tree-like) justification J' such that x is a root of J' and* $\mathrm{val}(J, x, \mathcal{I}) \leq_t \mathrm{val}(J', x, \mathcal{I})$ *for all interpretations $\mathcal{I}$.*

Proof. Take a node n labelled x. Let J' be the subgraph of J spanned by the nodes that are reachable from n. By definition J' is connected. Since J is locally complete, J will have no leaves with a defined fact as label, and so J' is locally complete. By construction $B_{J'}(x) \subseteq B_J(x)$, which means that $\mathrm{val}(J, x, \mathcal{I}) \leq_t \mathrm{val}(J', x, \mathcal{I})$ for all interpretations $\mathcal{I}$. □

Corollary 2.2.22. *To determine the supported value of a defined fact x, it suffices to look at the justifications that have x as root.*

It makes sense that open facts do not have a supported value since they are determined from the outside. We can however, extend the supported value to the whole of $\mathcal{F}$ by just taking the same value as the interpretation for open facts.

Definition 2.2.23. Let $\mathcal{JS}$ be a justification system. For any $\mathcal{F}$-interpretation $\mathcal{I}$, $\mathcal{S}_{\mathcal{JS}}(\mathcal{I})$ is the function $\mathcal{F} \to \mathcal{L}$ which maps $x \in \mathcal{F}$ to

- $\mathrm{SV}_{\mathcal{JS}}(x, \mathcal{I})$ if $x \in \mathcal{F}_d$;
- $\mathcal{I}(x)$ if $x \in \mathcal{F}_o$.

The function $\mathcal{S}_{\mathcal{JS}}$ is called the *support operator*. If $\mathcal{JS}$ consists of $\mathcal{JF}$ and $\mathcal{B}$, then we write $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}}$ for $\mathcal{S}_{\mathcal{JS}}$.

Assume we have an interpretation $\mathcal{I}$ such that $\mathcal{I}(x) = \mathbf{f}$, but there is a justification J with $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \mathbf{t}$. Then this interpretation seems to be incomplete by not utilising all the good explanations. Similarly, if $\mathcal{I}(x) = \mathbf{t}$, but the best justification for x has only the value $\mathbf{f}$, then $\mathcal{I}$ has unsupported claims. This brings us to models in justification theory.

Definition 2.2.24. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system. An $\mathcal{F}$-interpretation $\mathcal{I}$ is a *$\mathcal{JS}$-model* if for all $x \in \mathcal{F}_d$, $\mathrm{SV}_{\mathcal{JS}}(x, \mathcal{I}) = \mathcal{I}(x)$. If $\mathcal{JS}$ consists of $\mathcal{JF}$ and $\mathcal{B}$, then a $\mathcal{JS}$-model can be referred to as a *$\mathcal{B}$-model* of $\mathcal{JF}$.

In some cases, you might be interested only in models.

Definition 2.2.25. Two justification systems $\mathcal{JS}_1$ and $\mathcal{JS}_2$ are *equivalent* if the set of $\mathcal{JS}_1$-models and the set of $\mathcal{JS}_2$-models are equal. Two branch evaluations $\mathcal{B}_1$ and $\mathcal{B}_2$ are *equivalent* if for all justification frames $\mathcal{JF}$, the set of $\mathcal{B}_1$-models of $\mathcal{JF}$ and the of $\mathcal{B}_2$-models of $\mathcal{JF}$ are equal.

In some cases, one can also be concerned what the supported value is in a non-model.

Definition 2.2.26. Two justification systems $\mathcal{JS}_1$ and $\mathcal{JS}_2$ are *strongly equivalent* if $\mathcal{S}_{\mathcal{JS}_1} = \mathcal{S}_{\mathcal{JS}_2}$.

It is clear that strong equivalence implies equivalence since $\mathcal{JS}$-models are the fixpoints of $\mathcal{S}_{\mathcal{JS}}$.

2.3 Consistency of Justification Systems

Models can be thought of as fixpoints of the support operator, in the sense that $\mathcal{S}_{\mathcal{JS}}(\mathcal{I}) = \mathcal{I}$. However, we should note that the domain and range of the support operator are not equal: the range of the support operator is the set of functions from $\mathcal{F}$ to $\mathcal{L}$, while the domain is the set of $\mathcal{F}$-interpretations, which are functions from $\mathcal{F}$ to $\mathcal{L}$ with some additional properties such as $\mathcal{I}(\sim x) = \sim\mathcal{I}(x)$ for all $x \in \mathcal{F}$. Therefore, standard techniques of finding fixpoints by iterating the support operator do not work. This actually brings us to our first problem we try to solve in this thesis. Assume for some x we have a justification with value $\mathbf{t}$. This justification serves as an explanation of why x is true. This begs the question 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* for x with a value greater than $\mathbf{f}$. The question that then remains is: how to show that there are no such 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 $\mathrm{SV}(\sim x, \mathcal{I}) = \sim \mathrm{SV}(x, \mathcal{I})$. This is a fundamental property, otherwise justification semantics is unsound: we have a justification explaining x is ℓ, while there is no justification that explains that $\sim x$ has value $\sim\ell$.

Definition 2.3.1. A justification system $\mathcal{JS}$ is *consistent* if $\mathrm{SV}_{\mathcal{JS}}(\sim x, \mathcal{I}) = \sim \mathrm{SV}_{\mathcal{JS}}(x, \mathcal{I})$ for all $x \in \mathcal{F}_d$ and $\mathcal{F}$-interpretations $\mathcal{I}$.

Consistency is a reasonable assumption that, unfortunately, is not always satisfied. An obvious way to not satisfy it is by having unrelated rules for x and $\sim x$.

$$\left\{ \begin{array}{l} x \leftarrow \mathbf{t} \\ \sim x \leftarrow \mathbf{t} \end{array} \right\}$$

Of course, in this example we cannot expect that the justifications for x and $\sim x$ are related because their rules are contradictory.

The first and most important research question of this thesis is the following.

Consistency problem

When is a justification system consistent? In particular, what properties do branch evaluations and justification frames need to have to ensure that the justification system is consistent?

2.3.1 Equivalent Justification Frames

Let us first look at the conditions on the justification frames. We will see that in a lot of formalisms captured by justification theory, rules are only given for one of x or $\sim x$ and the rules for the other are implicitly constructed. For example, if we have the following rules:

$$\left\{ \begin{array}{l} \texttt{fly} \leftarrow \texttt{bird}, \sim\texttt{wounded} \\ \texttt{fly} \leftarrow \texttt{bat}, \sim\texttt{wounded} \end{array} \right\}$$

then it is obvious that the animal does not fly if it is either wounded or both not a bird and not a bat. This provides the following rules:

$$\left\{ \begin{array}{l} \sim\texttt{fly} \leftarrow \texttt{wounded} \\ \sim\texttt{fly} \leftarrow \sim\texttt{bird}, \sim\texttt{bat} \end{array} \right\}$$

What happens is that these rules are constructed by taking a fact from the body of each rule for `fly` and negating them. So you will also get the redundant rules

$$\left\{ \begin{array}{l} \sim\texttt{fly} \leftarrow \sim\texttt{bird}, \texttt{wounded} \\ \sim\texttt{fly} \leftarrow \sim\texttt{bat}, \texttt{wounded} \end{array} \right\}$$

With redundant we mean the following.

Definition 2.3.2. A rule $x \leftarrow A$ is *redundant* in a justification frame $\mathcal{JF}$ if there is a rule $x \leftarrow B$ with $B \subsetneq A$.

Adding redundant rules to a justification frame will not alter any justification model. Adding a redundant rule will allow for more justifications, but such justification will not be better than without redundant rules. If a justification uses a redundant rule, this rule can be replaced by a 'better' rule to get a new justification that is a subgraph of the original justification; hence will be as good as the original one.

This insensitivity to redundant rules is captured by an equivalence relation on justification frames with the same underlying fact space.

Definition 2.3.3. Two justification frames $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ and $\mathcal{JF}' = \langle \mathcal{F}, \mathcal{F}_d, R' \rangle$ are *equivalent*[4] if for every rule $x \leftarrow A \in R$, there is a rule $x \leftarrow B \in R'$ such that $B \subseteq A$, and likewise for every rule $x \leftarrow B \in R'$, there is a rule $x \leftarrow A \in R$ such that $A \subseteq B$.

Note that the underlying fact space and set of defined facts should be the same to be able to speak about equivalence. Equivalent frames have the same models because their justifications are related.

Proposition 2.3.4. *If $\mathcal{JF}$ and $\mathcal{JF}'$ are equivalent, then for every branch evaluation $\mathcal{B}$ we have that $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}} = \mathcal{S}^{\mathcal{B}}_{\mathcal{JF}'}$.*

Proof. For the purpose of simplicity, we only prove this for graph-like justifications. The proof is similar for tree-like justifications. A graph-like justification is just a set of rules. Therefore, by equivalence, any justification J in $\mathcal{JF}$ can be transformed to a justification J' in $\mathcal{JF}$, by choosing[5] for each rule $x \leftarrow A$ in J a rule $x \leftarrow B$ in R' such that $B \subseteq A$. This justification J' is a subgraph of J. This means that $B_{J'}(x) \subseteq B_J(x)$; hence $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) \leq_t \mathrm{val}_{\mathcal{B}}(J', x, \mathcal{I})$. So for every best justification for x in $\mathcal{JF}$, we can find a justification for x in $\mathcal{JF}'$ that is at least as good. Therefore, $\mathrm{SV}^{\mathcal{B}}_{\mathcal{JF}}(x, \mathcal{I}) \leq_t \mathrm{SV}^{\mathcal{B}}_{\mathcal{JF}'}(x, \mathcal{I})$. By swapping $\mathcal{JF}$ with $\mathcal{JF}'$ in what we proved above we get that $\mathrm{SV}^{\mathcal{B}}_{\mathcal{JF}}(x, \mathcal{I}) = \mathrm{SV}^{\mathcal{B}}_{\mathcal{JF}'}(x, \mathcal{I})$; hence $\mathcal{S}^{\mathcal{B}}_{\mathcal{JF}} = \mathcal{S}^{\mathcal{B}}_{\mathcal{JF}'}$, which concludes our proof. □

We have seen that by adding redundant rules we stay in the same equivalence class. Sometimes, however, it is not possible to remove all redundant rules, as shown by Denecker et al. (2015, Example 3).

[4] This is an equivalence relation: it is reflexive, symmetric and transitive.
[5] This is possible by the Axiom of Choice.

2.3.2 Complementarity of Rules

We have seen in Section 2.3.1 that usually only one of x or $\sim x$ has explicit rules and that rules for the other can be constructed from the other. We now make this formal. Rules are constructed by taking an element of each case. This is exactly what a selection function does.

Definition 2.3.5. Let $\mathcal{F}$ be a fact space and $R \subseteq \mathcal{F} \times (2^{\mathcal{F}} \setminus \emptyset)$ be a set of rules. For $x \in \mathcal{F}$, let $R[x]$ denote the set $\{A \mid x \leftarrow A \in R\}$. Let $\mathcal{F}_d$ be the subset of $\mathcal{F}$ for which $R[x] \neq \emptyset$. A *selection function* of $x \in \mathcal{F}_d$ in R is a function $\mathcal{S}\colon R[x] \to \mathcal{F}$ such that $\mathcal{S}(A) \in A$ for all $A \in R[x]$.

A selection function for x selects an element of each case of x. In general, selection functions exist when you assume the Axiom of Choice (which we do throughout this text). For a given selection function $\mathcal{S}$ of x, the set $\{\mathcal{S}(A) \mid A \in R[x]\}$ is denoted by $\mathrm{Im}(\mathcal{S})$. Once you have a selection function, it can be used to construct new rules.

Definition 2.3.6. Let $\mathcal{F}$, R, and $\mathcal{F}_d$ be as before. Define the *complement* $\mathrm{C}(R)$ of R as the set of elements of the form $\sim x \leftarrow \sim \mathrm{Im}(\mathcal{S})$ for $x \in \mathcal{F}_d$ and $\mathcal{S}$ a selection function of x in R. For a justification frame $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$, the *complement* $\mathrm{C}(\mathcal{JF})$ is defined as $\langle \mathcal{F}, \mathcal{F}_d, \mathrm{C}(R) \rangle$.

This construction allows to construct rules for $\sim x$ from rules for x. For example, in logic programming, only rules for positive facts are given, while in abstract argumentation, only rules for negative facts are given. Using the above construction, we can transform logic programs and abstract argumentation frames into justification frames. We will come back to both of these applications later on in Sections 2.7.1 and 2.7.3.

Going back to the consistency problem, we saw that the rules for x and $\sim x$ need to be related. If every rule is constructed by using selection functions, then the rules for x and $\sim x$ are definitely related. To make it more broadly applicable, we will only demand equivalence to the complement.

Definition 2.3.7. A justification frame $\mathcal{JF}$ is *complementary* if it is equivalent to $\mathrm{C}(\mathcal{JF})$.

By exploiting the equivalence in the definition of complementarity, we get the following characterisation.

Proposition 2.3.8. *Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame. Then $\mathcal{JF}$ is complementary if and only if for every $x \in \mathcal{F}_d$ the following hold:*

1. *for every selection function $\mathcal{S}$ of x in R, there exists an $A \in \mathcal{JF}(\sim x)$ such that $A \subseteq \sim \mathrm{Im}(\mathcal{S})$;*
2. *for every $A \in \mathcal{JF}(x)$, there exists a selection function $\mathcal{S}$ of $\sim x$ in R such that $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$.*

Proof. This follows directly from writing out the equivalence in the definition of complementary. □

The complement above does not keep the original rules.

Definition 2.3.9. The *complementation*[6] $\mathrm{CC}(\mathcal{JF})$ of $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ is defined as the justification frame $\langle \mathcal{F}, \mathcal{F}_d, R \cup \mathrm{C}(R) \rangle$.

Complementation can also be applied to any set of rules R. Most examples will only explicitly mention the rules for one of x or $\sim x$. In that case, the complementation of those rules is implicitly assumed.

Example 2.3.10. Take $\mathcal{F}_d = \{a, b, c, \sim a, \sim b, \sim c\}$, $\mathcal{F}_o = \mathcal{L}$, and

$$R = \left\{ \begin{array}{l} a \leftarrow b \\ a \leftarrow c \\ b \leftarrow \mathbf{t} \\ c \leftarrow \mathbf{t} \end{array} \right\}.$$

The complementation of R gives a complementary justification frame $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \cup \mathrm{C}(R) \rangle$, where

$$\mathrm{C}(R) = \left\{ \begin{array}{l} \sim a \leftarrow \sim b, \sim c \\ \sim b \leftarrow \mathbf{f} \\ \sim c \leftarrow \mathbf{f} \end{array} \right\}.$$

▲

The justification frame obtained in the example above is also complementary. This can be generalised to the following proposition.

Proposition 2.3.11. *Let $\mathcal{F}$ be a fact space and let $\mathcal{F}_d$ be a subset of $\mathcal{F}$ closed under $\sim$ that does not contain logical facts. Let $\{\mathcal{F}_1, \mathcal{F}_2\}$ be a partition of $\mathcal{F}_d$ such that $\sim \mathcal{F}_1 = \mathcal{F}_2$. Let R be a subset of $\mathcal{F}_1 \times (2^{\mathcal{F}} \setminus \emptyset)$ such that for each $x \in \mathcal{F}_1$, there is an $(x, A) \in R$. Then $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \cup \mathrm{C}(R) \rangle$ is complementary.*

[6] Denecker et al. (2015) call this *complement closure*, however this is not a closure operator since $\mathrm{CC}(\mathrm{CC}(\mathcal{JF})) = \mathrm{CC}(\mathcal{JF})$ is not always true. Therefore, we gave it a different name.

Proof. We use the characterisation in Proposition 2.3.8. By construction of $\mathrm{C}(R)$, (1) trivially holds for $x \in \mathcal{F}_1$ and (2) for $x \in \mathcal{F}_2$. We first prove (2) for $x \in \mathcal{F}_1$. Take $x \in A \in R \cup \mathrm{C}(R)$; hence $x \in A \in R$ since $x \in \mathcal{F}_1$. We construct a selection function $\mathcal{S}$ of $\sim x$ such that $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$. For any $\sim x \leftarrow C$ we have that $C = \sim \mathrm{Im}(\mathcal{T}_C)$ for some selection function $\mathcal{T}_C$ of x. Now define $\mathcal{S}(C) = \sim \mathcal{T}_C(A) \in \sim A \cap C$. Therefore, $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$, proving (2) for $x \in \mathcal{F}_1$. We now prove (1) for $x \in \mathcal{F}_2$. So take a selection function $\mathcal{S}$ of x in $\mathcal{JF}$. Suppose by contradiction that for all $A \in \mathcal{JF}(\sim x)$ it holds that $A \not\subseteq \sim \mathrm{Im}(\mathcal{S})$; hence there exists a $y_A \in A \setminus \mathrm{Im}(\mathcal{S})$. Define $\mathcal{T} \colon \mathcal{JF}(\sim x) \to \mathcal{F} \colon A \mapsto y_A$, which constitutes a selection function of $\sim x$ in $\mathcal{JF}$, and $\mathrm{Im}(\mathcal{T}) \cap \sim \mathrm{Im}(\mathcal{S}) = \emptyset$. Since $\sim x \in \mathcal{F}_1$, we have that (1) holds for $\sim x$, so there exists a $C \in \mathcal{JF}(x)$ such that $C \subseteq \sim \mathrm{Im}(\mathcal{T})$. This means that $C \cap \mathrm{Im}(\mathcal{S}) = \emptyset$. This, however, contradicts that $\mathcal{S}(C) \in C$; hence (1) holds for $x \in \mathcal{F}_2$. □

The partition $\{\mathcal{F}_1, \mathcal{F}_2\}$ is usually the partition $\{\mathcal{F}_+, \mathcal{F}_-\}$. In general the complementation of a justification frame is not always complementary.

Example 2.3.12. Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ with $\mathcal{F}_d = \{a, \sim a\}$, $\mathcal{F}_o = \mathcal{L}$, and

$$R = \left\{ \begin{array}{l} a \leftarrow \mathbf{t} \\ \sim a \leftarrow \mathbf{t} \end{array} \right\}.$$

The complementation of $\mathcal{JF}$ adds the following rules:

$$\left\{ \begin{array}{l} a \leftarrow \mathbf{f} \\ \sim a \leftarrow \mathbf{f} \end{array} \right\},$$

while the complement of the complementation of $\mathcal{JF}$ has the additional rules

$$\left\{ \begin{array}{l} a \leftarrow \mathbf{t}, \mathbf{f} \\ \sim a \leftarrow \mathbf{t}, \mathbf{f} \end{array} \right\}.$$

These two justification frames are not equivalent; hence the complementation of $\mathcal{JF}$ is not complementary. This is to be expected because the rules for a and $\sim a$ are contradictory. ▲

Luckily, complementation of a complementary justification frame does what you expect.

Proposition 2.3.13. *Let $\mathcal{JF}$ be a complementary justification frame. Then $\mathcal{JF}$ is equivalent with $\mathrm{CC}(\mathcal{JF})$ and $\mathrm{CC}(\mathcal{JF})$ is complementary.*

Proof. If $\mathcal{JF}$ is complementary, then complementation only adds redundant rules by (1) of Proposition 2.3.8. Therefore, $\mathcal{JF}$ is equivalent with $\mathrm{CC}(\mathcal{JF})$.

Take $x \leftarrow A \in \mathrm{C}(\mathrm{CC}(\mathcal{JF}))$. So there is a selection function $\mathcal{S}$ of $\sim x$ in $\mathrm{CC}(\mathcal{JF})$ such that $A = \sim \mathrm{Im}(\mathcal{S})$. We can restrict this selection function to a selection function $\mathcal{T}$ of $\sim x$ in $\mathcal{JF}$. Because $\mathcal{JF}$ is complementary, there is a $C \in \mathcal{JF}(\sim x)$ such that $C \subseteq \sim \mathrm{Im}(\mathcal{T})$. We have that $C \in \mathrm{CC}(\mathcal{JF})$ and thus $C \subseteq \sim \mathrm{Im}(\mathcal{T}) \subseteq \sim \mathrm{Im}(\mathcal{S}) = A$.

Take $x \in A \in \mathrm{CC}(\mathcal{JF})$. Therefore, by equivalence, there is a $B \in \mathcal{JF}(x)$ such that $B \subseteq A$. Since $\mathcal{JF}$ is complementary, there is a $C \in \mathrm{C}(\mathcal{JF})(x)$ by construction of complementation. Therefore, we proved that $\mathrm{C}(\mathrm{CC}(\mathcal{JF}))$ and $\mathrm{CC}(\mathcal{JF})$ are equivalent; hence $\mathrm{CC}(\mathcal{JF})$ is complementary. □

In principle, we can keep taking the complementation of a complementary justification frame. Because this operation is monotone, it halts at some point into a fixed point, i.e. a justification frame that is equal to its complementation.

Lemma 2.3.14. *Any complementary justification frame is equivalent to a complementary justification that is fixed under complementation.*

Proof. Complementation is a monotone operation with respect to the subset order, hence by the Knaster-Tarski fixpoint theorem (Tarski, 1955), there is an ordinal α such that the frame $\mathrm{CC}^{\alpha}(\mathcal{JF})$ is fixed under complementation. By Proposition 2.3.13, this frame is complementary and equivalent to the original frame. □

The supported value stays the same when you evaluate in an equivalent frame. Therefore, to prove the consistency it suffices to look at justification frames that are fixed under complementation. This fact is further exploited in Chapter 4.

In complementary justification systems, cases of a fact and its negation are intrinsically related.

Lemma 2.3.15. *If a justification frame is complementary, then for every rule $x \leftarrow A$ and $\sim x \leftarrow B$, we have $A \cap \sim B \neq \emptyset$.*

Proof. Take $A \in \mathcal{JF}(x)$ and $B \in \mathcal{JF}(\sim x)$. By complementarity, there exists a selection function $\mathcal{S}$ of $\sim x$ such that $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$. Therefore, $\sim \mathcal{S}(B) \in A$. On the other hand, $\mathcal{S}(B) \in B$; hence $\sim \mathcal{S}(B) \in A \cap \sim B$. □

This lemma actually expands to justifications.

Lemma 2.3.16. *Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a complementary justification frame and $x \in \mathcal{F}_d$. If J and K are locally complete justifications in $\mathfrak{J}(x)$ and $\mathfrak{J}(\sim x)$ respectively, then there exists a J-branch $\mathbf{b}$ starting with x such that $\sim \mathbf{b}$ is a K-branch starting in $\sim x$.*

Proof. For simplicity sake, we only prove this for graph-like justifications. The proof for tree-like justifications is similar. We incrementally define J-paths $\mathbf{b}_i$ and K-paths $\mathbf{b}_i^*$ of length i such that $\mathbf{b}_i = {\sim}\mathbf{b}_i^*$. Define $\mathbf{b}_1$ and $\mathbf{b}_1^*$ as the node x and ${\sim}x$ respectively. Now assume that we obtained $\mathbf{b}_i$ and $\mathbf{b}_i^*$. Let y be the end node of $\mathbf{b}_i$. If y is not defined, so is ${\sim}y$ and then $\mathbf{b}_i$ is the desired J-branch. So assume y is defined. We want to find a fact z such that z is a child of y in J and ${\sim}z$ is a child of ${\sim}y$ in K. By using the rules for $y \leftarrow A$ in J and ${\sim}y \leftarrow B$ in K we can use Lemma 2.3.15 to obtain that $A \cap {\sim}B \neq \emptyset$ because $\mathcal{JF}$ is complementary. Choose a z in $A \cap {\sim}B$. Then we construct $\mathbf{b}_{i+1} = \mathbf{b}_i \to z$ and $\mathbf{b}_{i+1}^* = \mathbf{b}_i^* \to {\sim}z$. Our required branch is then the limit of $\mathbf{b}_i$ for i going to infinity. □

This lemma has an important consequence. If x has a good justification J, i.e. $\mathrm{val}(J, x, \mathcal{I}) = \mathbf{t}$, then every justification K for ${\sim}x$ has the negation of a J-branch starting with x in K. To get consistent justification semantics, we want that every justification for ${\sim}x$ has a branch that evaluates to $\mathbf{f}$ under $\mathcal{I}$. A good candidate can be this shared negated branch obtained from Lemma 2.3.16. Therefore, it is not too far-fetched to demand that our branch evaluation has the following property.

Definition 2.3.17. A branch evaluation $\mathcal{B}$ *respects negation* if for all justification frames $\mathcal{JF}$ and all $\mathcal{JF}$-branches $\mathbf{b}$ it holds that $\mathcal{B}({\sim}\mathbf{b}) = {\sim}\mathcal{B}(\mathbf{b})$.

For a branch evaluation that has this property, if x has a $\mathbf{t}$ justification, then every justification for ${\sim}x$ will have value $\mathbf{f}$. This observation solves one direction of the consistency problem.

Theorem 2.3.18. *Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a complementary justification system with $\mathcal{B}$ respecting negation. Then*

$$\mathrm{SV}_{\mathcal{JS}}(x, \mathcal{I}) \leq_t {\sim}\mathrm{SV}_{\mathcal{JS}}({\sim}x, \mathcal{I})$$

for any $x \in \mathcal{F}_d$ and $\mathcal{F}$-interpretation.

Proof. Take x with $\mathrm{SV}_{\mathcal{JS}}(x, \mathcal{I}) = \ell$ for some $\ell \in \mathcal{L}$. This means there is a justification J such that $\mathrm{val}_{\mathcal{JS}}(J, x, \mathcal{I}) = \ell$. Take a justification K for ${\sim}x$. Therefore, by Lemma 2.3.16, there is a J-branch $\mathbf{b}$ starting with x such that ${\sim}\mathbf{b}$ is a K-branch starting in ${\sim}x$. If $\ell = \mathbf{t}$, then $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}$; hence K has a branch that is evaluated to $\mathbf{f}$. Therefore, $\mathrm{val}_{\mathcal{JS}}(K, {\sim}x, \mathcal{I}) = \mathbf{f}$. Since K was taken arbitrarily, we have that $\mathrm{SV}_{\mathcal{JS}}({\sim}x, \mathcal{I}) = \mathbf{f}$.

If $\ell = \mathbf{u}$, we need to prove that $\mathbf{u} \geq_t \mathrm{SV}_{\mathcal{JS}}({\sim}x, \mathcal{I})$. Similarly, every justification K for ${\sim}x$ has a branch ${\sim}\mathbf{b}$ starting in ${\sim}x$ such that $\mathbf{b}$ is a J-branch and $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathbf{u}$. This shows that $\mathcal{I}(\mathcal{B}({\sim}\mathbf{b})) \leq_t \mathbf{u}$. Therefore, $\mathrm{SV}_{\mathcal{JS}}({\sim}x, \mathcal{I}) \leq_t \mathbf{u}$.

For $\ell = \mathbf{f}$, nothing has to be proven. □

We now define a branch evaluation that will serve as a counterexample throughout this thesis.

Definition 2.3.19. The branch evaluation $\mathcal{B}_{\text{ex}}$ is defined as follows:

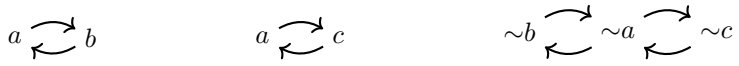
- $\mathcal{B}_{\text{ex}}(x_0 \to x_1 \to \cdots \to x_n) = x_n$;
- $\mathcal{B}_{\text{ex}}(x_0 \to x_1 \to \cdots) = \mathbf{f}$ if $\exists i_0 \in \mathbb{N}\colon \forall i, j > i_0\colon x_i \in \mathcal{F}_+$ and if $x_i = x_j$, then $x_{i+1} = x_{j+1}$;
- $\mathcal{B}_{\text{ex}}(x_0 \to x_1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

Using this branch evaluation we see that the converse of Theorem 2.3.18 is not always true as illustrated by the following example.

Example 2.3.20. Take $\mathcal{F}_d = \{a, \sim a, b, \sim b, c, \sim c\}$, $\mathcal{F}_o = \mathcal{L}$, $\mathcal{F}_+ = \{a, b, c\}$, and $\mathcal{F}_- = \{\sim a, \sim b, \sim c\}$. The set of rules is the complementation of

$$\left\{ \begin{array}{l} a \leftarrow b \\ a \leftarrow c \\ b \leftarrow a \\ c \leftarrow a \end{array} \right\}.$$

Under the branch evaluation $\mathcal{B}_{\text{ex}}$, we have that $\text{SV}_g(a, \mathcal{I}) = \mathbf{f}$, while $\text{SV}_g(\sim a, \mathcal{I}) = \mathbf{u}$ for any interpretation $\mathcal{I}$ of $\mathcal{F}$, i.e., graph-like consistency is not satisfied here. This can easily be checked by noting that the only connected graph-like justifications with a or $\sim a$ as internal node are the following:



Additionally, it can also be seen that $\text{SV}_t(a, \mathcal{I}) = \mathbf{u}$ and $\text{SV}_t(\sim a, \mathcal{I}) = \mathbf{u}$ 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 $\text{val}(J, a, \mathcal{I}) = \mathbf{u}$ 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 $\text{val}(J, \sim a, \mathcal{I}) = \mathbf{u}$ is given by unfolding (see below) of the graph-like justification for $\sim a$ with value $\mathbf{u}$ to a tree-like justification. ▲

2.4 Graph-Reducibility

Example 2.3.20 touches upon our second research question. We have two different kinds of justification. Both induce justification semantics. This raises the question how they are related. When you have a graph-like justification J, we can unroll it into a tree-like justification T such that $B_J(x) = B_T(x)$ for a given internal node x of J.

Definition 2.4.1. Let J be a graph-like justification with an internal fact x. Define the sequence of trees T_i as follows

- T_0 is the rule for x in J.
- T_{i+1} is T_i plus for each leaf y in T_i the rule for y in J.
- For a limit ordinal α, T_α is the union of all T_i with $i < \alpha$

The *unrolling* of J with respect to x is T_β for some ordinal β such that $T_\beta = T_{\beta+1}$. This is well-defined (such a β exists) by the Knaster-Tarski fixpoint theorem (Tarski, 1955) since T_i is contained in T_{i+1} for all ordinal numbers i.

Proposition 2.4.2. *Let J be a graph-like justification with an internal fact x and let T be the unrolling of J with respect to x. Then $B_J(x) = B_T(x)$.*

Proof. Follows directly from the construction of T. □

Since the value of justifications only depends on the branches starting with x we get the following proposition.

Proposition 2.4.3. *Take a locally complete graph-like justification J and an internal node x in J. Then there is a locally complete tree-like justification T such that $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \mathrm{val}_{\mathcal{B}}(T, x, \mathcal{I})$ for all branch evaluations $\mathcal{B}$ and $\mathcal{F}$-interpretations $\mathcal{I}$.*

Proof. The justification T is the unrolling of J with respect to x. □

This has the following consequence.

Corollary 2.4.4. *For any justification system $\mathcal{JS}$, we have for all $x \in \mathcal{F}_d$ and all $\mathcal{F}$-interpretations $\mathcal{I}$ that $\mathrm{SV}^g_{\mathcal{JS}}(x, \mathcal{I}) \leq_t \mathrm{SV}^t_{\mathcal{JS}}(x, \mathcal{I})$.*

Proof. This follows directly from the definition of supported value. □

This result states that tree-like justifications are more powerful than graph-like justifications. In Example 2.3.20, we see that the opposite direction is not always true: $\mathrm{SV}^g(a,\mathcal{I}) = \mathbf{f}$, while $\mathrm{SV}^t(a,\mathcal{I}) = \mathbf{u}$.

Definition 2.4.5. A justification system $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B}\rangle$ is *graph-reducible* if for all $x \in \mathcal{F}_d$ and $\mathcal{F}$-interpretations $\mathcal{I}$ it holds that $\mathrm{SV}^g_{\mathcal{JS}}(x,\mathcal{I}) = \mathrm{SV}^t_{\mathcal{JS}}(x,\mathcal{I})$. A branch evaluation is *graph-reducible* if for all complementary justification frames $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R\rangle$ the system $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B}\rangle$ is graph-reducible.

This raises the following question.

Graph-reducibility

Which justification systems are graph-reducible? In particular, what properties do branch evaluations need to have to be graph-reducible?

This is an important property for real-life applications that use justification theory. Tree-like justifications are almost always infinite, but if they are equivalent to graph-like justifications, they have a finite representation that can be used in computer programs.

2.4.1 Relation between Consistency and Graph-Reducibility

In Example 2.3.20, the branch evaluation $\mathcal{B}_{\mathrm{ex}}$ manages to break both consistency and graph-reducibility. This is not a coincidence as the two properties are related. For a complementary justification frame $\mathcal{JF}$, we have by Theorem 2.3.18 that

$$\mathrm{SV}^t(x,\mathcal{I}) \leq_t \sim \mathrm{SV}^t(\sim x,\mathcal{I}).$$

On the other hand, by Corollary 2.4.4 we have that

$$\mathrm{SV}^g(x,\mathcal{I}) \leq_t \mathrm{SV}^t(x,\mathcal{I}) \quad \text{and} \quad \mathrm{SV}^g(\sim x,\mathcal{I}) \leq_t \mathrm{SV}^t(\sim x,\mathcal{I}).$$

This gives us the following chain of inequalities.

Proposition 2.4.6. *For a complementary justification frame $\mathcal{JF}$ we have the following chain of inequalities for any $x \in \mathcal{F}_d$ and $\mathcal{F}$-interpretation $\mathcal{I}$*

$$\mathrm{SV}^g(x,\mathcal{I}) \leq_t \mathrm{SV}^t(x,\mathcal{I}) \leq_t \sim \mathrm{SV}^t(\sim x,\mathcal{I}) \leq_t \sim \mathrm{SV}^g(\sim x,\mathcal{I}).$$

Proof. This is a combination of Theorem 2.3.18 and Corollary 2.4.4. □

If the left side and the right side are equal (graph-like consistency), then everything else is equal as well.

Corollary 2.4.7. *Let $\mathcal{JS}$ be a justification system with an underlying complementary justification frame. Then $\mathcal{JS}$ is graph-like consistent if and only if it is tree-like consistent and graph-reducible.*

This result has important implications for our research goals: it suffices to prove that a system is graph-like consistent to get all the other properties. Therefore, in the next chapter we only prove graph-like consistency for the main branch evaluations.

The main goal of this thesis (see previous two research questions) is to identify properties of branch evaluations that imply consistency or graph-reducibility. Therefore, it is important to look at some basic types of branch evaluations.

2.5 Branch Evaluation Types

We already have encountered branch evaluations that respect negation. Branch evaluations can map branches to any fact. This fact then has to be interpreted in an interpretation. The simplest type of facts are the logical facts since they always have the same interpretation.

Definition 2.5.1. A branch evaluation is *logical* if every branch is mapped to a logical fact in $\mathcal{L}$.

A contrived example is the following.

Example 2.5.2. Define $\mathcal{B}$ as follows. If a branch is positive, it is mapped to $\mathbf{t}$, if it is negative to $\mathbf{f}$, and to $\mathbf{u}$ otherwise. ▲

The next simplest type of facts are the open facts, as they are determined from the outside.

Definition 2.5.3. A branch evaluation is *parametric* if every branch is mapped to an open fact.

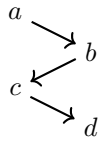
The prototypical examples are $\mathcal{B}_{\mathrm{KK}}$ and $\mathcal{B}_{\mathrm{wf}}$. The branch evaluation $\mathcal{B}_{\mathrm{ex}}$ is also parametric.

Since open facts are determined from the outside it is customary to have the interpretation of the open facts fixed. If this is done, then parametric branch evaluations have only one model.

Proposition 2.5.4. *Let $\mathcal{JF}$ be any justification frame and $\mathcal{B}$ a parametric branch evaluation. Let $\mathcal{I}_o$ be an interpretation of $\mathcal{F}_o$. There is at most one $\mathcal{B}$-model of $\mathcal{JF}$ that extends $\mathcal{I}_o$, i.e., $\mathcal{I}(x) = \mathcal{I}_o(x)$ for $x \in \mathcal{F}_o$. If $\mathcal{JF}$ together with $\mathcal{B}$ form a consistent system, then there is exactly one such model.*

Proof. The value of a justification only depends on the interpretation of the open facts; hence $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}')$ for every two interpretations $\mathcal{I}$ and $\mathcal{I}$ that agree on the open facts. This means that $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}')$. Therefore, if $\mathcal{JF}$ has a $\mathcal{B}$-model, then it is equal to $\mathcal{S}_{\mathcal{JS}}(\mathcal{I})$ for any interpretation $\mathcal{I}$ that extends $\mathcal{I}_o$. If $\mathcal{JF}$ together with $\mathcal{B}$ form a consistent system, then $\mathcal{S}_{\mathcal{JS}}(\mathcal{I})$ is the desired model. □

Consider the following justification.



In a $\mathcal{B}_{\mathrm{sp}}$-model, this is a good justification for a if d is true. Because if d is true, it is a good justification for c; hence c is true. Therefore, this is a good justification for b; hence b is true in a model and thus it is a good justification for a. The information of the branch was actually contained in the tail. This can be extended to infinite branches as well.

Definition 2.5.5. A branch evaluation $\mathcal{B}$ is *transitive* if for all branches $x_0 \to x_1 \to x_2 \to \cdots$, $\mathcal{B}(x_0 \to x_1 \to \cdots) = \mathcal{B}(x_1 \to x_2 \to \cdots)$.

The prototypical examples are $\mathcal{B}_{\mathrm{KK}}$ and $\mathcal{B}_{\mathrm{wf}}$. The branch evaluation $\mathcal{B}_{\mathrm{ex}}$ is also transitive. Transitive branch evaluations will be important in Chapter 6. An interesting observation to make is that in transitive branch evaluations, the information is in the tail of a branch.

Somewhat opposite to this, the information could be contained in the beginning of the branch. To be able to say what this means we have to introduce some concepts. For a finite branch $\mathbf{b}\colon x_0 \to \cdots \to x_n$, denote $\ell(\mathbf{b}) = n$. If $\mathbf{b}$ is infinite, we say $\ell(\mathbf{b}) = \infty$. Two branches $x_0 \to x_1 \to \cdots$ and $y_0 \to y_1 \to \cdots$ are *identical up to* n if for all $0 \leq i \leq n$, we have $x_i = y_i$. A $\mathcal{JF}$-branch is *decided* under $\mathcal{B}$ at $0 < n < \ell(\mathcal{B}) + 1$ if for every $\mathcal{JF}$-branch $\mathbf{b}'$ identical to $\mathbf{b}$ up to n, we have $\mathcal{B}(\mathbf{b}) = \mathcal{B}(\mathbf{b}')$. In this case, we call the $\mathcal{JF}$-path $x_0 \to \cdots \to x_n$ *decided* under $\mathcal{B}$. This means that in order to determine the value of $\mathbf{b}$ under $\mathcal{B}$, we

only need the first $n+1$ elements, so all relevant information is located at the beginning of the branch. This allows us to extend $\mathcal{B}$ to paths that are decided. For example, in $\mathcal{B}_{\text{st}}$, branches with sign switches are mapped to the first sign switch, and thus the path up to and including the sign switch is decided. This also means that every fact after the sign switch does not determine the value of the branch.

Definition 2.5.6. A branch evaluation is *decided* if every branch $\mathbf{b}$ is decided at all n with $0 < n < \ell(\mathbf{b})$.

This is the same as demanding that every branch is decided at 1, see Lemma 2.6.9. The only branch evaluation seen so far that is decided is $\mathcal{B}_{\text{sp}}$.

2.5.1 Dual Branch Evaluations

Non-logical facts are partitioned into positive and negative facts. Branch evaluations can make use of the sign of such facts to introduce asymmetries between positive and negative facts. Switching the roles of positive and negative facts reverses the asymmetry. This type of concept is usually called duality in mathematics.

Definition 2.5.7. Let $\mathcal{F}$ be a fact space. The *dual fact space* $\overline{\mathcal{F}}$ is defined to be $\mathcal{F}$ but with $\mathcal{F}_+$ and $\mathcal{F}_-$ swapped, i.e. $\overline{\mathcal{F}}_+ = \mathcal{F}_-$ and $\overline{\mathcal{F}}_- = \mathcal{F}_+$. For a justification frame $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$, $\overline{\mathcal{JF}}$ is the *dual justification system* $\langle \overline{\mathcal{F}}, \mathcal{F}_d, R \rangle$.

Each $\mathcal{JF}$-branch $\mathbf{b}$ is also a $\overline{\mathcal{JF}}$-branch. To make explicit the justification frame $\mathcal{JF}$ that we regard a branch $\mathbf{b}$ in, we denote $\mathbf{b}$ as $\mathbf{b}_{\mathcal{JF}}$.

Definition 2.5.8. Let $\mathbf{b}$ be a branch evaluation. The *dual branch evaluation* $\overline{\mathcal{B}}$ of $\mathbf{b}$ is defined as follows:

$$\overline{\mathcal{B}}(\mathbf{b}_{\mathcal{JF}}) = \mathcal{B}(\mathbf{b}_{\overline{\mathcal{JF}}})$$

for any justification frame $\mathcal{JF}$ and $\mathcal{JF}$-branch $\mathbf{b}$.

The dual branch evaluation of $\mathcal{B}_{\text{wf}}$ was already defined by Denecker et al. (2015).

Definition 2.5.9. The *co-well-founded* branch evaluation $\mathcal{B}_{\text{cwf}}$ maps finite branches to their last element, branches with a positive tail to $\mathbf{t}$, branches with a negative tail to $\mathbf{f}$, and all other branches to $\mathbf{u}$.

Note, that there aready exist some work on coinduction in logic programming, notably the work by Gupta et al. (2007); Min and Gupta (2009). There, the authors define coinductive SLDNF resolution. Moreover, the recent system s(ASP) for non-ground ASP (Arias et al., 2018) provides justification capabilities (Arias et al., 2020) and relies on a *dual program* transformation very similar to the operation of complementation (Definition 2.3.9).

Let us provide an example with the co-well-founded branch evaluation.

Example 2.5.10. Let S be a set of infinite lists on the set $\{a, b\}$. Such a list is represented using head/tail notation as $[H|T]$, where H is the head and T is the tail. Let $\mathcal{F}$ be the fact space with $\mathcal{F}_+ = \{p(s) \mid s \in S\}$. Define $\ell_{s_1=s_2}$ to be $\mathbf{t}$ if $s_1 = s_2$ and $\mathbf{f}$ otherwise. Let R be the complementation of $\{p([H|T]) \leftarrow p(T), \ell_{H=a} \mid [H|T] \in S\}$. The unique $\mathcal{B}_{\text{cwf}}$-model will assign $\mathbf{t}$ to $p(s)$ iff and only if s is the infinite list consisting solely out of as. The unique $\mathcal{B}_{\text{wf}}$-model, on the other hand, will assign $\mathbf{f}$ to $p(s)$ for all $s \in S$. ▲

Branch evaluations, such as $\mathcal{B}_{\text{sp}}$ and $\mathcal{B}_{\text{KK}}$, that do not depend on the signs are self-dual. Apart from the co-well-founded branch evaluation, we constructed a new semantics: the co-stable branch evaluation $\mathcal{B}_{\text{cst}} := \overline{\mathcal{B}_{\text{st}}}$. It differs with $\mathcal{B}_{\text{st}}$ only on infinite branches that have the same sign throughout. In this case, it is the negation of $\mathcal{B}_{\text{st}}$. Where stable semantics gives a default value of false to atoms, co-stable semantics gives a default value of true to atoms. We have seen that $\mathcal{B}_{\text{cwf}}$ is useful for coinductive reasoning in the same manner that $\mathcal{B}_{\text{wf}}$ is useful for inductive reasoning. Therefore, $\mathcal{B}_{\text{cst}}$ can have be used in a similar role as $\mathcal{B}_{\text{st}}$ for $\mathcal{B}_{\text{wf}}$. Hence, co-stable semantics could be useful in the context of stream reasoning (Beck et al., 2017).

Example 2.5.11. Take the same fact space as in Example 2.5.10. Let R be the complementation of $\{p([H|T]) \leftarrow \ell_{H=a}, {\sim}p(T) \mid [H, T] \in S\}$. A $\mathcal{B}_{\text{cst}}$-model is an interpretation that maps $p(s)$ to $\mathbf{t}$ if and only if s is the alternating infinite list $[a, b, a, b, \ldots]$. ▲

As usual with duality, taking the dual of the dual returns the original one. Similarly, we get the following lemma.

Lemma 2.5.12. *The justification systems $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ and $\langle \overline{\mathcal{F}}, \mathcal{F}_d, R, \overline{\mathcal{B}} \rangle$ have the same support operator.*

Proof. The set of facts are the same in both systems and they have the same set of rules. This means that the set of justifications is equal as well. Moreover, the value of a justification is also the same since branches are evaluated equally. This proves that the supported values are equal and thus that both systems have the same support operator. □

While the proof of this lemma is (almost) trivial, it has important consequences for our two research questions.

Corollary 2.5.13. *If $\mathcal{B}$ is graph-like (respectively tree-like) consistent, then $\overline{\mathcal{B}}$ is graph-like (respectively) consistent. If $\mathcal{B}$ is a graph-reducible, then $\overline{\mathcal{B}}$ is graph-reducible.*

Proof. It follows directly by noting that if $\mathcal{JF}$ is complementary, then $\overline{\mathcal{JF}}$ is complementary as well. □

Most properties that branch evaluations have, are also transferred to their dual branch evaluations. This holds for logical, parametric, transitive, and decided branch evaluations.

2.6 Pasting Justifications

Most of the time, we take a separate justification for each fact. If we have two graph-like justifications that have no nodes in common, we can take the union of the two to get a justification for both facts. If the justifications overlap, this is a more difficult question. In this section, we define what it means to paste justifications together and what the value of the pasting is. The pasting results given in this section are only for graph-like justifications. Similar results can be proven for tree-like justifications, but we do not need them in the rest of the text; hence the restriction to graph-like justification. Therefore, when we write justification, we mean a graph-like justification.

Any graph-like justification J can be seen as a partial function from $\mathcal{F}_d$ to R such that when x is in the domain of J, $J(x)$ is a rule for x. Write $\mathrm{dom}(J)$ for the domain of J. This is just the set of internal nodes of J.

Definition 2.6.1. A justification K is an *extension* of a justification J if K coincides with J on the domain of J seen as partial functions.

An easy way to extend a justification is by taking parts from another justification.

Definition 2.6.2. For any two justifications J and K, the justification $J \uparrow K$ is defined as the justification J on $\mathrm{dom}(J)$ and K on $\mathrm{dom}(K) \setminus \mathrm{dom}(J)$. We say $J \uparrow K$ is the *extension* of J with K.

It is obvious that $J \uparrow K$ is an extension of J. This construction can be used to broaden the definition of the value of a justification to justifications that are not locally complete.

Definition 2.6.3. Let J be a non locally complete justification. Define $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I})$ as the minimum of $\mathrm{val}_{\mathcal{JS}}(K, x, \mathcal{I})$ for every locally complete extension K of J with respect to $\leq_t$.

This definition is compatible with the original definition by the following lemma.

Lemma 2.6.4. *If J is locally complete, then $\mathrm{val}_{\mathcal{JS}}(J, x, \mathcal{I})$ is the minimum of $\mathrm{val}_{\mathcal{JS}}(K, x, \mathcal{I})$ for every locally complete extension K of J with respect to $\leq_t$.*

Proof. Since J is locally complete, any extension of J does not add branches starting in x for internal nodes x in J. The result then follows immediately. □

This implies that for determining the supported value, you can also look at non locally complete justifications. This will be useful in the next chapter.

Going back to extending a justification with another justification, the following lemma serves to be useful.

Lemma 2.6.5. *If J and K are two locally complete justifications, then $J \uparrow K$ is locally complete. Moreover, every $(J \uparrow K)$-branch is either a K-branch or a concatenation of a K-path with a J-branch.*

Proof. A defined leaf of $J \uparrow K$ is a defined leaf of J or K. Hence $J \uparrow K$ is locally complete because J and K are. Take a $(J \uparrow K)$-branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ such that $\mathbf{b}$ is not a K-branch. This means there is a least i such that $x_{i+1} \notin K(x_i)$. This means that $x_{i+1} \in J(x_i)$. And if $x_j \in \mathrm{dom}(J)$, then $x_{j+1} \in \mathrm{dom}(J)$ or x_{j+1} is a leaf of J. Therefore, by induction we proved that $x_i \to x_{i+1} \to \cdots$ is a J-branch. The result then follows since i was taken smallest. □

The branches that have the form $\mathbf{p} \to \mathbf{b}$ with $\mathbf{p}$ a non-empty K-path and $\mathbf{b}$ a J-branch are important to calculate the value of $J \uparrow K$. If the branch evaluation is transitive, then it would have the same value as $\mathbf{b}$. But it could be that this particular branch has its information in the beginning. This amounts to being decided somewhere in $\mathbf{p}$. Similar to the definition of decided, we can say a branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ is *transitive under $\mathcal{B}$ at* $0 \leq n < \ell(\mathbf{b})$ if $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_n \to x_{n+1} \to \cdots)$.

Definition 2.6.6. A branch $\mathbf{b}$ is *splittable* under $\mathcal{B}$ at $0 \leq n < \ell(\mathbf{b}) + 1$ if it is either decided or transitive at n under $\mathcal{B}$.[7] A branch evaluation $\mathcal{B}$ is *splittable* if for every justification frame $\mathcal{JF}$ and $\mathcal{JF}$-branch $\mathbf{b}$, $\mathbf{b}$ is splittable at i under $\mathcal{B}$ for all i with $0 \leq i < \ell(\mathbf{b}) + 1$.

[7] Note that decided at 0 is not defined, so splittable at 0 means transitive at 0.

Intuitively, if a branch is splittable at n, then the information needed to evaluate the branch is either in the tail or start, but not in both. In formula form, $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ is splittable at n if either $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_n \to x_{n+1} \to \cdots)$ or $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_0 \to \cdots \to x_n)$ for the decided path $x_0 \to \cdots \to x_n$. We already have seen a few splittable branch evaluations, such as $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{wf}}$, and $\mathcal{B}_{\mathrm{ex}}$.

Proposition 2.6.7. *Any decided or transitive branch evaluation is splittable.*

Proof. Follows directly from the definitions. □

All important branch evaluations seen so far are splittable.

Proposition 2.6.8. *The branch evaluations $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{st}}$, $\mathcal{B}_{\mathrm{wf}}$, $\mathcal{B}_{\mathrm{ex}}$, and their duals are splittable.*

Proof. By Proposition 2.6.7, it is left to prove that $\mathcal{B}_{\mathrm{st}}$ is splittable. On positive or negative branches, $\mathcal{B}_{\mathrm{st}}$ is everywhere transitive. So it suffices to prove that mixed branches are everywhere splittable. Take a branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ with a first sign switch at i. It is straightforward that $\mathbf{b}$ is decided at $j \geq i$ and transitive at $j < i$; hence $\mathbf{b}$ is everywhere splittable. □

There is an alternative characterisation of splittable that might be more intuitive. But before we get to that, we need some lemmas and terminology.

Lemma 2.6.9. *If a branch $\mathbf{b}$ is decided at n, then it is also decided at m for $n \leq m < \ell(\mathbf{b}) + 1$.*

If a branch is decided, this means there is a first point in which it is decided.

Definition 2.6.10. A branch $\mathbf{b}$ is *first decided* at $0 < i < \ell(\mathbf{b}) + 1$ if $\mathbf{b}$ is decided at $i = 1$ or $\mathbf{b}$ is decided at $i > 1$ and $\mathbf{b}$ is not decided at $i - 1$.

Under $\mathcal{B}_{\mathrm{st}}$, this point is on the first sign switch. For a general branch evaluation, any branch $\mathbf{b} : x_0 \to x_1 \to \cdots$ has a least initial segment $\mathbf{b}^*$ that is decided. Indeed, if $\mathbf{b}$ is first decided at i, then $x_0 \to \cdots \to x_i$ is this segment. If $\mathbf{b}$ is nowhere decided, then $\mathbf{b}$ itself is this segment. Using this segment, we can give an equivalent characterisation of splittable.

Proposition 2.6.11. *A branch evaluation $\mathcal{B}$ is splittable if and only if for every branch $\mathbf{b}$, we have that every final segment of $\mathbf{b}^*$ is decided and has the same evaluation as $\mathbf{b}$.*

Proof. Suppose first that $\mathcal{B}$ is splittable. Take a branch $\mathbf{b}$. If $\mathbf{b}$ is nowhere decided, then $\mathbf{b}^* = \mathbf{b}$ and $\mathbf{b}$ is totally transitive. Thus, every final segment of $\mathbf{b}^*$ has the same evaluation as $\mathbf{b}$. So suppose $\mathbf{b}$ is somewhere decided; hence there is a minimal j so that $\mathbf{b}$ is decided at j. In this case $\mathbf{b}^*$ is equal to $x_0 \to \cdots \to x_j$. Take $0 \leq i < j$. Since $\mathbf{b}$ is totally splittable, $\mathbf{b}$ is transitive at i by minimality of j. Take any path $\mathbf{p}$ so that $x_i \to \cdots \to x_j \to \mathbf{p}$ is a branch. Since $\mathbf{b}$ is decided at j, we get that

$$\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_0 \to \cdots \to x_i \to \cdots \to x_j \to \mathbf{p}).$$

The branch in the right-hand side cannot be decided at i, otherwise $\mathbf{b}$ would also be decided at i. By splittability, this branch is transitive at i; hence we get that

$$\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_i \to \cdots \to x_j \to \mathbf{p}).$$

Therefore, $x_i \to \cdots \to x_j$ is a decided path with the same evaluation as $\mathbf{b}$.

Suppose conversely that for every branch $\mathbf{b}$, it holds that every final segment of $\mathbf{b}^*$ is decided and has the same evaluation as $\mathbf{b}$. Take a branch $\mathbf{b}$. If $\mathbf{b}^*$ is infinite, then every final segment of $\mathbf{b}^*$ is a tail of $\mathbf{b}$. So if every final segment of $\mathbf{b}^*$ has the same evaluation as $\mathbf{b}$, we get that $\mathbf{b}$ is totally transitive, i.e. totally splittable. So suppose $\mathbf{b}^*$ is finite. Let $j = \ell(\mathbf{b}^*)$. Then $\mathbf{b}$ is decided at $j \geq i$ by Lemma 2.6.9. So we only need to prove that $\mathbf{b}$ is transitive at $0 \leq i < j$. We have that

$$\mathcal{B}(x_i \to x_{i+1} \to \cdots) = \mathcal{B}(x_i \to \cdots \to x_j) = \mathcal{B}(\mathbf{b}),$$

which shows that $\mathbf{b}$ is transitive at $0 \leq i < j$. □

As a consequence, the information of a decided branch under a splittable branch evaluation is condensed to a decided path of length 1.

Corollary 2.6.12. *Let $\mathcal{B}$ be a splittable branch evaluation. If a branch $\mathbf{b}$ is first decided at j. Then for all $0 \leq i < j$ the path $x_i \to \cdots \to x_j$ is decided and has the same evaluation as $\mathbf{b}$. This includes the path $x_{j-1} \to x_j$.*

The opposite also holds.

Lemma 2.6.13. *Let $\mathcal{B}$ be a splittable branch evaluation. If a $\mathcal{JF}$-branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ is not decided at $i > 0$, then the path $x_{i-1} \to x_i$ is not decided.*

Proof. There is a $\mathcal{JF}$-branch $\mathbf{b}^*\colon x_0 \to \cdots \to x_i \to \mathbf{p}$ with $\mathbf{p}$ a path, such that $\mathcal{B}(\mathbf{b}) \neq \mathcal{B}(\mathbf{b}^*)$. Since $\mathbf{b}$ is not decided at i, then it is also not decided at $i-1$ and thus it is transitive at $i-1$. It also means that $\mathbf{b}^*$ is not decided at

$i-1$. Therefore if $x_{i-1} \to x_i$ is decided, then $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_{i-1} \to x_i \to \cdots) = \mathcal{B}(x_{i-1} \to x_i) = \mathcal{B}(x_{i-1} \to x_i \to \mathbf{p}) = \mathcal{B}(\mathbf{b}^*)$, which is a contradiction; hence $x_{i-1} \to x_i$ is not decided. □

Even though, the requirement for splittability seems stringent, our counter-example branch evaluation $\mathcal{B}_{\text{ex}}$ is splittable since it is transitive.

The splittability of $\mathcal{B}_{\text{sp}}$, $\mathcal{B}_{\text{KK}}$, $\mathcal{B}_{\text{st}}$, and $\mathcal{B}_{\text{wf}}$ will be used in Chapter 3 to prove that these evaluations are graph-like consistent, and in Chapter 5 to prove the correspondence between justification theory and approximation fixpoint theory.

For the rest of this section, we show that for a $\mathcal{B}$-model there is a single justification that explains the whole model if $\mathcal{B}$ is splittable. This is done by carefully pasting justifications together.

In transitive branch evaluations, if we have two internal nodes x and y in a justification and we can reach y from x in J, then it holds that $\text{val}_{\mathcal{B}}(J, y, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}}(J, x, \mathcal{I})$. This is fairly straightforward to see, any J-branch $\mathbf{b}$ starting with y can be extended to a J-branch $\mathbf{p} \to \mathbf{b}$ starting with x. Then by the transitivity of $\mathcal{B}$, we see that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(\mathbf{p} \to \mathbf{b})) \geq_t \text{val}_{\mathcal{B}}(J, x, \mathcal{I})$. Therefore, $\text{val}_{\mathcal{B}}(J, y, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}}(J, x, \mathcal{I})$.

This does not hold for splittable branch evaluations.

Example 2.6.14. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B}_{\text{sp}} \rangle$ be the justification system with $\mathcal{F}_d = \{a, \sim a, b, \sim b\}$, $\mathcal{F}_o = \mathcal{L} \cup \{c, \sim c\}$, and

$$R = \left\{ \begin{array}{l} a \leftarrow b \\ b \leftarrow c \\ \sim a \leftarrow \sim b \\ \sim b \leftarrow \sim c \end{array} \right\}.$$

Take the interpretation $\mathcal{I}$ with $\mathcal{I}(a) = \mathbf{t}$, $\mathcal{I}(b) = \mathbf{t}$, and $\mathcal{I}(c) = \mathbf{f}$. The single locally complete justification J for a is equal to

$$a \to b \to c.$$

Even though b is reachable from a, $\mathbf{f} = \text{val}_{\mathcal{B}_{\text{sp}}}(J, b, \mathcal{I}) < \text{val}_{\mathcal{B}_{\text{sp}}}(J, a, \mathcal{I}) = \mathbf{t}$. ▲

The values of a and b in this example are different. With splittable branch evaluations we have a similar property when we limit ourselves to equally valued facts.

Definition 2.6.15. Let $\ell \in \mathcal{L}$. A justification J is *ℓ-domain supporting* in $\mathcal{I}$ if $\text{val}_{\mathcal{B}}(J, y, \mathcal{I}) \geq_t \ell$ for all internal nodes y of J.

Not every justification J with $\mathrm{SV}_{\mathcal{B}}(x,\mathcal{I}) = \mathrm{val}_{\mathcal{B}}(J,x,\mathcal{I})$ is ℓ-domain supporting: in Example 2.6.14, the justification is not $\mathbf{t}$-domain supporting. In this example, it is not possible to have such a justification for a that is locally complete. The justification with the node c removed is, however, a $\mathbf{t}$-domain supporting justification. In general, if $\mathcal{B}$ is splittable such ℓ-domain supporting justifications exist for any $x \in \mathcal{F}_d$ with $\ell \leq_t \mathrm{SV}_{\mathcal{B}}(x,\mathcal{I})$.

Lemma 2.6.16. *Let $\mathcal{B}$ be a splittable branch evaluation, $\mathcal{I}$ an interpretation, $\ell \in \mathcal{L}$, and take $x \in \mathcal{F}_d$ with $\mathrm{SV}_{\mathcal{B}}(x,\mathcal{I}) \geq_t \ell$. There is an ℓ-domain supporting justification J with x as internal node.*

Proof. Take a locally complete justification K with $\mathrm{val}_{\mathcal{B}}(K,x,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}}(x,\mathcal{I}) \geq_t \ell$. Let D be the set of internal nodes y in K such that there exists a K-path $x \to \cdots \to y$ that is not decided at y. Note that we have $x \in D$. Define J to be the restriction of K with domain D. The internal nodes of J are exactly the elements of D. Take $y \in D$ and a locally complete extension M of J. Let $\mathbf{b}\colon y \to y_1 \to y_2 \to \cdots$ be an M-branch. We prove that $\mathrm{val}_{\mathcal{B}}(M,y,\mathcal{I}) \geq_t \ell$ by proving that $\mathcal{I}(\mathbf{b}) \geq_t \ell$ for all $\mathbf{b} \in B_M(y)$. Since M is chosen arbitrary, this means that $\mathrm{val}_{\mathcal{B}}(J,y,\mathcal{I}) \geq_t \ell$.

Assume first that $\mathbf{b}$ is a J-branch, then it is also a K-branch. If $y = x$, then $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}_{\mathcal{B}}(K,x,\mathcal{I}) \geq_t \ell$. If $y \neq x$, then there is a K-path $x \to \cdots \to y$ that is not decided at y. Hence, $x \to \cdots \to y \to y_1 \to y_2 \to \cdots$ is transitive at y by splittability of $\mathcal{B}$. Therefore, $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x \to \cdots \to y \to y_1 \to y_2 \to \cdots)$. The latter is a K-branch starting with x, thus $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \ell$.

Therefore, we can assume that $\mathbf{b}$ is not a J-branch; hence there is a least i such that y_i is not an internal node of J. This means that y_i is a leaf of J. Since J is a subgraph of K, this means that y_i is either a leaf or an internal node of K. In the former case, y_i is open, thus that $\mathbf{b}$ is a J-branch, which contradicts our assumption. Hence, y_i is an internal node of K. If $y = x$, then y_i is an internal node of K outside of D. Therefore, $\mathbf{b}$ is decided at y_i, so the evaluation of $\mathbf{b}$ is equal to the evaluation of a K-path starting from x. This means that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(\mathbf{b}'))$ for some K-branch $\mathbf{b}'$ starting at x; hence $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \ell$. If $y \neq x$, then there exists a K-branch $\mathbf{b}'\colon x \to x_1 \to \cdots \to x_m \to y \to y_1 \to y_2 \to \cdots$ that is not decided at y. By splittability of $\mathcal{B}$, $\mathbf{b}'$ is transitive at y: $\mathcal{B}(\mathbf{b}') = \mathcal{B}(\mathbf{b})$. Since $\mathbf{b}'$ is a K-branch starting at x, we have that $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \ell$. □

Note that these justifications do not need to be locally complete.

Example 2.6.17. In the justification system from Example 2.6.14, the $\mathbf{t}$-domain supporting justification for a is equal to $a \to b$. It cannot contain b as internal node because b has supported value equal to $\mathbf{f}$. ▲

The interesting part about these justifications is that they can be pasted together without losing any power with respect to any of their internal nodes as shown in the next lemma.

Lemma 2.6.18. *Let J and K be two ℓ-domain supporting justifications. The justification $J \uparrow K$ is also ℓ-domain supporting.*

Proof. Take a locally complete extension M of $J \uparrow K$ and an internal node y of $J \uparrow K$. We prove that $\mathrm{val}_{\mathcal{B}}(M, y, \mathcal{I}) \geq_t \ell$. If y is an internal node of J, then $\mathrm{val}_{\mathcal{B}}(M, y, \mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}}(J, y, \mathcal{I}) \geq_t \ell$ because M is also a locally complete extension of J. So we can assume that y is not an internal node of J; hence y is an internal node of K. We prove for every M-branch $\mathbf{b}$ starting with y that $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \ell$. If $\mathbf{b}$ is a K-branch, then $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}_{\mathcal{B}}(K, y, \mathcal{I}) \geq_t \ell$. So we can assume that $\mathbf{b}$ is not a K-branch; hence there is a least i such that y_i is not an internal node of K.

Assume first that y_i is an internal node of J. If $\mathbf{b}$ is transitive at i, then $\mathcal{B}(\mathbf{b}) = \mathcal{B}(y_i \to y_{i+1} \to \cdots)$. The latter branch is an M-branch starting with y_i, which is an internal node of J. Hence, by the result above we have that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(y_i \to y_{i+1} \to \cdots) \geq_t \mathrm{val}_{\mathcal{B}}(M, y_i, \mathcal{I}) \geq_t \ell$. Therefore, we can assume that $\mathbf{b}$ is decided at i; hence the evaluation of $\mathbf{b}$ depends on the evaluation of a K-path starting at y. This implies that $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}_{\mathcal{B}}(K, y, \mathcal{I}) \geq_t \ell$.

Finally, we can assume that y_i is not an internal node of J. If for every j, y_j is not an internal node of J, then $\mathbf{b}$ is a $(K \uparrow M)$-branch. Since $K \uparrow M$ is an extension of K, we have that $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}_{\mathcal{B}}(K, y, \mathcal{I}) \geq_t \ell$. Therefore, we can assume there is a least j such that y_j is an internal node of J. If $\mathbf{b}$ is transitive at j, then $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}_{\mathcal{B}}(M, y_j, \mathcal{I}) \geq_t \ell$ by a result we proved above. If $\mathbf{b}$ is decided at j, then $\mathcal{B}(\mathbf{b})$ is decided by a $(K \uparrow M)$-path; hence $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}_{\mathcal{B}}(K \uparrow M, y, \mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}}(K, y, \mathcal{I}) \geq_t \ell$. This concludes that $\mathrm{val}_{\mathcal{B}}(M, y, \mathcal{I}) \geq_t \ell$. Since M is taken arbitrary, this proves that $\mathrm{val}_{\mathcal{B}}(J \uparrow K, y, \mathcal{I}) \geq_t \ell$. □

Using the previous two lemmas, we can construct a justification J that contains all elements with a given supported value as internal nodes. This justification can also contain elements with a higher supported value though. This construction is formalised in the next lemma.

Lemma 2.6.19. *Let $\mathcal{I}$ be an interpretation and $\ell \in \mathcal{L}$. There is a justification J (not necessarily locally complete) such that*

- $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) \geq_t \ell$ *if and only if x is an internal node of J;*
- $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) \geq_t \ell$ *for all internal nodes x of J.*

Proof. Let $X = \{x \in \mathcal{F}_d \mid \mathrm{SV}_{\mathcal{B}}(x,\mathcal{I}) \geq_t \ell\}$. If $X = \emptyset$, take the empty justification. By Lemma 2.6.16, for all $x \in X$, there is a justification J_x with x as internal node such that $\mathrm{val}_{\mathcal{B}}(J_x, y, \mathcal{I}) \geq_t \ell$ for all internal nodes y of J_x. By the well-ordering theorem[8], fix a well-order on $X = \{x_i \mid i \leq \beta\}$. Define

1. $K_0 = J_{x_0}$;
2. $K_{i+1} = K_i \uparrow J_{x_{i+1}}$ for any ordinal $i < \beta$;
3. $K_\alpha = \bigcup_{i<\alpha} K_i$ for any limit ordinal $\alpha \leq \beta$.

This is an increasing sequence of justifications. We prove for all $i \leq \beta$ that $\mathrm{val}_{\mathcal{B}}(K_i, y, \mathcal{I}) \geq_t \ell$ for all internal nodes y of K_i. We do this by transfinite induction. The base case is trivial by Lemma 2.6.16. The successor case follows immediately from Lemma 2.6.18. Take a limit ordinal $\alpha \leq \beta$, a locally complete extension M of K_α, and an internal node y of K_α. There is an $i < \alpha$ so that y is an internal node of K_i. Since $\mathrm{val}_{\mathcal{B}}(K_i, y, \mathcal{I}) \geq_t \ell$ and M is also an extension of K_i, we have that $\mathrm{val}_{\mathcal{B}}(M, y, \mathcal{I}) \geq_t \ell$. Since M is taken arbitrary, we have that $\mathrm{val}_{\mathcal{B}}(K_\alpha, y, \mathcal{I}) \geq_t \ell$.

By construction K_β contains exactly the elements in X as internal nodes, completing the proof, by setting $J = K_\beta$. □

These three justifications can also be pasted together to form one justification that explains everything. Since the internal nodes of these justifications is exactly $\mathcal{F}_d$, the resulting justification is necessarily locally complete.

Theorem 2.6.20. *If $\mathcal{B}$ is splittable, then for every interpretation $\mathcal{I}$ there is a locally complete justification J such that $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$ for all $x \in \mathcal{F}_d$.*

Proof. By Lemma 2.6.19, there are justifications $J_{\mathbf{t}}$, $J_{\mathbf{u}}$, and $J_{\mathbf{f}}$ such that

- J_ℓ is ℓ-domain supporting
- $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) \geq_t \ell$ if and only if y is an internal node of J_ℓ.

Define K to be equal to $J_{\mathbf{t}} \uparrow J_{\mathbf{u}}$, which is an extension of $J_{\mathbf{t}}$. This means that for all $x \in \mathcal{F}_d$ with $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) = \mathbf{t}$, we have that $\mathrm{val}_{\mathcal{B}}(K, x, \mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}}(J_{\mathbf{t}}, x, \mathcal{I}) = \mathbf{t}$. Hence, $\mathrm{val}_{\mathcal{B}}(K, x, \mathcal{I}) = \mathbf{t} = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$. Take $x \in \mathcal{F}_d$ with $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) = \mathbf{u}$. By Lemma 2.6.18, we have that $\mathrm{val}_{\mathcal{B}}(K, x, \mathcal{I}) \geq_t \mathbf{u}$ because $J_{\mathbf{t}}$ and $J_{\mathbf{u}}$ are

[8] The well-ordering theorem is equivalent to the axiom of choice, which we assume in this thesis.

u-domain supporting. By the definition of supported value, we have that $\mathrm{val}_{\mathcal{B}}(K, x, \mathcal{I}) \leq_t \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) = \mathbf{u}$; hence $\mathrm{val}_{\mathcal{B}}(K, x, \mathcal{I}) = \mathbf{u} = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$. Define J to be equal to $K \uparrow J_{\mathbf{f}}$. Take $x \in \mathcal{F}_d$ such that $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) \in \{\mathbf{u}, \mathbf{t}\}$. Since J is an extension of K, we have that $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}}(K, x, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$; hence $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$ by definition of supported value. Take $x \in \mathcal{F}_d$ with $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) = \mathbf{f}$. Therefore $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) = \mathbf{f} = \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$.

It is straightforward that J is locally complete since it contains all $x \in \mathcal{F}_d$ as internal nodes. □

Splittability is necessary as shown in the following example.

Example 2.6.21. Let $\mathcal{B}$ be the branch evaluation that maps $x_0 \to x_1 \to x_2 \to \cdots$ to x_2 if it exists and to $\sim x_1$ otherwise. Let $\mathcal{F}_d = \{a, b, \sim a, \sim b\}$ and $\mathcal{F}_o = \mathcal{L} \cup \{c, d, \sim c, \sim d\}$. Let R be the complementation of

$$\left\{ \begin{array}{l} a \leftarrow b \\ b \leftarrow c \\ b \leftarrow d \end{array} \right\}.$$

Define $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ and let $\mathcal{I}$ be the interpretation such that $\mathcal{I}(c) = \mathbf{t}$ and $\mathcal{I}(d) = \mathbf{f}$. The only connected justifications that contain a and b as internal nodes are

$$\begin{array}{cc} a & a \\ \downarrow & \downarrow \\ b & b \\ \downarrow & \downarrow \\ c & d \end{array}$$

Let $\mathcal{I}$ be an interpretation such that $\mathcal{I}(c) = \mathbf{f}$, and $\mathcal{I}(d) = \mathbf{t}$. Then the left justification supports b, while the right justification supports a. But neither justification supports both a and b. This is because these two justification cannot be put together without ruining the good values of a and b. If $\mathcal{B}$ was splittable, this was possible. However, $\mathcal{B}$ is not splittable: every branch with more than two elements is neither decided or transitive at the second element. ▲

2.7 Applications

2.7.1 Logic Programming

Syntactically, logic programs are very similar to justification frames. There are a few important differences. Justification theory allows for parameters to exist,

while regular logic programming does not. The second distinction is that logic programs do not allow negation in the head and thus only rules for atoms are given. In this section, we only consider normal logic programs.

Definition 2.7.1. A *(propositional) normal logic program* Π over a set of atoms Σ is a set of rules having the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

where a and all b_i and c_j are in Σ. A logic program is called *positive* if no rule has a negative literal in its body.

An *interpretation* I of the set of atoms Σ is a subset of Σ. Intuitively, $a \in I$ means that a is true in I and $a \notin I$ means a is false in I. Therefore, an interpretation is two-valued. For any literal ℓ over Σ, ℓ^I denotes the truth value of ℓ in I, that is, for an $a \in \Sigma$, $a^I = \mathbf{t}$ if $a \in I$ and $a^I = \mathbf{f}$ if $a \notin I$. This can extended to any set B of literals over Σ: $B^I = \mathbf{t}$ if for all $b \in B$, $b^I = \mathbf{t}$; otherwise $B^I = \mathbf{f}$.

With each logic program Π, we associate a *(two-valued) immediate consequence operator* (van Emden and Kowalski, 1976) T_Π that maps a structure I to

$$T_\Pi(I) = \left\{ a \in \Sigma \;\middle|\; \exists r \in \Pi \colon head(r) = p \wedge body(r)^I = \mathbf{t} \right\}.$$

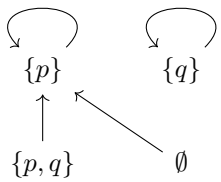
This operator is used to define supported models.

Definition 2.7.2. The *supported models* of Π are the fixpoints of T_Π.

Example 2.7.3. Take the logic program Π over $\Sigma = \{p, q\}$

$$\left\{ \begin{array}{l} p \leftarrow p \\ q \leftarrow \neg p \end{array} \right\}.$$

The operator T_Π does the following on the Σ-interpretations.



Therefore Π has two supported models: $\{p\}$ and $\{q\}$. ▲

A thee-valued (or partial) interpretation I of a set of atoms Σ is a function from Σ to $\mathcal{L} = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. Again, for any literal ℓ over Σ, ℓ^I denotes the truth value of

ℓ in I. This can be extended to sets of literals over Σ by using Kleene's strong three-valued logic (Kleene, 1952). The negation maps **f** to **t**, **t** to **f**, and **u** to **u**. A conjunction is true if all conjuncts are true, it is false if some conjunct is false, and it is unknown otherwise. The disjunction is defined dually to the conjunction. The two-valued immediate consequence operator T_Π defined by van Emden and Kowalski (1976) can be extended to be three-valued as done by Fitting (2002).

Definition 2.7.4. The *three-valued immediate consequence operator* associated with normal logic program Π is the function Φ_Π that maps a three-valued interpretation I to

$$\Phi_\Pi(I)(a) = \max_{\{r \in \Pi \mid head(r) = a\}} body(r)^I,$$

where max is with respect to $\leq_t$.

This operator is monotone with respect to $\leq_p$ and thus by a generalisation of the Knaster-Tarski fixpoint theorem (Cousot and Cousot, 1979), it has a $\leq_p$-least fixpoint, which is called the Kripke-Kleene fixpoint of Π (Fitting, 1985, 1986, 1991).

Definition 2.7.5. The $\leq_p$-least fixpoint of Φ_Π is the *Kripke-Kleene fixpoint* of Π.

Example 2.7.6. Let Π be the logic program over $\Sigma = \{p, q\}$

$$\left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}.$$

The $\leq_p$-least fixpoint of Φ_Π is equal to $\{p^{\mathbf{u}}, q^{\mathbf{u}}\}$. ▲

Definition 2.7.7. A three-valued Σ-interpretation is a *model* of Π if for all $x \leftarrow B \in \Pi$ we have that $x^I \geq_t B^I$.

Proposition 2.7.8 (Przymusinski, 1990, Theorem 3.1)**.** *Every positive logic program* Π *has a unique* $\leq_t$*-least three-valued model.*

The Gelfond-Lifschitz transformation (Gelfond and Lifschitz, 1988) can be extended to three-valued interpretation to define three-valued stable models (sometime called stationary models), see the work by Przymusinski (1990).

Definition 2.7.9. Let Π be a normal logic program and I a three-valued Σ-interpretation. The program Π_I is the logic program obtained from Π where each negative literal $\neg a$ in a body is replaced by $\sim a^I$. The program Π_I is positive and thus has a unique $\leq_t$-least model, let $\Gamma(I)$ be this model. A fixpoint of Γ is a *three-valued stable model* of Π.

Usually, only two-valued stable models are considered, so when we say stable model we mean a two-valued stable model. For our purposes, well-founded semantics is defined using three-valued stable models.

Definition 2.7.10. The *well-founded model* of Π is the $\leq_p$-least three-valued stable model of Π.

Now we come to the point that we construct a justification frame associated to a normal logic program. Negative literals do not appear in the head of a rule in a normal logic program. In Section 2.3.2, we have seen that by using complementation we get rules for negative literals. Another discrepancy between justification theory is that in normal logic programs every atom has a rule, either explicitly, or implicitly. If no rules are supplied for an atom a, then we assume an implicit rule $a \leftarrow \mathbf{f}$. If the body of a rule for the head a is empty, this corresponds to a rule $a \leftarrow \mathbf{t}$. Taking this into account, we have enough to define a justification frame that represents the normal logic program at hand.

Definition 2.7.11. Let Π be a propositional logic program over a set of atoms Σ. The fact space $\mathcal{F}_\Pi$ associated with Π is the set of all literals over Σ plus the set $\mathcal{L}$ with $\mathcal{F}_+ = \Sigma$ and the negation as usual. The justification frame $\mathcal{JF}_\Pi$ associated with Π is equal to $\langle \mathcal{F}_\Pi, \mathcal{F}_\Pi \setminus \mathcal{L}, R \rangle$, where R is the complementation of the rules in Π after potentially adding rules of the form $x \leftarrow \mathbf{f}$ when there are no rules with x as the head and replacing rules for x with empty body with the rule $x \leftarrow \mathbf{t}$.

Similar things can be done for non-propositional logic programs by first grounding since justification theory can handle infinite fact spaces, infinitely many rules for a certain head, and infinite bodies.

Justification theory always works with three-valued interpretations. If $\mathcal{I}$ is a $\mathcal{F}_\Pi$-interpretation, then it corresponds to three-valued interpretation I of Σ by stating $I(a) = \mathcal{I}(a)$ for all $a \in \Sigma$. If $\mathcal{I}$ only maps $\mathbf{u}$ to $\mathbf{u}$, then $\mathcal{I}$ is essentially two-valued and corresponds to a two-valued Σ-interpretation as follows: $I = \{a \in \Sigma \mid \mathcal{I}(a) = \mathbf{t}\}$. This correspondence is implicitly used in the following theorem that shows that the main logic programming semantics can be captured with justification theory.

Theorem 2.7.12. *Let Π be a propositional logic program, then we have the following correspondences.*

- *A two-valued interpretation $\mathcal{I}$ is a $\mathcal{B}_{\mathrm{sp}}$-model of $\mathcal{JF}_\Pi$ if and only if I is a supported model of Π.*
- *An interpretation $\mathcal{I}$ is the unique $\mathcal{B}_{\mathrm{KK}}$-model of $\mathcal{JF}_\Pi$ if and only if I is the Kripke-Kleene model of Π.*

- *A two-valued interpretation $\mathcal{I}$ is a $\mathcal{B}_{\text{st}}$-model of $\mathcal{JF}_\Pi$ if and only if I is a stable model of* Π.
- *An interpretation $\mathcal{I}$ is the unique $\mathcal{B}_{\text{wf}}$-model of $\mathcal{JF}_\Pi$ if and only if I is the well-founded model of* Π.

Proof. Follows from the correspondence from Chapter 5 and the characterisation of the various logic programming semantics with AFT as done for example by Denecker et al. (2012). □

2.7.2 Differences between answer-set programming

In contrast to ASP, our framework does not allow disjunction in the head, but allows parameters. On top of that, the intuition behind the negation is different. In justification theory, negative literals are treated as completely symmetric to positive ones, so that double negation $\sim\sim p$ is just equal to p. This is because, $\sim$ is classical three-valued logic. This is rooted in the idea that justifications capture construction processes (or at least generalisations thereof). But there are other logics such as Gödel's G3 logic (Gödel, 1932), where this does not hold. In particular, G3 is equivalent to the intermediate logic of Here-and-There (Bochman, 2012; Cabalar and Ferraris, 2007; Lifschitz et al., 1999, 2001), used by a part of the ASP community both for foundations and as a reference for popular tools such as Clingo. In G3, double negation $\sim\sim p$ has quite a different meaning than p. There, the rule operator $\leftarrow$ becomes regular implication in the logic of Here-and-There, while negation is classical. In justification theory, negation is classical and the rule operator $\leftarrow$ is not classical implication but some logical connective intimately tied to the construction processes that justifications represent.

2.7.3 Abstract Argumentation

Argumentation Frameworks (AFs) (Dung, 1995) are a simple formalism for representing arguments and attacks between these arguments. In AFs, the contents of the arguments do not matter and are thus abstracted away. This section is inspired by the work of Strass (2013).

Definition 2.7.13. An *argumentation framework* $\mathcal{A}$ is a pair (A, X) where A is a set of atomic arguments and $X \subseteq A \times A$ is a relation representing attacks on arguments, i.e. if $(a, b) \in X$, then the argument a attacks b.

The goal of argumentation frameworks is to determine a set of arguments that are accepted to some standard (cf. a model in justification theory). This acceptance depends on which semantics are used. For example, if a set of arguments is consistent (no argument in the set attack each other) and if it can defend against any argument from outside, then we can consider it to be a good set of arguments. It is reasonable to assume that good sets have no internal conflicts.

Definition 2.7.14. A set $S \subseteq A$ is *conflict-free* in $\mathcal{A}$ if there are no $a, b \in S$ with $(a, b) \in X$.

For the rest of the discussion it is useful to define the following two sets.

Definition 2.7.15. For $a \in A$, $\text{Attackers}_{\mathcal{A}}(a)$ is the set $\{b \in A \mid (b, a) \in X\}$. We define $\text{Attacked}_{\mathcal{A}}(S) = \{a \in A \mid \exists b \in S \colon (b, a) \in X\}$. A set S *defends* a in $\mathcal{A}$ if $\text{Attackers}_{\mathcal{A}}(a) \subseteq \text{Attacked}_{\mathcal{A}}(S)$.

Definition 2.7.16. A set S is a *stable extension* of $\mathcal{A}$ if $S = A \setminus \text{Attacked}_{\mathcal{A}}(S)$.

All semantics defined by Dung (1995) can be captured using two operators. The first one is the *characteristic function* $C_{\mathcal{A}}$ for an argumentation framework $\mathcal{A} = (A, X)$, which is the function:

$$2^A \to 2^A \colon S \mapsto \{a \in A \mid S \text{ defends } a \text{ in } \mathcal{A}\}.$$

Fixpoints are of special interest.

Definition 2.7.17. A *complete extension* of $\mathcal{A}$ is a conflict-free set S such that $S = C_{\mathcal{A}}(S)$. A *preferred extension* of $\mathcal{A}$ is a $\subseteq$-maximal complete extension of $\mathcal{A}$.

The characteristic function is monotone with respect to $\subseteq$; therefore by the Knaster-Tarski fixpoint theorem (Tarski, 1955), it has a $\subseteq$-least fixpoint.

Definition 2.7.18. The *grounded extension* of $\mathcal{A}$ is the $\subseteq$-least fixpoint of $C_{\mathcal{A}}$.

If $S \subseteq C_{\mathcal{A}}(S)$, then every $a \in S$ is defended by S.

Definition 2.7.19. A conflict-free set S is *admissible* if $S \subseteq C_{\mathcal{A}}(S)$.

The second operator is the *unattacked function* $U_{\mathcal{A}}$, which is the function

$$2^A \to 2^A \colon S \mapsto A \setminus \text{Attacked}_{\mathcal{A}}(S).$$

Fixpoints of this function are exactly the stable extensions of $\mathcal{A}$. Strass (2013) defined a single function that captures both the operator $C_{\mathcal{A}}$ and $U_{\mathcal{A}}$.

Definition 2.7.20. Define $\mathscr{F}_{\mathcal{A}} \colon 2^A \times 2^A \to 2^A \times 2^A \colon (S,T) \mapsto (U_{\mathcal{A}}(T), U_{\mathcal{A}}(S))$

Similar to $\leq_p$ on interpretations, we can define $\leq_p$ on $2^A \times 2^A$.[9] We say that $(S_1, S_2) \leq_p (T_1, T_2)$ if $S_1 \subseteq T_1$ and $S_2 \supseteq T_2$. The operator $\mathscr{F}_{\mathcal{A}}$ can be used to characterise every type of model we have seen so far (Strass, 2013).

Proposition 2.7.21. *Let $\mathcal{A} = (A, X)$, then $S \subseteq A$ is*

- *a stable extension of $\mathcal{A}$ if and only if (S,S) is a fixpoint of $\mathscr{F}_{\mathcal{A}}$.*
- *a complete extension of $\mathcal{A}$ if and only if there exists a $T \subseteq A$ so that (S,T) is a fixpoint of $\mathscr{F}_{\mathcal{A}}$.*
- *a grounded extension of $\mathcal{A}$ if and only if there exists a $T \subseteq A$ so that (S,T) is $\leq_p$-least fixpoint of $\mathscr{F}_{\mathcal{A}}$.*
- *a preferred extension of $\mathcal{A}$ if and only if there exists a $T \subseteq A$ so that (S,T) is fixpoint of $\mathscr{F}_{\mathcal{A}}$ where S is $\subseteq$-maximal.*
- *admissible if and only if $(S, U_{\mathcal{A}}(S)) \leq_p \mathscr{F}_{\mathcal{A}}(S, U_{\mathcal{A}}(S))$*

Correspondence

Each argument b that attacks a is a reason why a does not hold. Therefore, we could represent this by a rule $\sim a \leftarrow b$. This means that only negative elements occur in the head of rules. Taking the complement of these rules, we get rules of the form $a \leftarrow \sim\text{Attackers}_{\mathcal{A}}(a)$. Intuitively, such rules state that a holds if every argument b attacking a does not hold.

Definition 2.7.22. Let $\mathcal{A} = (A, X)$ be an argumentation framework. The fact space $\mathcal{F}_{\mathcal{A}}$ corresponding to $\mathcal{A}$ is the set $A \cup \sim A \cup \mathcal{L}$, with $(\mathcal{F}_{\mathcal{A}})_+ = A$. The justification frame $\mathcal{JF}_{\mathcal{A}}$ is the frame $\langle \mathcal{F}_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}} \setminus \mathcal{L}, R \rangle$ where R is the complementation of the set

$$\{\sim a \leftarrow b \mid (b,a) \in X\} \cup \{\sim a \leftarrow \mathbf{f} \mid \text{Attackers}_{\mathcal{A}}(a) = \emptyset\}.$$

This justification frame has some extra properties. Positive facts have a single rule, where each body element is negative (or $\mathbf{t}$) On the other hand, every rule for a negative element has a body with a single element that is positive (or $\mathbf{f}$). This means that every branch of a justification alternates between positive and negative elements (possibly with $\mathbf{f}$ or $\mathbf{t}$ as its last element). Therefore, $\mathcal{B}_{\text{sp}}$

[9] This similar becomes apparent in Chapter 5, where we decompose interpretations as pairs of sets of positive facts.

and $\mathcal{B}_{\text{st}}$ are equal. Similarly, $\mathcal{B}_{\text{KK}}$ and $\mathcal{B}_{\text{wf}}$ are equal since they only differ on non-mixed loops and every infinite branch is mixed.

The correspondence between justification theory and AF is similar to that of Proposition 2.7.21.

Theorem 2.7.23. *Let $\mathcal{A} = (A, X)$ be an argumentation framework. A subset S of A is*

- *a stable extension of $\mathcal{A}$ if and only if there is a two-valued $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}$ of $\mathcal{JF}_{\mathcal{A}}$ with $S = \{x \in \mathcal{F}_+ \mid \mathcal{I}(x) = \mathbf{t}\}$;*
- *a complete extension of $\mathcal{A}$ if and only if there is a $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}$ of $\mathcal{JF}_{\mathcal{A}}$ with $S = \{x \in \mathcal{F}_+ \mid \mathcal{I}(x) = \mathbf{t}\}$;*
- *the grounded extension of $\mathcal{A}$ if and only if there exists $\mathcal{B}_{\text{KK}}$-model $\mathcal{I}$ of $\mathcal{JF}_{\mathcal{A}}$ with $S = \{x \in \mathcal{F}_+ \mid \mathcal{I}(x) = \mathbf{t}\}$;*
- *a preferred extension of $\mathcal{A}$ if and only if there exists a $\leq_p$-maximal $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}$ of $\mathcal{JF}_{\mathcal{A}}$ with $S = \{x \in \mathcal{F}_+ \mid \mathcal{I}(x) = \mathbf{t}\}$;*
- *admissible if and only if for the interpretation $\mathcal{I}$ defined by $S = \{x \in \mathcal{F}_+ \mid \mathcal{I}(x) = \mathbf{t}\}$ and $U_{\mathcal{A}}(S) = \{x \in \mathcal{F}_+ \mid \mathcal{I}(x) \geq_t \mathbf{u}\}$ it holds that $\mathcal{I} \leq_p \mathcal{S}^{\mathcal{B}_{\text{sp}}}_{\mathcal{JF}_{\mathcal{A}}}(\mathcal{I})$.*

Proof. We first prove that $\mathscr{F}_{\mathcal{A}}$ is equal to the approximator $A_{\mathcal{JF}_{\mathcal{A}}}$ associated to the justification $\mathcal{JF}_{\mathcal{A}}$ (see Chapter 5 for definitions). Take $a \in A$. Assume first that $\text{Attackers}_{\mathcal{A}}(a) \neq \emptyset$. Then $a \in A_{\mathcal{JF}_{\mathcal{A}}}(I_1, I_2)_1$ iff for all $b \in \text{Attackers}_{\mathcal{A}}(a)$ we have $\mathcal{I}(b) = \mathbf{f}$ iff $\text{Attackers}_{\mathcal{A}}(a) \cap I_2 = \emptyset$ iff $a \in U_{\mathcal{A}}(I_2)$ iff $a \in \mathscr{F}_{\mathcal{A}}(I_1, I_2)_1$. Similarly, $a \in A_{\mathcal{JF}_{\mathcal{A}}}(I_1, I_2)_2$ iff for all $b \in \text{Attackers}_{\mathcal{A}}(a)$ we have $\mathcal{I}(a) \leq_t \mathbf{u}$ iff $\text{Attackers}_{\mathcal{A}}(a) \cap I_1 = \emptyset$ iff $a \in U_{\mathcal{A}}(I_1)$ iff $a \in \mathscr{F}_{\mathcal{A}}(I_1, I_2)$. If $\text{Attackers}_{\mathcal{A}}(a) = \emptyset$, then $\mathcal{JF}_{\mathcal{A}}$ contains the rule $a \leftarrow \mathbf{t}$. Then $a \in A_{\mathcal{JF}_{\mathcal{A}}}(I_1, I_2)_1$ and $a \in U_{\mathcal{A}}(I_2)$, hence $a \in \mathscr{F}_{\mathcal{A}}(I_1, I_2)_1$. This proves that $\mathscr{F}_{\mathcal{A}} = A_{\mathcal{JF}_{\mathcal{A}}}$. The rest of the proof then follows from the correspondence in Chapter 5 and Proposition 2.7.21. □

2.8 Conclusion

In this chapter, we explained how justification semantics work and provided precise definitions to work with. Then we discussed consistency of justification systems by noting that not all justification semantics are sound. The consistency problem we posed in that section will be the driving force for the following chapters, especially Chapters 3 and 4. We examined how rules for a fact and its

negation are related and worked out some details surrounding complementarity. In the next section our second research question popped up: when are graph-like and tree-like justifications equivalent? We were able to reduce this back to the consistency problem. A few basic branch evaluation types are given as well, while we will encounter a few more in coming chapters. We took a brief look at dual branch evaluations and showed a few results for pasting justifications together. This chapter is closed off by giving two well-known formalisms expressed through justification theory.

In the following chapter, we will take a deeper look into the consistency problem for the main branch evaluations $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{st}}$, and $\mathcal{B}_{\mathrm{wf}}$.

Chapter 3

Basic Properties of Justification Theory

3.1 Introduction

Most applications known so far of justification theory use the four main branch evaluations $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{st}}$, $\mathcal{B}_{\mathrm{wf}}$, and their duals. In this chapter, we start by proving the consistency of these branch evaluations. This consistency is then used to relate models of these evaluations with each other. When justification theory was first formalised for graph-like justification by Denecker et al. (2015), variants for $\mathcal{B}_{\mathrm{sp}}$ and $\mathcal{B}_{\mathrm{st}}$ were given instead. We prove that they give rise to the same models as our definitions. Since graph-like consistency implies tree-like consistency, it suffices to focus ourselves on graph-like consistency. Every justification in this chapter is a graph-like justification.

The text of this chapter is mainly from (Marynissen et al., 2018a,b) but updated to fit in the current formalisation. The section on relations between different models and branch evaluations is taken from (Marynissen et al., 2021).

3.2 Consistency Revisited

We have seen in the previous chapter that for one side of the consistency: $\mathrm{SV}(x, \mathcal{I}) \leq_t \sim\mathrm{SV}(\sim x, \mathcal{I})$, we need complementarity. Similarly, to prove the other direction we will demand complementarity as well. One special kind of

complementary justification frames are the ones closed under complementation. By Lemma 2.3.14, it suffices to consider justification frames that are closed under complementation. Let us first give a few stronger properties that do not always hold in every complementary justification frame. The following proposition is a stronger result than Lemma 2.3.15.

Proposition 3.2.1. *If a justification frame $\mathcal{JF}$ is fixed under complementation, then for every rule $x \leftarrow A$ and $y \in A$, there is a rule $\sim x \leftarrow B$ such that $\sim y \in B$.*

Proof. Take a selection function $\mathcal{S}$ of x so that $\mathcal{S}(A) = y$. Then $B \coloneqq \sim \mathrm{Im}(\mathcal{S})$ is an element in $\mathcal{JF}(\sim x)$ so that $\sim y \in B$ since $\mathcal{JF}$ is closed under complementation. □

This has as a consequence that if $x \to y$ is a path in some justification, that $\sim x \to \sim y$ is also path in some justification. Branch evaluations can also be applied to branches that cannot occur in a justification as they are independent of the given justification frame. This is why the following lemma does not have the fixed under complementation restriction.

Lemma 3.2.2. *Let $\mathcal{B}$ be a branch evaluation that respects negation and $\mathcal{JF}$ a complementary justification frame. If a $\mathcal{JF}$-path $\mathbf{p}$ is decided, then $\sim p$ is decided as well. Moreover, $\mathcal{B}(\sim \mathbf{p}) = \sim \mathcal{B}(\mathbf{p})$.*

Proof. Take any $\mathcal{JF}$-path $\sim \mathbf{p} \to \mathbf{b}$. Then $\mathcal{B}(\sim \mathbf{p} \to \mathbf{b}) = \sim \mathcal{B}(\mathbf{p} \to \sim \mathbf{b}) = \sim \mathcal{B}(\mathbf{p})$ since $\mathbf{p}$ is decided. This means that $\sim \mathbf{p}$ is decided and $\mathcal{B}(\sim \mathbf{p}) = \sim \mathcal{B}(\mathbf{p})$. □

The one direction for the consistency we have not yet proven is $\mathrm{SV}(x, \mathcal{I}) \geq_t \sim \mathrm{SV}(\sim x, \mathcal{I})$. This is the same as $\sim \mathrm{SV}(x, \mathcal{I}) \leq_t \mathrm{SV}(\sim x, \mathcal{I})$. If $\mathrm{SV}(x, \mathcal{I}) = \mathbf{f}$, i.e. there is no good justification for x, then $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{t}$, i.e. there is a good justification for $\sim x$. There are similar consequences if $\mathrm{SV}(x, \mathcal{I}) \neq \mathbf{f}$. However, the condition with $\mathrm{SV}(x, \mathcal{I}) = \mathbf{f}$ is enough according the following proposition (which is the contraposition of that statement).

Proposition 3.2.3. *Let $\mathcal{JF}$ be a complementary justification frame. If for all $x \in \mathcal{F}_d$ it holds that $\mathrm{SV}(\sim x, \mathcal{I}) \leq_t \mathbf{u}$ implies that $\mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}$, then we have graph-like consistency.*

Proof. By Theorem 2.3.18, it suffices to prove for all $x \in \mathcal{F}_d$ that $\sim \mathrm{SV}(\sim x, \mathcal{I}) \leq_t \mathrm{SV}(x, \mathcal{I})$, or that $\mathrm{SV}(\sim x, \mathcal{I}) \geq_t \sim \mathrm{SV}(x, \mathcal{I})$.

We make a case distinction on the value of $\mathrm{SV}(\sim x, \mathcal{I})$. If $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{t}$, then we have a vacuous claim.

If $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{u}$, then we have on the one hand that $\mathrm{SV}(\sim x, \mathcal{I}) \leq_t \mathbf{u}$; hence $\mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}$. On the other hand, $\mathbf{u} = \sim \mathrm{SV}(\sim x, \mathcal{I}) \geq_t \mathrm{SV}(x, \mathcal{I})$ by Theorem 2.3.18. This proves that $\mathrm{SV}(x, \mathcal{I}) = \mathbf{u}$.

If $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{f}$, then we have that $\mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}$ since $\mathrm{SV}(\sim x, \mathcal{I}) \leq_t \mathbf{u}$. If $\mathrm{SV}(x, \mathcal{I}) = \mathbf{u}$, then $\mathrm{SV}(\sim x, \mathcal{I}) \geq_t \mathbf{u}$, which is a contradiction. Therefore, $\mathrm{SV}(x, \mathcal{I}) = \mathbf{t}$, completing the proof. □

This characterisation shows us that we should find consequences of $\mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}$. The following result shows that there is a rule $x \leftarrow A$ with some properties. In the results after this, we will use this rule to construct a good justification for x.

Lemma 3.2.4. *Let $\mathcal{B}$ be a splittable branch evaluation that respects negation. Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a complementary justification frame. Take $x \in \mathcal{F}_d$. If $\mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}$, then there is an $A \in \mathcal{JF}(x)$ such that for all $y \in A$ at least one of the following holds:*

- *the path $x \to y$ is decided and $\mathcal{I}(\mathcal{B}(x \to y)) \geq_t \mathbf{u}$;*
- *the path $x \to y$ is not decided, and $\mathrm{SV}(y, \mathcal{I}) \geq_t \mathbf{u}$ or $\mathrm{SV}(\sim y, \mathcal{I}) \neq \mathbf{t}$.*

Proof. Define $\mathfrak{m}_{\mathcal{I}}$ to be the subset of $\mathcal{F}_d$ such that $x \in \mathfrak{m}_{\mathcal{I}}$ if and only if for all $A \in \mathcal{JF}(x)$, there is a $y \in A$ such that both the following holds:

- if the path $x \to y$ is decided, then $\mathcal{I}(\mathcal{B}(x \to y)) = \mathbf{f}$;
- if the path $x \to y$ is not decided, then $\mathrm{SV}(y, \mathcal{I}) = \mathbf{f}$ and $\mathrm{SV}(\sim y, \mathcal{I}) = \mathbf{t}$.

The lemma states that if $\mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}$, then $x \notin \mathfrak{m}_{\mathcal{I}}$. We prove the contraposition: if $x \in \mathfrak{m}_{\mathcal{I}}$, then $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{t}$. Let $X = \mathfrak{m}_{\mathcal{I}} \setminus \{x \mid \mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{t}\}$. Suppose by contradiction that $X \neq \emptyset$. For any $x \in \mathfrak{m}_{\mathcal{I}}$, there is a selection function $\mathcal{T}_x$ that maps $A \in \mathcal{JF}(x)$ to a $y \in A$ as provided in the definition of $\mathfrak{m}_{\mathcal{I}}$.

Take a $\mathbf{t}$-domain supporting justification J^* with the internal nodes of J^* exactly the elements y with $\mathrm{SV}(y, \mathcal{I}) = \mathbf{t}$. Take J a locally complete extension of J^* such that $J(\sim x) \subseteq \sim \mathrm{Im}(\mathcal{T}_x)$ for all $x \in X$. This is possible since $\mathcal{JF}$ is complementary and because no element of X is an internal node of J^*. Take any $x_0 \in X$. Take a J-branch $\mathbf{b} \colon \sim x_0 \to \sim x_1 \to \cdots$. We prove that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}$.

Assume that $\mathbf{b}$ does not lie completely in $\sim X$. Let $\sim x_i$ be the first such that $x_i \notin X$. Note that $i > 0$. Suppose $\mathbf{b}$ is first decided at $j \leq i$; hence by Corollary 2.6.12 the path $\sim x_{j-1} \to \sim x_j$ is decided. By Lemma 3.2.2, the path

$x_{j-1} \to x_j$ is also decided and $\mathcal{B}(\sim x_{j-1} \to \sim x_j) = \sim\mathcal{B}(x_{j-1} \to x_j)$. Since $x_{j-1} \in X$, it holds that $x_j \in \mathrm{Im}(\mathcal{T}_{x_{j-1}})$. Therefore, by construction of $\mathcal{T}_{x_{j-1}}$ it holds that $\mathcal{I}(\mathcal{B}(x_{j-1} \to x_j)) = \mathbf{f}$; hence $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(\sim x_{j-1} \to \sim x_j)) = \mathbf{t}$. This means we can assume that $\mathbf{b}$ is not decided at j for $j \leq i$. Therefore, $\sim x_{i-1} \to \sim x_i$ is not decided by Lemma 2.6.13. Moreover, $x_{i-1} \to x_i$ is not decided by Lemma 3.2.2. Since $x_{i-1} \in \mathfrak{m}_{\mathcal{I}}$, we get that $\mathrm{SV}(x_i, \mathcal{I}) = \mathbf{f}$ and $\mathrm{SV}(\sim x_i, \mathcal{I}) = \mathbf{t}$. By splittability, $\mathbf{b}$ is transitive at i. Therefore, $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(\sim x_i \to \sim x_{i+1} \to \cdots)) = \mathbf{t}$ since $\mathrm{SV}(\sim x_i, \mathcal{I}) = \mathbf{t}$ and $\mathrm{val}(J, \sim x_i, \mathcal{I}) = \mathrm{val}(J^*, \sim x_i, \mathcal{I}) = \mathbf{t}$.

Therefore, we can assume that $\mathbf{b}$ lies completely in $\sim X$. If $\mathbf{b}$ is first decided at some k, then the path $\sim x_{k-1} \to \sim x_k$ is decided as well. This means that $\mathcal{I}(\mathbf{b}(x_{k-1} \to x_k)) = \mathbf{f}$ and that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}$. So we can assume that $\mathbf{b}$ is nowhere decided. This means that the path $x_{k-1} \to x_k$ is not decided; hence $\mathrm{SV}(x_k, \mathcal{I}) = \mathbf{f}$ and $\mathrm{SV}(\sim x_k, \mathcal{I}) = \mathbf{t}$. This contradicts that $x_k \in X$.

This proves that $\mathrm{val}(J, \sim x_0, \mathcal{I}) = \mathbf{t}$; hence $\mathrm{SV}(\sim x_0, \mathcal{I}) = \mathbf{t}$. This is a contradiction; hence $X = \emptyset$. □

If $\mathcal{B}$ is graph-like consistent, then for each defined x with $\mathrm{SV}(x, \mathcal{I}) = \mathbf{f}$, we have that $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{t}$. A somewhat weird way to state this is the following.

$$\mathcal{F}_d = \{x \in \mathcal{F}_d \mid \mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}\} \cup \sim \{x \in \mathcal{F}_d \mid \mathrm{SV}(x, \mathcal{I}) = \mathbf{t}\}.$$

The following lemma exploits this property by constructing a weird branch when this equality does not hold.

Lemma 3.2.5. *Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system and $\mathcal{B}$ splittable. Let $X = \mathcal{F}_d \setminus (\{x \in \mathcal{F}_d \mid \mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}\} \cup \sim \{x \in \mathcal{F}_d \mid \mathrm{SV}(x, \mathcal{I}) = \mathbf{t}\})$. If X is non-empty, there is a locally complete justification J such $\mathrm{val}(J, x, \mathcal{I}) \geq_t \mathbf{u}$ for all x with $\mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}$ so that for any $x \in X$ and for any J-branch $\mathbf{b}$ starting with x such that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{f}$, it holds that $\mathbf{b}$ is nowhere decided and all elements of $\mathbf{b}$ are in X. Moreover, such a branch always exists for any such x internal in J.*

Proof. By Lemma 3.2.4, for every $x \in \mathcal{F}_d$ with $\mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}$ there is an $A \in \mathcal{JF}(x)$ such that for all $y \in A$, we have that either

- $x \to y$ is decided and $\mathcal{I}(\mathcal{B}(x \to y)) \geq_t \mathbf{u}$ or
- $x \to y$ is not decided, and $\mathrm{SV}(y, \mathcal{I}) \geq_t \mathbf{u}$ or $\mathrm{SV}(\sim y, \mathcal{I}) \neq \mathbf{t}$.

We call $\mathcal{A}_x$ the set of these rule bodies.

Take an $\mathbf{u}$-domain supporting justification J^* such that its internal nodes are exactly the elements y with $\mathrm{SV}(y,\mathcal{I}) \geq_t \mathbf{u}$. This is possibly due to Lemma 2.6.19. Take any locally complete extension J of J^* such that $J(x) \in \mathcal{A}_x$ for all $x \in X$. Remark that $\mathrm{val}(J,y,\mathcal{I}) = \mathrm{val}(J^*,y,\mathcal{I})$ for internal nodes y of J^*.

Take any $x_0 \in X$. Since $\mathrm{SV}(x_0,\mathcal{I}) = \mathbf{f}$, there is a J-branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ such that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{f}$.

We first prove that $\mathrm{SV}(x_i,\mathcal{I}) = \mathbf{f}$ for all x_i. Take by contradiction the lowest i such that $\mathrm{SV}(x_i,\mathcal{I}) \geq_t \mathbf{u}$. Remark that $i > 0$. If $\mathbf{b}$ is transitive at i, then $\mathcal{I}(\mathcal{B}(x_i \to x_{i+1} \to \cdots)) = \mathcal{B}(\mathbf{b}) = \mathbf{f}$; contradicting that $\mathrm{val}(J,x_i,\mathcal{I}) \geq_t \mathbf{u}$ since x_i is an internal node of J^*.

Therefore, by splittability, $\mathbf{b}$ is decided at i. Let j be the lowest at which $\mathbf{b}$ is decided; hence $\mathbf{b}$ is transitive at $j-1$. Remark that $j > 0$ since branches cannot be decided at 0. So $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x_{j-1} \to x_j \to \cdots)$. By Corollary 2.6.12, the path $x_{j-1} \to x_j$ is decided. Suppose $\mathrm{SV}(\sim x_{j-1},\mathcal{I}) \neq \mathbf{t}$. Then $x_j \in J(x_{j-1}) \in \mathcal{A}_{x_{j-1}}$. Therefore, $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(x_{j-1} \to x_j)) \geq_t \mathbf{u}$, which contradicts that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{f}$. Therefore, $\mathrm{SV}(\sim x_{j-1},\mathcal{I}) = \mathbf{t}$. Take k to be the lowest such that $\mathrm{SV}(\sim x_k,\mathcal{I}) = \mathbf{t}$. Since $k < j$, $\mathbf{b}$ is not decided at k, so also $x_{k-1} \to x_k$ is not decided by Lemma 2.6.13. Remark that $k > 0$ since $\mathrm{SV}(\sim x_0,\mathcal{I}) \neq \mathbf{t}$. Since $\mathrm{SV}(\sim x_{k-1},\mathcal{I}) \neq \mathbf{t}$, it holds that $x_k \in J(x_{k-1}) \in \mathcal{A}_{x_{k-1}}$ and thus $\mathrm{SV}(x_k,\mathcal{I}) \geq_t \mathbf{u}$ or $\mathrm{SV}(\sim x_k,\mathcal{I}) \neq \mathbf{t}$. Since $\mathrm{SV}(\sim x_k,\mathcal{I}) = \mathbf{t}$, we get that $\mathrm{SV}(x_k,\mathcal{I}) \geq_t \mathbf{u}$, which contradicts the minimality of i.

Secondly, we prove that $\mathrm{SV}(\sim x_i,\mathcal{I}) \neq \mathbf{t}$ for all i. Take the lowest i such that $\mathrm{SV}(\sim x_i,\mathcal{I}) = \mathbf{t}$. Note that $i > 0$. If $x_{i-1} \to x_i$ is not decided, then, since $\mathrm{SV}(\sim x_i,\mathcal{I}) = \mathbf{t}$ and $x_i \in J(x_{i-1}) \in \mathcal{A}_{x_i}$, it holds that $\mathrm{SV}(x_i,\mathcal{I}) \geq_t \mathbf{u}$, which is a contradiction. Therefore, $x_{i-1} \to x_i$ is decided and thus $\mathcal{I}(\mathcal{B}(x_{i-1} \to x_i)) \geq_t \mathbf{u}$. If $\mathbf{b}$ is first decided at $j < i$, then $x_{j-1} \to x_j$ is decided and thus $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(x_{j-1} \to x_j)) \geq_t \mathbf{u}$, which is a contradiction; hence $\mathbf{b}$ is transitive in $i-1$. Therefore, $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(x_{i-1} \to x_i \to \cdots)) = \mathcal{I}(\mathcal{B}(x_{i-1} \to x_i)) \geq_t \mathbf{u}$, which is a contradiction.

Thirdly, we prove that $\mathbf{b}$ is nowhere decided. Suppose $\mathbf{b}$ is first decided at i. Note that $i > 0$. Then by Corollary 2.6.12, the path $x_{i-1} \to x_i$ is decided and $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(x_{i-1} \to x_i)) \geq_t \mathbf{u}$ since $\mathrm{SV}(\sim x_{i-1},\mathcal{I}) \neq \mathbf{t}$ and $x_i \in J(x_{i-1}) \in \mathcal{A}_{x_{i-1}}$. This is a contradiction, so $\mathbf{b}$ is nowhere decided. □

The existence of a false branch nowhere decided with all elements inside

$$\mathcal{F}_d \setminus (\{x \in \mathcal{F}_d \mid \mathrm{SV}(x,\mathcal{I}) \geq_t \mathbf{u}\} \cup {\sim}\{x \in \mathcal{F}_d \mid \mathrm{SV}(x,\mathcal{I}) = \mathbf{t}\})$$

is enough to get a contradiction for $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, and $\mathcal{B}_{\mathrm{st}}$. Note that a branch that is nowhere decided is infinite since finite branches are decided at their last element.

Theorem 3.2.6. *The evaluations $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, and $\mathcal{B}_{\mathrm{st}}$ are graph-like consistent.*

Proof. By the contraposition of Proposition 3.2.3, it suffices to prove that if $\mathrm{SV}(x,\mathcal{I}) = \mathbf{f}$, then $\mathrm{SV}(\sim x,\mathcal{I}) = \mathbf{t}$. Suppose by contradiction that $\mathrm{SV}(\sim x,\mathcal{I}) \neq \mathbf{t}$. Then by Lemma 3.2.5, there is a $\mathcal{JF}$-branch starting with x such that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{f}$, nowhere decided and all elements of $\mathbf{b}$ lie in $\mathcal{F}_d \setminus (\{x \in \mathcal{F}_d \mid \mathrm{SV}(x) \geq_t \mathbf{u}\} \cup \sim\{x \in \mathcal{F}_d \mid \mathrm{SV}(x) = \mathbf{t}\})$.

If $\mathcal{B} = \mathcal{B}_{\mathrm{sp}}$, this is a contradiction since every branch is decided at the first element. If $\mathcal{B} = \mathcal{B}_{\mathrm{KK}}$, then $\mathcal{B}(\mathbf{b}) = \mathbf{u}$ since $\mathbf{b}$ is infinite, which is a contradiction. If $\mathcal{B} = \mathcal{B}_{\mathrm{st}}$, then $\mathbf{b}$ has everywhere the same sign, otherwise, it would have been decided somewhere; hence $\mathbf{b}$ is everywhere positive because $\mathcal{B}(\mathbf{b}) = \mathbf{f}$.

Take $y \in \mathcal{F}_d \setminus (\{x \mid \mathrm{SV}(x,\mathcal{I}) \geq_t \mathbf{u}\} \cup \sim\{x \mid \mathrm{SV}(x,\mathcal{I}) = \mathbf{t}\})$ and $A \in \mathcal{JF}(y)$. If $A \in \mathcal{A}_y$, then there is a branch as provided by Lemma 3.2.5. Define z_A to be the second element of this branch. If $A \notin \mathcal{A}_y$, then there is a $z_A \in A$ such that $y \to z_A$ satisfies the conditions for $\mathfrak{m}_{\mathcal{I}}$. Define $\mathcal{T}_y \colon \mathcal{JF}(y) \to \mathcal{F} \colon A \mapsto z_A$. Take a $\mathbf{t}$-domain supporting justification K such that the internal nodes of K are exactly the elements y such that $\mathrm{SV}(y,\mathcal{I}) = \mathbf{t}$. This is possible due to Lemma 2.6.19. Let J be a locally complete extension of K such that $J(\sim y) \subseteq \sim \mathrm{Im}(\mathcal{T}_y)$ for all $y \in \mathcal{F}_d \setminus (\{x \mid \mathrm{SV}(x) \geq_t \mathbf{u}\} \cup \sim\{x \mid \mathrm{SV}(x) = \mathbf{t}\})$. This is possible because $\sim y$ is not internal in K because $\mathrm{SV}(\sim y,\mathcal{I}) \neq \mathbf{t}$.

Take a J-branch $\mathbf{b}' \colon \sim x_0 \to \sim x_1 \to \cdots$ with $x_0 = x$. We prove that $\mathcal{I}(\mathcal{B}(\mathbf{b}')) = \mathbf{t}$. Suppose first that all elements of $\mathbf{b}'$ lie in

$$\sim(\mathcal{F}_d \setminus (\{x \in \mathcal{F}_d \mid \mathrm{SV}(x,\mathcal{I}) \geq_t \mathbf{u}\} \cup \sim\{x \in \mathcal{F}_d \mid \mathrm{SV}(x,\mathcal{I}) = \mathbf{t}\}))\,.$$

This means that $\sim x_{i+1} \in J(\sim x_i) \subseteq \sim \mathrm{Im}(\mathcal{T}_{x_i})$ for any $i \geq 0$. If $\mathbf{b}'$ is first decided at $i > 0$, then by Corollary 2.6.12 and Lemma 3.2.2, the path $x_{i-1} \to x_i$ is decided. Thus, $\mathcal{I}(\mathcal{B}(\mathbf{b}')) = \sim\mathcal{I}(\mathcal{B}(x_{i-1} \to x_i)) = \mathbf{t}$. So we can assume that $\mathbf{b}'$ is nowhere decided and thus infinite. This means that $\mathbf{b}'$ has everywhere the same sign and a different sign than $\mathbf{b}$. Therefore $\mathcal{I}(\mathcal{B}(\mathbf{b}')) = \sim\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}$. This means that $\mathrm{val}(J, \sim x_0, \mathcal{I}) = \mathbf{t}$, which is in contradiction with our assumption. □

3.3 Consistency of Well-Founded Semantics

Transitive branch evaluations have the nice property that if a justification is good for x, then it is also good for any y reachable from x in that justification. The following lemma is a continuation of this property.

Lemma 3.3.1. *Let $\mathcal{B}$ be a transitive branch evaluation such that finite branches are mapped to their last element. Let $\mathcal{JF}$ be a complementary justification frame. Then for any $\mathcal{F}$-interpretation and $x \in \mathcal{F}_d$*

- $\mathrm{SV}(x, \mathcal{I}) = \mathbf{t}$ *if and only if there is an $A \in \mathcal{JF}(x)$ such that for all $a \in A$ $\mathrm{SV}(a, \mathcal{I}) = \mathbf{t}$ if a is defined and $\mathcal{I}(a) = \mathbf{t}$ if a is open.*
- $\mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}$ *if and only if there is an $A \in \mathcal{JF}(x)$ such that for all $a \in A$, $\mathrm{SV}(\sim a, \mathcal{I}) \neq \mathbf{t}$ if a is defined and $\mathcal{I}(a) \geq_t \mathbf{u}$ if a is open.*

Proof. If $\mathrm{SV}(x, \mathcal{I}) = \mathbf{t}$, then there is a locally complete justification J with $\mathrm{val}(J, x, \mathcal{I}) = \mathbf{t}$. Then by transitivity and the fact that finite branches are mapped to their last element we have the result.

Now, take such a case A. Let J_{-1} be the justification consisting only of the rule $x \leftarrow A$. Let Y be the set $A \cap \mathcal{F}_d$. For all $y \in Y$, we have $\mathrm{SV}(y, \mathcal{I}) = \mathbf{t}$; hence there is a locally complete justification J_y with $\mathrm{val}(J_y, y, \mathcal{I}) = \mathbf{t}$. Fix a well-order on $Y = \{y_i \mid i \leq \beta\}$. Define inductively justifications J_i for $0 \leq i \leq \beta$:

- $J_{i+1} \coloneqq J_i \uparrow J_{y_{i+1}}$ for any ordinal $i < \beta$;
- $J_\alpha \coloneqq \cup_{i<\alpha} J_i$ for a limit ordinal $\alpha \leq \beta$.

Any J_β-branch starting with x is either of the form $x \to a$ with $a \in A \cap \mathcal{F}_o$ or $x \to \mathbf{b}$ with $\mathbf{b}$ a J_y-branch for some $y \in A \cap \mathcal{F}_d$. By our assumptions, both types are mapped to $\mathbf{t}$ by $\mathcal{I}$ under $\mathcal{B}$; hence $\mathrm{val}(J_\beta, x, \mathcal{I}) = \mathbf{t}$, which shows that $\mathrm{SV}(x, \mathcal{I}) = \mathbf{t}$.

If $\mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}$, then by the above, for all $B \in \mathcal{JF}(\sim x)$ there is a defined $z_B \in B$ such that $\mathrm{SV}(z_B, \mathcal{I}) \neq \mathbf{t}$ or an open $z_B \in B$ such that $\mathcal{I}(z_B) \neq \mathbf{t}$. Define $\mathcal{T} : \mathcal{JF}(\sim x) \to \mathcal{F} : B \mapsto z_B$. Since $\mathcal{JF}$ is complementary, there exists an $A \subseteq \sim \mathrm{Im}(\mathcal{T})$. This means that for every defined $a \in A$, $\mathrm{SV}(\sim a, \mathcal{I}) \neq \mathbf{t}$ and for every open $a \in A$, $\mathcal{I}(\sim a) \neq \mathbf{t}$, i.e. $\mathcal{I}(a) \geq_t \mathbf{u}$.

Now assume that there is such an $A \in \mathcal{JF}(x)$ and suppose by contradiction that $\mathrm{SV}(\sim x, \mathcal{I}) = \mathbf{t}$. Then there is a $B \in \mathcal{JF}(\sim x)$ such that every defined fact in B has true supported value and every open in B is true under $\mathcal{I}$. Therefore, by Lemma 2.3.15, $A \cap \sim B \neq \emptyset$, which is a contradiction. □

This lemma motivates the following definition.

Definition 3.3.2. A subset $\mathcal{P}$ of $\mathcal{F}$ is *compositional* on $X \subseteq \mathcal{F}_d$ if for all $x \in X$ it holds that $x \in \mathcal{P}$ if and only if there is an $A \in \mathcal{JF}(x)$ such that $A \subseteq \mathcal{P}$.

Intuitively, a property is compositional on X if that property propagates to all the elements of the body of at least one rule and vice versa.

In compositional property terms, Lemma 3.3.1 becomes the following corollary.

Corollary 3.3.3. *Let $\mathcal{B}$ be a transitive branch evaluation mapping finite branches to their last element and $\mathcal{JF}$ a complementary justification frame. The following sets are compositional on any subset of $\mathcal{F}_d$:*

- $\{x \in \mathcal{F}_d \mid \mathrm{SV}(x, \mathcal{I}) = \mathbf{t}\} \cup \{x \in \mathcal{F}_o \mid \mathcal{I}(x) = \mathbf{t}\}$,
- $\{x \in \mathcal{F}_d \mid \mathrm{SV}(\sim x, \mathcal{I}) \neq \mathbf{t}\} \cup \{x \in \mathcal{F}_o \mid \mathcal{I}(x) \geq_t \mathbf{u}\}$.

In complementary justification frames, we can transform a compositional property on a set X to a dual compositional property on the set $\sim X$.

Lemma 3.3.4. *Let $\mathcal{JF}$ be a complementary justification frame. If $\mathcal{P}$ is compositional on X, then $\mathrm{d}(\mathcal{P}) \coloneqq \mathcal{F} \setminus \sim\mathcal{P}$ is compositional on $\sim X$.*

Proof. Let $\sim x \in \sim X$. First, assume $\sim x \in \mathrm{d}(\mathcal{P})$, that is, $x \notin \mathcal{P}$; hence for every $A \in \mathcal{JF}(x)$, there is a $z_A \in A \setminus \mathcal{P}$. Therefore, $\sim z_A \in \mathrm{d}(\mathcal{P})$. Define the selection function $\mathcal{T}$ of x as $\mathcal{T} \colon \mathcal{JF}(x) \to \mathcal{F} \colon A \mapsto z_A$. Then by complementarity, there is a $B \in \mathcal{JF}(\sim x)$ such that $B \subseteq \sim \mathrm{Im}(\mathcal{T}) \subseteq \mathrm{d}(\mathcal{P})$.

We prove the converse by contraposition, i.e. if $\sim x \notin \mathrm{d}(\mathcal{P})$, then for all $B \in \mathcal{JF}(\sim x)$ it holds that $B \nsubseteq \mathrm{d}(\mathcal{P})$. So assume that $\sim x \notin \mathrm{d}(\mathcal{P})$, thus that $x \in \mathcal{P}$; hence there exists an $A \in \mathcal{JF}(x)$ such that $A \subseteq \mathcal{P}$. Take a $B \in \mathcal{JF}(\sim x)$. By Lemma 2.3.15, $\sim A \cap B \neq \emptyset$. If $B \subseteq \mathrm{d}(\mathcal{P})$, then

$$\emptyset \neq \sim A \cap B \subseteq \sim\mathcal{P} \cap \mathrm{d}(\mathcal{P}) = \emptyset,$$

which is a contradiction, concluding the proof. □

Compositional properties are excellent tools to construct justifications with nice properties as shown by the following lemma.

Lemma 3.3.5. *If $\mathcal{P}$ is compositional on $X \subseteq \mathcal{F}_d$, then for each $x \in \mathcal{P} \cap X$, there is a justification J with x as internal node such that*

- *The internal nodes of J are in $\mathcal{P} \cap X$;*

- *The leaves of J are in $\mathcal{P}$;*
- *The defined leaves of J do not lie in X.*

Proof. Start with the justification J_0 with a single rule $x \leftarrow A$ so that $A \subseteq \mathcal{P}$. We construct justifications J_{i+1} by induction. For every defined leaf y of J_i that lies in X, there exists a rule $y \leftarrow A$ such that $A \subseteq \mathcal{P}$. Define J_{i+1} as the extension of J_i that adds these rules. Then $J = \cup_{i \geq 0} J_i$. □

In the proof that $\mathcal{B}_{\text{wf}}$ is consistent, we will paste together justifications that are formed as in the previous lemma. However, these justifications are not necessarily locally complete. Therefore, we need a result stating how the branches are formed, similar to Lemma 2.6.5.

Proposition 3.3.6. *Let K be any justification and J a locally complete justification such that $J \uparrow K$ is locally complete. Then every $(J \uparrow K)$-branch either is a K-branch or has a tail that is a J-branch.*

Proof. Take a $(J \uparrow K)$-branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$ that is not a K-branch and not a J-branch. Then there is an i such that x_i is internal in K, but $x_{i+1} \notin K(x_i)$. This means that x_i is internal in J. In general if x_j is internal in J, then $x_{j+1} \in J(x_j) = (J \uparrow K)(x_j)$. So x_{j+1} is either an open leaf of J or x_{j+1} is internal in J since J is locally complete. Then by induction $x_i \to x_{i+1} \to \cdots$ is a J-branch and a tail of $\mathbf{b}$. □

Extending a justification with no infinite branches with another justification with no infinite branches can create a justification with infinite branches. This can happen when both justifications are not locally complete. For example, extending the justification $x \to y$ with the justification $y \to x$ would create a justification with the branch $x \to y \to x \to y \to \cdots$. The following lemma states a sufficient condition for which this cannot happen.

Lemma 3.3.7. *Let J and K be any two justifications without infinite branches such that there is an $X \subseteq \mathcal{F}_d$ so that the internal nodes of J and K are in X and J and K have no defined leaves in X. Then $J \uparrow K$ has no infinite branches as well.*

Proof. By contradiction, take an infinite $(J \uparrow K)$-branch $\mathbf{b}\colon x_0 \to x_1 \to \cdots$. Every element in $\mathbf{b}$ is defined and thus an internal node of J or K; hence every element in $\mathbf{b}$ lies in X. This means that x_i is not a leaf of J and K since defined leaves lie outside X. Since K has no infinite branches, we have that $\mathbf{b}$ is not a K-branch. So take the least i such that $x_{i+1} \notin K(x_i)$; hence $x_{i+1} \in J(x_i)$.

If x_{i+1} is not internal in J, then we have that x_{i+1} is a leaf of J, but this contradicts that $x_{i+1} \in X$. Hence $x_{i+2} \in J(x_{i+1})$. By an induction argument, $\mathbf{b}$ has a tail that is a J-branch, contradicting that $\mathbf{b}$ is infinite. □

The conditions on the justifications in this lemma are very similar to the properties that justifications constructed from Lemma 3.3.5 have. This will be a crucial point in proving that the following property is compositional.

Lemma 3.3.8. *The set $\mathcal{P}$ defined as*

- $\mathcal{P} \cap \mathcal{F}_o = \{x \in \mathcal{F}_o \mid \mathcal{I}(x) \geq_t \mathbf{u}\}$;
- $\mathcal{P} \cap \mathcal{F}_- = \{x \in \mathcal{F}_- \mid \mathrm{SV}(\sim x, \mathcal{I}) \leq_t \mathbf{u}\}$;
- *for $x \in \mathcal{F}_+$, $x \in \mathcal{P}$ if and only if there is a justification J with x as internal node such that*
  - *J does not have infinite branches;*
  - *the internal nodes of J are in $\mathcal{F}_+$;*
  - *defined leaves of J lie in $\mathcal{F}_-$;*
  - *all leaves of J lie in $\mathcal{P}$.*

  We call such a justification a $*$-justification for x.

is compositional on $\mathcal{F}_+$.

Proof. Take $x \in \mathcal{F}_+$. First, assume that $x \in \mathcal{P}$. So there exists a $*$-justification J for x. We show that any $y \in J(x)$ lies in $\mathcal{P}$. If y is a leaf of J, then $y \in \mathcal{P}$. If y is not a leaf, then $y \in \mathcal{F}_+$. In this case, J is also a $*$-justification for y. So we can conclude that $J(x) \subseteq \mathcal{P}$.

Conversely, assume there is an $A \in \mathcal{JF}(x)$ such that $A \subseteq \mathcal{P}$. Suppose first that $A \cap \mathcal{F}_+ = \emptyset$. In that case, the justification J consisting of only the rule $x \leftarrow A$ is a $*$-justification for x. So we can assume that $A \cap \mathcal{F}_+ \neq \emptyset$. For any $y \in A \cap \mathcal{F}_+$, there is a $*$-justification J_y for y. Fix a well-order on $A \cap \mathcal{F}_+ = \{y_i \mid i \leq \beta\}$. Define J_0 to be the justification consisting only of the rule $x \leftarrow A$. Inductively define

- $J_{i+1} = J_i \uparrow J_{y_i}$ for any ordinal $i < \beta$;
- $J_\alpha = \cup_{i<\alpha} J_i$ for any limit ordinal $\alpha \leq \beta$.

We prove that J_β is a $*$-justification for x.

We only prove that J_β has no infinite branches. The other properties should be straightforward. By Lemma 3.3.7, it suffices to prove for any limit ordinal $\alpha \leq \beta$ that J_α has no infinite branches if J_i has no infinite branches for all $i < \alpha$. Suppose that J_α contains an infinite branch $x_0 \to x_1 \to \cdots$. Then $x_j \in \mathcal{F}_+$ for all j since x_j is internal of J_i for some $i < \alpha$. By construction, there is a $\gamma < \alpha$ such that $J_\alpha(x_0) = J_\gamma(x_0)$. Since $x_1 \in \mathcal{F}_+$, $x_1 \in \mathrm{dom}(J_\gamma)$; otherwise x_1 is a leaf of J_γ and thus $x_1 \in \mathcal{F}_-$. By induction all x_i's lie in $\mathrm{dom}(J_\gamma)$; which contradicts that J_γ has no infinite branches.

We proved that J_β is a $*$-justification for x. Therefore, $x \in \mathcal{P}$, which ends the proof that $\mathcal{P}$ is compositional on $\mathcal{F}_+$. □

Now, we have accumulated the necessary results to prove that $\mathcal{B}_{\mathrm{wf}}$ is graph-like consistent.

Theorem 3.3.9. *The branch evaluation $\mathcal{B}_{\mathrm{wf}}$ is graph-like consistent.*

Proof. Because of Proposition 3.2.3, it suffices to prove that if $\mathrm{SV}(\sim x, \mathcal{I}) \leq_t \mathbf{u}$, then $\mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}$. We first show that there exists a justification J_x with x internal in J_x such that

- all internal elements have the same sign as x;
- all open leaves y have $\mathcal{I}(y) \geq_t \mathbf{u}$;
- all defined leaves y have $\mathrm{SV}(\sim y, \mathcal{I}) \leq_t \mathbf{u}$ and a different sign than x;
- all infinite branches have a negative tail.

We first prove it for $x \in \mathcal{F}_-$. By Corollary 3.3.3, the set

$$\{y \in \mathcal{F}_d \mid \mathrm{SV}(\sim y, \mathcal{I}) \neq \mathbf{t}\} \cup \{y \in \mathcal{F}_o \mid \mathcal{I}(y) \geq_t \mathbf{u}\}$$

is compositional on $\mathcal{F}_-$. Then by Lemma 3.3.5 such a justification J_x exists.

So assume $x \in \mathcal{F}_+$. It suffices to prove that there is a $*$-justification for x, i.e. $x \in \mathcal{P}$ with $\mathcal{P}$ the compositional set of Lemma 3.3.8. Assume by contradiction that $x \notin \mathcal{P}$, i.e. $\sim x \in \mathrm{d}(\mathcal{P})$. By Lemma 3.3.4, $\mathrm{d}(\mathcal{P})$ is compositional on $\mathcal{F}_-$. Therefore, by Lemma 3.3.5, there is a justification J with $\sim x$ internal in J such that all internal nodes of J are in $\mathcal{F}_-$, all defined leaves are positive, and all leaves lie in $\mathrm{d}(\mathcal{P})$. This means that for all open leaves y, it holds that $\mathcal{I}(y) = \mathbf{t}$. For positive defined leaves y, we have that $\sim y \notin \mathcal{P}$, which means that $\mathrm{SV}(y, \mathcal{I}) = \mathbf{t}$. Take a locally complete justification J_y such that $\mathrm{val}(J_y, y, \mathcal{I}) = \mathbf{t}$.

Remark that we can take J_y so that every element in J_y is reachable from y by removing unreachable elements.

Define $\mathcal{Q}$ to be the set of positive defined leaves of J. There is a locally complete justification J^* such that $\mathrm{val}(J^*, y, \mathcal{I}) = \mathbf{t}$ for all $y \in \mathcal{Q}$ and the set of internal nodes of J^* is exactly the union of the internal nodes of J_y for $y \in \mathcal{Q}$. Such a justification can be constructed by pasting together J_y for $y \in \mathcal{Q}$. Define $K = J^* \uparrow J$. Suppose K is not locally complete: it has a defined leaf z. Since J^* is locally complete, z is a leaf of J. Therefore, $z \in \mathcal{Q}$, which contradicts the construction of K; hence K is locally complete.

Then by Proposition 3.3.6, every K-branch is either a J-branch or has a tail that is a J^*-branch. In the first case, every infinite branch consists solely out of negative defined facts, hence is evaluated to $\mathbf{t}$ by $\mathcal{I}$ under $\mathcal{B}$. Every finite branch is mapped to an open element y in $\mathrm{d}(\mathcal{P})$, i.e. $\mathcal{I}(y) = \mathbf{t}$. In the second case, for every such branch $\mathbf{b}$, $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathcal{I}(\mathcal{B}(\mathbf{b}'))$ for a J^*-branch $\mathbf{b}'$ by transitivity of $\mathcal{B}_{\mathrm{wf}}$. Since $\mathrm{val}(J^*, y, \mathcal{I}) = \mathbf{t}$ for any internal node y of J^*, we have that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}$. Therefore, we proved that $\mathrm{val}(K, {\sim}x, \mathcal{I}) = \mathbf{t}$; hence $\mathrm{SV}({\sim}x, \mathcal{I}) = \mathbf{t}$. This is a contradiction, hence there is a justification J_x for $x \in \mathcal{F}_+$.

By pasting the justifications J_x together for all $x \in \mathcal{F}_d$ with $\mathrm{SV}({\sim}x, \mathcal{I}) \neq \mathbf{t}$, we get a locally complete justification K^* such that any internal nodes y in K, we have that $K^*(y) = J_x(y)$ for some defined x with $\mathrm{SV}({\sim}x, \mathcal{I}) \neq \mathbf{t}$. Take a finite K^*-branch $x_0 \to \cdots \to x_n$. Since x_n is an open leaf of K^*, x_n is an open leaf of some J_x, i.e. $\mathcal{I}(x_n) \geq_t \mathbf{u}$. Therefore, $\mathcal{I}(\mathcal{B}(x_0 \to \cdots \to x_n)) \geq_t \mathbf{u}$. Suppose there is an infinite K^*-branch $x_0 \to x_1 \to \cdots$ with a positive tail. Without loss of generality, we can take all the x_i's to be positive. There is a defined z such that $K^*(x_0) = J_z(x_0)$. Now, let j be the smallest index such that $K^*(x_j) \neq J_z(x_j)$. Such an index exists since J_z has no infinite branches with a positive tail. This means that x_j is a defined leaf of J_z. Therefore, it has a different sign than x_0, which is negative. This is a contradiction, so we can conclude that K^* has no infinite branches with a positive tail. Therefore, we have for an infinite K^*-branch $\mathbf{b}$ that $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathbf{u}$. This proves that $\mathrm{val}(K^*, x, \mathcal{I}) \geq_t \mathbf{u}$ for all x with $\mathrm{SV}({\sim}x, \mathcal{I}) \leq_t \mathbf{u}$. This concludes the proof that $\mathrm{SV}(x, \mathcal{I}) \geq_t \mathbf{u}$. □

The proof for $\mathcal{B}_{\mathrm{wf}}$ is a lot more complex than for the other branch evaluations. This is because we construct a justification J with $\mathrm{val}(J, x, \mathcal{I}) \geq_t \mathbf{u}$, while for the other branch evaluations, it was sufficient to get a contradiction when such a justification does not exist. We must note, however, that the construction is not exactly deterministic; both in the choice of some justifications as well as the order in which justifications are pasted together. The latter is not really a problem since any total order would suffice. The former, however, is more

troublesome as one would need to find small good justifications for a bunch of elements. Searching for such justifications will certainly take more time than directly searching for a justification for $\sim x$. Now that we proved that $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{st}}$, and $\mathcal{B}_{\mathrm{wf}}$ are graph-like consistent, we get the following consequence.

Corollary 3.3.10. *Any evaluation in* $\{\mathcal{B}_{\mathrm{sp}}, \mathcal{B}_{\mathrm{KK}}, \mathcal{B}_{\mathrm{st}}, \mathcal{B}_{\mathrm{wf}}, \mathcal{B}_{\mathrm{cst}}, \mathcal{B}_{\mathrm{cwf}}\}$ *is graph-like consistent, tree-like consistent, and graph-reducible.*

Proof. This is a combination of Theorems 3.2.6 and 3.3.9 and Corollaries 2.4.7 and 2.5.13. □

In the next section, we will use the consistency to show relations between the various models.

3.4 Alternative Branch Evaluations

In 2015, Denecker et al. (2015) gave an alternative definition for $\mathcal{B}_{\mathrm{st}}$. In a similar manner, $\mathcal{B}_{\mathrm{sp}}$ can have an alternative as well.

Definition 3.4.1. The branch evaluation $\mathcal{B}'_{\mathrm{sp}}$ is equal to $\mathcal{B}_{\mathrm{sp}}$ on infinite branches and maps finite branches to their last element. The branch evaluation $\mathcal{B}'_{\mathrm{st}}$ is equal to $\mathcal{B}_{\mathrm{st}}$ on infinite branches and maps finite branches to their last element.

In this section, we prove that $\mathcal{B}_{\mathrm{sp}}$ and $\mathcal{B}'_{\mathrm{sp}}$ are equivalent and that $\mathcal{B}_{\mathrm{st}}$ and $\mathcal{B}'_{\mathrm{st}}$ are equivalent. Recall that equivalent means they have the same models in each justification frame.

The reason why we prefer $\mathcal{B}_{\mathrm{sp}}$ and $\mathcal{B}_{\mathrm{st}}$ is that they have better qualities, such as splittability. The branch evaluation $\mathcal{B}'_{\mathrm{sp}}$ is not splittable because an infinite branch is neither decided nor transitive at the second element. Similarly, $\mathcal{B}'_{\mathrm{sp}}$ is not splittable because an infinite branch is neither decided nor transitive at its first sign.

The following lemma provides a simpler way to determine if an interpretation is a model.

Lemma 3.4.2. *Let* $\mathcal{JF}$ *be any justification frame and* $\mathcal{B}$ *a branch evaluation that respects negation. An* $\mathcal{F}$*-interpretation* $\mathcal{I}$ *such that* $\mathrm{SV}_{\mathcal{B}}(x, \mathcal{I}) \geq_t \mathcal{I}(x)$ *for all* $x \in \mathcal{F}_d$*, is a* $\mathcal{B}$*-model of* $\mathcal{JF}$*.*

Proof. For all $x \in \mathcal{F}_d$ we have that $\mathrm{SV}_{\mathcal{B}}(\sim x, \mathcal{I}) \geq_t \mathcal{I}(\sim x)$. By Theorem 2.3.18, we have $\mathrm{SV}_{\mathcal{B}}(\sim x, \mathcal{I}) \leq_t \sim \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$. Therefore, $\mathcal{I}(\sim x) \leq_t \sim \mathrm{SV}_{\mathcal{B}}(x, \mathcal{I})$, or that

$\mathrm{SV}_{\mathcal{B}}(x,\mathcal{I}) \leq_t \sim\mathcal{I}(\sim x) = \mathcal{I}(x)$. This completes the proof that $\mathrm{SV}_{\mathcal{B}}(x,\mathcal{I}) = \mathcal{I}(x)$, i.e., $\mathcal{I}$ is a $\mathcal{B}$-model of $\mathcal{JF}$. □

To prove that the alternative branch evaluations for $\mathcal{B}_{\mathrm{sp}}$ and $\mathcal{B}_{\mathrm{st}}$ are equivalent, we need a result similar to Theorem 2.6.20. But before we do that, we provide a different characterisation of true facts in $\mathcal{B}'_{\mathrm{sp}}$-models.

Proposition 3.4.3. *Let $\mathcal{I}$ be a $\mathcal{B}'_{\mathrm{sp}}$-model. For $x \in \mathcal{F}_d$, $\mathcal{I}(x) = \mathbf{t}$ if and only if there exists a rule $x \leftarrow A$ such that for all $a \in A$, $\mathcal{I}(a) = \mathbf{t}$.*

Proof. If $\mathcal{I}(x) = \mathbf{t}$, we have that $\mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{t}$; hence there is a justification J such that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J,x,\mathcal{I}) = \mathbf{t}$. Let A be the case $J(x)$. Take $y \in A$. If $x \to y$ is part of an infinite branch in $B_J(x)$, then it is obvious that $\mathcal{I}(y) = \mathbf{t}$. So assume this is not the case, then we have that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J,y,\mathcal{I}) = \mathbf{t}$ since every branch in $B_J(y)$ is part of a finite branch in $B_J(x)$ that maps to $\mathbf{t}$ under $\mathcal{I}$. Therefore, $\mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(y,\mathcal{I}) = \mathbf{t}$. Since $\mathcal{I}$ is a $\mathcal{B}'_{\mathrm{sp}}$-model, we get that $\mathcal{I}(y) = \mathbf{t}$.

Assume now that there exists such a rule $x \leftarrow A$. Since $\mathcal{I}(y) = \mathbf{t}$ for all $y \in A$. We have by the above proof that we have a true rule for y. By iteratively adding these rules we get a locally complete justification J. We prove that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J,x,\mathcal{I}) = \mathbf{t}$. Since every element of this justification is mapped to $\mathbf{t}$ under $\mathcal{I}$, this is straightforward. Now since $\mathcal{I}$ is a $\mathcal{B}'_{\mathrm{sp}}$-model, we get that $\mathcal{I}(x) = \mathbf{t}$. □

The right hand condition is exactly the same as $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{t}$. A similar thing can be proven for false facts in $\mathcal{B}'_{\mathrm{sp}}$-models.

Proposition 3.4.4. *Let $\mathcal{I}$ be a $\mathcal{B}'_{\mathrm{sp}}$-model. For $x \in \mathcal{F}_d$, $\mathcal{I}(x) = \mathbf{f}$ if and only if for every rule $x \leftarrow A$ there is an $a \in A$ such that $\mathcal{I}(a) = \mathbf{f}$.*

Proof. The right-hand side is equivalent to $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{f}$. By consistency of $\mathcal{B}_{\mathrm{sp}}$, this is equivalent to $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(\sim x,\mathcal{I}) = \mathbf{t}$. By Proposition 3.4.3, this is equivalent to $\mathcal{I}(\sim x) = \mathbf{t}$; hence $\mathcal{I}(x) = \mathbf{f}$, which completes the proof. □

These two propositions actually imply that every $\mathcal{B}'_{\mathrm{sp}}$-model is a $\mathcal{B}_{\mathrm{sp}}$-model, which is one direction of the equivalence.

Corollary 3.4.5. *Every $\mathcal{B}'_{\mathrm{sp}}$-model is a $\mathcal{B}_{\mathrm{sp}}$-model.*

Proof. Take a $\mathcal{B}'_{\mathrm{sp}}$-model $\mathcal{I}$. Take any $x \in \mathcal{F}_d$. We prove that $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathcal{I}(x)$. If $\mathcal{I}(x) = \mathbf{t}$ or $\mathcal{I}(x) = \mathbf{f}$, this follows from Propositions 3.4.3 and 3.4.4. So assume $\mathcal{I}(x) = \mathbf{u}$. If $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{f}$, then by Proposition 3.4.4, we have that $\mathcal{I}(x) = \mathbf{f}$. If $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{t}$, then by Proposition 3.4.3, we have that $\mathcal{I}(x) = \mathbf{t}$. Both are not possible, hence $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{u}$. □

We can now prove a similar result to Theorem 2.6.20 for $\mathcal{B}'_{\mathrm{sp}}$, but only in $\mathcal{B}'_{\mathrm{sp}}$-models.

Theorem 3.4.6. *For every $\mathcal{B}'_{\mathrm{sp}}$-model $\mathcal{I}$, there is a locally complete justification J such that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J,x,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(x,\mathcal{I})$ for all $x \in \mathcal{F}_d$.*

Proof. Every x with $\mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(x,\mathcal{I}) = \mathbf{t}$ has a true rule $x \leftarrow A$. By combining all these rules, we get a locally complete justification $J_{\mathbf{t}}$ such that for every internal node y of $J_{\mathbf{t}}$ we get that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J_{\mathbf{t}},y,\mathcal{I}) = \mathbf{t}$. By a similar construction we get a justification $J_{\mathbf{u}}$. This is not a locally complete justification. However, the justification $J_{\mathbf{t}} \uparrow J_{\mathbf{u}}$ is locally complete. For every internal node y of $J_{\mathbf{t}}$ we have that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J_{\mathbf{t}} \uparrow J_{\mathbf{u}},y,\mathcal{I}) = \mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J_{\mathbf{t}},y,\mathcal{I}) = \mathbf{t}$. For every internal node y of $J_{\mathbf{u}}$ we have that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J_{\mathbf{t}} \uparrow J_{\mathbf{u}},y,\mathcal{I}) = \mathbf{u}$. By extending the justification $J_{\mathbf{t}} \uparrow J_{\mathbf{u}}$ to a complete justification, we do not change the value of the internal nodes of $J_{\mathbf{t}} \uparrow J_{\mathbf{u}}$; hence this provides a justification J such that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J,y,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(y,\mathcal{I})$ for all $y \in \mathcal{F}_d$. □

The existence of such a justification will not be used to prove the equivalence, but for some later results. We can, however, prove the equivalence at this point.

Proposition 3.4.7. *The two supported branch evaluations $\mathcal{B}_{\mathrm{sp}}$ and $\mathcal{B}'_{\mathrm{sp}}$ are equivalent.*

Proof. By Corollary 3.4.5, it suffices to prove that any $\mathcal{B}_{\mathrm{sp}}$-model is a $\mathcal{B}'_{\mathrm{sp}}$-model. Take a $\mathcal{B}_{\mathrm{sp}}$-model $\mathcal{I}$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\mathrm{sp}}$, there is a justification J such that $\mathrm{val}_{\mathcal{B}_{\mathrm{sp}}}(J,x,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) = \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$. For any rule $x \leftarrow D$ in J, we have that $\mathcal{I}(x) \leq_t \mathcal{I}(z)$ for all $z \in D$. Therefore, by iteratively applying this result, it holds that for all leaves z in J reachable from x, we have that $\mathcal{I}(x) \leq_t \mathcal{I}(z)$. This means that $\mathrm{val}_{\mathcal{B}'_{\mathrm{sp}}}(J,x,\mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}_{\mathrm{sp}}}(J,x,\mathcal{I}) = \mathcal{I}(x)$, which implies that $\mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(x,\mathcal{I}) \geq_t \mathcal{I}(x)$. This holds for all $x \in \mathcal{F}_d$; hence by Lemma 3.4.2 we have that $\mathcal{I}$ is a $\mathcal{B}'_{\mathrm{sp}}$-model.

□

Also for $\mathcal{B}'_{\mathrm{st}}$, a result similar to Theorem 2.6.20 holds, except only in $\mathcal{B}'_{\mathrm{st}}$-models.

Theorem 3.4.8. *For every $\mathcal{B}'_{\mathrm{st}}$-model $\mathcal{I}$, there is a locally complete justification J such that $\mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J,x,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}'_{\mathrm{st}}}(x,\mathcal{I})$ for all $x \in \mathcal{F}_d$.*

Proof. Take $\ell \in \mathcal{L}$. Take $x \in \mathcal{F}_d$ such that $\mathrm{SV}_{\mathcal{B}'_{\mathrm{st}}}(x,\mathcal{I}) = \ell$; hence there is a locally complete justification J_x such that $\mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J_x,x,\mathcal{I}) = \ell$. Let J_x^* be the justification J restricted to the domain of elements y reachable from x through a J_x-path of the same sign (this also means that x and y have the same sign).

Note that leaves can have different signs. Every internal node y of J_x^* has $\mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J_x, y, \mathcal{I}) \geq_t \ell$. Indeed, the evaluation of every branch in $B_{J_x}(y)$ is equal to the evaluation of a path from x to y with the same sign concatenated with this branch; hence this branch is evaluated to $\geq_t \mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J_x, x, \mathcal{I}) \geq_t \ell$; hence $\mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J_x, y, \mathcal{I}) \geq_t \ell$. Therefore, $\mathcal{I}(y) \geq_t \ell$ because $\mathcal{I}$ is a $\mathcal{B}'_{\mathrm{st}}$-model. For every leaf y of J_x^*, we have $\mathcal{I}(y) \geq_t \ell$ as well. Indeed, if y is part of an infinite J_x-path starting with x, then this is obvious. If y is not, then $B_{J_x}(y)$ only contains finite branches, and the result follows.

If we paste together all these J_x^* for all $x \in \mathcal{F}_d$ such that $\mathrm{SV}_{\mathcal{B}'_{\mathrm{st}}}(x, \mathcal{I}) = \ell$, we get a not necessarily locally complete justification J_ℓ such that every node y in J_ℓ has $\mathcal{I}(y) \geq_t \ell$.

Moreover, $J_{\mathbf{t}}$ is locally complete. Our final justification is then $J_{\mathbf{t}} \uparrow J_{\mathbf{u}} \uparrow J_{\mathbf{f}}$. By construction everything works out. □

In contrast with the completion equivalence, the proof of the stable equivalence makes use of such a justification.

Proposition 3.4.9. *The two stable branch evaluations $\mathcal{B}_{\mathrm{st}}$ and $\mathcal{B}'_{\mathrm{st}}$ are equivalent.*

Proof. Take a $\mathcal{B}_{\mathrm{st}}$-model $\mathcal{I}$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\mathrm{st}}$, there is a justification J such that $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, x, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x, \mathcal{I}) = \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$. For any finite branch $\mathbf{b}$ in J starting with x, we have for the first sign switch y, that $\mathcal{I}(x) \leq_t \mathcal{I}(y)$ since $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, x, \mathcal{I}) = \mathcal{I}(x)$. Therefore, by applying induction, we have that $\mathcal{I}(x) \leq_t \mathcal{I}(\text{last element of } \mathbf{b})$. This means that J is a justification for x with $\mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J, x, \mathcal{I}) \geq_t \mathcal{I}(x)$; hence $\mathrm{SV}_{\mathcal{B}'_{\mathrm{st}}}(x, \mathcal{I}) \geq_t \mathcal{I}(x)$. Then $\mathcal{I}$ is a $\mathcal{B}'_{\mathrm{st}}$-model by Lemma 3.4.2.

Take a $\mathcal{B}'_{\mathrm{st}}$-model $\mathcal{I}$. By Theorem 3.4.8, there is a justification J such that $\mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J, x, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}'_{\mathrm{st}}}(x, \mathcal{I}) = \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$. Take a finite branch $\mathbf{b}$ with a first sign switch y. We prove that $\mathcal{I}(y) \geq_t \mathcal{I}(x)$ since that would prove that $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, x, \mathcal{I}) \geq_t \mathcal{I}(x)$, and by Lemma 3.4.2 this proves that $\mathcal{I}$ is $\mathcal{B}_{\mathrm{st}}$-model. If y is the first sign switch of an infinite branch $\mathbf{b}^*$ starting with x, then $\mathcal{I}(x) = \mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J, x, \mathcal{I}) \leq_t \mathcal{I}(\mathcal{B}'_{\mathrm{st}}(\mathbf{b}^*)) = \mathcal{I}(y)$. So we can assume that this is not the case. It suffices to prove that $\mathcal{I}(\mathcal{B}'_{\mathrm{st}}(\mathbf{b}^*)) \geq_t \mathcal{I}(x)$ for every $\mathbf{b}^*$ in $B_J(y)$ since it implies that $\mathcal{I}(y) = \mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J, y, \mathcal{I}) \geq_t \mathcal{I}(x)$. The branch $\mathbf{b}^*$ cannot be infinite since then the infinite J-branch $x \to \cdots \to \mathbf{b}^*$ has y as first sign switch. If $\mathbf{b}^*$ is finite, then there is a finite branch $\mathbf{b}'$ in $B_J(x)$ so that $\mathcal{B}'_{\mathrm{st}}(\mathbf{b}^*) = \mathcal{B}'_{\mathrm{st}}(\mathbf{b}')$ and thus $\mathcal{I}(\mathcal{B}'_{\mathrm{st}}(\mathbf{b}^*)) \geq_t \mathrm{val}_{\mathcal{B}'_{\mathrm{st}}}(J, x, \mathcal{I}) = \mathcal{I}(x)$. □

Both of these equivalences show that we can choose which branch evaluation to use to find models.

3.5 Links between Different Justification Models

In the previous chapter, we have already proved that $\mathcal{B}_{\text{KK}}$ and $\mathcal{B}_{\text{wf}}$ have a single model if the interpretation of the opens is fixed. In this section, we show the relation between the models of the main justification semantics. Many results in this section rely on Theorem 2.6.20. To get started, we have that every $\mathcal{B}_{\text{KK}}$-model is a $\mathcal{B}_{\text{sp}}$-model.

Proposition 3.5.1. *A $\mathcal{B}_{\text{KK}}$-model is a $\mathcal{B}_{\text{sp}}$-model.*

Proof. Let $\mathcal{I}$ be a $\mathcal{B}_{\text{KK}}$-model of $\mathcal{JF}$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\text{KK}}$, there is a justification J such that $\text{val}_{\mathcal{B}_{\text{KK}}}(J, x, \mathcal{I}) = \text{SV}_{\mathcal{B}_{\text{KK}}}(x, \mathcal{I}) = \mathcal{I}(x)$. By Lemma 3.4.2, it suffices to prove that $\text{val}_{\mathcal{B}_{\text{sp}}}(J, x, \mathcal{I}) \geq_t \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$.

Since $\mathcal{B}_{\text{KK}}$ is transitive, we have for every node y reachable from x in J that $\mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{KK}}}(J, x, \mathcal{I}) \leq_t \text{val}_{\mathcal{B}_{\text{KK}}}(J, y, \mathcal{I}) = \mathcal{I}(y)$. The branch evaluation $\mathcal{B}_{\text{sp}}$ always maps to a node reachable from the start node. Therefore, $\text{val}_{\mathcal{B}_{\text{sp}}}(J, x, \mathcal{I}) \geq_t \mathcal{I}(x)$, which concludes the proof. □

If the interpretation of the opens is fixed, the unique $\mathcal{B}_{\text{KK}}$-model is the least precise $\mathcal{B}_{\text{sp}}$-model.

Proposition 3.5.2. *If the interpretations of the opens is fixed, the unique $\mathcal{B}_{\text{KK}}$-model is the $\leq_p$-least $\mathcal{B}_{\text{sp}}$-model.*

Proof. Let $\mathcal{I}_{\mathcal{B}_{\text{KK}}}$ be the unique $\mathcal{B}_{\text{KK}}$-model. Take a $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}$ and assume by contradiction that $\mathcal{I}_{\mathcal{B}_{\text{KK}}} \not\leq_p \mathcal{I}$. This means there is an $x \in \mathcal{F}_d$ such that $\mathcal{I}_{\mathcal{B}_{\text{KK}}}(x) \neq \mathbf{u}$ and $\mathcal{I}_{\mathcal{B}_{\text{KK}}}(x) \neq \mathcal{I}(x)$. Because the supported value under $\mathcal{B}_{\text{KK}}$ only depends on the interpretation of the opens, we have that $\mathbf{u} \neq \mathcal{I}_{\mathcal{B}_{\text{KK}}}(x) = \text{SV}_{\mathcal{B}_{\text{KK}}}(x, \mathcal{I}_{\mathcal{B}_{\text{KK}}}) = \text{SV}_{\mathcal{B}_{\text{KK}}}(x, \mathcal{I})$. If $\text{SV}_{\mathcal{B}_{\text{KK}}}(x, \mathcal{I}) = \mathbf{t}$, then there is a justification J with x as internal node without infinite branches. This means that $\text{SV}_{\mathcal{B}'_{\text{sp}}}(x, \mathcal{I}) = \mathbf{t}$. Then by Proposition 5.3.5, $\mathcal{I}(x) = \text{SV}_{\mathcal{B}_{\text{sp}}}(x, \mathcal{I}) = \mathbf{t}$, which is a contradiction. So we can assume that $\text{SV}_{\mathcal{B}_{\text{KK}}}(x, \mathcal{I}) = \mathbf{f}$. In that case, any justification has a false branch, which under $\mathcal{B}_{\text{KK}}$ means a finite branch ending in a false branch. Again, this proves that $\text{SV}_{\mathcal{B}'_{\text{sp}}}(x, \mathcal{I}) = \mathbf{f}$. Then by Proposition 5.3.5, $\mathcal{I}(x) = \text{SV}_{\mathcal{B}_{\text{sp}}}(x, \mathcal{I}) = \mathbf{f}$, which is a contradiction. This proves that $\mathcal{I}_{\mathcal{B}_{\text{KK}}} \leq_p \mathcal{I}$. □

All $\mathcal{B}_{\text{st}}$-models are $\mathcal{B}_{\text{sp}}$-models.

Proposition 3.5.3. *Every $\mathcal{B}_{\text{st}}$-model is a $\mathcal{B}_{\text{sp}}$-model.*

Proof. Take a $\mathcal{B}_{\mathrm{st}}$-model $\mathcal{I}$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\mathrm{st}}$, there is a justification J such that $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J,x,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x,\mathcal{I}) = \mathcal{I}(x)$. Let y be a child of x in J. We prove that $\mathcal{I}(x) \leq_t \mathcal{I}(y)$. If $y \in \mathcal{F}_o$, then by definition of J, we have that $\mathcal{I}(x) \leq_t \mathcal{I}(y)$. If $y \in \mathcal{F}_d$, there are two possibilities: If y has a different sign than x, then by definition of J, we have $\mathcal{I}(x) \leq_t \mathcal{I}(y)$. If y has the same sign as x, then for every $\mathbf{b} \in B_J(y)$ we have that $x \to \mathbf{b}$ is in $B_J(x)$ with $\mathcal{B}(\mathbf{b}) = \mathcal{B}(x \to \mathbf{b})$. Therefore, $\mathcal{I}(y) = \mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J,y,\mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J,x,\mathcal{I}) = \mathcal{I}(x)$. Therefore, we have proved that $\mathrm{val}_{\mathcal{B}_{\mathrm{sp}}}(J,x,\mathcal{I}) \geq_t \mathcal{I}(x)$; hence $\mathrm{SV}_{\mathcal{B}_{\mathrm{sp}}}(x,\mathcal{I}) \geq_t \mathcal{I}(x)$. Then, by Lemma 3.4.2, $\mathcal{I}$ is a $\mathcal{B}_{\mathrm{sp}}$-model. □

Well-founded models are also stable models as shown by the following proposition.

Proposition 3.5.4. *Any $\mathcal{B}_{\mathrm{wf}}$-model of $\mathcal{JF}$ is a $\mathcal{B}_{\mathrm{st}}$-model of $\mathcal{JF}$.*

Proof. Let $\mathcal{I}$ be a $\mathcal{B}_{\mathrm{wf}}$-model of $\mathcal{JF}$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\mathrm{wf}}$, there is a justification J such that $\mathrm{val}_{\mathcal{B}_{\mathrm{wf}}}(J,x,\mathcal{I}) = \mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x,\mathcal{I}) = \mathcal{I}(x)$. By Lemma 3.4.2, it suffices to prove that $\mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x,J,\mathcal{I}) \geq_t \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$.

For any internal node y reachable from x in J we have that $\mathcal{I}(x) \leq_t \mathcal{I}(y)$. Indeed, since $\mathcal{B}_{\mathrm{wf}}$ is transitive, we have that for every $\mathbf{b} \in B_J(y)$ there is a branch $\mathbf{b}'$ in $B_J(x)$ so that $\mathcal{B}(\mathbf{b}) = \mathcal{B}(\mathbf{b}')$. This means that $\mathcal{I}(y) = \mathrm{val}_{\mathcal{B}_{\mathrm{wf}}}(J,y,\mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}_{\mathrm{wf}}}(J,x,\mathcal{I}) = \mathcal{I}(x)$.

For every $\mathbf{b}$ in $B_J(x)$ we have that $\mathcal{B}_{\mathrm{st}}(\mathbf{b})$ is mapped to an element in $\mathbf{b}$ or that $\mathcal{B}_{\mathrm{st}}(\mathbf{b}) = \mathcal{B}_{\mathrm{wf}}(\mathbf{b})$. In both cases, we know that $\mathcal{I}(x) \leq_t \mathcal{I}(\mathcal{B}_{\mathrm{st}}(\mathbf{b}))$. This means that $\mathcal{I}(x) \leq_t \mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J,x,\mathcal{I}) \leq_t \mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x,\mathcal{I})$. This concludes the proof that $\mathcal{I}$ is a stable model of $\mathcal{JF}$. □

Of course, this means that well-founded models are supported models.

Corollary 3.5.5. *Any $\mathcal{B}_{\mathrm{wf}}$-model is a $\mathcal{B}_{\mathrm{sp}}$-model.*

Proof. Follows from Propositions 3.5.3 and 3.5.4. □

If the interpretation of the opens is fixed, then there is only one well-founded model, which is also a stable model. The following results states that in a stable model, the well-founded supported value is the same in some cases.

Lemma 3.5.6. *Let $\mathcal{I}$ be a $\mathcal{B}_{\mathrm{st}}$-model. If $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x,\mathcal{I}) = \mathbf{f}$, then $\mathcal{I}(x) = \mathbf{f}$. If $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x,\mathcal{I}) = \mathbf{t}$, then $\mathcal{I}(x) = \mathbf{t}$.*

Proof. For the first, it suffices to prove that $\mathcal{I}(x) \geq_t \mathbf{u}$ implies $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x, \mathcal{I}) \geq_t \mathbf{u}$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\mathrm{st}}$, there is a justification J such that $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, y, \mathcal{I}) = \mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(y, \mathcal{I}) = \mathcal{I}(y)$ for all defined facts y.

So take a defined x with $\mathcal{I}(x) \geq_t \mathbf{u}$. The only way that we have $\mathrm{val}_{\mathcal{B}_{\mathrm{wf}}}(J, x, \mathcal{I}) = \mathbf{f}$ is when $B_J(x)$ contains a branch $\mathbf{b}$ with a positive tail. Let $y_1, \ldots, y_n$ be the sign switches in $\mathbf{b}$ and $y_0 = x$. It is easy to see that $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, y_n, \mathcal{I}) = \mathbf{f}$ since $B_J(y_n)$ contains a positive branch. For i with $0 \leq i < n$ we have that $\mathcal{I}(y_i) = \mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(y_i, \mathcal{I}) = \mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, y_i, \mathcal{I}) \leq_t \mathcal{I}(y_{i+1})$. Therefore, we obtained that $\mathcal{I}(x) = \mathbf{f}$, which is a contradiction. Therefore, we have that $\mathcal{I}(x) \geq_t \mathbf{u}$ implies that $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x, \mathcal{I}) \geq_t \mathbf{u}$.

The second statement follows by the consistency of $\mathcal{B}_{\mathrm{wf}}$: if $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x, \mathcal{I}) = \mathbf{t}$, then $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(\sim x, \mathcal{I}) = \mathbf{f}$. By the first statement, this means that $\mathcal{I}(\sim x) = \mathbf{f}$; hence $\mathcal{I}(x) = \mathbf{t}$. □

This results implies that the unique well-founded model with a given interpretation of the opens is the least precise stable model with the same interpretation of the opens.

Proposition 3.5.7. *If the interpretation of the opens is fixed, the unique $\mathcal{B}_{\mathrm{wf}}$-model is the $\leq_p$-least $\mathcal{B}_{\mathrm{st}}$-model.*

Proof. Let $\mathcal{I}_{\mathcal{B}_{\mathrm{wf}}}$ be the unique $\mathcal{B}_{\mathrm{wf}}$-model and let $\mathcal{I}$ be any $\mathcal{B}_{\mathrm{st}}$-model. Lemma 3.5.6 states that if $\mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x, \mathcal{I}) \neq \mathbf{u}$, then $\mathcal{I}(x) = \mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x, \mathcal{I})$. Therefore, $\mathcal{I}_{\mathcal{B}_{\mathrm{wf}}}(x) = \mathrm{SV}_{\mathcal{B}_{\mathrm{wf}}}(x, \mathcal{I}) \leq_p \mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x, \mathcal{I}) = \mathcal{I}(x)$. Hence, $\mathcal{I}_{\mathcal{B}_{\mathrm{wf}}} \leq_p \mathcal{I}$ for all stable models $\mathcal{I}$. Then Proposition 3.5.4 finishes the proof. □

This shows that justification theory can capture least fixpoint semantics. Similar results exist for $\mathcal{B}_{\mathrm{cst}}$ and $\mathcal{B}_{\mathrm{cwf}}$.

Proposition 3.5.8. *Every $\mathcal{B}_{\mathrm{cwf}}$-model is a $\mathcal{B}_{\mathrm{cst}}$-model. Every $\mathcal{B}_{\mathrm{cst}}$-model is a $\mathcal{B}_{\mathrm{sp}}$-model. If the interpretation of the opens is fixed, then the unique $\mathcal{B}_{\mathrm{cwf}}$-model is the $\leq_p$-least $\mathcal{B}_{\mathrm{cst}}$-model.*

Proof. The proof that every $\mathcal{B}_{\mathrm{cst}}$-model is a $\mathcal{B}_{\mathrm{sp}}$-model is exactly the same as the proof of Proposition 3.5.3. The proof of Proposition 3.5.4 is also a valid for proving that every $\mathcal{B}_{\mathrm{cwf}}$-model is a $\mathcal{B}_{\mathrm{cst}}$-model. Lemma 3.5.6 also holds for $\mathcal{B}_{\mathrm{cst}}$ and $\mathcal{B}_{\mathrm{cwf}}$. And the proof of Proposition 3.5.7 also carries to $\mathcal{B}_{\mathrm{cst}}$ and $\mathcal{B}_{\mathrm{cwf}}$. □

Proposition 3.5.9. *If the interpretation of the opens is fixed, then $\mathcal{B}_{\mathrm{st}}$-models are $\leq_t$-minimal $\mathcal{B}_{\mathrm{sp}}$-models.*

Proof. Let $\mathcal{I}$ be a $\mathcal{B}_{\text{st}}$-model. Assume by contradiction that there is a $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}'$ with $\mathcal{I}' <_t \mathcal{I}$. This means that for all $y \in \mathcal{F}_+$, $\mathcal{I}'(x) \leq_t \mathcal{I}(x)$ and for all $y \in \mathcal{F}_-$, $\mathcal{I}'(x) \geq_t \mathcal{I}(x)$. Since $\mathcal{I}' \neq \mathcal{I}$, there exist an $x \in \mathcal{F}_- \cap \mathcal{F}_d$ with $\mathcal{I}'(x) >_t \mathcal{I}(x)$. Let $\ell = \mathcal{I}'(x)$. Since $\mathcal{I}'$ is a $\mathcal{B}_{\text{sp}}$-model, there is a rule $x \leftarrow A$ so that for all $a \in A, \mathcal{I}'(a) \geq_t \ell$. Similarly, such rules exists for $a \in A$. By induction, these rules can be combined into a justification J so that for all facts y in J we have $\mathcal{I}'(y) \geq_t \ell$. This justification does not have infinite positive branches starting with x, since x is negative. Therefore, $\text{val}_{\mathcal{B}_{\text{st}}}(J, x, \mathcal{I}') \geq_t \ell = \mathcal{I}'(x)$. All branches of J are either mapped to a positive fact or an open fact. For positive facts y, we know that $\mathcal{I}'(y) \leq_t \mathcal{I}(y)$ and for open facts, $\mathcal{I}'(y) = \mathcal{I}(y)$. This means that $\text{val}_{\mathcal{B}_{\text{st}}}(J, x, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}_{\text{st}}}(J, x, \mathcal{I}')$. Now, $\mathcal{I}$ is a $\mathcal{B}_{\text{st}}$-model; hence $\mathcal{I}(x) \geq_t \text{val}_{\mathcal{B}_{\text{st}}}(J, x, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}_{\text{st}}}(J, x, \mathcal{I}') \geq_t \mathcal{I}'(x)$, that is, $\mathcal{I}(x) \geq_t \mathcal{I}'(x)$. This contradicts that $\mathcal{I}'(x) >_t \mathcal{I}(x)$; hence there is no $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}'$ with $\mathcal{I}' <_t \mathcal{I}$. □

Proposition 3.5.10. *If the interpretation of the opens is fixed, then $\mathcal{B}_{\text{cst}}$-models are $\leq_t$-maximal $\mathcal{B}_{\text{sp}}$-models.*

Proof. Let $\mathcal{I}$ be a $\mathcal{B}_{\text{cst}}$-model. Assume by contradiction that there is a $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}'$ with $\mathcal{I}' >_t \mathcal{I}$. This means that for all $y \in \mathcal{F}_+$, $\mathcal{I}'(x) \geq_t \mathcal{I}(x)$ and for all $y \in \mathcal{F}_-$, $\mathcal{I}'(x) \leq_t \mathcal{I}(x)$. Since $\mathcal{I}' \neq \mathcal{I}$, there exists an $x \in \mathcal{F}_+ \cap \mathcal{F}_d$ with $\mathcal{I}'(x) >_t \mathcal{I}(x)$. Let $\ell = \mathcal{I}'(x)$. Since $\mathcal{I}'$ is a $\mathcal{B}_{\text{sp}}$-model, there is a rule $x \leftarrow A$ so that for all $a \in A, \mathcal{I}'(x) \geq_t \ell$. Similarly, such rules exists for $a \in A$. By induction, these rules can be combined into a justification J so that for all facts y in J we have $\mathcal{I}'(y) \geq_t \ell$. This justification does not have infinite negative branches starting with x since x is positive. Therefore, $\text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}') \geq_t \ell = \mathcal{I}'(x)$. All branches of J are either mapped to a negative fact or an open fact. For negative facts y, we know that $\mathcal{I}'(y) \leq_t \mathcal{I}(y)$ and for open facts $\mathcal{I}'(y) = \mathcal{I}(y)$. This means that $\text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}')$. Now, $\mathcal{I}$ is a $\mathcal{B}_{\text{cst}}$-model; hence $\mathcal{I}(x) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}') \geq_t \mathcal{I}'(x)$, that is, $\mathcal{I}(x) \geq_t \mathcal{I}'(x)$. This contradicts that $\mathcal{I}'(x) <_t \mathcal{I}(x)$; hence there is no $\mathcal{B}_{\text{sp}}$-model $\mathcal{I}'$ with $\mathcal{I}' >_t \mathcal{I}$. □

3.6 Conclusion

Proving that a particular branch evaluation is graph-like consistent is a difficult task. This is even true for tree-like justifications as shown by the length of the proofs in the PhD thesis of Denecker (1993). There are many different ways to go about proving it. One way is to directly construct a good justification for x if $\text{SV}(\sim x, \mathcal{I}) = \mathbf{f}$. It requires a great deal of care to make sure everything works out with your construction. We used the pasting result from the previous chapter numerous times for proving that the main branch evaluations are graph-like consistent. And we used it even more in the sections on the relations

between the various semantics. These results, however, are important for the significance of justification semantics. We have seen that splittable branch evaluations have a great deal of interesting properties that can be exploited to construct justifications. However, as illustrated by $\mathcal{B}_{\text{ex}}$, splittability does not imply consistency or graph-reducibility. In particular, we have not yet identified properties on branch evaluations that imply consistency. In the next chapter, we take a look at a few properties that do this, but only in a finite setting, which for practical applications often suffices.

In this chapter, we also settled the differences between alternative definitions of supported and stable branch evaluations. Apart from that, we showed a number of relations between different justification semantics. Most of these results are already known in other fields such as logic programming. However, we should note that these relations in justification theory are deeper. This is because justification theory is more general than any semantics it can capture due to the presence of open facts, and because justification theory allows for an infinite number of rules for a fact with potentially an infinite number of elements in the body.

Many of the proofs were very similar, so some future work might analyse these proofs and extract some properties between two branch evaluations $\mathcal{B}_1$ and $\mathcal{B}_2$, so that every $\mathcal{B}_1$-model is a $\mathcal{B}_2$-model. Something else that can be investigated is if $\mathcal{B}_1$ is parametric, then when fixing the interpretation of the opens will the unique $\mathcal{B}_1$-model be a $\leq_p$-least $\mathcal{B}_2$-model, akin to the relation between $\mathcal{B}_{\text{sp}}$ and $\mathcal{B}_{\text{KK}}$, between $\mathcal{B}_{\text{st}}$ and $\mathcal{B}_{\text{wf}}$, and between $\mathcal{B}_{\text{cst}}$ and $\mathcal{B}_{\text{cwf}}$.

Chapter 4

Exploiting Game Theory for Analysing Justifications

4.1 Introduction

In the previous chapter, we proved the graph-like consistency of some important branch evaluations. The proofs, however, are rather complex. This means that if a new branch evaluation is invented, quite a bit of work has to be done to ensure the associated semantics is consistent. What we would like instead is an easy to check condition on the branch evaluation such that consistency is ensured. The results in this chapter 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 chapter are as follows: 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-like consistency holds. While this means that our results do not apply, e.g., to infinitary semantics, for practical applications finiteness usually suffices. Finally, we show that all branch evaluations corresponding to the major semantics of logic programming are indeed selective and monotone. As a consequence, with our results, proving that a branch evaluation is graph-like consistent is much easier: all that is

needed is to show that it is monotone and selective. This is in sharp contrast with the proofs of graph-like consistency of Chapter 3 and the work on tree-like justification by Denecker (1993), which span at least a couple of pages for each semantics.

As justification theory has its root in logic programming, this chapter establishes a link between logic programming and game theory. This chapter is not the first time that a relation between logic programming and game theory is established. However, the relation in this chapter is different than existing results. The discussion on related work is postponed to the conclusion of this chapter.

The text of this chapter is solely based on (Marynissen et al., 2020), but the proofs and a few examples are new material. A few results in case of infinite games are also added.

4.2 Game Theory

Game theory is a very versatile field. Many different types of games are considered. In this section, we define the games we are considering: antagonistic games over graphs (Gimbert and Zielonka, 2005). Our formalisation slightly differs from the one of Gimbert and Zielonka (2005), but the differences are minor and non-fundamental. Intuitively a game unfolds as follows: There are two players **T** and **F**, with opposite interests. These players are called the *true* and *false player*. Let G be a directed 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 4.2.1. 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.

Players do not necessarily alternate plays, but in what follows we will assume that. Note that paths of a game graph are reminiscent of branches in a

justification frame. When this game is played, the trajectory of the stone is a path of special interest.

Definition 4.2.2. 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.

Note that a play of this game can be infinite and thus cannot be played in real life. 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. We will see later that plays correspond to branches. The true player would like branches to be evaluated to **t** and the false player would like branches to be evaluated to **f**.

Definition 4.2.3. 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.

Intuitively, if $p \sqsubseteq_P q$, then the player P prefers the play q over 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 $\sqsubseteq_P$. Since the players **T** and **F** have opposite interests, we will assume that $p \sqsubseteq_{\mathbf{T}} q$ if and only if $q \sqsubseteq_{\mathbf{F}} p$. A game graph combined with the preference relation for the players is setup for our games.

Definition 4.2.4. 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 **T**. We define $\sqsubseteq_{\mathbf{F}}$ to be the inverse relation of $\sqsubseteq_{\mathbf{T}}$.

Two-player games in which the players have exactly opposite goals are called *antagonistic*. When playing, players usually follow some strategy to try getting a play that is as preferable as possible. That is, their incentive is to get a play that is most preferable. In game theory, many different type of strategies exist. We are interested in just two types of strategies. The first type takes into account the entire play history to determine the next move. These strategies are called *general*. For the second type of strategy, the next move is made solely on the information where the stone is currently at. Since the strategy only depends on the position of the stone, they are called *positional* strategies. To execute such a strategy you do not need to remember the previous states the stone has been. Therefore, they are also called *memoryless* strategies. Note that a positional strategy is also a general strategy, but a less 'smart' one since it just does not use the memory of the previous states the stone has visited.

The difference between these two types of strategies is very similar to the distinction between graph-like and tree-like justifications. Indeed, every graph-like justification is also a tree-like justifications. Also, storing the strategy in

memory, the positional ones will take much less space just as with graph-like justifications. Let us now formalise these strategies.

Definition 4.2.5. 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 $\sigma_P : \{p \in \mathrm{Path}_G \mid \mathrm{target}(p) \in S_P\} \to E$ such that $\sigma_P(p) \in \mathrm{target}(p)E$. 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 $\mathfrak{S}_p(P)$.

As mentioned before a positional strategy can be seen as a general strategy. In formal terms, it goes as follows. 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 $\mathfrak{S}_p(P) \subseteq \mathfrak{S}_g(P)$.

When a player sticks to a strategy, many different plays can result because of the other player. Such plays are called *consistent* with that strategy. This is formalised below.

Definition 4.2.6. 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 $s_i \in S_P$.

If both players stick to their strategy, then there is only one play that is consistent with both strategies. This is because strategies are deterministic. Given a state s and strategies σ and τ for players $\mathbf{T}$ and $\mathbf{F}$, there exists a unique play in G starting with s consistent with both σ and τ. We denote this play by $p_G(s, \sigma, \tau)$. If we look at all the plays starting with a state s consistent with a strategy, then they form interesting structures.

Definition 4.2.7. 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 with 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 with x consistent with σ_p.

The observant reader might already have noticed that this is very similar to the distinction between tree-like and graph-like justifications. This is further worked out in the next section.

Under the assumption that $\mathbf{T}$ and $\mathbf{F}$ are rational agents, trying to maximise the value of the resulting play in their preference relation ($\sqsubseteq_{\mathbf{T}}$ respectively $\sqsubseteq_{\mathbf{F}}$),

some strategies will not be followed. For example, if player **T** follows a strategy σ that is good if **F** follows strategy τ, but very bad if **F** follows strategy τ', then **T** will not follow that strategy because he knows that **F** will follow τ'. Taking this idea to the next level, the players tend toward strategies such that neither player has an incentive to unilaterally deviate from their strategy. In game theoretic terms, these are called *Nash equilibria* and are often regarded as solutions to games. Note that the strategies we consider completely determine how the game is played. In regular game theory, these are called *pure* strategies. Often, Nash equilibria are so-called *mixed* strategies, which are assignments of probabilities to pure strategies. In this text, we only consider pure strategies. These pure Nash equilibria are formally defined as follows.

Definition 4.2.8. 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

$$p_G(s, \sigma, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(s, \sigma^*, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(s, \sigma^*, \tau).$$

As already mentioned, in an optimal pair (σ^*, τ^*) of strategies, no player can benefit by changing only their strategy. So if **T** changes their strategy σ^* to σ, then $p_G(s, \sigma, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(s, \sigma^*, \tau^*)$, or that $p_G(s, \sigma, \tau^*) \sqsupseteq_{\mathbf{F}} p_G(s, \sigma^*, \tau^*)$; hence **F** has an advantage. Similarly, if **F** changes their strategy τ^* to τ, then $p_G(s, \sigma^*, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(s, \sigma^*, \tau)$, and thus **T** has an advantage.

Finding optimal pairs has been a common thread in game theory. This is no different for us. In this chapter, we are most interested in the existence of such pairs, not per se the actual values of the strategies. In particular, we are interested whether an optimal pair of positional strategies exist. We mean an optimal pair (σ^*, τ^*) of strategies like the definition above such that σ^* and τ^* are positional (recall that every positional strategy is a general one as well).

4.3 Justifications as Strategies

In this section, we 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. This then translates to the relation between supported values and the value of the play determined by an optimal pair of strategies, which serves as a tool to shed light on the consistency problem from a different angle.

The following serves as a running example for this section.

Example 4.3.1. 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

$$\left\{ \begin{array}{l} p \leftarrow \sim q \\ q \leftarrow \sim p \\ p \leftarrow r \end{array} \right\}.$$

▲

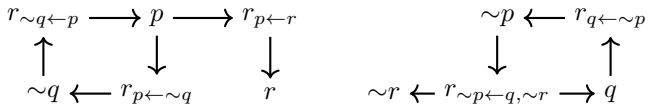
4.3.1 Games Associated to justifications

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 4.3.2. 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 $(r_{x \leftarrow A}, y)$.

Notice that the open facts are also owned by **T**, but these do not have outgoing edges in the game graph, so essentially they are not owned by any player.

Example 4.3.3. For the justification frame from Example 4.3.1, the game graph (without isolated nodes) is visualised below.



▲

It is worth noting that any play in such a game will alternate between rule symbols and elements in $\mathcal{F}$, and so the players **T** and **F** alternate turns. The outgoing edges of a defined fact x will go to rule symbols with x as head. Similarly, the outgoing edges of a rule symbol $r_{x \leftarrow A}$ goes to elements in the body of the corresponding rule.

Therefore, 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. When the strategies

for both **T** and **F** are chosen, essentially the true player will be choosing a justification, while the false player chooses a branch in that justification.

Any play in the game graph corresponds to a $\mathcal{JF}$-branch as follows: Let $s_0 s_1 \dots$ 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 $\mathbf{b}_p$. We can use these branches to define a preference relation for the player **T**, which together with the game graph defines a game.

Definition 4.3.4. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system and $\mathcal{I}$ an $\mathcal{F}$-interpretation. The *justification game* $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ 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 $\mathcal{I}(\mathcal{B}(\mathbf{b}_p)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}_q))$.

So far, we have shown only that a justification system can be transformed into an antagonistic game. Next, we establish semantic relations.

4.3.2 Strategies are Justifications

We have seen that strategies give rise to play graphs or trees. These structures are very similar to justifications except for the presence of rule symbols. This relation is explored in this section. 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. The following propositions make the relation between strategies for **T** and justifications precise.

Proposition 4.3.5. *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 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.*

Proof. Every leaf node of $J_\sigma(x)$ is not defined, because plays never stop in defined nodes; hence it is locally complete. We take an arbitrary internal node n with label y in $J_\sigma(x)$. This means that $y \in \mathcal{F}_d$. We prove that the set of

[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.

labels of the children of n are a case of y. By definition, $y \in S_{\mathbf{T}}$, hence there is only one outgoing transition from n in the play graph. This transition goes to a node m with a rule symbol $r_{y \leftarrow A} \in S_{\mathbf{F}}$ as label. This means that for all $a \in A$ that there is an edge in the play graph from m to a node n_a labelled a. This means for all $a \in A$ that (n, n_a) is an edge in $J_\sigma(x)$ and that these are the only edges starting with n, which proves that $J_\sigma(x)$ is a justification. Since the play graph/tree has a node labelled x as root, so does $J_\sigma(x)$. □

Example 4.3.6. Let $\mathcal{JF}$ be the justification frame from Example 4.3.1 and let σ be a positional strategy for $\mathbf{T}$ 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.

▲

The opposite also holds.

Proposition 4.3.7. *Any 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 $\mathbf{T}$.*

Proof. We define a strategy σ for $\mathbf{T}$. Let p be a finite path in $G_{\mathcal{JF}}$ such that $\text{source}(p) = x$ and $\text{target}(p) \in S_{\mathbf{T}}$. Then by filtering out the rule symbols, we get a path p^* in $\mathcal{JF}$. It suffices to define $\sigma(p)$ only for paths p such that p^* is a path in J. Since J is a justification, the set of labels of the children of $\text{target}(p)$ form a case A for the label y of $\text{target}(p)$. We define $\sigma(p) = r_{y \leftarrow A}$. It is clear that $J_\sigma(x)$ is equal to J. □

Before continuing let use note that the preference relations in games induced by justification systems, only have three equivalence classes. One for each value in $\mathcal{L}$. Therefore, to simply notation, we introduce a *utility function* defined as follows:

$$u(x, \sigma, \tau) \coloneqq \mathcal{I}(\mathcal{B}(\mathbf{b}_{p_G(x,\sigma,\tau)})).$$

If $u(x, \sigma, \tau) = \mathbf{t}$, then we say that $\mathbf{T}$ has won the game. If $u(x, \sigma, \tau) = \mathbf{f}$, then $\mathbf{F}$ wins the game. And the game 'ends' in a draw when $u(x, \sigma, \tau) = \mathbf{u}$.

The correspondence of justifications and strategies for $\mathbf{T}$ raises the question what the role of $\mathbf{F}$ is. Intuitively, $\mathbf{T}$ chooses for each node in the justification a set of children corresponding to a rule and $\mathbf{F}$ then chooses for each such node a single child. This means that $\mathbf{F}$ actually chooses a branch in the justification determined by a strategy of $\mathbf{T}$. Moreover, the value of that branch is equal to

$u(x, \sigma, \tau)$. The reverse is also true, any branch in $J_\sigma(x)$ corresponds to a strategy for $\mathbf{F}$. 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 must be elements of the same case). Taking these considerations into account, we can rephrase the value of a justification in terms of strategies, where $\mathbf{T}$ has to play positionally to obtain graph-like justifications but $\mathbf{F}$ is always allowed to use general strategies.

Lemma 4.3.8. *Let σ be a positional (respectively general) strategy for $\mathbf{T}$. For any interpretation $\mathcal{I}$ and $x \in \mathcal{F}_d$, it holds that* $\mathrm{val}(J_\sigma(x), x, \mathcal{I}) = \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau)$*, where* min *is with respect to $\leq_t$.*

Proof. Follows from the correspondence between general strategies for $\mathbf{F}$ and branches of $J_\sigma(x)$. □

Remark 4.3.9. Even for a positional strategy σ, it is the minimum over all general strategies τ. ▲

By Proposition 2.2.21, the supported value of x only depends on the value of locally complete justifications with x as root. Therefore, if every locally complete justification with x is of the form $J_\sigma(x)$ for some strategy σ for $\mathbf{T}$, then we can calculate the supported value of x only using the values of x in $J_\sigma(x)$ for strategies σ for $\mathbf{T}$. This is exactly what Proposition 4.3.7 states. As a consequence, the supported value can be computed using play values of strategies.

Theorem 4.3.10. *For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, the following holds*

$$\mathrm{SV}_t(x, \mathcal{I}) = \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau)$$

$$\mathrm{SV}_g(x, \mathcal{I}) = \max_{\sigma \in \mathfrak{S}_p(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau)$$

where max *and* min *are with respect to $\leq_t$.*

Proof. By Proposition 2.2.21 the supported value of x is equal to the maximum of the values of locally complete justifications with root x. By Proposition 4.3.7, these are exactly the justifications of the form $J_\sigma(x)$ with σ a strategy for $\mathbf{T}$. By using Lemma 4.3.8, we then conclude the proof. □

Strategies for F Are Also Justifications

We have seen how strategies for $\mathbf{T}$ correspond to justifications and that strategies for $\mathbf{F}$ 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 before, is a condition satisfied in all practical applications of justification theory. But in this case, we will demand the stronger property that our justification frames are fixed under complementation. Every complementary frame is equivalent to a justification frame fixed under complementation by Proposition 2.3.13. Therefore, demanding this will not reduce the strength of the result below.

Let τ be a strategy for **F**. This means that τ chooses for every rule symbol $r_{y \leftarrow A}$ an element from A. This corresponds to a selection function $\mathcal{S}$ for y. Since our justification frame is fixed under complementation, this selection function provides a rule for $\sim y$: $\sim y \leftarrow \sim \mathrm{Im}(\mathcal{S})$. Therefore, the strategy τ indirectly chooses a rule for every $\sim y$, which thus again induces a justification.

Proposition 4.3.11. *Let $\mathcal{JF}$ be a justification frame fixed under complementation 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 $\sim y$, note that $\sim(\sim y) = y$), is a 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.*

Proof. By construction, $J_\tau(x)$ is locally complete. Take an internal node n of $J_\tau(x)$ with label y, hence $y \in \mathcal{F}_d \cap S_{\mathbf{T}}$. This node corresponds to a node n^* with label $\sim y$. This means that in the play tree of τ there is an edge from n^* to a node $m^*_{\sim y \leftarrow A}$ with label $r_{\sim y \leftarrow A}$ for all rules of the form $\sim y \leftarrow A$. Since $r_{\sim y \leftarrow A} \in S_{\mathbf{F}}$, in the play tree of τ there is exactly one outgoing edge from $m^*_{\sim y \leftarrow A}$ to a node labelled $a \in A$. This defines a selection function $\mathcal{S}$ for $\sim y$. Therefore, in $J_\tau(x)$ the set of labels of the children of n is exactly $\sim \mathrm{Im}(\mathcal{S})$. Since $\mathcal{JF}$ is a complementary justification frame, it means that $J_\tau(x)$ is a justification. That $J_\tau(x)$ has $\sim x$ as root follows directly from the fact that x is a root of the play graph/tree. □

Example 4.3.12. Let $\mathcal{JF}$ be the justification frame from Example 4.3.1. 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.

▲

Again, this raises the question of what is the role of the strategy of $\mathbf{T}$ in this justification $J_\tau(x)$. In this correspondence, a strategy τ for $\mathbf{F}$ chooses for each x a rule $\sim x \leftarrow A$. A strategy σ for $\mathbf{T}$ on the other hand chooses for each $\sim x$ a rule $\sim x \leftarrow B$. Since $\mathcal{JF}$ is fixed under complementation, there is a selection function $\mathcal{S}$ of x such that $\mathrm{Im}(\mathcal{S}) = \sim B$. Hence σ chooses an element from B for every $x \leftarrow B$. In contrast, for every $y \in A$, we have by Proposition 3.2.1 that there is a rule $x \leftarrow B$ such that $\sim y \in B$. So the appropriate strategy σ for $\mathbf{T}$ will choose $\sim y$ from B. Hence we can view σ as choosing the negation of an element of the body of each rule $\sim x \leftarrow A$. Therefore, any general strategy σ for $\mathbf{T}$ correspond to the negation of a branch in $J_\tau(x)$ as formally shown by the following two lemmas.

Lemma 4.3.13. *Let $\mathcal{JF}$ be a justification frame fixed under complementation and τ a positional (respectively general) strategy for $\mathbf{F}$, and let σ be a general strategy for $\mathbf{T}$. The branch corresponding to the play $p_G(x, \sigma, \tau)$ is the negation of a branch in $J_\tau(x)$.*

Proof. Take a general strategy σ for $\mathbf{T}$. Let $p = s_0 s_1 s_2 \ldots$ be the play $p_G(x, \sigma, \tau)$. Since rule symbols in p occur at the odd indices, we need to prove that $s_0 s_2 s_4 \ldots$ is the negation of a $J_\tau(x)$ branch. Take an even i. Then $s_i = y$ and $s_{i+2} = z$ for some $y, z \in \mathcal{F}_d$. It suffices to prove that $\sim y \to \sim z$ is a path in $J_\tau(x)$. The strategy τ chooses for each rule symbol $r_{y \leftarrow A}$ an element $a \in A$. This is the same as a selection function $\mathcal{S}$ of y. Since $\mathcal{JF}$ is fixed under complementation, this means that $\sim y \leftarrow \sim \mathrm{Im}(\mathcal{S})$ is a rule. This is exactly the rule for $\sim y$ in $J_\tau(x)$. Therefore, it suffices to prove that $\sim z \in \sim \mathrm{Im}(\mathcal{S})$. The strategy σ maps $s_0 \ldots s_i$ to a rule $y \leftarrow B$, and the strategy τ maps $s_0 \ldots s_i r_{y \leftarrow B}$ to z; hence $z \in \mathcal{S}(y \leftarrow B)$. This means that $\sim z \in \sim \mathrm{Im}(\mathcal{S})$, finishing the proof. □

The negation of each branch in $J_\tau(x)$ can be constructed using plays consistent with τ.

Lemma 4.3.14. *Let $\mathcal{JF}$ be a justification frame fixed under complementation and τ a positional (respectively general) strategy for $\mathbf{F}$. Every branch in $J_\tau(x)$ starting with $\sim x$ is the negation of a branch corresponding to the play $p_G(x, \sigma, \tau)$ for some general strategy σ for $\mathbf{T}$.*

Proof. Take a $J_\tau(x)$-branch $\mathbf{b}$: $\sim x_0 \to \sim x_1 \to \cdots$ with $x_0 = x$. This corresponds to a path $p = x_0 r_{x_0 \leftarrow A_0} x_1 r_{x_1 \leftarrow A_1} \ldots$ in the play graph (tree) of τ. Note that $\tau(x_0 r_{x_0 \leftarrow A_0} \ldots x_i r_{x_i \leftarrow A_i}) = x_{i+1}$. Take any general strategy σ for $\mathbf{F}$ such that $\sigma(x_0 r_{x_0 \leftarrow A_0} \ldots x_{i-1} r_{x_{i-1} \leftarrow A_{i-1}} x_i) = r_{x_i \leftarrow A_i}$. Then p is equal to $p_G(x, \sigma, \tau)$, finishing the proof. □

These results allow us to view the value of $J_\tau(x)$ with respect to $\sim x$ in terms of strategies.

Lemma 4.3.15. *Let $\mathcal{JF}$ be a justification frame fixed under complementation and τ a positional (respectively general) strategy for $\mathbf{F}$. For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, it holds that* $\mathrm{val}(J_\tau(x), \sim x, \mathcal{I}) = \min_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \mathcal{I}(\mathcal{B}(\sim \mathbf{b}_{p_G(x,\sigma,\tau)}))$.

The value of $\mathcal{I}(\mathcal{B}(\sim \mathbf{b}_{p_G(x,\sigma,\tau)}))$ cannot be simplified further in general. However, when our branch evaluation respects negation we can simplify.

Lemma 4.3.16. *Let $\mathcal{JF}$ be a justification frame fixed under complementation, $\mathcal{B}$ a branch evaluation that respects negation, and τ a positional (respectively general) strategy for $\mathbf{F}$. For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, it holds that* $\mathrm{val}(J_\tau(x), \sim x, \mathcal{I}) = \sim \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$.

Proof. Since $\mathcal{B}$ respects negation, we have $\mathcal{I}(\mathcal{B}(\sim \mathbf{b}_{p_G(x,\sigma,\tau)})) = \sim u(x, \sigma, \tau)$. The result follows by Lemma 4.3.15 and by noting that $\sim \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau) = \min_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \sim u(x, \sigma, \tau)$. □

This result allows to view the supported value in yet a different light using the justification induced by strategies for $\mathbf{F}$. However, we first need to following result.

Proposition 4.3.17. *Let $\mathcal{JF}$ be a justification frame fixed under complementation. Any 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 $\mathbf{F}$.*

Proof. The proof is completely similar to the proof of Proposition 4.3.7. □

Completely analogously to Theorem 4.3.10, we thus obtain a method to compute the supported value of the negation of a fact in terms of strategies.

Theorem 4.3.18. *Let $\mathcal{JF}$ be a justification frame fixed under complementation and $\mathcal{B}$ a branch evaluation that respects negation. For any $x \in \mathcal{F}_d$ and interpretation $\mathcal{I}$, the following holds*

$$\mathrm{SV}_t(\sim x, \mathcal{I}) = \sim \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$$

$$\mathrm{SV}_g(\sim x, \mathcal{I}) = \sim \min_{\tau \in \mathfrak{S}_p(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$$

Proof. By Proposition 2.2.21 the supported value of $\sim x$ is equal to the maximum of the values of locally complete justifications with $\sim x$ as root. By Proposition 4.3.17, these are exactly the justifications of the form $J_\tau(x)$ with τ a strategy for $\mathbf{F}$. Using Lemma 4.3.16 and by noting that $\max_{\tau\in\mathfrak{S}(\mathbf{F})} \sim \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau) = \sim \min_{\tau\in\mathfrak{S}(\mathbf{F})} \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau)$ concludes the proof. □

4.4 Consistency Revisited

4.4.1 Minimax

In the previous section we related the supported value to play values. In particular, Theorem 4.3.10 provides a characterisation for $\mathrm{SV}(x,\mathcal{I})$ in terms of play values. On the other hand, Theorem 4.3.18 provides a characterisation for $\sim \mathrm{SV}(\sim x,\mathcal{I})$. The consistency of a justification system is exactly the equality between the two characterisations.

Corollary 4.4.1. *Let $\mathcal{JF}$ be a justification frame fixed under complementation. If for all $x \in \mathcal{F}_d$*

$$\max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} \min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau) = \min_{\tau\in\mathfrak{S}_g(\mathbf{F})} \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau),$$

then $\mathcal{JS}$ is tree-like consistent. If for all $x \in \mathcal{F}_d$

$$\max_{\sigma\in\mathfrak{S}_p(\mathbf{T})} \min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau) = \min_{\tau\in\mathfrak{S}_p(\mathbf{F})} \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau),$$

then $\mathcal{JS}$ is graph-like consistent.

Such equalities in game theory are called minimax problems. A famous theorem by John Nash states that every zero-sum two-player game satisfies such equalities. With zero-sum, we mean that one player's gain is equivalent to the other player's loss. Because the preference relation of $\mathbf{T}$ and $\mathbf{F}$ are opposite, our games can be considered zero-sum games. However, this theorem was proved in the context of mixed strategies. Furthermore, the equation for graph-like consistency above is not exactly a minimax problem due to slight differences between the index sets ($\mathfrak{S}_p$ vs $\mathfrak{S}_g$). One direction of the minimax is a fairly well-known mathematical identity:

$$\max_{a\in A} \min_{b\in B} f(a,b) \leq \min_{b\in B} \max_{a\in A} f(a,b).$$

This is analogous with the easy direction of the consistency problem, see Theorem 2.3.18. Minimax problems in game theory are equivalent to the

existence of optimal strategies. This is also the case for our games. Let us first show what the values of optimal plays are.

Proposition 4.4.2. *If a pair* (σ^*, τ^*) *of general strategies is optimal, then*

$$u(x, \sigma^*, \tau^*) = \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau) = \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau),$$

where max *and* min *are with respect to the order* $\leq_t$*. If* (σ^*, τ^*) *is also positional, then*

$$u(x, \sigma^*, \tau^*) = \max_{\sigma \in \mathfrak{S}_p(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau) = \min_{\tau \in \mathfrak{S}_p(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau).$$

Proof. Assume (σ^*, τ^*) is an optimal pair of strategies. By optimality we have that for all general strategies τ that $p_G(x, \sigma^*, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(x, \sigma^*, \tau)$. This is the same as $u(x, \sigma^*, \tau^*) \leq_t u(x, \sigma^*, \tau)$. Therefore,

$$u(x, \sigma^*, \tau^*) \leq_t \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma^*, \tau) \leq_t \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau).$$

On the other hand, by optimality we have for all general strategies σ that $p_G(x, \sigma, \tau^*) \sqsubseteq_{\mathbf{T}} p_G(x, \sigma^*, \tau^*)$, which is the same as $u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*)$. Therefore, $\max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*)$. We also have that $\min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau) \leq_t u(x, \sigma, \tau^*)$; hence that

$$\max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau) \leq_t \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*).$$

So we proved that $u(x, \sigma^*, \tau^*) = \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau)$.

By optimality we have that $u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*)$ for all general strategies σ. This means that $\max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*)$. We have that

$$\min_{\tau \in \mathfrak{S}_g(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau) \leq_t \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*).$$

Again by optimality we have for all general strategies τ that $u(x, \sigma^*, \tau^*) \leq_t u(x, \sigma^*, \tau)$. This means that $u(x, \sigma^*, \tau^*) \leq_t \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma^*, \tau)$. We also have that $u(x, \sigma^*, \tau) \leq_t \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$. Therefore,

$$u(x, \sigma^*, \tau^*) \leq_t \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma^*, \tau) \leq_t \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau).$$

This proves that $u(x, \sigma^*, \tau^*) = \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$.

Now suppose (σ^*, τ^*) is a positional pair of optimal strategies.

By optimality, we have that $\max_{\sigma\in\mathfrak{S}_p(\mathbf{T})} u(x,\sigma,\tau^*) \leq_t u(x,\sigma^*,\tau^*)$ because $\mathfrak{S}_p(\mathbf{T})$ embeds into $\mathfrak{S}_g(\mathbf{T})$. Therefore,

$$\max_{\sigma\in\mathfrak{S}_p(\mathbf{T})}\min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau) \leq_t \max_{\sigma\in\mathfrak{S}_p(\mathbf{T})} u(x,\sigma,\tau^*) \leq_t u(x,\sigma^*,\tau^*).$$

By optimality, $u(x,\sigma^*,\tau^*) \leq_t \min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma^*,\tau)$. Since σ^* is positional, we have that

$$u(x,\sigma^*,\tau^*) \leq_t \min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma^*,\tau) \leq_t \max_{\sigma\in\mathfrak{S}_p(\mathbf{T})}\min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau),$$

which proves that $u(x,\sigma^*,\tau^*) = \max_{\sigma\in\mathfrak{S}_p(\mathbf{T})}\min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma,\tau)$.

By optimality, $\max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau^*) \leq_t u(x,\sigma^*,\tau^*)$. Since τ^* is positional, we have

$$\min_{\tau\in\mathfrak{S}_p(\mathbf{F})}\max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau) \leq_t \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau^*) \leq_t u(x,\sigma^*,\tau^*).$$

Again by optimality we have that $u(x,\sigma^*,\tau^*) \leq_t \min_{\tau\in\mathfrak{S}_p(\mathbf{F})} u(x,\sigma^*,\tau)$ since $\mathfrak{S}_p(\mathbf{F})$ embeds into $\mathfrak{S}_g(\mathbf{F})$. Therefore,

$$u(x,\sigma^*,\tau^*) \leq_t \min_{\tau\in\mathfrak{S}_p(\mathbf{F})} u(x,\sigma^*,\tau) \leq_t \min_{\tau\in\mathfrak{S}_p(\mathbf{F})}\max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau),$$

which proves that $u(x,\sigma^*,\tau^*) = \min_{\tau\in\mathfrak{S}_p(\mathbf{F})}\max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau)$ □

These equations are exactly the equations from Corollary 4.4.1, so as a result we get that the existence of optimal pairs implies consistency.

Corollary 4.4.3. *Let $\mathcal{JF}$ be a justification frame fixed under complementation and $\mathcal{B}$ a branch evaluation that respects negation. The system $\mathcal{JS}$ is tree-like (graph-like) consistent if there is an optimal pair (σ^*,τ^*) of (positional) strategies.*

Proof. Follows directly from Proposition 4.4.2 and Corollary 4.4.1. □

4.4.2 Existence of Optimal Pairs of Positional Strategies

At the end of the previous section, we showed that if an optimal pair of positional strategies exist, then the justification system is graph-like consistent. Since graph-like consistency implies tree-like consistency and graph-reducibility, the existence of an optimal pair of positional strategies is paramount to solving our

research problems. In particular, we want conditions on our branch evaluation that guarantee the existence of positional optimal pairs. Inspired by Gimbert and Zielonka (2004, 2005), we will introduce two properties that branch evaluations should have. The first property is *monotonicity*. We are looking for positional strategies. In particular, we are looking for good positional strategies. For such a strategy to decide what to do in a state, it should not matter what happened before. So if the play so far was bad or good should not matter in the choice, i.e. if the play so far is good, then the decision should ruin this advantageous play and if the play so far is bad, the decision should not make it worse. This is exactly what monotonicity is about. In terms of branches it 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 quality of the branches are respected. In formal terms, it goes as follows.

Definition 4.4.4. A branch evaluation $\mathcal{B}$ is *monotone* if for every two branches $\mathbf{b}_1$ and $\mathbf{b}_2$ starting with the same fact x and all finite paths $\mathbf{p}$ the following holds:

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

Intuitively, this is some sort of 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).

Attentive readers might see a correlation with splittable branch evaluations. In fact, splittability implies monotonicity.

Proposition 4.4.5. *A splittable branch evaluation is monotone.*

Proof. Take branches $\mathbf{b}_1$ and $\mathbf{b}_2$ starting with x such that $\mathcal{I}(\mathcal{B}(\mathbf{b}_1)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}_2))$. Let $\mathbf{p}$ be any finite path. If $\mathbf{p} \to \mathbf{b}_1$ is decided at x, then $\mathbf{p} \to x$ is decided and $\mathcal{I}(\mathcal{B}(\mathbf{p} \to \mathbf{b}_1)) = \mathcal{I}(\mathcal{B}(\mathbf{p} \to x)) = \mathcal{I}(\mathcal{B}(\mathbf{p} \to \mathbf{b}_2))$. So assume it is not decided at x. Then by splittability, $\mathbf{p} \to \mathbf{b}_1$ is transitive at x. Similarly, $\mathbf{p} \to \mathbf{b}_2$ is transitive at x as well. Therefore $\mathcal{I}(\mathcal{B}(\mathbf{p} \to \mathbf{b}_1)) = \mathcal{I}(\mathcal{B}(\mathbf{b}_1)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}_2)) = \mathcal{I}(\mathcal{B}(\mathbf{p} \to \mathbf{b}_2))$. □

This means that we already have a number of monotone branch evaluations.

Corollary 4.4.6. *The branch evaluations* $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{st}}$, $\mathcal{B}_{\mathrm{wf}}$, $\mathcal{B}_{\mathrm{ex}}$, *and their duals are monotone.*

The second property we introduce is *selectivity*. The defining property of positional strategies is that they always make the same choice. For example, a positional strategy of **T** always moves the stone to the same rule symbol. Basically, it makes the same choice over and over, no matter how it got there. So to ensure that good positional strategies exist, we want that if always the same choice is made, the strategy does not become worse. So if a choice was made to go from s_1 to s_2 and later the stone returned back to s_1, we need that moving to s_2 remains the best choice. Similarly, we want that if a strategy is not positional we can improve it by making it positional.

In order to formally define selective branch evaluations, we need some extra notation. A *finite loop* starting with x is a finite path $\mathbf{p}$ starting with x such that every element is defined. If M is a set of loops starting with the same x, then 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 4.4.7. A branch evaluation $\mathcal{B}$ is *selective* with respect to $\mathcal{I}$ if for all finite paths p, all facts $x \in \mathcal{F}_d$, for all sets M, N of finite loops starting with x, for all sets K of branches starting with 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 $\mathcal{I}(\mathcal{B}(\mathbf{b}_*)) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b})) \leq_t \mathcal{I}(\mathcal{B}(\mathbf{b}^*))$.

One might expect to only see the existence of the branch $\mathbf{b}^*$. However, in justification theory we have a partition of positive and negative elements. In branch evaluations that respect negation (all examples so far) we would get the following: $\mathcal{I}(\mathcal{B}(\sim\mathbf{b})) \geq_t \mathcal{I}(\mathcal{B}(\sim\mathbf{b}^*))$.

Intuitively, selectivity means that the best strategy can be consistent with it choices. This is most easily seen in case M, N and K are singletons. 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 consistently, to obtain a branch that is at least as good ($\mathbf{b}^*$) and one that is at least as bad ($\mathbf{b}_*$).

All the branch evaluations we proved to be consistent so far are selective.

Proposition 4.4.8. *The branch evaluations $\mathcal{B}_{\text{sp}}$, $\mathcal{B}_{\text{KK}}$, $\mathcal{B}_{\text{st}}$, $\mathcal{B}_{\text{wf}}$, and their duals are selective.*

Proof.

$\mathcal{B}_{\text{sp}}$ We only need to look at the case that p is an empty path because $\mathcal{B}_{\text{sp}}$ is decided on the second element. So take a branch $\mathbf{b} \in (M \cup N)^* K \cup$

$(M \cup N)^\omega$. Since $\mathcal{B}_{\mathrm{sp}}(\mathbf{b})$ is the second element of $\mathbf{b}$, the value $\mathcal{I}(\mathcal{B}_{\mathrm{sp}}(\mathbf{b}))$ is already determined when choosing M, N or K. This proves that $\mathcal{B}_{\mathrm{sp}}$ is selective.

$\mathcal{B}_{\mathrm{KK}}$ Take $\mathbf{b} \in p(M \cup N)^* K \cup p(M \cup N)^\omega$. If $\mathbf{b}$ is infinite, then $\mathcal{B}_{\mathrm{KK}}(\mathbf{b}) = \mathbf{u}$. If K contains infinite branches, then for any branch $\mathbf{b}^*$ in pK it holds that $\mathcal{I}(\mathcal{B}_{\mathrm{KK}}(\mathbf{b}^*)) = \mathcal{I}(\mathcal{B}_{\mathrm{KK}}(\mathbf{b}))$. If K does not contain infinite branches, then either pM^ω or pN^ω is not empty. Therefore, there is a branch $\mathbf{b}^* \in pM^\omega \cup pN^\omega$ such that $\mathcal{I}(\mathcal{B}_{\mathrm{KK}}(\mathbf{b}^*)) = \mathcal{I}(\mathcal{B}_{\mathrm{KK}}(\mathbf{b}))$.

Assume now that $\mathbf{b}$ is finite. This means that the tail of $\mathbf{b}$ is contained in K. Then by transitivity of $\mathcal{B}_{\mathrm{KK}}$, the result follows.

$\mathcal{B}_{\mathrm{wf}}$ Take $\mathbf{b} \in p(M \cup N)^* K \cup p(M \cup N)^\omega$. If $\mathbf{b}$ is finite, then a tail $\mathbf{b}'$ of $\mathbf{b}$ lies in K. This means that $\mathcal{B}_{\mathrm{wf}}(\mathbf{b}) = \mathcal{B}_{\mathrm{wf}}(\mathbf{b}') = \mathcal{B}_{\mathrm{wf}}(p \to \mathbf{b}')$ by transitivity. So we can assume that $\mathbf{b}$ is infinite. We can also assume by reasoning like above, that no tail of $\mathbf{b}$ lies in K. This means that $\mathbf{b} \in p(M \cup N)^\omega$. If $\mathbf{b}$ has a tail of equal signs, then there is a loop in either M or N with the same sign. Therefore, there is a branch $\mathbf{b}^*$ in either pM^ω or pN^ω with the same value as $\mathbf{b}$. It rests us now the case that $\mathcal{B}_{\mathrm{wf}}(\mathbf{b}) = \mathbf{u}$. We distinguish three cases.

Case 1 Either M or N contains a mixed element. This means that there is an element in pM^ω or in pN^ω that is mapped to $\mathbf{u}$ under $\mathcal{B}_{\mathrm{wf}}$.

Case 2 Every element in M and N is completely positive or completely negative. Not all elements in M and N can be completely negative, since $\mathbf{b} \in p(M \cup N)^\omega$ and $\mathcal{B}_{\mathrm{wf}}(\mathbf{b}) = \mathbf{u}$. Therefore, there is a positive loop q^+ and a negative loop q^- in $M \cup N$. This means that $\mathbf{f} = \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(p(q^+)^\omega)) \leq_t \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(\mathbf{b})) \leq_t \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(p(q^-)^\omega)) = \mathbf{t}$.

$\mathcal{B}_{\mathrm{st}}$ Take a branch $\mathbf{b} \in p(M \cup N)^* K \cup p(M \cup N)^\omega$.

Case 1 The branch $\mathbf{b}$ is infinite and every element of $\mathbf{b}$ has the same sign. Then $\mathbf{b}$ has a tail in K, or there are loops in M or N with the same sign. Thus we have a branch $\mathbf{b}^* \in pM^\omega \cup pN^\omega \cup pK$ such that $\mathcal{I}(\mathcal{B}_{\mathrm{wf}}(\mathbf{b}^*)) = \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(\mathbf{b}))$.

Case 2 The branch $\mathbf{b}$ is finite and every element except maybe the last has the same sign. This means that $\mathbf{b}$ has a tail $\mathbf{b}'$ in K such that $\mathcal{I}(\mathcal{B}_{\mathrm{wf}}(\mathbf{b})) = \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(p \to \mathbf{b}'))$. It holds that $p \to \mathbf{b}' \in pK$.

Case 3 The branch $\mathbf{b}$ has a first sign switch before the last element. Assume y is the first sign switch in $\mathbf{b}$. Then either y is in a loop q_M in M, a loop q_N in N, or in a element q_K of K. In all cases, we get that $\mathcal{I}(\mathcal{B}_{\mathrm{wf}}(\mathbf{b})) = \mathcal{I}(y) = \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(p \to (q_M \to)^\omega)) = \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(p \to (q_N \to)^\omega)) = \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(p \to q_K))$.

□

The branch evaluation $\mathcal{B}_{\text{ex}}$ from Definition 2.3.19 is not selective. In Example 2.3.20, 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., $(a \to b \to a \to c \to)^\omega$.

The following proposition states if $\mathcal{B}$ is monotone and selective, then to find an optimal pair of positional strategies, it suffices to find optimal pairs of positional strategies in certain subgames, which breaks our problem down into smaller parts. These subgames are formed by splitting the outgoing edges of an state owned by $\mathbf{T}$ into two.

Proposition 4.4.9. *Let $x \in S_{\mathbf{T}}$ such that $|xE| > 1$. Let $xE = A_1 \cup A_2$ a partition of non-empty sets. Let $E_1 = E \setminus A_2$ Let $E_2 = E \setminus A_1$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the subgames of $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ with the same states as $G_{\mathcal{JF}}$ and edges E_1 respectively E_2. If $\mathcal{B}$ is monotone and selective and there are positional optimal pairs (σ_i^*, τ_i^*) in the games $\mathcal{G}_i$ for $i \in \{1, 2\}$, then there is an optimal pair (σ^*, τ^*) in $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ such that σ^* is positional.*

Proof. The proof is heavily based on the proof by Gimbert and Zielonka (2004).

Without loss of generality, we can assume that $u(x, \sigma_2^*, \tau_2^*) \leq_t u(x, \sigma_1^*, \tau_1^*)$. We prove that $\sigma^* \coloneqq \sigma_1^*$ is optimal for $\mathbf{T}$ in G, i.e. there is a τ^* such that (σ_1^*, τ^*) is optimal. We now construct a strategy τ^* for $\mathbf{F}$ such that for all strategies σ in G and all vertices y we have

$$u(y, \sigma, \tau^*) \leq_t u(y, \sigma_1^*, \tau_1^*). \tag{4.1}$$

Define a mapping $b : \text{Path}_G \to \{1, 2\}$. Let $p \in \text{Path}_G$ be a finite path in G. Set $b(p) = 1$ if either p does not contain any edge with the source x or the last edge of p with source x belongs to G_1 (A_1). Set $b(p) = 2$ if the last edge of p with the source v belongs to G_2 (A_2). Define $\tau^*(p) = \tau_i^*(\text{target}(p))$ if $b(p) = i$.

A finite or infinite path p is called *homogeneous* if p never visits x, or if each edge in p with source x belongs to E_1, or if each edge in p with source x belong to E_2.

The proof of Eq. (4.1) happens in 4 cases.

Case 1 $y = x$ and the play $p_G(y, \sigma, \tau^*)$ is of the form $p_0 p_1 \dots p_n q$, where p_i are finite non empty homogeneous paths with source x and q is a homogeneous play with source x.

Since p is consistent with τ^* and p_i are homogeneous, each play p_i^ω is either consistent with τ_1^* (if p_i contains only edges of G_1) or with τ_2^* (if p_i contains only edges of G_2). Assume without loss of generality that p_i^ω is consistent with τ_2^*. This means there is a σ' such that $p_i^\omega = p_{G_1}(x, \sigma', \tau_2^*)$; hence $u(p_i^\omega) = u(x, \sigma', \tau_2^*)$. By optimality, we get that $u(p_i^\omega) \leq_t u(x, \sigma_2^*, \tau_2^*)$. By assumption, we have that $u(p_i^\omega) \leq_t u(x, \sigma_2^*, \tau_2^*) \leq_t u(x, \sigma_1^*, \tau_1^*)$.

In the case that p_i^ω is consistent with τ_1^*, we also get that $u(p_i^\omega) \leq_t u(x, \sigma_1^*, \tau_1^*)$. Similarly, $u(q) \leq_t u(x, \sigma_1^*, \tau_1^*)$. By selectivity, we have that $u(x, \sigma, \tau^*)$ is smaller or equal to some $u(p_i^\omega)$ or $u(q)$. Then by the results above, we have that $u(x, \sigma, \tau^*) \leq_t u(x, \sigma_1^*, \tau_1^*)$.

Case 2 $y = x$ and the play $p_G(y, \sigma, \tau^*)$ is of the form $p_0 p_1 p_2 \ldots$, where p_i is homogeneous non-empty path with source x and $p_i p_{i+1}$ is not homogeneous.

Let M be the set of plays $p_{i_1} p_{i_2} \ldots$ such that $i_1, i_2, \ldots$ are even. Let N be the set of plays $p_{i_1} p_{i_2} \ldots$ such that $i_1, i_2, \ldots$ are odd. All plays in M are homogeneous and are either all consistent with τ_1^* or all consistent with τ_2^*. If all plays in M are all consistent with τ_i^*, then all plays in N are all consistent with τ_{3-i}^*. The plays in N are all homogeneous.

Assume without loss of generality that M is consistent with τ_2^*. By optimality of τ_2^*, we have that $u(q) \leq_t u(x, \sigma_2^*, \tau_2^*) \leq_t u(x, \sigma_1^*, \tau_1^*)$. By optimality of τ_1^*, we have that $u(q) \leq_t u(x, \sigma_1^*, \tau_1^*)$. By selectivity, we know that $u(x, \sigma, \tau^*)$ is smaller than or equal to $u(q)$ for some $q \in M \cup N$. Therefore, $u(x, \sigma, \tau^*) \leq_t u(x, \sigma_1^*, \tau_1^*)$.

Case 3 $y \neq x$ and the play $p \coloneqq p_G(y, \sigma, \tau^*)$ never passes through x. Let σ_1 be the restriction of σ to G_1. Then p is consistent with σ_1 and τ_1^* since $b(q) = 1$ for any initial segment of p. By optimality of τ_1^* we get $u(p) = u(y, \sigma_1, \tau_1^*) \leq_t u(y, \sigma_1^*, \tau_1^*)$.

Case 4 $y \neq x$ and the play $p \coloneqq p_G(y, \sigma, \tau^*)$ passes at least once the vertex x. We can factorise p as rq where r is the initial segment of p that does not contain x and the source of q is x. Let q^* be the play $p_{G_1}(x, \sigma_1^*, \tau_1^*)$. Of course $u(q^*) = u(x, \sigma_1^*, \tau_1^*)$. The play q is consistent with τ^* with source x, and thus by applying case 1 or case 2, we know that $u(q) \leq_t u(x, \sigma_1^*, \tau_1^*)$. This means that $u(q) \leq_t u(q^*)$. By monotonicity, we get that $u(p) = u(rq) \leq_t u(rq^*)$.

The play rq^* starts in y and is consistent with τ_1^*. By optimality of τ_1^*, we get that $u(rq^*) \leq_t u(y, \sigma_1^*, \tau_1^*)$. This proves that $u(p) \leq_t u(y, \sigma_1^*, \tau_1^*)$.

So we have proven that $u(y, \sigma, \tau^*) \leq_t u(y, \sigma_1^*, \tau_1^*)$ for all σ. The play $p_G(y, \sigma^*, \tau^*)$ does not have a transition in A_2 and thus is in G_1; hence $p_G(y, \sigma^*, \tau^*) =$

$p_G(y, \sigma_1^*, \tau_1^*)$. So we have proven that $u(y, \sigma, \tau^*) \leq_t u(y, \sigma^*, \tau^*)$ for all σ. This is the left-hand side of the optimality equation.

Take a general strategy τ. Let τ_1 be the restriction of τ to G_1. Since the play $p_G(y, \sigma^*, \tau)$ does not have a transition in A_2 since $\sigma^* = \sigma_1^*$, we have that this play is in G_1. Therefore, $u(y, \sigma^*, \tau) = u(y, \sigma_1^*, \tau_1)$. Now, by optimality of (σ_1^*, τ_1^*) we have that $u(y, \sigma_1^*, \tau_1^*) \leq_t u(y, \sigma_1^*, \tau_1)$, which shows that $u(y, \sigma^*, \tau^*) \leq_t u(y, \sigma^*, \tau)$. This concludes the proof that (σ^*, τ^*) is an optimal pair of strategies, where σ^* is positional. □

Completely analogous to Proposition 4.4.9 for outgoing edges of a state owned by **F** holds as well.

Proposition 4.4.10. *Let $x \in S_{\mathbf{F}}$ such that $|xE| > 1$. Let $xE = A_1 \cup A_2$ a partition of non-empty sets. Let $E_1 = E \setminus A_2$ Let $E_2 = E \setminus A_1$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the subgames of $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ with the same states as $G_{\mathcal{JF}}$ and edges E_1 respectively E_2. If $\mathcal{B}$ is monotone and selective and there are positional optimal pairs (σ_i^*, τ_i^*) in the games $\mathcal{G}_i$ for $i \in \{1, 2\}$, then there is an optimal pair (σ^*, τ^*) in $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ such that τ^* is positional.*

These two results can then be used to prove that the game $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ has an optimal pair of positional strategies if $\mathcal{B}$ is monotone and selective. If every state owned by **T** has at most one outgoing transition, then there is only one strategy σ for **T**. This strategy is positional and there is a general strategy τ for **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 4.4.11. *If $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ is finite (a finite amount of edges) and $\mathcal{B}$ is monotone and selective, then there is a positional optimal pair.*

Proof. We prove this by induction on the number edges. As base case we have games where every state has at most one outgoing edge. Because then every pair of strategies for **T** and **F** is optimal and positional. So assume now by induction that every subgame with fewer edges has an optimal pair of positional strategies. If every state owned by **T** has at most one outgoing edge, then there is only one strategy σ^* for **T**. This strategy is positional and there exists a general strategy τ such that (σ^*, τ) is optimal. If there is a state x owned by **T** such that $|xE| > 1$, then by using the induction assumption and Proposition 4.4.9 there exist a positional strategy σ^* and a general strategy τ such that (σ^*, τ) is optimal.

A similar reasoning can be done using Proposition 4.4.10, to get a general strategy σ and a positional strategy τ^* such that (σ, τ^*) is optimal. Therefore, also (σ^*, τ^*) is optimal, which concludes the proof. □

Corollary 4.4.12. *If $\mathcal{JS}$ is finite ($\mathcal{F}$ finite) and $\mathcal{B}$ is monotone and selective, then $\mathcal{JS}$ is graph-like consistent, tree-like consistent and graph-reducible.*

Proof. The underlying justification frame is equivalent to a justification frame fixed under complementation. This frame is also finite. Then the result follows from Theorem 4.4.11 and Corollaries 4.4.3 and 2.4.7. □

These results do not immediately prove for some known branch evaluation to be graph-like consistent as we already proved that $\mathcal{B}_{\mathrm{sp}}$, $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{st}}$, $\mathcal{B}_{\mathrm{wf}}$, and their duals are consistent and graph-reducible. However, when new branch evaluations arise, it can be an easy check to see if they behave correctly in a finite setting. For practical problems, the finiteness requirement does not pose any problem.[2]

4.5 Infinite Games

In this section we take a different look at the existence of optimal pairs, but limit ourselves to general optimal pairs. Let us first provide a different characterisation of optimal pairs.

Lemma 4.5.1. *A pair (σ^*, τ^*) of general strategies is optimal if and only if* $\max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau^*) = u(x, \sigma^*, \tau^*) = \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma^*, \tau)$.

[2] Finitists would even consider both of our research questions solved

Proof. This follows directly by noting that (σ^*, τ^*) is an optimal pair if and only if $u(x, \sigma, \tau^*) \leq_t u(x, \sigma^*, \tau^*) \leq_t u(x, \sigma^*, \tau)$ for all strategies σ and τ for **T** and **F**. □

Let us now introduce Gale-Stewart games (Gale and Stewart, 1953) as done by Martin (1975).

Definition 4.5.2. Let A be a set and Y a set of finite sequences in A such that every initial segment (including the empty one) of an element of Y belongs to Y. Let $\mathfrak{F}(Y)$ be the collection of infinite sequences all of whose finite initial segments belong to Y. For each $P \subseteq \mathfrak{F}(Y)$ we define a two-player game $\mathcal{G}_Y^P$. Two players, I and II, take turns by choosing an $a \in A$ such that all time, the elements chosen form a sequence in Y. This determines a sequence s in $\mathfrak{F}(Y)$. If $s \in P$, then player I wins, otherwise player II wins (there are no draws). The set P is called the *payoff set*.

Compared to the games associated with justification systems, Gale-Stewart games do not allow for a finite plays. However, this restriction can be alleviated, which we will discuss in a moment.

Since Gale-Stewart games are two-valued, it is customary to talk about winning strategies. A *winning strategy* for a player is a function that dictates which element from A the player has to choose at each point in the game in order to always win. That is, no matter what the other player does, the strategy guarantees the player to win.

Definition 4.5.3. A Gale-Stewart game $\mathcal{G}_Y^P$ is *determined* if some player has a winning strategy. In that case P is also said to be *determined*.

A famous result by Martin (1975, 1985) provides a condition for a payoff set to be determined. In order to be able to state this result, we need to introduce some terminology. We can endow the set $\mathfrak{F}(Y)$ with a topology[3].

Definition 4.5.4. A subset of $\mathfrak{F}(Y)$ is a *basic open set* if it is of the form $\{s \mid s \in \mathfrak{F}(Y) \colon p \text{ is an initial segment of } s\}$. An *open set* in $\mathfrak{F}(Y)$ is an arbitrary union of basic open sets in $\mathfrak{F}(Y)$.

The set of *Borel sets* of $\mathfrak{F}(Y)$ is the smallest set containing open sets of $\mathfrak{F}(Y)$ that is closed under

- complement (with respect to $\mathfrak{F}(Y)$);

[3] If we let A have the discrete topology, then the topology $\mathfrak{F}(Y)$ is the subset topology of A^ω with the product topology.

- countable unions.

Theorem 4.5.5 (Borel determinacy (Martin, 1975, 1985))**.** *If P is Borel in $\mathfrak{F}(Y)$, then $\mathcal{G}_Y^P$ is determined.*

This is the strongest determinacy result for Gale-Stewart games that is provable in Zermelo–Fraenkel set theory with the axiom of Choice (ZFC) since the determinacy of the next higher Wadge class is not provable in ZFC (Kechris et al., 2012).

To be able to use these results, we need to transform our games to Gale-Stewart games. However, since our games are three-valued, we will need more than one Gale-Stewart game to capture our games. Let us first bring the notion of determinacy to our games.

Definition 4.5.6. A game $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ is *determined* in x by $A \subseteq \mathcal{L}$ if one of the following holds:

- There is a $\sigma^* \in \mathfrak{S}_g(\mathbf{T})$ such that for all $\tau \in \mathfrak{S}_g(\mathbf{F})$ we have that $u(x, \sigma^*, \tau) \in A$.
- There is a $\tau^* \in \mathfrak{S}_g(\mathbf{F})$ such that for all $\sigma \in \mathfrak{S}_g(\mathbf{T})$ we have that $u(x, \sigma, \tau^*) \notin A$.

Remark 4.5.7. At most one of the two sentences hold. ▲

The notion of determinacy allows us to look at our games as a pair of two-valued games.

Proposition 4.5.8. *If $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ is determined in x by $\{\mathbf{t}\}$ and $\{\mathbf{t}, \mathbf{u}\}$, then there is an optimal pair of general strategies.*

Proof. Assume first there is a $\sigma^* \in \mathfrak{S}_g(\mathbf{T})$ such that for all $\tau \in \mathfrak{S}_g(\mathbf{F})$ we have that $u(x, \sigma^*, \tau) = \mathbf{t}$. This means that $\min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma^*, \tau) = \mathbf{t}$. Now for any $\tau \in \mathfrak{S}_g(\mathbf{F})$ we have that $\mathbf{t} = u(x, \sigma^*, \tau) \leq_t \max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau)$. Therefore, $\max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau) = \mathbf{t}$. By Lemma 4.5.1, (σ^*, τ) is optimal.

Assume now that there is a $\tau^* \in \mathfrak{S}_g(\mathbf{F})$ such that for all $\sigma \in \mathfrak{S}_g(\mathbf{T})$ we have that $u(x, \sigma, \tau^*) = \mathbf{f}$ ($\notin \{\mathbf{t}, \mathbf{u}\}$). This means that $\max_{\sigma \in \mathfrak{S}_g(\mathbf{T})} u(x, \sigma, \tau^*) = \mathbf{f}$. For any $\sigma \in \mathfrak{S}_g(\mathbf{T})$, we have that $\mathbf{f} = u(x, \sigma, \tau^*) \geq_t \min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau)$. Therefore, $\min_{\tau \in \mathfrak{S}_g(\mathbf{F})} u(x, \sigma, \tau) = \mathbf{f}$. By Lemma 4.5.1, (σ, τ^*) is optimal.

There is only one case left: there is a $\tau^* \in \mathfrak{S}_g(\mathbf{F})$ such that for all $\sigma \in \mathfrak{S}_g(\mathbf{T})$ it holds that $u(x, \sigma, \tau^*) \neq \mathbf{t}$ and there is a $\sigma^* \in \mathfrak{S}_g(\mathbf{T})$ such that for all $\tau \in \mathfrak{S}_g(\mathbf{F})$ it holds that $u(x, \sigma^*, \tau) \neq \mathbf{f}$. We prove that (σ^*, τ^*) is an

optimal pair. We know that $\max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau^*) <_t \mathbf{t}$ because max is a maximum. Likewise, $\min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma^*,\tau) >_t \mathbf{f}$ because min is a minimum. Since $\mathbf{f} \neq u(x,\sigma^*,\tau^*) \neq \mathbf{t}$, we have that $u(x,\sigma^*,\tau^*) = \mathbf{u}$. We also have that $\mathbf{f} <_t \min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma^*,\tau) \leq_t u(x,\sigma^*,\tau^*) \leq_t \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau^*) <_t \mathbf{t}$; hence $\min_{\tau\in\mathfrak{S}_g(\mathbf{F})} u(x,\sigma^*,\tau) = \max_{\sigma\in\mathfrak{S}_g(\mathbf{T})} u(x,\sigma,\tau^*) = u(x,\sigma^*,\tau^*) = \mathbf{u}$. By Lemma 4.5.1, the proof is concluded. □

Our plan is thus to prove that our games are determined by $\{\mathbf{t}\}$ and $\{\mathbf{t},\mathbf{u}\}$. We do this by mapping our games to Gale-Stewart games. The games associated with a justification system alternate between two players **T** and **F**. Therefore, I is **T** and II is **F**. Take $Y' = \mathrm{Path}_{\mathcal{G}_{\mathcal{JS},\mathcal{I}}}$. Let Y be Y' where we added paths that repeat the last open fact a finite number of times. There is a one-to-one correspondence between elements in $\mathfrak{F}(Y)$ starting with x and plays in $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ starting with x. This constructs a Gale-Stewart game $X_{\mathcal{JS},\mathcal{I}}$ corresponding to $\mathcal{JS}$ and $\mathcal{I}$. This game is equivalent to $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$.

Proposition 4.5.9. *The game $\mathcal{G}_{\mathcal{JS},\mathcal{I}}$ is determined in x by $A \subseteq \mathcal{L}$ if and only if $\{\mathbf{b} \mid \mathcal{B}(\mathbf{b}) \in A\}$ is determined in $\mathfrak{F}(Y)$.*

Proof. Winning strategies can easily be transferred from one game to the other. □

We can also put a similar topology on the set of branches that follow the structure of $\mathcal{JF}$. This allows us to state the following proposition.

Proposition 4.5.10. *If the sets $\{\mathbf{b} \mid \mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}\}$ and $\{\mathbf{b} \mid \mathcal{I}(\mathcal{B}(\mathbf{b})) \neq \mathbf{f}\}$ are Borel, then there is a optimal pair of strategies.*

Corollary 4.5.11. *If the sets $\{\mathbf{b} \mid \mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}\}$ and $\{\mathbf{b} \mid \mathcal{I}(\mathcal{B}(\mathbf{b})) \neq \mathbf{f}\}$ are Borel, then $\mathcal{JS}$ is tree-like consistent.*

Corollary 4.5.12. *If the sets $\{\mathbf{b} \mid \mathbf{b}$ is nowhere decided$, \mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}\}$ and $\{\mathbf{b} \mid \mathbf{b}$ is nowhere decided$, \mathcal{I}(\mathcal{B}(\mathbf{b})) \neq \mathbf{f}\}$ are Borel, then $\mathcal{JS}$ is tree-like consistent.*

Proof. For each decided path $\mathbf{p}$ with $\mathcal{I}(\mathcal{B}(\mathbf{p})) = \mathbf{t}$, the set of branches starting with $\mathbf{p}$ is an basic open set. Therefore, the set

$$\{\mathbf{b} \mid \mathbf{b} \text{ is somewhere decided and } \mathcal{I}(\mathcal{B}(\mathbf{b})) = \mathbf{t}\}$$

is an open set. Therefore, the set of branches that have value $\mathbf{t}$ is a Borel set since it is the union of two Borel sets. A similar reasoning can be made for $\neq \mathbf{f}$. □

4.6 Conclusion

In this chapter, we studied the relation between justification theory and game theory to answer both of our research problems: consistency and graph-reducibility. All of our work was developed in a general setting except for one (crucial) result: in order to prove the existence of an optimal pair of positional strategies, 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. At the end of this chapter we have established the relation between our games and Gale-Stewart games (Gale and Stewart, 1953; Soare, 2016). Using the Borel-determinacy result (Martin, 1975, 1985), we have a potential way to prove tree-like consistency. However, for graph-like consistency this result will not help, as strategies considered in Gale-Stewart games have perfect information. There have been results on memoryless determinacy of parity games (Emerson and Jutla, 1991; Roux, 2019; Zielonka, 1998) and of ω-automata (Jutla, 1997). Some work capturing parity games with justification theory has already been done by Lapauw et al. (2020, 2021). Therefore, future work could try to generalise this results to our type of games. Another possibility is to find a condition on the branch evaluation for which the justification game is equivalent to a parity game.

Several earlier papers have already established connections between logic programming and games to study properties of logic programs. One of the earliest results is by van Emden (1986), in which a probabilistic version of logic programming is worked out whose proof theory is described using a two-person game. In 1995, Blair (1995) presented a family of simple two-player games, where the players are represented as almost independent logic programs. This allowed for complexity results to transfer from game theory to logic programming. Later on, a game semantics was developed by Cosmo et al. (1998) where to each logic program a simple game is associated in which one player tries to prove the goal, while the opponent tries to disprove it. Loddo and Cosmo (2000) then showed that the alpha-beta algorithm can also be applied to constraint logic programming. So far, these results only consider positive logic programs. Vos and Vermeir (1999, 2001) showed that stable models of choice programs correspond to Nash equilibria of strategic games. Galanaki et al. (2008) presented a game semantics for well-founded negation. Galanaki et al. (2013, 2017) developed a game semantics for non-monotonic intensional

logic programming. Tsouanas (2013) made a game semantics for disjunctive logic programming.

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. 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.

Chapter 5

Embedding Justification Theory in Approximation Fixpoint Theory

5.1 Introduction

In the introduction and the second chapter, we have seen that justification theory is a versatile semantic framework. However, justification theory is not the only unifying framework for non-monotonic logics. Another framework, invented by Denecker, Marek, and Truszczyński (2000), is Approximation Fixpoint Theory (AFT). In this chapter, we take a look at how justification theory can fit into this framework. In contrast to justification theory, AFT is mainly focused on completion, Kripke-Kleene, stable, and well-founded semantics.[1] Therefore, in most of this chapters, we are focussed on on these semantics. We will show that justification theory for these semantics can be embedded into approximation fixpoint theory. Consequentially, we bring ultimate semantics, a concept from AFT, to justification theory. This is done in such a way, that is actually applicable to any conceivable justification semantics.

The text of this chapter is solely based on (Marynissen et al., 2021). Full proofs are added, plus additional results for $\mathcal{B}_{\text{cst}}$ and $\mathcal{B}_{\text{cwf}}$ are added as well.

[1] Groundedness is another important category in AFT.

5.1.1 Approximation Fixpoint Theory

In the 1980s and 90s, the area of Non-Monotonic Reasoning (NMR) saw fierce debates about formal semantics. In several subareas, researchers sought to formalise common-sense intuitions about knowledge of introspective agents. In these areas, appeals to similar intuitions were made, resulting in the development of similar mathematical concepts. Despite the obvious similarity, the precise relation between these concepts remained elusive. AFT was founded in the early 2000s by Denecker, Marek, and Truszczyński (2000) as a way of unifying semantics that emerged in these different subareas. The main contribution of AFT was to demonstrate that, by moving to an algebraic setting, the common principles behind these concepts can be isolated and studied in a general way. This breakthrough allowed results that were achieved in the context of one of these languages to be easily transferred to another. In the early stages, AFT was applied to default logic (Reiter, 1980), auto-epistemic logic (Moore, 1985), and logic programming (Denecker et al., 2000, 2003; van Emden and Kowalski, 1976). In recent years, interest in AFT has gradually increased, with applications in various other domains, encompassing abstract argumentation (Strass, 2013), extensions to deal with inconsistencies (Bi et al., 2014), higher-order logic programs (Charalambidis et al., 2018), and active integrity constraints (Bogaerts and Cruz-Filipe, 2018).

The foundations of AFT lie in Tarski's fixpoint theory of monotone operators on a complete lattice (Tarski, 1955). AFT demonstrates that by moving from the original lattice L to the bilattice L^2, Tarski's theory can be generalised into a fixpoint theory for arbitrary (i.e., also non-monotone) operators. Crucially, all that is required to apply AFT to a formalism and obtain several semantics is to define an appropriate approximating operator $L^2 \to L^2$ on the bilattice; the algebraic theory of AFT then takes care of the rest. For instance, to characterise the major logic programming semantics using AFT, it suffices to define Fitting's four-valued immediate consequence operator (Fitting, 2002). The (partial) stable fixpoints of that operator (as defined by AFT) are exactly the partial stable models of the original program; the well-founded fixpoint of the operator is the well-founded model of the program, etc.

5.1.2 Correspondence

AFT and justification theory were designed with similar intentions in mind, namely to unify different (mainly non-monotonic) logics. One major difference between them is that justification theory is defined logically while AFT is defined purely algebraically. This makes justification frameworks less abstract and easier to grasp, but also in a certain sense less general, as demonstrated by

the fact that logics such as autoepistemic logic have no justification semantics (yet). On the other hand, justification theory has a larger freedom to define semantics by providing a branch evaluation, as well as the notion of nesting (see Chapter 6) to integrate the semantics of nested least and nested greatest fixpoint definitions.

Despite the differences, certain correspondences between the theories show up: several definitions in justification frameworks seem to have an algebraical counterpart in AFT. This is evident from the fact that many results on justifications are formulated in terms of fixpoints of the support operator (see Definition 2.2.23) that happens, for the case of logic programming, to coincide with (Fitting's three-valued version (Fitting, 2002) of) the immediate consequence operator for logic programs (see Section 2.7.1). Of course, now the question naturally arises whether this correspondence can be made formal, i.e., whether it can formally be shown that semantics induced by justification theory will always coincide with their equally-named counterpart in AFT.

Correspondence between AFT and Justification Theory

How are AFT and justification theory related? Is the support operator in justification theory a (consistent) approximator in AFT.

If we can answer this question, then it will allow us to translate results between the two theories. Formalising this correspondence is the key contribution of this chapter.

5.2 Approximation Fixpoint Theory

Given a complete lattice $\langle L, \leq \rangle$, AFT Denecker et al. (2000) uses the *bilattice* $L^2 = L \times L$. We define projection functions as usual: $(x, y)_1 = x$ and $(x, y)_2 = y$. Pairs $(x, y) \in L^2$ are used to approximate elements in the interval $[x, y] = \{z \mid x \leq z \leq y\}$. We call $(x, y) \in L^2$ *consistent* if $x \leq y$; i.e., if $[x, y]$ is not empty. The set of consistent elements is denoted L^c. Pairs (x, x) are called *exact* since they approximate only the element x. The *precision order* $\leq_p$ on L^2 is defined as $(x, y) \leq_p (u, v)$ if $x \leq u$ and $y \geq v$. If (u, v) is consistent, this means that $[u, v] \subseteq [x, y]$. If $\langle L, \leq \rangle$ is a complete lattice, then so is $\langle L^2, \leq_p \rangle$. AFT studies fixpoints of operators $O\colon L \to L$ through operators approximating O. An operator $A\colon L^2 \to L^2$ is an *approximator* of O if it is $\leq_p$-monotone and has the property that $A(x, x) = (O(x), O(x))$ for all $x \in L$. Approximators are internal in L^c (i.e., map L^c into L^c). We often restrict our attention to *symmetric* approximators: approximators A such that, for all x

and y, $A(x,y)_1 = A(y,x)_2$. Denecker et al. (2004) showed that the consistent fixpoints of interest of a symmetric approximator are uniquely determined by an approximator's restriction to L^c and hence, that it usually suffices to define approximators on L^c. Such a restriction is called a *consistent approximator*. As mentioned before, AFT studies fixpoints of O using fixpoints of A. The main type of fixpoints that concern us are given here.

- A *partial supported fixpoint* of A is a fixpoint of A.
- The *Kripke-Kleene fixpoint* of A is the $\leq_p$-least fixpoint of A; it approximates all fixpoints of A.
- A *partial stable fixpoint* of A is a pair (x,y) such that $x = \text{lfp}(A(\cdot,y)_1)$ and $y = \text{lfp}(A(x,\cdot)_2)$, where $A(\cdot,y)_1$ denotes the function $L \to L\colon z \mapsto A(z,y)_1$ and analogously $A(x,\cdot)_2$ stands for $L \to L\colon z \mapsto A(x,z)_2$.
- The *well-founded fixpoint* of A is the least precise partial stable fixpoint of A.

Other type of fixpoints do exist, such as grounded fixpoints Bogaerts et al. (2015a), but we do not consider them. We add two new type of fixpoints for our purposes

- A *partial co-stable fixpoint* of A is a pair (x,y) such that $x = \text{gfp}(A(\cdot,y)_1)$ and $y = \text{gfp}(A(x,\cdot)_2)$.
- The *co-well-founded fixpoint* of A is the least precise partial co-stable fixpoint of A.

5.3 The Embedding

We now turn our attention to the main topic of this chapter, namely, a formal proof of the correspondence between justification theory and AFT. We start by showing how to obtain an approximator out of a justification frame.

5.3.1 The Approximator

Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame, fixed throughout this section. Our first goal is to define, from a given justification frame, an approximator on a suitable lattice. Following the correspondence with how this is done

in logic programming, we will take as lattice the set of *exact* interpretations (interpretations that map no facts to **u** except for **u** itself). It is easy to see that such interpretations correspond directly to subsets of $\mathcal{F}_+$. In other words, we will use the lattice $\langle L = 2^{\mathcal{F}_+}, \subseteq \rangle$. Now, the set L^c is isomorphic to the set of three-valued interpretations of $\mathcal{F}$; under this isomorphism, a consistent pair $(I, J) \in L^c$ corresponds to the three-valued interpretation $\mathcal{I}$ such that for positive facts $x \in \mathcal{F}_+$, $\mathcal{I}(x) = \mathbf{t}$ if $x \in I$, $\mathcal{I}(x) = \mathbf{f}$ if $x \notin J$, and $\mathcal{I}(x) = \mathbf{u}$ otherwise.

Definition 5.3.1. The *operator* $O_{\mathcal{JF}} \colon L \to L$ of $\mathcal{JF}$ maps a subset I of $\mathcal{F}_+$ to

$$O_{\mathcal{JF}}(I) = \{x \in \mathcal{F}_+ \mid \exists x \leftarrow A \in R \colon \forall a \in A \colon (I, I)(a) = \mathbf{t}\}\,.$$

The *approximator* $A_{\mathcal{JF}}$ of $\mathcal{JF}$ is defined as follows

$$A_{\mathcal{JF}}(\mathcal{I})_1 = \{x \in \mathcal{F}_+ \mid \exists x \leftarrow A \in R \colon \forall a \in A \colon \mathcal{I}(a) = \mathbf{t}\}\,;$$

$$A_{\mathcal{JF}}(\mathcal{I})_2 = \{x \in \mathcal{F}_+ \mid \exists x \leftarrow A \in R \colon \forall a \in A \colon \mathcal{I}(a) \geq_t \mathbf{u}\}\,.$$

Proposition 5.3.2. *If no rule body in $\mathcal{JF}$ contains* **u**, *then $A_{\mathcal{JF}}$ is a consistent approximator of $O_{\mathcal{JF}}$.*

Proof. It is clear that $A_{\mathcal{JF}}$ coincides with $O_{\mathcal{JF}}$ on exact interpretations. It suffices to prove that $A_{\mathcal{JF}}(\mathcal{I})(x) \neq \mathbf{u}$ if $\mathcal{I}$ is exact. This follows directly from the fact that no rule body in $\mathcal{JF}$ contains **u**. □

The requirement that rule bodies cannot contain **u** is an immediate consequence of the fact that approximators in AFT ought to be symmetric. Quoting Denecker et al. (2004, page 14)

> While it is possible to develop a generalisation of the theory presented in this paper without the symmetry assumption, we chose to adopt it because the motivating examples, that is, operators occurring in knowledge representation, are symmetric.

Similarly, we are not aware of practical examples with bodies containing **u** in unnested justification systems.[2] However, in nested systems, as we will see in Chapter 6, bodies containing **u** can occur quite easily due to the construction of the compression (Definition 6.2.17). No proof, except for Proposition 5.3.2, in this chapter makes use of the fact that **u** does not appear in bodies of rules. This means that once consistent AFT is worked out for asymmetric operators, we

[2] You and Yuan (1990) argue that **u** itself is needed only for a very special type of logic programs.

can remove this restriction. For this chapter, we assume that every justification frame does not have **u** in a rule body. It turns out that in case our justification frame is complementary, the approximator is the same as the support operator of $\mathcal{B}_{\text{sp}}$.

Lemma 5.3.3. *For a complementary justification frame $\mathcal{JF}$, the function $A_{\mathcal{JF}}$ and the support operator $\mathcal{S}_{\mathcal{JF}}^{\mathcal{B}_{\text{sp}}}$ are equal.*

Proof. Take an interpretation $\mathcal{I}$. For any $x \in \mathcal{F}_+$, it is obvious that $A_{\mathcal{JF}}(\mathcal{I})(x) = \mathcal{S}_{\mathcal{JF}}^{\mathcal{B}_{\text{sp}}}(\mathcal{I})(x)$. Take $x \in \mathcal{F}_-$. We have that $A_{\mathcal{JF}}(\mathcal{I})(x) = \sim A_{\mathcal{JF}}(\mathcal{I})(\sim x) = \sim \mathcal{S}_{\mathcal{JF}}^{\mathcal{B}_{\text{sp}}}(\mathcal{I})(\sim x) = \mathcal{S}_{\mathcal{JF}}^{\mathcal{B}_{\text{sp}}}(\mathcal{I})(x)$, where the last step is by the consistency of $\mathcal{B}_{\text{sp}}$ (Theorem 3.2.6). □

5.3.2 Semantic Correspondence

The central result of this section is the following theorem, which essentially states that for all major semantics, the branch evaluation in justification theory corresponds to the definitions of AFT.

Theorem 5.3.4. *Take a complementary justification frame $\mathcal{JF}$. If the interpretation of the open facts is fixed, then the following results hold.*

- *The partial supported fixpoints of $A_{\mathcal{JF}}$ are exactly the $\mathcal{B}_{\text{sp}}$-models of $\mathcal{JF}$.*
- *The Kripke-Kleene fixpoint of $A_{\mathcal{JF}}$ is the unique $\mathcal{B}_{\text{KK}}$-model of $\mathcal{JF}$.*
- *The partial stable fixpoints of $A_{\mathcal{JF}}$ are exactly the $\mathcal{B}_{\text{st}}$-models of $\mathcal{JF}$.*
- *The well-founded fixpoint of $A_{\mathcal{JF}}$ is the unique $\mathcal{B}_{\text{wf}}$-model of $\mathcal{JF}$.*
- *The partial co-stable fixpoints of $A_{\mathcal{JF}}$ are exactly the $\mathcal{B}_{\text{cst}}$-models of $\mathcal{JF}$.*
- *The co-well-founded fixpoint of $A_{\mathcal{JF}}$ is the unique $\mathcal{B}_{\text{cwf}}$-model of $\mathcal{JF}$.*

These points are proven independently; the first follows directly from our observation that $A_{\mathcal{JF}}$ and $\mathcal{S}_{\mathcal{JF}}^{\mathcal{B}_{\text{sp}}}$ are in fact the same operator.

Proposition 5.3.5. *The partial supported fixpoints of $A_{\mathcal{JF}}$ are exactly the supported models of $\mathcal{JF}$.*

Proof. Follows directly from Lemma 5.3.3. □

Given the correspondence between supported semantics, the result for Kripke-Kleene semantics follows quite easily.

Proposition 5.3.6. *The Kripke-Kleene fixpoint of $A_{\mathcal{JF}}$ is equal to the unique $\mathcal{B}_{\mathrm{KK}}$-model of $\mathcal{JF}$.*

Proof. Let $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}$ be the unique $\mathcal{B}_{\mathrm{KK}}$-model of $\mathcal{JF}$. By Propositions 3.5.1 and 3.4.7, it suffices to prove that $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}$ is the least precise fixpoint of $A_{\mathcal{JF}}$. Assume there is a fixpoint $\mathcal{I}$ of $A_{\mathcal{JF}}$ such that $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}} \not\leq_p \mathcal{I}$. This means there exists an $x \in \mathcal{F}_d$ such that $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) \not\leq_p \mathcal{I}(x)$. There are exactly four cases

1. $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{t}$ and $\mathcal{I}(x) = \mathbf{f}$;
2. $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{f}$ and $\mathcal{I}(x) = \mathbf{t}$;
3. $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{f}$ and $\mathcal{I}(x) = \mathbf{u}$;
4. $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{t}$ and $\mathcal{I}(x) = \mathbf{u}$.

A fact x satisfies the first case if and only if $\sim x$ satisfies the second case. The same holds for the third and fourth case. Therefore, it suffices to look at the following two cases:

1. $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{f}$ and $\mathcal{I}(x) = \mathbf{t}$;
2. $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{f}$ and $\mathcal{I}(x) = \mathbf{u}$.

In both cases, we have that $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(x) = \mathbf{f}$; hence every justification J with x as internal node has a finite branch starting with x mapped to an open fact y with $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}(y) = \mathcal{I}(y) = \mathbf{f}$. Therefore, $\mathcal{I}(x) = \mathrm{SV}_{\mathcal{B}'_{\mathrm{sp}}}(x, \mathcal{I}) = \mathbf{f}$, contradicting both cases; hence $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}} \leq_p \mathcal{I}$, proving that $\mathcal{I}_{\mathcal{B}_{\mathrm{KK}}}$ is the least precise fixpoint of $A_{\mathcal{JF}}$. □

The proof of the third point of Theorem 5.3.4 is split in two parts, proven separately in the following propositions.

Proposition 5.3.7. *Each $\mathcal{B}_{\mathrm{st}}$-model of $\mathcal{JF}$ is a partial stable fixpoint of $A_{\mathcal{JF}}$.*

Proof. Let $\mathcal{I} = (I_1, I_2)$ be a $\mathcal{B}_{\mathrm{st}}$-model of $\mathcal{JF}$. We prove that $\mathrm{lfp}(A_{\mathcal{JF}}(\cdot, I_2)_1) = I_1$ and that $\mathrm{lfp}(A_{\mathcal{JF}}(I_1, \cdot)_2) = I_2$. By Proposition 3.5.3, it holds that $A_{\mathcal{JF}}(\mathcal{I}) = \mathcal{S}^{\mathcal{B}_{\mathrm{sp}}}_{\mathcal{JF}}(\mathcal{I}) = \mathcal{I}$. Therefore, we have that $A_{\mathcal{JF}}(I_1, I_2)_1 = I_1$ and $A_{\mathcal{JF}}(I_1, I_2)_2 = I_2$.

Take $I'_1 \subsetneq I_1$ and assume by contradiction that $A_{\mathcal{JF}}(I'_1, I_2)_1 = I'_1$. Define $\mathcal{I}' = (I'_1, I_2)$. Therefore, $\mathcal{I}' <_p \mathcal{I}$ and $\mathcal{I}(x) = \mathbf{t}$ and $\mathcal{I}'(x) = \mathbf{u}$ for all $x \in I_1 \setminus I'_1$.

By Theorem 2.6.20 and splittability of $\mathcal{B}_{\text{st}}$, there is a justification J so that $\text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) = \text{SV}_{\mathcal{B}_{\text{st}}}(x,\mathcal{I}) = \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$. Define the partial order $\preceq_J$ on I_1: $y \preceq_J x$ if y is reachable in J from x through positive facts. Since J does not contain infinite positive branches starting in a fact $x \in I_1$, we have that $\preceq_J$ does not have infinitely descending chains; hence $\preceq_J$ is well-founded. The set $I_1 \setminus I_1'$ is not empty, hence has a minimal element x with respect to $\preceq_J$. Take a child y of x in J. We prove that $\mathcal{I}'(y) = \mathbf{t}$. If y is open, then $\mathbf{t} = \mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) \leq_t \mathcal{I}(y) = \mathcal{I}'(y)$. If y has a different sign than x, then $\mathbf{t} = \mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) \leq_t \mathcal{I}(y)$. This means that $\sim y \notin I_2$ since $y \in \mathcal{F}_-$; hence $\mathcal{I}'(y) = \mathbf{t}$. If y has the same sign as x, then $\mathbf{t} = \mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) \leq_t \text{val}_{\mathcal{B}_{\text{st}}}(J,y,\mathcal{I}) = \mathcal{I}(y)$. Therefore, $y \in I_1$, which implies that $y \prec_J x$. This means that $y \notin I_1 \setminus I_1'$. We can conclude that $y \in I_1'$; hence $\mathcal{I}'(y) = \mathbf{t}$. This shows that $\text{val}_{\mathcal{B}_{\text{sp}}}(J,x,\mathcal{I}') = \mathbf{t}$, hence $\text{SV}_{\mathcal{B}_{\text{sp}}}(x,\mathcal{I}') = \mathbf{t}$. This implies that $x \in A_{\mathcal{JF}}(I_1', I_2)_1 = I_1'$, which contradicts that $x \notin I_1'$; hence $I_1 = \text{lfp}(A_{\mathcal{JF}}(\cdot, I_2)_1)$.

Take $I_1 \subseteq I_2' \subsetneq I_2$ and assume by contradiction that $A_{\mathcal{JF}}(I_1, I_2')_2 = I_2'$. Define $\mathcal{I}'' = (I_1, I_2')$. Therefore, $\mathcal{I} <_p \mathcal{I}''$ and $\mathcal{I}(x) = \mathbf{u}$ and $\mathcal{I}''(x) = \mathbf{f}$ for all $x \in I_2 \setminus I_2'$.

Define the partial order $\preceq_J'$ on I_2 the same as before. This order is also well-founded for a similar reason. The set $I_2 \setminus I_2'$ is not empty, hence has a minimal element x with respect to $\preceq_J'$. Take a child y of x in J. We prove that $\mathcal{I}''(y) \geq_t \mathbf{u}$. If y is open, then $\mathbf{u} = \mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) \leq_t \mathcal{I}(y) = \mathcal{I}''(y)$. If y has a different sign as x, then $\mathbf{u} = \mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) \leq_t \mathcal{I}(y)$. This means that $\sim y \notin I_2$ or $\sim y \in I_2 \setminus I_1$. Therefore, $\sim y \notin I_2'$ or $\sim y \in I_2' \setminus I_1$, thus $\mathcal{I}''(y) \geq_t \mathbf{u}$. If y has the same sign as x, then $\mathbf{u} = \mathcal{I}(x) = \text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) \leq_t \text{val}_{\mathcal{B}_{\text{st}}}(J,y,\mathcal{I}) = \mathcal{I}(y)$. Therefore, $y \in I_2$, which implies that $y \prec_J' x$. This means that $y \notin I_2 \setminus I_2'$, hence $y \in I_2'$. We conclude that $\mathcal{I}''(y) \geq_t \mathbf{u}$. This shows that $\text{val}_{\mathcal{B}_{\text{sp}}}(J,x,\mathcal{I}'') \geq_t \mathbf{u}$, hence $\text{SV}_{\mathcal{B}_{\text{sp}}}(x,\mathcal{I}'') \geq_t \mathbf{u}$. This implies that $x \in A_{\mathcal{JF}}(I_1, I_2')_2 = I_2'$, which contradicts that $x \notin I_2'$, concluding that $I_2 = \text{lfp}(A_{\mathcal{JF}}(I_1, \cdot)_2)$. This finishes the proof that $\mathcal{I}$ is a partial stable fixpoint of $A_{\mathcal{JF}}$. □

For the other direction, we first need the following lemma.

Lemma 5.3.8. *Let $\mathcal{I}$ be a $\mathcal{B}_{\text{sp}}$-model and $x \in \mathcal{F}_+$ with $\text{SV}_{\mathcal{B}_{\text{sp}}}(x,\mathcal{I}) = \mathbf{f}$. It holds that $\text{SV}_{\mathcal{B}_{\text{st}}}(x,\mathcal{I}) = \mathbf{f}$.*

Proof. Take an arbitrary justification J with x as internal node. There is a child y of x with $\mathcal{I}(y) = \mathbf{f}$. If y has a different sign than x, then $\text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) = \mathbf{f}$. Otherwise, we can construct either a finite branch ending in a z with $\mathcal{I}(z) = \mathbf{f}$ or an infinite branch such that every element z in $\mathbf{b}$ has $\mathcal{I}(z) = \mathbf{f}$ and z has the same sign as x. Since x is positive, this means that $\text{val}_{\mathcal{B}_{\text{st}}}(J,x,\mathcal{I}) = \mathbf{f}$. This concludes the proof that $\text{SV}_{\mathcal{B}_{\text{st}}}(x,\mathcal{I}) = \mathbf{f}$. □

Proposition 5.3.9. *Each partial stable fixpoint of $A_{\mathcal{JF}}$ is a $\mathcal{B}_{\mathrm{st}}$-model of $\mathcal{JF}$.*

Proof. Let $\mathcal{I} = (I_1, I_2)$ be a partial stable fixpoint of $A_{\mathcal{JF}}$. We prove that $\mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x, \mathcal{I}) = \mathcal{I}(x)$ for $x \in \mathcal{F}_+$. By consistency of $\mathcal{B}_{\mathrm{st}}$ (Theorem 3.2.6), this proves that $\mathcal{I}$ is a $\mathcal{B}_{\mathrm{st}}$-model of $\mathcal{JF}$. We prove our claim in three parts.

Part 1: $\mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x, \mathcal{I}) = \mathcal{I}(x) = \mathbf{t}$ for all $x \in I_1$. Since I_1 is $\mathrm{lfp}(A_{\mathcal{JF}}(\cdot, I_2)_1)$, there is an increasing sequence $(K_i)_{i \leq \beta}$ for some ordinal number β so that

- $K_0 = \emptyset$;
- $K_{i+1} = A_{\mathcal{JF}}(K_i, I_2)_1$ for all $i < \beta$;
- $K_\alpha = \bigcup_{i<\alpha} K_i$ for all limit ordinals $\alpha \leq \beta$;
- $K_\beta = I_1$.

Such a sequence exists by a constructive version of the Knaster-Tarski fixpoint theorem (Cousot and Cousot, 1979). Now, for every $x \in I_1$, there is a least ordinal i_x such that $x \notin K_{i_x}$, while $x \in K_{i_x+1}$. This means that there is a rule $x \leftarrow A_x$ such that for all $y \in A_x$ we have that $(K_{i_x}, I_2)(y) = \mathbf{t}$. Since $(K_{i_x}, I_2) \leq_p \mathcal{I}$, we have that $\mathcal{I}(y) = \mathbf{t}$ for all $y \in A_x$.

We now define J to be the justification with exactly the rules $x \leftarrow A_x$ for $x \in I_1$. For a $y \in \mathcal{F}_+ \cap A_x$, we have that $y \in K_{i_x}$; hence $i_y < i_x$. This means that every infinite J-branch $x = x_0 \to x_1 \to \cdots$ implies the existence of strictly decreasing sequence of ordinals $(i_{x_0}, i_{x_1}, \ldots)$. Therefore, every J-branch is finite and ending in an element in $\mathcal{F}_- \cup \mathcal{F}_o$. Consequently, the value of any J-branch starting from x is an element of J, hence $\mathrm{val}_{\mathcal{B}_{\mathrm{st}}}(J, x, \mathcal{I}) = \mathbf{t}$.

Part 2: $\mathrm{SV}_{\mathcal{B}_{\mathrm{st}}}(x, \mathcal{I}) = \mathcal{I}(x) = \mathbf{u}$ for all $x \in I_2 \setminus I_1$. Since I_2 is $\mathrm{lfp}(A_{\mathcal{JF}}(I_1, \cdot)_2)$, there is an increasing sequence $(M_i)_{i \leq \beta}$ for some ordinal number β so that

- $M_0 = I_1$;
- $M_{i+1} = A_{\mathcal{JF}}(I_1, M_i)_2$ for $i < \beta$;
- $M_\alpha = \bigcup_{i<\alpha} M_i$ for limit ordinal $\alpha \leq \beta$;
- $M_\beta = I_2$.

For every $x \in I_2 \setminus I_1$, there is a least ordinal j_x such that $x \notin M_{j_x}$, while $x \in M_{j_x+1}$. Therefore, there is a rule $x \leftarrow C_x$ such that for all $y \in C_x$ we have that $(I_1, M_{j_x})(y) \geq_t \mathbf{u}$. Since $\mathcal{I} \leq_p (I_1, M_{j_x})$ we have that $\mathcal{I}(y) \geq_t \mathbf{u}$ for all $y \in C_x$. Define J' to be the justification with exactly the rules $x \leftarrow C_x$ for

$x \in I_2 \setminus I_1$. By a similar reasoning as in the first part, we have that every J'-branch is finite and ending in $I_1 \cup \mathcal{F}_- \cup \mathcal{F}_o$. Define J^* as $J' \uparrow J$, with J the justification from the first part. A J^*-branch is either a J'-branch or a concatenation of a J'-branch with a J-branch. This means that J^* does not have infinite branches. By construction, we have for every element y in J^* that $\mathcal{I}(y) \geq_t \mathbf{u}$. The evaluation of a J^*-branch is equal to an element in J^*; hence $\text{val}_{\mathcal{B}_{\text{st}}}(J^*, x, \mathcal{I}) \geq_t \mathbf{u}$.

However, every justification for x has a branch $\mathbf{b}$ such that $\mathcal{I}(\mathcal{B}_{\text{st}}(\mathbf{b})) \leq_t \mathbf{u}$; hence $\text{SV}_{\mathcal{B}_{\text{st}}}(x, \mathcal{I}) = \mathbf{u} = \mathcal{I}(x)$. Indeed, for every y with $\mathcal{I}(y) \leq_t \mathbf{u}$ and rule $y \leftarrow C$ there is a $c \in C$ with $\mathcal{I}(x) \leq_t \mathbf{u}$. This constructs a branch $\mathbf{b}$ such that for every element y in $\mathbf{b}$ we have that $\mathcal{I}(y) \leq_t \mathbf{u}$. We know that $\mathcal{B}_{\text{st}}(\mathbf{b}) \neq \mathbf{t}$; otherwise $\mathbf{b}$ is completely negative or ending in $\mathbf{t}$. Therefore, $\mathcal{B}_{\text{st}}$ maps $\mathbf{b}$ to an element in $\mathbf{b}$ or to $\mathbf{f}$ or $\mathbf{u}$. This proves that $\mathcal{I}(\mathcal{B}_{\text{st}}(\mathbf{b})) \leq_t \mathbf{u}$.

Part 3: $\text{SV}_{\mathcal{B}_{\text{st}}}(x, \mathcal{I}) = \mathcal{I}(x) = \mathbf{f}$ **for all** $x \in \mathcal{F}_+ \setminus I_2$**.** This is immediate from Lemma 5.3.8. □

Example 5.3.10. Let $\mathcal{F} = \{x, \sim x, y, \sim y, z, \sim z\} \cup \mathcal{L}$ and let R be the complementation of

$$\left\{ \begin{array}{l} x \leftarrow y \\ y \leftarrow \sim z \\ z \leftarrow \sim x, \sim y \end{array} \right\}.$$

The approximator $A_{\mathcal{JF}}$ has three partial stable fixpoints: $(\{x, y\}, \{x, y\})$, $(\{z\}, \{z\})$, and $(\emptyset, \{x, y, z\})$.

Let us take a look at the fixpoint $(\{x, y\}, \{x, y\})$. Since it is a stable fixpoint, we know that $(\{x, y\}, \{x, y\})$ is a least fixpoint of $A_{\mathcal{JF}}(\cdot, \{x, y\})$. This operator is monotone with respect to $\subseteq$; hence we can construct the fixpoint by iteratively applying the operator on $(\emptyset, \{x, y\})$. This produces the following sequence.

$$(\emptyset, \{x, y\}) \rightarrow (\{y\}, \{x, y\}) \rightarrow (\{x, y\}, \{x, y\})$$

The first step uses the rule $y \leftarrow \sim z$, while the second step uses the rule $x \leftarrow y$. Combining the two we get the justification $x \rightarrow y \rightarrow \sim z$, which has only true nodes in the model $(\{x, y\}, \{x, y\})$, only positive internal nodes and every defined leaf is negative. This illustrates the first step of the proof of Proposition 5.3.9. By extending the found justification, we get a locally complete justification with the same value as the supported value. ▲

The proof of the fourth point of Theorem 5.3.4 follows directly from the third point and Proposition 3.5.7.

The correspondence for co-stable is proven similarly as for the stable correspondence. First, we show that every $\mathcal{B}_{\text{cst}}$-model is a partial co-stable fixpoint.

Proposition 5.3.11. *Each $\mathcal{B}_{\text{cst}}$-model of $\mathcal{JF}$ is a partial co-stable fixpoint of $A_{\mathcal{JF}}$.*

Proof. Take $\mathcal{I} = (I_1, I_2)$ to be a $\mathcal{B}_{\text{cst}}$-model of $\mathcal{JF}$. We prove that $\text{gfp}(A_{\mathcal{JF}}(\cdot, I_2)_1) = I_1$ and $\text{gfp}(A_{\mathcal{JF}}(I_1, \cdot)_2) = I_2$. By Proposition 3.5.8, it holds that $A_{\mathcal{JF}}(\mathcal{I}) = \mathcal{S}^{\mathcal{B}_{\text{sp}}}_{\mathcal{JF}}(\mathcal{I}) = \mathcal{I}$. Therefore, we have that $A_{\mathcal{JF}}(I_1, I_2)_1 = I_1$ and $A_{\mathcal{JF}}(I_1, I_2)_2 = I_2$.

Take I_2' with $I_2 \subsetneq I_2'$ and assume by contradiction that $A_{\mathcal{JF}}(I_1, I_2')_2 = I_2'$. Define $\mathcal{I}' = (I_1, I_2')$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\text{cst}}$, there is a locally complete justification J so that $\text{SV}_{\mathcal{B}_{\text{cst}}}(x, \mathcal{I}) = \text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}) = \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$. Define the relation $\prec_J$ on $\mathcal{F}_- \setminus {\sim} I_1$ as follows: $y \prec_J x$ holds if y is reachable from x in J through negative facts. The relation $\prec_J$ is well-founded, since $\mathcal{F}_- \setminus {\sim} I_1$ is the set of negative facts not false in $\mathcal{I}$, and for those facts, J has no infinite negative branches. Since ${\sim} I_2' \setminus {\sim} I_2$ is a subset of $\mathcal{F}_- \setminus {\sim} I_1$, ${\sim} I_2' \setminus {\sim} I_2$ has a minimal element z for the relation $\prec_J$. Note that $\mathcal{I}(z) = \mathbf{t}$ and $\mathcal{I}'(z) = \mathbf{u}$. Take a child y of z in J. We prove that $\mathcal{I}'(y) = \mathbf{t}$. If y is open, then $\mathcal{I}'(y) = \mathcal{I}(y) \geq_t \mathcal{I}(z) = \mathbf{t}$. If y has a different sign than z, then $\mathcal{I}(y) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, z, \mathcal{I}) = \mathcal{I}(z) = \mathbf{t}$. Therefore, $\mathcal{I}(y) = \mathbf{t}$ and thus $y \in I_1$, which means that $\mathcal{I}'(y) = \mathbf{t}$ as well. If y has the same sign as z, then $\mathcal{I}(y) = \text{val}_{\mathcal{B}_{\text{cst}}}(J, y, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, z, \mathcal{I}) = \mathcal{I}(z) = \mathbf{t}$. This means that ${\sim} y \notin I_2$; hence $y \in \mathcal{F}_- \setminus {\sim} I_2 \subseteq \mathcal{F}_- \setminus {\sim} I_1$. Consequently, we have that $y \prec_J z$. However, since z is minimal, we should have that $y \notin {\sim} I_2' \setminus {\sim} I_2$. Thus, $\mathcal{I}(y) = \mathcal{I}'(y)$. Because y was taken arbitrary, we have that $\text{val}_{\mathcal{B}_{\text{sp}}}(J, z, \mathcal{I}') = \mathbf{t}$ and thus $\text{SV}_{\mathcal{B}_{\text{sp}}}({\sim} z, \mathcal{I}') = \mathbf{f}$. This means that ${\sim} z \notin A_{\mathcal{JF}}(I_1, I_2')_2 = I_2'$, which is in contradiction with $z \in {\sim} I_2' \setminus {\sim} I_2$. Therefore, $I_2 = \text{gfp}(A_{\mathcal{JF}}(I_1, \cdot)_2)$.

Take I_1' with $I_1 \subsetneq I_1' \subseteq I_2$ and assume by contradiction that $A_{\mathcal{JF}}(I_1', I_2)_1 = I_1'$. Define $\mathcal{I}' = (I_1', I_2)$. By Theorem 2.6.20 and splittability of $\mathcal{B}_{\text{cst}}$, there is a locally complete justification J so that $\text{SV}_{\mathcal{B}_{\text{cst}}}(x, \mathcal{I}) = \text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}) = \mathcal{I}(x)$ for all $x \in \mathcal{F}_d$. Let $\prec_J$ be the same relation on $\mathcal{F}_- \setminus {\sim} I_1$ as defined above. Take a minimal element z of ${\sim} I_1' \setminus {\sim} I_1$ with respect to $\prec_J$. This means that $\mathcal{I}(z) \geq_t \mathbf{u}$ and $\mathcal{I}'(z) = \mathbf{f}$. Take a child y of z in J. We prove that $\mathcal{I}'(y) \geq_t \mathbf{u}$. If y is open, then $\mathcal{I}'(y) = \mathcal{I}(y) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, z, \mathcal{I}) = \mathcal{I}(z) \geq_t \mathbf{u}$. If y has a different sign than z, then $\mathcal{I}(y) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, z, \mathcal{I}) = \mathcal{I}(z) \geq_t \mathbf{u}$. This means that $y \in I_2$ and thus $\mathcal{I}'(y) \geq_t \mathbf{u}$ as well. If y has the same sign as z, then $\mathcal{I}(y) = \text{val}_{\mathcal{B}_{\text{cst}}}(J, y, \mathcal{I}) \geq_t \text{val}_{\mathcal{B}_{\text{cst}}}(J, z, \mathcal{I}) = \mathcal{I}(z) \geq_t \mathbf{u}$. This means that ${\sim} y \in \mathcal{F}_+ \setminus I_1$ or that $y \in \mathcal{F}_- \setminus {\sim} I_1$. Since z is minimal, we should have that $y \notin {\sim} I_1' \setminus {\sim} I_1$. Therefore, $\mathcal{I}(y) = \mathcal{I}'(y)$. Because y was taken arbitrary,

we have that $\text{val}_{\mathcal{B}_{\text{sp}}}(J, z, \mathcal{I}') \geq_t \mathbf{u}$ and thus $\text{SV}_{\mathcal{B}_{\text{sp}}}(\sim z, \mathcal{I}') \leq_t \mathbf{u}$. This means that $\sim z \notin A_{\mathcal{JF}}(I_1', I_2)_1 = I_1'$, which is in contradiction with $z \in \sim I_1' \setminus \sim I_1$. Therefore, $I_1 = \text{gfp}(A_{\mathcal{JF}}(\cdot, I_2)_1)$. This finishes the proof that $\mathcal{I}$ is a partial co-stable fixpoint of $A_{\mathcal{JF}}$. □

To prove the other direction, we need the following lemma.

Lemma 5.3.12. *Let $\mathcal{I}$ be a $\mathcal{B}_{\text{sp}}$-model and $x \in \mathcal{F}_-$ with $\text{SV}_{\mathcal{B}_{\text{sp}}}(x, \mathcal{I}) = \mathbf{f}$. It holds that $\text{SV}_{\mathcal{B}_{\text{cst}}}(x, \mathcal{I}) = \mathbf{f}$.*

Proof. Take an arbitrary justification J with x as internal node. There is a child y of x with $\mathcal{I}(y) = \mathbf{f}$. If y has a different sign than x, then $\text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}) = \mathbf{f}$. Otherwise, we can construct either a finite branch ending in a z with $\mathcal{I}(z) = \mathbf{f}$ or an infinite branch such that every element z in $\mathbf{b}$ has $\mathcal{I}(z) = \mathbf{f}$ and z has the same sign as x. Since x is negative, this means that $\text{val}_{\mathcal{B}_{\text{cst}}}(J, x, \mathcal{I}) = \mathbf{f}$. This concludes the proof that $\text{SV}_{\mathcal{B}_{\text{cst}}}(x, \mathcal{I}) = \mathbf{f}$. □

Now, we are able to prove that every partial co-stable fixpoint is a $\mathcal{B}_{\text{cst}}$-model.

Proposition 5.3.13. *Each partial co-stable fixpoint of $A_{\mathcal{JF}}$ is a $\mathcal{B}_{\text{cst}}$-model of $\mathcal{JF}$.*

Proof. Let $\mathcal{I} = (I_1, I_2)$ be a partial co-stable fixpoint of $A_{\mathcal{JF}}$. We prove that $\text{SV}_{\mathcal{B}_{\text{cst}}}(x, \mathcal{I}) = \mathcal{I}(x)$ for $x \in \mathcal{F}_-$. By consistency of $\mathcal{B}_{\text{cst}}$ (Corollary 3.3.10), this proves that $\mathcal{I}$ is a $\mathcal{B}_{\text{cst}}$-model of $\mathcal{JF}$. We prove our claim in three parts.

Part 1: $\text{SV}_{\mathcal{B}_{\text{cst}}}(x, \mathcal{I}) = \mathcal{I}(x) = \mathbf{t}$ for all $x \in \mathcal{F}_- \setminus \sim I_2$. Since I_2 is $\text{gfp}(A_{\mathcal{JF}}(I_1, \cdot)_2)$, there is a decreasing sequence $(M_i)_{i \leq \beta}$ for some ordinal number β so that

- $M_0 = \mathcal{F}_+$;
- $M_{i+1} = A_{\mathcal{JF}}(I_1, M_i)_2$ for $i < \beta$;
- $M_\alpha = \bigcap_{i<\alpha} M_i$ for limit ordinal $\alpha \leq \beta$;
- $M_\beta = I_2$.

Such a sequence exists by a constructive version of the Knaster-Tarski fixpoint theorem (Cousot and Cousot, 1979). For every $x \in \mathcal{F}_- \setminus \sim I_2$, there is a least ordinal j_x such that $\sim x \in M_{j_x}$, while $\sim x \notin M_{j_x+1} = A_{\mathcal{JF}}(I_1, M_{j_x})_2$. Therefore, there is a rule $x \leftarrow C_x$ such that for all $y \in C_x$, we have that $(I_1, M_{j_x})(y) = \mathbf{t}$.

Define J to be the justification with exactly the rules $x \leftarrow C_x$ for $x \in \mathcal{F}_- \setminus {\sim}I_2$. For a $y \in \mathcal{F}_- \cap C_x$, we have that ${\sim}y \notin M_{j_x}$. This means that $j_y < j_x$. This means that every infinite J-branch implies the existence of a strictly decreasing sequence of ordinals. Therefore, every J-branch is finite and ending in an element in $\mathcal{F}_+ \cup \mathcal{F}_o$. Consequently, the value of any J-branch starting with x is an element of J, hence $\mathrm{val}_{\mathcal{B}_{\mathrm{cst}}}(J, x, \mathcal{I}) = \mathbf{t}$.

Part 2: $\mathrm{SV}_{\mathcal{B}_{\mathrm{cst}}}(x, \mathcal{I}) = \mathcal{I}(x) = \mathbf{u}$ **for all** $x \in {\sim}I_2 \setminus {\sim}I_1$**.** Since I_1 is $\mathrm{gfp}(A_{\mathcal{JF}}(\cdot, I_2)_1)$ and by a constructive version of the Knaster-Tarski fixpoint theorem (Cousot and Cousot, 1979), there is a decreasing sequence $(K_i)_{i \leq \beta}$ for some ordinal number β so that

- $K_0 = I_2$;
- $K_{i+1} = A_{\mathcal{JF}}(K_i, I_2)_1$ for all $i < \beta$;
- $K_\alpha = \bigcap_{i<\alpha} K_i$ for all limit ordinals $\alpha \leq \beta$;
- $K_\beta = I_1$.

Now, for every $x \in {\sim}I_2 \setminus {\sim}I_1$, there is a least ordinal i_x such that ${\sim}x \in K_{i_x}$, while ${\sim}x \notin K_{i_x+1} = A_{\mathcal{JF}}(K_{i_x}, I_2)_1$. This means that there is a rule $x \leftarrow A_x$ such that for all $y \in A_x$ we have that $(K_{i_x}, I_2)(y) \geq_t \mathbf{u}$. Since, $(K_{i_x}, I_2) \geq_p \mathcal{I}$, we have that $\mathcal{I}(y) \geq_t \mathbf{u}$ for all $y \in A_x$.

Define J' to be the justification with exactly the rules $x \leftarrow A_x$ for $x \in {\sim}I_2 \setminus {\sim}I_1$. For a $y \in \mathcal{F}_- \cap A_x$, we have that ${\sim}y \notin K_{i_x}$; hence $i_y < i_x$. This means that every infinite J'-branch implies the existence of a strictly decreasing sequence of ordinals. Therefore, every J'-branch is finite and ending in an element in $\mathcal{F}_- \setminus {\sim}I_2 \cup \mathcal{F}_+ \cup \mathcal{F}_o$. Define J^* as $J' \uparrow J$ with J the justification from the first part. A J^*-branch is either a J'-branch or a concatenation of a J'-branch with a J-branch. This means that J^* has no infinite branches. By construction, we have for every element y in J^* that $\mathcal{I}(y) \geq_t \mathbf{u}$. The evaluation of J^*-branch is equal to an element in J^* because it has no infinite branches. Therefore, $\mathrm{val}_{\mathcal{B}_{\mathrm{cst}}}(J^*, x, \mathcal{I}) \geq_t \mathbf{u}$.

Every justification for x has a branch $\mathbf{b}$ such that $\mathcal{I}(\mathcal{B}_{\mathrm{cst}}(\mathbf{b})) \leq_t \mathbf{u}$; hence $\mathrm{SV}_{\mathcal{B}_{\mathrm{cst}}}(x, \mathcal{I}) = \mathbf{u} = \mathcal{I}(x)$. Indeed, for every y with $\mathcal{I}(y) \leq_t \mathbf{u}$ and rule $y \leftarrow C$ there is a $c \in C$ with $\mathcal{I}(x) \leq_t \mathbf{u}$. This construct a branch $\mathbf{b}$ such that for every element y in $\mathbf{b}$ we have that $\mathcal{I}(y) \leq_t \mathbf{u}$. We know that $\mathcal{B}_{\mathrm{cst}}(\mathbf{b}) \neq \mathbf{t}$; otherwise $\mathbf{b}$ is completely positive or ending in $\mathbf{t}$. Therefore, $\mathcal{B}_{\mathrm{cst}}$ maps $\mathbf{b}$ to an element in $\mathbf{b}$ or to $\mathbf{f}$ or $\mathbf{u}$. This proves that $\mathcal{I}(\mathcal{B}_{\mathrm{cst}}(\mathbf{b})) \leq_t \mathbf{u}$.

Part 3: $\mathrm{SV}_{\mathcal{B}_{\mathrm{cst}}}(x, \mathcal{I}) = \mathcal{I}(x) = \mathbf{f}$ **for all** $x \in {\sim}I_1$**.** This is immediate from Lemma 5.3.12. □

The proof of the sixth point of Theorem 5.3.4 follows directly from the fifth point and Proposition 3.5.8.

5.4 Application: Ultimate Semantics

When applying AFT to new domains, there is not always a clear choice of approximator to use; the operator on the other hand is often more clear. Denecker et al. (2004) studied the space of approximators and observed that consistent approximators can naturally be ordered according to their precision, where more precise approximator also yield more precise results (e.g., if approximator A is more precise than B, then the A-well-founded fixpoint is guaranteed to be more precise than B's). They also observed that the space of consistent approximators of O has a most precise element, called the *ultimate approximator*, denoted $\mathrm{U}(O)$. This induces ultimate versions of the various AFT fixpoints.[3] In the context of logic programming, the step from the standard approximator (which is Fitting's partial immediate consequence operator (Fitting, 2002)) to the ultimate approximator roughly boils down to using *supervaluations* (Fitting, 1994; van Fraassen, 1966). The ultimate approximator of an operator $O\colon L \to L$ has the following form (Denecker et al., 2004):

$$\mathrm{U}(O)\colon L^c \to L^c\colon (x,y) \mapsto \left(\bigwedge_{x \leq z \leq y} O(z), \bigvee_{x \leq z \leq y} O(z) \right),$$

where $\bigwedge$ (respectively $\bigvee$) are the greatest lower (respectively least upper) bound with respect to $\leq$. If an approximator A approximates an operator O, then we abuse notation by defining $\mathrm{U}(A) \coloneqq \mathrm{U}(O)$.

In the context of justification theory, the justification frame uniquely determines the approximator at hand. Still, we show that it is possible to obtain ultimate semantics here as well. To do so, we will develop a method to transform a justification frame $\mathcal{JF}$ into its ultimate frame $\mathrm{U}(\mathcal{JF})$. We will then show that the approximator associated to $\mathrm{U}(\mathcal{JF})$ is indeed the ultimate approximator of $O_{\mathcal{JF}}$. The result is a generic mechanism to go from any semantics induced by justification theory (for arbitrary branch evaluations – not just for those that have an AFT counterpart) to an ultimate variant thereof. Our construction is as follows:

[3] This added precision has an increased computational cost (Denecker et al., 2004, Theorems 6.12 and 6.13).

Definition 5.4.1. Let $\mathcal{JF}$ be a complementary justification frame. Let X be the set of rules with a positive head. Let X^* be the least (w.r.t. $\subseteq$) set containing X that is closed under the addition of rules $x \leftarrow A$

- if there is a rule $x \leftarrow B$ with $B \subseteq A$, or
- if there are rules $x \leftarrow \{y\} \cup A$ and $x \leftarrow \{\sim y\} \cup A$.

Let Y be the complementation of X. Then $\mathrm{U}(\mathcal{JF})$ is defined to be the complementary justification frame $\langle \mathcal{F}, \mathcal{F}_d, Y \rangle$. We call $\mathrm{U}(\mathcal{JF})$ the *ultimate frame* of $\mathcal{JF}$.

Let us look at an example.

Example 5.4.2. Let $\mathcal{F}_d = \{x, \sim x\}$ and $\mathcal{F}_o = \{a, \sim a, b, \sim b\} \cup \mathcal{F}_o$. Take R to be the complementation of

$$\left\{ \begin{array}{l} x \leftarrow a, x \\ x \leftarrow b, \sim x \end{array} \right\},$$

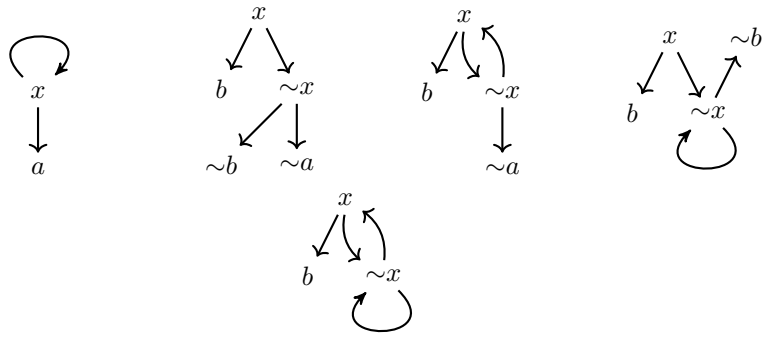
which adds the following rules

$$\left\{ \begin{array}{l} \sim x \leftarrow \sim a, \sim b \\ \sim x \leftarrow \sim a, x \\ \sim x \leftarrow b, \sim x \\ \sim x \leftarrow x, \sim x \end{array} \right\}.$$

To determine the supported value, it suffices to take non-redundant rules into account. Recall that a rule $x \leftarrow A$ is non-redundant if there is no rule $x \leftarrow B$ with $B \subsetneq A$. The justification frame $\mathrm{U}(\mathcal{JF})$ has exactly the following non-redundant rules:

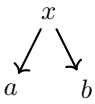
$$\left\{ \begin{array}{l} x \leftarrow a, x \\ x \leftarrow b, \sim x \\ x \leftarrow a, b \\ \sim x \leftarrow \sim a, \sim b \\ \sim x \leftarrow \sim a, x \\ \sim x \leftarrow \sim b, \sim x \end{array} \right\}.$$

Of course, it contains many redundant rules, for example $x \leftarrow a, b, x$. The justifications in the original system containing x as root are exactly the following:



Assume from now on we are working under $\mathcal{B}_{\text{st}}$. The value of the upper left justification for x is $\mathbf{f}$ in every interpretation. The values of the other justifications for x are at most $\mathbf{u}$ in $\mathcal{B}_{\text{st}}$-models. If it would be $\mathbf{t}$, then the value of these justifications for x is equal to the value of $\sim x$, which is $\mathbf{f}$.

By taking the ultimate justification frame, the non-redundant rule $x \leftarrow a, b$ is added and the non-redundant rule $\sim x \leftarrow x, \sim x$ is removed. This allows for the justification



If the interpretation of a and b is $\mathbf{t}$, then the value of this justification for x is $\mathbf{t}$. Therefore, $(\{a, b, x\}, \{a, b, x\})$ is an ultimate stable model, while not a stable model. Note that the lower justification is not a justification in $\text{U}(\mathcal{JF})$. If it would be, then this is a true justification for $\sim x$ contradicting the consistency. ▲

It can be seen that the construction adds rules to $\mathcal{JF}$ in two cases. For the first type, if $x \leftarrow B$ is a rule in R with $B \subseteq A$, then if B is sufficient to derive x, clearly so is A. The second type of rule addition essentially performs some sort of case splitting. It states that if a set of facts A can be used with either y or $\sim y$ to derive x, then the essence for deriving x is the set A itself. In that case, the rule $x \leftarrow A$ is added to the ultimate frame. This case splitting is actually a bit more general as shown in the following proposition.

Proposition 5.4.3. $\text{U}(\mathcal{JF})$ *can also be constructed using the following variation: closed under the addition of*

- *a rule* $x \leftarrow A$ *if there is a rule* $x \leftarrow B$ *with* $B \subseteq A$*, or*

- *a rule* $x \leftarrow A \cup B$ *if there are rules* $x \leftarrow A \cup \{y\}$ *and* $x \leftarrow B \cup \{\sim y\}$.

Proof. It suffices to prove that the second type is implied by the standard rules. Take two rules $x \leftarrow A \cup \{y\}$ and $x \leftarrow B \cup \{\sim y\}$. This means that the rules $x \leftarrow A \cup B \cup \{y\}$ and $x \leftarrow A \cup B \cup \{\sim y\}$ are also in the frame. Therefore, $x \leftarrow A \cup B$ is also in the frame. □

It turns out that this rule of case splitting is indeed sufficient to reconstruct the ultimate semantics in justification theory. This is formalised in the main theorem of this section, for which the proof will be postponed a bit.

Theorem 5.4.4. *For any complementary frame* $\mathcal{JF}$, $A_{\mathrm{U}(\mathcal{JF})} = \mathrm{U}(O_{\mathcal{JF}})$.

An immediate corollary is, for instance that the set of stable models of $\mathrm{U}(\mathcal{JF})$ equals the set of ultimate stable fixpoints of $O_{\mathcal{JF}}$, and similarly for other semantics.

Recall that, in the context of lattices with the subset order, which is what we are concerned with here, the ultimate approximator is equal to (Denecker et al., 2004):

$$\mathrm{U}(O)(I_1, I_2) = \left(\bigcap_{I_1 \subseteq K \subseteq I_2} O(K), \bigcup_{I_1 \subseteq K \subseteq I_2} O(K) \right). \tag{5.1}$$

The proof of Theorem 5.4.4 makes use of the following intermediate results.

Lemma 5.4.5. *Let* $\mathcal{I}$ *be an interpretation and* $x \in \mathcal{F}_d$. *If* $O_{\mathcal{JF}}(\mathcal{I}')(x) = \mathbf{t}$ *(respectively* $\mathbf{f}$*) for all exact interpretations* $\mathcal{I}'$ *with* $\mathcal{I}' \geq_p \mathcal{I}$, *then* $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(x) = \mathbf{t}$ *(respectively* $\mathbf{f}$*).*

Proof. Let $\mathcal{I} = (I_1, I_2)$. We first prove this for the case $\mathbf{t}$. Define $X = \{y \in \mathcal{F}_d \mid \mathcal{I}(x) = \mathbf{u}\}$. We prove for all $Y \subseteq X$ and all complete consistent subsets A over $X \setminus Y$ that the rule $x \leftarrow \{\mathbf{t}\} \cup I_1 \cup \sim(\mathcal{F}_+ \setminus I_2) \cup A$ is a rule in $\mathrm{U}(\mathcal{JF})$. A complete consistent subset A of Z is a subset such that for each $z \in Z$ exactly one of z and $\sim z$ is in A. If $Y = X$, then we get that $x \leftarrow \{\mathbf{t}\} \cup I_1 \cup \sim(\mathcal{F}_+ \setminus I_2)$ is a rule in $\mathrm{U}(\mathcal{JF})$ with every element y in its body we have $\mathcal{I}(y) = \mathbf{t}$, completing our proof. We prove our claim by transfinite induction. Assume $Y = \emptyset$. Take a complete consistent set A over X. Define $K = I_1 \cup (A \cap \mathcal{F}_+)$. Since $\mathcal{I} \leq_p (K, K)$, we have a rule $x \leftarrow B$ in $\mathcal{JF}$ with $(K, K)(b) = \mathbf{t}$ for all $b \in B$. The true facts under B are equal to $\{\mathbf{t}\} \cup I_1 \cup \sim(\mathcal{F}_+ \setminus I_2) \cup A$. Take $b \in B$. Therefore, we can extend this rule to $x \leftarrow I_1 \cup \sim(\mathcal{F}_+ \setminus I_2) \cup A$ in $\mathrm{U}(\mathcal{JF})$.

Take $Y \neq \emptyset$. Assume by induction the claim holds for all $Y' \subsetneq Y$. Take any $y \in Y$ and a complete consistent set A over $X \setminus Y$. Then $A \cup \{y\}$ and $A \cup \{\sim y\}$ are complete consistent sets over $X \setminus (Y \setminus \{y\})$. So by induction, there are rules $x \leftarrow I_1 \cup (\mathcal{F}_+ \setminus I_2) \cup A \cup \{y\}$ and $x \leftarrow I_1 \cup (\mathcal{F}_+ \setminus I_2) \cup A \cup \{y\}$ in $\mathrm{U}(\mathcal{JF})$. This means that $x \leftarrow I_1 \cup (\mathcal{F}_+ \setminus I_2) \cup A$ is a rule in $\mathrm{U}(\mathcal{JF})$.

The case for **f** follows easily from the **t** case. We have for all exact interpretations $\mathcal{I}'$ with $\mathcal{I} \leq_p \mathcal{I}'$ that $A_{\mathcal{JF}}(\mathcal{I}')(x) = \mathbf{f}$. Then by consistency, $A_{\mathcal{JF}}(I, I)(\sim x) = \mathbf{t}$. By Lemma 5.4.5, we have that $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(\sim x) = \mathbf{t}$; hence $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(x) = \mathbf{f}$. □

Combining this lemma with Eq. (5.1) of the ultimate approximator immediately yields that the operator $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})$ is as least as precise as the ultimate approximator of $O_{\mathcal{JF}}$.

Lemma 5.4.6. *For all interpretations $\mathcal{I}$ we have* $\mathrm{U}(O_{\mathcal{JF}})(\mathcal{I}) \leq_p A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})$.

Proof. Take $x \in \mathcal{F}_+$. If $\mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x) = \mathbf{u}$, then it is obvious that $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(x) \leq_p \mathbf{u} = \mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x)$.

If $\mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x) = \mathbf{t}$, then $x \in \cap_{I_1 \subseteq K \subseteq I_2} A_{\mathcal{JF}}(K, K)_1$; hence $A_{\mathcal{JF}}(K, K)(x) = \mathbf{t}$ for all exact interpretations $(K, K) \geq_p \mathcal{I}$. Therefore, by Lemma 5.4.5, we have that $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(x) = \mathbf{t} = \mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x)$.

If $\mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x) = \mathbf{f}$, then $x \notin \cup_{I_1 \subseteq K \subseteq I_2} A_{\mathcal{JF}}(K, K)_1$; hence $A_{\mathcal{JF}}(K, K)(x) = \mathbf{f}$ for all exact interpretations $(K, K) \geq_p \mathcal{I}$. By Lemma 5.4.5 it follows that $A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(x) = \mathbf{f} = \mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x)$.

Similarly, we can prove for all $x \in \mathcal{F}_-$ that $\mathrm{U}_{\mathcal{JF}}(\mathcal{I})(x) \leq_p A_{\mathrm{U}(\mathcal{JF})}(\mathcal{I})(x)$. □

Since the ultimate approximator is the most precise approximator of any given operator, all that is left to prove, to indeed obtain Theorem 5.4.4 is that $A_{\mathrm{U}(\mathcal{JF})}$ indeed approximates $O_{\mathcal{JF}}$. That is the content of the last lemma.

Lemma 5.4.7. *$A_{\mathrm{U}(\mathcal{JF})}$ is an approximator of $O_{\mathcal{JF}}$.*

Proof. Take $I \subseteq \mathcal{F}_+$. Adding a rule $x \leftarrow B$ to $\mathcal{JF}$ if $x \leftarrow A$ is in $\mathcal{JF}$ with $A \subsetneq B$, does not change $A_{\mathcal{JF}}(I, I)$. Similarly, adding a rule $x \leftarrow A$ to $\mathcal{JF}$ if $x \leftarrow A \cup \{y\}$ and $x \leftarrow A \cup \{\sim y\}$ are in $\mathcal{JF}$ does not change $A_{\mathcal{JF}}(I, I)$. This means that $O_{\mathcal{JF}}(I) = A_{\mathcal{JF}}(I, I) = A_{\mathrm{U}(\mathcal{JF})}(I, I)$. □

5.5 Conclusion

In this chapter, we presented a general mechanism to translate justification frames into approximating operators and showed that this transformation preserves all semantics the two formalisms have in common. The correspondence we established provides ample opportunity for future work and in fact probably generates more questions than it answers.

By embedding justification theory in AFT, justification theory gets access to a rich body of theoretical results developed for AFT, but of course said results are only directly applicable to branch evaluations that have a counterpart in AFT. A question that immediately arises is whether results such as stratification results also apply to other branch evaluations, and which assumptions on branch evaluations would be required for that. Another question that pops up on the justification theory side is whether concepts such as *groundedness* Bogaerts et al. (2015a) can be transferred.

On the AFT side, this embedding calls for a general algebraic study of *explanations*. Indeed, for certain approximators, namely those that "come from" a justification frame, our results give us a method for answering certain *why* questions in a graph-based manner (with justifications). Lifting this notion of explanation to general approximators would benefit domains of logics that are covered by AFT but not by justification theory.

A last question that emerges is how nesting of justification frames (see Chapter 6) fits into this story, and whether it can give rise to notions of nested operators on the AFT side.

Chapter 6

Nested Justification Systems

6.1 Introduction

An important property in knowledge representation is modularity. Being able to compose different semantics is key to achieving modularity. In justification theory, composition is obtained by *nesting* systems, which was introduced by Denecker et al. (2015), but not extensively discussed. This chapter further explores nested justification systems. Denecker et al. (2015) defined the semantics of nested systems by *compressing* to an unnested system. One shortcoming of this approach is that we consider justifications in the compressed system, and thus information is lost. We introduce an alternative view on nested systems that avoids this loss of information. We prove that this alternative view is strongly equivalent to the compression approach when we are dealing with tree-like justifications, and we provide some partial results in case of graph-like justifications. The consistency of the compression has interesting relations between graph-reducibility and tree-like consistency. As a bonus, we completely solve the tree-like consistency. The chapter is closed by showing how aggregates, first-order logic definitions and fixpoint definitions can be captured with nested justification systems.

Apart from the recalling of the definitions of nested systems and their compression (Denecker et al., 2015), the text of this chapter is entirely new and unpublished work.

6.2 Basic Definitions

In order to understand the need for nested systems, we need to take a look at some inductive definitions. Essentially facts are propositional in nature, but because we allow infinitely many in rule bodies, we can use compound formulae. Suppose we want to inductively define the reachability relation of graph over a set of nodes N. A fairly well-known representation as an inductive definition is given below.

$$\forall x \in N, y \in N \colon reachable(x, y) \leftarrow edge(x, y).$$

$$\forall x \in N, y \in N \colon reachable(x, y) \leftarrow \exists z \colon edge(x, z) \wedge reachable(z, y).$$

Intuitively, the first rule states the base state for the induction: if there is an edge from x to y, then y is reachable from x. The second rule states the induction step: if there is an edge from x to a z such that y is reachable from z, then surely y must be reachable from x. We can represent this in a justification frame as follows: Take $\mathcal{F}_+ \cap \mathcal{F}_d = \{reachable(x, y) \mid x, y \in N\}$, $\mathcal{F}_o \cap \mathcal{F}_+ = \{edge(x, y) \mid x, y \in N\}$. Then the justification frame is the complementation of the set

$$\{reachable(x, y) \leftarrow edge(x, y) \mid x, y \in N\}$$

$$\cup \{reachable(x, y) \leftarrow edge(x, z), reachable(z, y) \mid x, y, z \in N\}.$$

The universal quantifications are actually transformed into a number of rules. Note that the facts $edge(x, y)$ (and $\sim edge(x, y)$) are open facts and thus serve as parameters. This means that depending on the interpretation of the opens, it represents the reachability definition of a different graph; hence the graph is a parameter of this system. Since induction is associated with well-founded semantics (Denecker, 1998; Denecker and Vennekens, 2014), we can use $\mathcal{B}_{\text{wf}}$. If the interpretation of $\mathcal{F}_o$ is fixed and two-valued, that is, the graph on N is known, then the unique $\mathcal{B}_{\text{wf}}$-model is two-valued since our rules do not allow for mixed loops. In that case, the unique $\mathcal{B}_{\text{wf}}$-model can be computed by iteratively applying the approximator $A_{\mathcal{JF}}$ until fixpoint starting from the interpretation that is $\mathbf{u}$ on all defined facts. This corresponds neatly to applying the different rules in the inductive definition. So because of our AFT connection from Chapter 5, any inductive definition that can be faithfully represented as a propositional inductive definition can be captured by justification theory, see Section 6.5.3.

Justification theory handles inductive definitions well if the body is a set of literals. This begs the question at what happens if the bodies contain complex

formulae. Let us now look at when the edges are not determined from outside, but from the inside. Suppose we want to look at the following graph[1] on $\mathbb{N}$:

- $(x, x/2)$ if x is even
- $(x, 3x + 1)$ if x is odd.

Let $\ell_{(x,y)}$ be $\mathbf{t}$ if (x, y) is in this graph and $\mathbf{f}$ otherwise. The graph can be represented by the complementation of the following rule set

$$\left\{ edge(x, y) \leftarrow \ell_{(x,y)} \;\middle|\; x, y \in \mathbb{N} \right\}.$$

Adding this rule set to the justification system above will produce a system that captures this definition correctly. This technique also works for any graph and no matter how the graph is defined (as long its definition can be captured with well-founded semantics). Two problems exists with this approach. First, it does not make a distinction between the definition of the reachability relation and the definition of the graph; the two are mushed together and the knowledge representation is not modular. Second, if the definition of the graph cannot be captured with well-founded semantics, then this might not work.

We illustrate these problems in another related area: *predicate introduction* (Vennekens et al., 2007b). Predicate introduction is the act of replacing a complex formula by a newly defined predicate. This can be used to simplify or eliminate redundancy of a theory. However, predicate introduction might not be equivalence preserving.

Example 6.2.1. Suppose you have a justification frame with the rule $p \leftarrow p$. If we now want to introduce a new predicate T representing $\sim p$, then it will introduce a rule $T \leftarrow \sim p$. Because p is equal to $\sim\sim p$, we can replace p with $\sim T$, so we replace the rule $p \leftarrow p$ with $p \leftarrow \sim T$. So our predicate introduction produces the justification frame

$$\left\{ \begin{array}{l} p \leftarrow \sim T \\ T \leftarrow \sim p \end{array} \right\}$$

Under $\mathcal{B}_{\mathrm{wf}}$, this will have the model in which p is $\mathbf{u}$ instead of $\mathbf{f}$ in the original system. ▲

This situation is analogous to the graph reachability definition. The definition of p corresponds to the definition of graph reachability itself, while the definition of T corresponds to the definition of the graph. Putting together both definitions

[1] This is the graph for the Collatz conjecture, a famous unsolved mathematical problem stating for every $x \in \mathbb{N}$, $(x, 1)$ is in the reachability relation of this graph.

does not provide the desired effect. Instead, we want a way that makes the definition of T inferior to p, first evaluating what the value of T is and then filling it in. If we can achieve this, then we can perform any predicate introduction in an equivalence preserving manner.

We have seen that introducing a new predicate might not be equivalence-preserving, see Example 6.2.1. We have hinted that we can solve this by having a local (the definition for the newly introduced predicate) justification frame for which we find the model and then substitute the predicate with the value of the found model in the main justification frame. Stretching this idea further, it might be that the local justification frame has a different semantics than the main justification frame, allowing for predicate introduction with a different semantics. This brings us to the definition of nested justification systems, which were invented by Denecker et al. (2015). Nested justification systems are essentially a tree structure of justification systems, meaning that some systems are local to certain others.

Definition 6.2.2. Let $\mathcal{F}$ be a fact space. A *nested justification system* on $\mathcal{F}$ is a tuple $\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$ such that

1. $\langle \mathcal{F}, \mathcal{F}_{dl}, R, \mathcal{B}\rangle$ is a parametric justification system;
2. for each i, $\mathcal{JS}^i$ is a nested justification system $\langle \mathcal{F}^i, \mathcal{F}^i_d, \mathcal{F}^i_{dl}, R^i, \mathcal{B}^i, \ldots\rangle$ (we write $\mathcal{F}^i_o$ for $\mathcal{F}^i \setminus \mathcal{F}^i_d$ as usual);
3. $\mathcal{F}_d$ is partitioned into $\{\mathcal{F}_{dl}, \mathcal{F}^1_d, \ldots, \mathcal{F}^k_d\}$;
4. $\mathcal{F} = \cup_{i=1}^k \mathcal{F}^k$;
5. $\mathcal{F}^i_o = \mathcal{F}_o \cup \mathcal{F}_{dl}$, where $\mathcal{F}_o = \mathcal{F} \setminus \mathcal{F}_d$ as usual.

This defines a tree of nested justification systems, where the leaves have $\mathcal{F}_{dl} = \mathcal{F}_d$ and $k = 0$, which corresponds to an unnested justification system. Every fact is either defined locally in the top system ($\mathcal{F}_{dl}$), or in one of the subsystems $\mathcal{F}^i_d$. The open facts in $\mathcal{JS}^i$ are the open facts of the root augmented with the locally defined facts (defined in the root). This has the consequence that facts defined in $\mathcal{JS}^i$ do not appear as open facts in $\mathcal{JS}^j$ if $i \neq j$.

Lemma 6.2.3. *Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$ be a nested justification system. If $i \neq j$, then $\mathcal{F}^i_d \cap \mathcal{F}^j = \emptyset$.*

Proof. Take $i \neq j$. Because $\mathcal{F}_d$ is partitioned into $\{\mathcal{F}_{dl}, \mathcal{F}^1_d, \ldots, \mathcal{F}^k_d\}$, it suffices to prove that $\mathcal{F}^i_d \cap \mathcal{F}^j_o = \emptyset$. This follows from (5) in Definition 6.2.2. □

To make the nesting structure more apparent, nested systems can also be viewed as a tree of unnested justification systems.

Proposition 6.2.4. *A nested justification system*

$$\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$$

corresponds to a tree of justification systems, called the nesting tree *of* $\mathcal{JS}$*, such that*

1. *The root of the tree is* $\langle \mathcal{F}, \mathcal{F}_{dl}, R, \mathcal{B}\rangle$*;*
2. *Each element in* $\mathcal{F}_d$ *is a defined fact in exactly one system;*
3. *Each element in* $\mathcal{F}_o$ *is an open fact in every system;*
4. *A fact defined in a system* $\mathcal{JS}_1$ *occurs as open fact in another system* $\mathcal{JS}_2$ *if and only if* $\mathcal{JS}_2$ *is an ancestor or descendant of* $\mathcal{JS}_1$*;*
5. *The root has* k *children and for each* $1 \leq i \leq k$ *the subtree rooted in the* i*th child of the root is the nesting tree of* $\mathcal{JS}^i$*.*

Proof. By combining the first and last point, the tree is uniquely determined. The rest of the conditions follow from the definition of nested justification systems. □

Most of the time, we will refer to the unnested systems in the nesting tree, instead of the subsystems $\mathcal{JS}^i$ from the definition of nested systems. A system that is closer to the root than another system is said to have a higher level. The depth of a nested system is the depth of the nesting tree, that is, the length of the longest path in the nesting tree. For a unnested system, this is one. Let us look at a simple example of a nested justification system.

Example 6.2.5. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \mathcal{JS}^2\}\rangle$ be a nested system with $\mathcal{F}_+ = \{x, a, b, c, d\}$, $\mathcal{F}_{dl} = \{x, \sim x\}$, $\mathcal{B} = \mathcal{B}_{\mathrm{KK}}$, and R equal to the complementation of

$$\{\ x \leftarrow a, b\ \}.$$

The inner system $\mathcal{JS}^1$ is equal to the unnested system with $\mathcal{F}^1_{dl} = \{a, \sim a\}$, $\mathcal{B}^1 = \mathcal{B}_{\mathrm{cwf}}$, and R^1 equal to the complementation of

$$\{\ a \leftarrow a\ \}.$$

The inner system $\mathcal{JS}^2$ is a nested system $\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^3, \mathcal{JS}^4\}\rangle$ with $\mathcal{F}^2_+ = \{b, c, d\}$, $\mathcal{F}^2_{dl} = \{b, \sim b\}$, $\mathcal{B}^2 = \mathcal{B}_{\mathrm{KK}}$, and R^2 equal to the complementation of

$$\{\ b \leftarrow c, d\ \}.$$

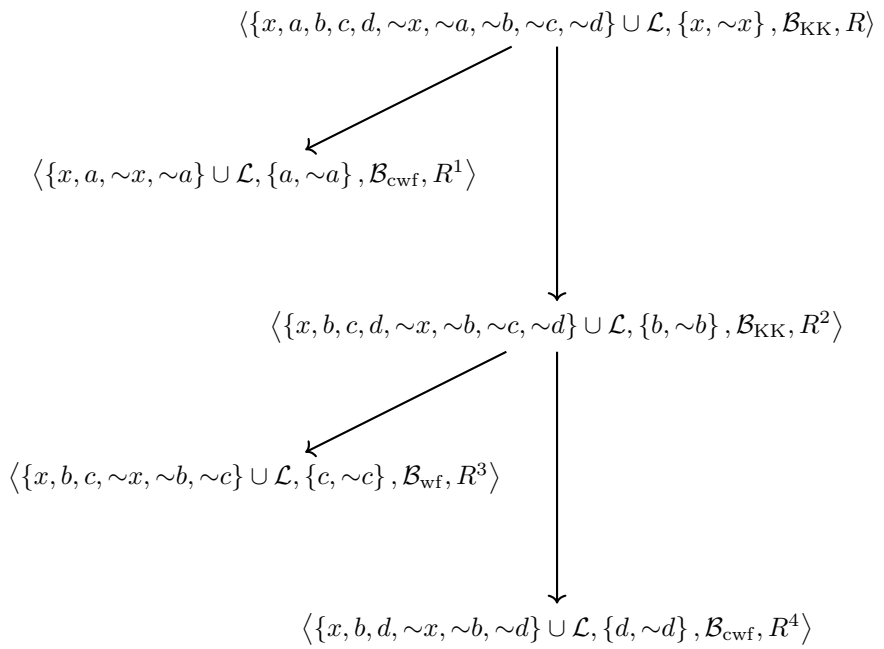
The system $\mathcal{JS}^3$ is unnested with $\mathcal{F}^3_{dl} = \{c, \sim c\}$, $\mathcal{B}^3 = \mathcal{B}_{\mathrm{wf}}$, and R^3 equal to the complementation of

$$\left\{\ c \leftarrow c\ \right\}.$$

The system $\mathcal{JS}^4$ is the unnested system with $\mathcal{F}^4_{dl} = \{d, \sim d\}$, $\mathcal{B}^4 = \mathcal{B}_{\mathrm{cwf}}$, and R^4 equal to the complementation of

$$\left\{\ d \leftarrow d\ \right\}.$$

The system $\mathcal{JS}$ has depth 3 and its nesting tree is equal to



▲

Using the nesting tree to represent nested systems graphically is a bit unwieldy. A lot of information in this tree is superfluous. For example, the set of defined facts can be extracted from the set of rules. The open facts are exactly the defined facts of ancestor or descendant nodes plus the facts that are open everywhere. In principle, just the set of rules is enough to represent a nested system. If we are working with complementary rule sets, then we can also leave out the rules for one of x and $\sim x$. This is illustrated in the following two examples

Example 6.2.6. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1\}\rangle$ be a nested justification system with $\mathcal{F}_+ = \{z, x, w, v\}$, $\mathcal{F}_{dl} = \{z, w, \sim z, \sim w\}$, and R equal to the complementation of

$$\left\{ \begin{array}{l} z \leftarrow x \\ w \leftarrow x \end{array} \right\}.$$

The inner system has $\mathcal{F}^1 = \mathcal{F}$, $\mathcal{F}^1_{dl} = \mathcal{F}^1_d = \{x, v, \sim x, \sim v\}$, R^1 the complementation of

$$\left\{ \begin{array}{l} x \leftarrow v, w \\ v \leftarrow v \end{array} \right\}.$$

We can represent this as follows:

$$\left\{ \begin{array}{l} z \leftarrow x \\ w \leftarrow x \\ \left\{ \begin{array}{l} x \leftarrow v, w \\ v \leftarrow v \end{array} \right\} \end{array} \right\}$$

The corresponding nesting tree is a graph with two nodes, where the top level system has the rules for $(\sim)z$ and $(\sim)w$, while the leaf system has the rules for $(\sim)x$ and $(\sim)v$. Here, we see that x is an open fact in the top level. Similarly, w is an open fact in the lowest level. ▲

In the example above, the information of the branch evaluations is left out. This can be implicit or can be made explicit by adding '$\mathcal{B}$: ' before the braces.

Example 6.2.7. The graphical representation of the nested system of Example 6.2.5 is as follows:

$$\mathcal{B}_{\mathrm{KK}}: \left\{ \begin{array}{l} x \leftarrow a, b \\ \mathcal{B}_{\mathrm{cwf}}: \left\{ a \leftarrow a \right\} \\ \mathcal{B}_{\mathrm{KK}}: \left\{ \begin{array}{l} b \leftarrow \sim c, d \\ \mathcal{B}_{\mathrm{wf}}: \left\{ c \leftarrow c \right\} \\ \mathcal{B}_{\mathrm{cwf}}: \left\{ d \leftarrow d \right\} \end{array} \right\} \end{array} \right\}$$

▲

Nesting so far has been syntactically, and we still need to attach meaning to it. The way Denecker et al. (2015) do this, is by transforming it to a regular justification system. Essentially, the transformation compresses the nested system into a normal one, by starting from the leaves of the nesting tree.

Since every branch evaluation in a nested justification system is parametric, we know that there is an unique model if the interpretation of the open facts is fixed (see Proposition 2.5.4). The value of a fact in this model depends solely

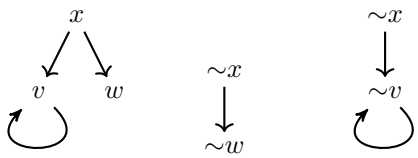
on the values of the open facts. Therefore, we can represent the model by a set of rules for which the body only contains open facts. This representation is formed by transforming each justification into a single rule, by an operation called flattening.

Definition 6.2.8. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system. The *flattening* $\mathrm{Flat}(\mathcal{JS})$ is the justification system $\langle \mathcal{F}, \mathcal{F}_d, R^f, \mathcal{B} \rangle$, where

$$R^f = \{x \leftarrow A \mid x \in \mathcal{F}_d, J \in \mathfrak{J}_x, A = \{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\}\}.$$

This is called flattening because every locally complete justification is reduced to a single rule, and because justifications in this new system have, when $\mathcal{B}$ is parametric, only branches of length two.

Example 6.2.9. Take the leaf system of the nesting tree of Example 6.2.6 and evaluate in $\mathcal{B}_{\mathrm{wf}}$. The only connected locally complete justifications are



The left justification gives use the rules $x \leftarrow \mathbf{f}, w$ and $v \leftarrow \mathbf{f}$, the middle one $\sim x \leftarrow \sim w$, and the right one $\sim x \leftarrow \mathbf{t}$ and $\sim v \leftarrow \mathbf{t}$. ▲

Lemma 6.2.10. *If $\mathcal{B}$ is parametric, then every justification in* $\mathrm{Flat}(\mathcal{JS})$ *only has branches of length two.*

Proof. Take $x \in \mathcal{F}_d$. If $x \to y$ is the start of a J-branch, then there is a locally complete justification J^* in $\mathcal{JS}$ with x as internal node such that $y = \mathcal{B}(\mathbf{b})$ for some J^*-branch starting with x. Since $\mathbf{b}$ is parametric, $\mathcal{B}(\mathbf{b})$ is open; concluding our proof. □

If these simple branches are mapped to their last element, then the flattening is strongly equivalent to the original system.

Proposition 6.2.11. *If $\mathcal{B}$ is parametric and maps branches $x \to y$ to y, then $\mathcal{S}_{\mathcal{JS}} = \mathcal{S}_{\mathrm{Flat}(\mathcal{JS})}$, i.e. $\mathcal{JS}$ is strongly equivalent to* $\mathrm{Flat}(\mathcal{JS})$.

Proof. Take an $\mathcal{F}$-interpretation $\mathcal{I}$ and $x \in \mathcal{F}_d$. Let J be a $\mathcal{JS}$-justification such that $\mathrm{SV}_{\mathcal{JS}}(x, \mathcal{I}) = \mathrm{val}(J, x, \mathcal{I})$. Then by construction of $\mathrm{Flat}(\mathcal{JS})$, this

J corresponds to a rule $x \leftarrow A$ in $\text{Flat}(\mathcal{JS})$ such that $A \subseteq \mathcal{F}_o$ and for all $y \in A$ it holds that $\mathbf{b}(x \to y) \geq_t \text{val}(J, x, \mathcal{I})$. The rule $x \leftarrow A$ is a locally complete justification in $\text{Flat}(\mathcal{JS})$. Therefore, $\text{SV}_{\text{Flat}(\mathcal{JS})}(x, \mathcal{I}) \geq_t \text{val}(J, x, \mathcal{I}) = \text{SV}_{\mathcal{JS}}(x, \mathcal{I})$. Similarly, every locally complete justification in $\text{Flat}(\mathcal{JS})$ with x as root is a single rule $x \leftarrow A$ such that $A = \{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\}$ for some locally complete justification J in $\mathcal{JS}$. Therefore, $\text{SV}_{\mathcal{JS}}(x, \mathcal{I}) \geq_t \text{SV}_{\text{Flat}(\mathcal{JS})}(x, \mathcal{I})$. □

As a consequence, flattening preserves the properties of the justification at hand. We should note, however, that the structure of the justification is lost: justifications in $\text{Flat}(\mathcal{JS})$ are condensed down. We will come back to this later.

Proposition 6.2.12. *Let $\mathcal{JS}_1$ and $\mathcal{JS}_2$ be two parametric justification systems that map $x \to y$ to y. Then $\mathcal{JS}_1$ and $\mathcal{JS}_2$ are (strongly) equivalent if and only if $\text{Flat}(\mathcal{JS}_1)$ and $\text{Flat}(\mathcal{JS}_2)$ are (strongly) equivalent.*

Proof. The strong equivalence result follows from Proposition 6.2.11 and by looking at the diagram below.

$$\begin{array}{ccc} \mathcal{S}_{\mathcal{JS}_1} & = & \mathcal{S}_{\text{Flat}(\mathcal{JS}_1)} \\ \| & & \| \\ \mathcal{S}_{\mathcal{JS}_2} & = & \mathcal{S}_{\text{Flat}(\mathcal{JS}_2)} \end{array}$$

Proposition 6.2.11 implies that the models of $\mathcal{JS}$ and $\text{Flat}(\mathcal{JS})$ are equal, i.e. $\mathcal{JS}$ and $\text{Flat}(\mathcal{JS})$ are equivalent. The regular equivalence result follows then by looking at the diagram above by interpreting the equality signs as equivalences between the underlying systems instead. □

Because of these nice properties, we will actually demand that finite branches should be mapped to their last element by all branch evaluations involved in nested systems. Denecker et al. (2015) did not require this because they demanded that the flattening[2] was to be evaluated using $\mathcal{B}_{\text{sp}}$, which maps finite branches $x \to y$ to y. This suggest that they understood the issue that the flattening is not always strongly equivalent to the original system. By introducing the property that finite branches should be mapped to their last element, we circumvent this problem. Moreover, this property will also be important in Section 6.3, when we introduce an alternative view on nested systems.

[2]The flattening construction was only implicitly used in that paper.

So far, we have a way to evaluate inner definitions, but not a way to eliminate the inner symbols from the outside definitions. The following example shows how to do that.

Example 6.2.13. Let us take another look at Example 6.2.6. By replacing the inner system with its flattening, we get

$$\left\{\begin{array}{l} z \leftarrow x \\ w \leftarrow x \\ \sim z \leftarrow \sim x \\ \sim w \leftarrow \sim x \\ \left\{\begin{array}{l} x \leftarrow \mathbf{f}, w \\ v \leftarrow \mathbf{f} \\ \sim x \leftarrow \mathbf{t} \\ \sim x \leftarrow \sim w \\ \sim v \leftarrow \mathbf{t} \end{array}\right\} \end{array}\right\}.$$

In the inner system, we see that the value of x is determined by the value of a case of x. By replacing x in the top level by a case of x, we eliminate x from the the top level. For example, $z \leftarrow x$ becomes $z \leftarrow \mathbf{f}, w$. Doing this for all rules and all cases for defined facts in the lower level we get the following system:

$$\left\{\begin{array}{l} z \leftarrow \mathbf{f}, w \\ w \leftarrow \mathbf{f}, w \\ \sim z \leftarrow \mathbf{t} \\ \sim z \leftarrow \sim w \\ \sim w \leftarrow \mathbf{t} \\ \sim w \leftarrow \sim w \end{array}\right\}.$$

Note that the rules for $\sim z$ and $\sim w$ are doubled since $\sim x$ has two cases. The newly formed system will take the facts of the lowest level implicitely into account. If you want this to be explicit, the inner system can be added as well. ▲

The construction in this example where rules are transformed by replacing facts of a lower level by a case for that fact is called unfolding and is formally defined below.

Definition 6.2.14. Let R be a set of rules, and R_ℓ a set of rules for the facts X (the elements of X are the heads of the rules in R_ℓ). Take a rule $x \leftarrow A$ in R. Let f be any function with domain $A \cap X$ such that for all $y \in A \cap X$, $y \leftarrow f(y)$ is a rule in R_ℓ (the function f chooses a rule for each $y \in A \cap X$). The *unfolding* of $x \leftarrow A$ with respect to f is the rule

$$\mathrm{Unf}_f\,(x \leftarrow A) = x \leftarrow (A \setminus X) \cup \bigcup_{y \in A \cap X} f(y).$$

Let $F_{x \leftarrow A}$ be the set of such functions f. Then the *unfolding* of $x \leftarrow A$ with respect to R_ℓ is the set

$$\mathrm{Unf}_{R_\ell}(x \leftarrow A) = \{\mathrm{Unf}_f(x \leftarrow A) \mid f \in F_{x \leftarrow A}\}.$$

The *unfolding* of R with respect to R_ℓ is

$$\mathrm{Unfold}_{(X,R_\ell)}(R) = \bigcup_{x \leftarrow A \in R} \mathrm{Unf}_{R_\ell}(x \leftarrow A).$$

In Example 6.2.13, the last set of rules is the unfolding of the inner set of rules on the outer set of rules. In that example, a nested justification system is transformed to an unnested one by unfolding, so we can define unfolding also on systems with a single nesting. However, in order to keep the set of defined facts the same, we keep the rules of the inner systems.

Definition 6.2.15. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$ be a two-level nested justification system. The *unfolding* of $\mathcal{JS}$ is

$$\mathrm{Unfold}(\mathcal{JS}) = \langle \mathcal{F}, \mathcal{F}_d, R^s \cup \mathrm{Unfold}_{(\mathcal{F}_d \setminus \mathcal{F}_{dl}, R^s)}(R), \mathcal{B}\rangle,$$

where $R^s = \cup_{i=1}^k R^i$.

Intuitively, it unfolds the rules of the child systems into the top system and keeps the rules for the lower level facts (R^s).

Example 6.2.16. The unfolding of the nested system from Example 6.2.13 is given below (the rules for negative facts are left out for brevity).

$$\left\{ \begin{array}{l} z \leftarrow \mathbf{f}, w \\ w \leftarrow \mathbf{f}, w \\ x \leftarrow \mathbf{f}, w \\ v \leftarrow \mathbf{f} \end{array} \right\}$$

▲

This operation reduces the depth of the nesting tree and thus will serve as a basis for defining the compression of a nested justification system.

Definition 6.2.17. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$ be a nested justification system. The *compression* $\mathrm{Compress}(\mathcal{JS})$ is defined inductively to be the justification system

$$\mathrm{Unfold}\left(\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathrm{Flat}(\mathrm{Compress}(\mathcal{JS}^1)), \ldots, \mathrm{Flat}(\mathrm{Compress}(\mathcal{JS}^k))\}\rangle\right).$$

Note that this definition is well-defined since the subsystems $\mathcal{JS}^i$ will have a smaller depth than $\mathcal{JS}$. The compression starts by flattening the leaf systems and unfolding them onto their parents. This is repeated until there is no nesting. Denecker et al. (2015) demand that all branch evaluations involved should be parametric, but in principle, the top level justification system does not need to be parametric. Denecker et al. (2015) use the compression to give meaning to nested justification systems: to get notions such as justification, supported value, and models into nested systems realm. Let us take a look at an example.

Example 6.2.18. Let us look at Example 6.2.1, where we used an auxiliary symbol for $\sim p$. We can use the following nested system:

$$\left\{ \begin{array}{l} p \leftarrow \sim T \\ \left\{ \begin{array}{l} T \leftarrow \sim p \end{array} \right\} \end{array} \right\}.$$

The compression of this system is

$$\left\{ \begin{array}{l} p \leftarrow p \\ T \leftarrow \sim p \end{array} \right\}.$$

By evaluating the compression under $\mathcal{B}_{\text{wf}}$, we get the intended model in which p is $\mathbf{f}$. The justification for p will not contain any trace of T. In this case, this is fine, but in more complicated predicate introductions or reifications, understanding the justifications becomes difficult. ▲

Example 6.2.19. Let $\mathcal{JS}$ be the nested system from Example 6.2.5 (the graphical representation is shown in Example 6.2.7). The compression starts by flattening the leaf systems of the nesting tree. So we obtain the system

$$\mathcal{B}_{\text{KK}}: \left\{ \begin{array}{l} x \leftarrow a, b \\ \mathcal{B}_{\text{cwf}}: \left\{ \begin{array}{l} a \leftarrow \mathbf{t} \end{array} \right\} \\ \mathcal{B}_{\text{KK}}: \left\{ \begin{array}{l} b \leftarrow \sim c, d \\ \mathcal{B}_{\text{wf}}: \left\{ \begin{array}{l} c \leftarrow \mathbf{f} \end{array} \right\} \\ \mathcal{B}_{\text{cwf}}: \left\{ \begin{array}{l} d \leftarrow \mathbf{t} \end{array} \right\} \end{array} \right\} \end{array} \right\}.$$

Then we unfold the inner system

$$\mathcal{B}_{\text{KK}}: \left\{ \begin{array}{l} b \leftarrow \sim c, d \\ \mathcal{B}_{\text{wf}}: \left\{ \begin{array}{l} c \leftarrow \mathbf{f} \end{array} \right\} \\ \mathcal{B}_{\text{cwf}}: \left\{ \begin{array}{l} d \leftarrow \mathbf{t} \end{array} \right\} \end{array} \right\}$$

to get the system

$$\mathcal{B}_{\text{KK}}: \left\{ \begin{array}{l} b \leftarrow \mathbf{t} \\ c \leftarrow \mathbf{f} \\ d \leftarrow \mathbf{t} \end{array} \right\}$$

So far, we have

$$\mathcal{B}_{\text{KK}}\colon \left\{ \begin{array}{l} x \leftarrow a, b \\ \mathcal{B}_{\text{cwf}}\colon \{\ a \leftarrow \mathbf{t}\ \} \\ \mathcal{B}_{\text{KK}}\colon \left\{ \begin{array}{l} b \leftarrow \mathbf{t} \\ c \leftarrow \mathbf{f} \\ d \leftarrow \mathbf{t} \end{array} \right\} \end{array} \right\}.$$

Since both inner systems are already flattened, we unfold this system to get the compression of $\mathcal{JS}$:

$$\mathcal{B}_{\text{KK}}\colon \left\{ \begin{array}{l} x \leftarrow \mathbf{t} \\ a \leftarrow \mathbf{t} \\ b \leftarrow \mathbf{t} \\ c \leftarrow \mathbf{f} \\ d \leftarrow \mathbf{t} \end{array} \right\}.$$

The single justification with x as root is equal to

$$\begin{array}{c} x \\ \downarrow \\ \mathbf{t} \end{array}$$

The only thing this justification explains is that x has to be true in a model, but it gives no reason why this is the case. This information is lost because we have compressed the nested system. One can expect to lose more information if there is a large nesting depth. ▲

6.3 Alternative View on Nested Systems

Example 6.2.19 shows that the justifications in Compress($\mathcal{JS}$) completely lost their use to serve as explanations. However, we identified this to be one of the motivating properties for why to use justification theory. Therefore, it raises the question if we can evaluate nested justification systems differently, without compressing and losing details in the process. If we want all the possible details available in the justifications, we should put all the rules of each system together in a single unnested justification frame. This new frame, then, should be evaluated with a new branch evaluation. This brings us to our final research question.

Nested Justification Semantics without Quality Loss

By combining all the rules in a nested system, we get a single unnested justification frame. The question remains if there is a branch evaluation

such that this new system is strongly equivalent with the compression. If such a branch evaluation exists, then nested systems can be looked at from an alternative standpoint without the loss of quality in justifications which the compression suffers from.

Let us go back to Example 6.2.18. If all the rules are thrown together, we get the complementation of

$$\left\{ \begin{array}{l} p \leftarrow \sim T \\ T \leftarrow \sim p \end{array} \right\}.$$

In the compression, p has supported value $\mathbf{f}$. But now we get branches $p \to \sim T \to p \to \sim T \to \cdots$, which is evaluated to $\mathbf{u}$ under $\mathcal{B}_{\mathrm{wf}}$. The well-founded semantics tries to minimise truth. It becomes unclear if we want to minimise the truth of p or T. Well-founded semantics cannot make such a distinction. The nesting, however, makes it clear which symbol is more important to minimise truth for: the fact that is closest to the root system is minimised first. In order to simulate the compression in the new branch evaluation, we can leave out all the occurrences of $\sim T$ to get the branch $p \to p \to \cdots$.

A similar idea can be achieved in arbitrary nestings.

Definition 6.3.1. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\} \rangle$ be a nested justification system. Take an $\mathcal{F}$-branch $\mathbf{b}$. We define the *merge branch evaluation* $\mathcal{B}^*$ so that

- if $\mathbf{b}$ is finite, then $\mathcal{B}^*(\mathbf{b})$ is equal to the last element of $\mathbf{b}$.
- if $\mathbf{b}$ is infinite, then let $\mathcal{JS}'$ be the highest level justification system in the nesting tree of $\mathcal{JS}$ such that $\mathbf{b}$ contains infinitely many elements of locally defined facts of $\mathcal{JS}'$ (Such a system $\mathcal{JS}'$ exists, see discussion below definition). Let $\mathbf{b}'$ be the branch obtained from $\mathbf{b}$ by removing all facts outside $\mathcal{JS}'$. Then $\mathcal{B}^*(\mathbf{b}) = \mathcal{B}'(\mathbf{b}')$, where $\mathcal{B}'$ is the branch evaluation of $\mathcal{JS}'$.

The *merge* $\mathrm{Merge}(\mathcal{JS})$ of $\mathcal{JS}$ is the justification system $\langle \mathcal{F}, \mathcal{F}_d, R^*, \mathcal{B}^* \rangle$, where R^* is the union of all the rules in $\mathcal{JS}$.

Technically, this does not constitute a branch evaluation as in Definition 2.2.11. However, by demanding that the fact space has a nesting tree structure of defined facts attached to it, it is just as general. We can say that branches in a nested justification system are sequences of facts $x_0 \to x_1 \to \cdots$ such that the level of x_i is equal to, a descendant, or ancestor of the level of x_{i+1}. With this restriction, the highest level in the definition of the merge branch evaluation

always exist. If two such levels exist that are not descendant or ancestor of each other, then the branch will have a higher level that occurs infinitely often. This level will be a common ancestor of the two.

Note that the definition of merge does not have the limitation that the underlying branch evaluations are parametric as in the compression.

Example 6.3.2. Take a nested justification system where the inner system is not parametric.

$$\mathcal{B}_{\text{wf}}\colon \left\{ \begin{array}{l} r \leftarrow x \\ s \leftarrow y \\ \mathcal{B}_{\text{st}}\colon \left\{ \begin{array}{l} x \leftarrow {\sim}y \\ y \leftarrow {\sim}x \end{array} \right\} \end{array} \right\}$$

The inner system has three models $\mathcal{I}$ with $\mathcal{I}(x) = {\sim}\mathcal{I}(y)$, where $\mathcal{I}(x) \in \{\mathbf{t}, \mathbf{u}, \mathbf{f}\}$. Intuitively, you would expect that each model of the inner system gives rise to a model of the nesting by substituting the values of x and y into the outer system. When taking the merge, this is exactly what happens, even though the inner system is not parametric: The only justification for r in the merge is

$$r \to x \to {\sim}y \to x \to {\sim}y \to \cdots$$

The merge branch evaluates this to $\mathcal{B}_{\text{st}}(x \to {\sim}y \to x \to {\sim}y \to \ldots) = {\sim}y$; hence $\text{SV}_{\text{Merge}(\mathcal{JS})}(r, \mathcal{I}) = {\sim}\mathcal{I}(y)$. Similarly, $\text{SV}_{\text{Merge}(\mathcal{JS})}(s, \mathcal{I}) = {\sim}\mathcal{I}(x)$. This shows that each model of the merge is produced by substituting a model of the inner system into the outer system. ▲

As of now, we do not have any applications for nestings of non-parametric branch evaluations. So for the remainder of this text, we will assume that all branch evaluations involved are parametric.

Note that the merge can also be seen as a recursive definition like the compression.

Lemma 6.3.3. *Let* $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$ *be a nested system. Then* $\text{Merge}(\mathcal{JS})$ *and*

$$\text{Merge}(\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\text{Merge}(\mathcal{JS}^1), \ldots, \text{Merge}(\mathcal{JS}^k)\}\rangle)$$

are equal (not just strongly equivalent).

Our research question then becomes, when are $\text{Merge}(\mathcal{JS})$ and $\text{Compress}(\mathcal{JS})$ strongly equivalent.

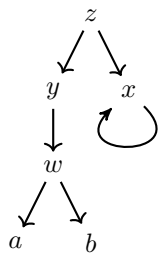
6.3.1 Shrinking Justifications

To prove the strong equivalence between Compress($\mathcal{JS}$) and Merge($\mathcal{JS}$) we need a way to convert justifications from one system to the other. Going from Merge($\mathcal{JS}$) to Compress($\mathcal{JS}$) should be relatively easy since justifications in Merge($\mathcal{JS}$) contain more information. By removing this extra information, we can reduce the justification. This is illustrated in the next example.

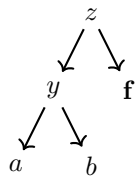
Example 6.3.4. Take the nested system

$$\left\{ \begin{array}{l} z \leftarrow x, y \\ y \leftarrow w \\ \left\{ \begin{array}{l} x \leftarrow x \\ w \leftarrow a, b \end{array} \right\} \end{array} \right\},$$

where both levels are evaluated with $\mathcal{B}_{\text{wf}}$. Let us look at the following justification for z in the merge system.



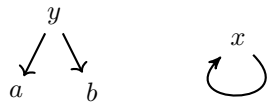
The loop on the right side is evaluated to $\mathbf{f}$. So to eliminate x from the justification, we can replace this loop with the fact $\mathbf{f}$. On the left side, w can be replaced by a and b. This will produce the justification



This is a justification of the compression

$$\left\{ \begin{array}{l} z \leftarrow \mathbf{f}, y \\ y \leftarrow a, b \end{array} \right\}.$$

A justification in the merge also contains justifications in the unnested systems of the nesting tree. In the example above, these are



These justifications can be reduced to a single set, by evaluating their branches, in the same manner as in the flattening. Then replacing facts of lower systems by such a set, we get a justification in compression as above. ▲

The idea in this example is formalised in the *shrinking* of a justification. In the example above, we evaluated certain subjustifications.

Definition 6.3.5. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\} \rangle$ and J a locally complete justification in Merge($\mathcal{JS}$). Let $\mathcal{JS}^*$ be a node of the nesting tree of $\mathcal{JS}$ that has children, but no grandchildren. Let $\mathcal{JS}^1, \ldots, \mathcal{JS}^m$ be the children of $\mathcal{JS}^*$ and let X be the set of internal nodes of J that are defined in some $\mathcal{JS}^i$. For each $x \in X$ defined in $\mathcal{JS}^i$, let $P(x)$ be the set of maximal J-paths starting with x that consist of

- an infinite number of defined facts of $\mathcal{JS}^i$, or
- a finite number of defined facts of $\mathcal{JS}^i$ and an open fact in $\mathcal{JS}^i$.

This means that $P(x)$ forms a justification J_x inside $\mathcal{JS}^i$. The justification J_x is called the *subjustification* for x in J.

In correspondence with the flattening in the compression, each of the subjustifications can be reduced to a single rule.

Definition 6.3.6. Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\} \rangle$ and J a locally complete justification in Merge($\mathcal{JS}$). Let $\mathcal{JS}^*$ be a node of the nesting tree of $\mathcal{JS}$ that has children, but no grandchildren. Let $\mathcal{JS}^1, \ldots, \mathcal{JS}^m$ be the children of $\mathcal{JS}^*$ and let X be the set of internal nodes of J that are defined in some $\mathcal{JS}^i$. Each subjustification J_x for x in J corresponds to a rule $x \leftarrow B_x$ in Flat($\mathcal{JS}^i$). Define Shrink$_{\mathcal{JS}^*}(J)$ to be the justification obtained from J where

- every rule $y \leftarrow A$ in J is retained if y is defined outside $\mathcal{JS}^*$ or its children.
- every rule $y \leftarrow A$ in J with y defined in $\mathcal{JS}^*$ is replaced by $y \leftarrow A \setminus X \cup \cup_{x \in A \cap X} B_x$;
- together with the rules $x \leftarrow B_x$.

The construction of Shrink follows the construction of compression: the evaluation of the subjustifications is in line with the flattening, while the shrinking itself is like the unfolding. The addition of the rules $x \leftarrow B_x$ corresponds to adding the rules of the flattening in the compression.

When shrinking we start with a locally complete justification, and so we would like to end up with a locally complete justification. If the shrinking is not locally complete, then there is a subjustification J_x that has a branch $\mathbf{b}$ starting with x that is mapped to a defined fact that is not internal in $\mathrm{Shrink}_{\mathcal{JS}^*}(J)$. If $\mathbf{b}$ is finite, then by locally completeness of J, the last element of $\mathbf{b}$ is also internal in J. Therefore, it is only possible that $\mathbf{b}$ is infinite. All branch evaluations considered are parametric, and thus $\mathbf{b}$ is mapped to an open fact. Branch evaluations tend to map branches to some element in the branch (or the negation) or some logical fact. For a branch evaluation to map to another fact requires for that fact to have a special status in the given justification frame, by adding extra structure on the underlying facts space. This is because branch evaluations are defined independent of justification frames. Since we are dealing with parametric branch evaluations, we cannot map to elements of an infinite branch since they are all defined. Therefore, the only feasible image of an infinite branch is a logical fact.

Lemma 6.3.7. *If the branch evaluation of the children of $\mathcal{JS}^*$ map infinite branches into $\mathcal{L}$, then* $\mathrm{Shrink}_{\mathcal{JS}^*}(J)$ *is locally complete.*

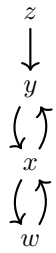
Proof. If $\mathrm{Shrink}_{\mathcal{JS}^*}(J)$ has a defined leaf, then there is a finite branch $\mathbf{b}^*$ in $\mathrm{Shrink}_{\mathcal{JS}^*}(J)$ ending in a defined leaf. This means that there is either a finite J-branch ending in the same leaf, or an infinite J-branch for which the highest level with an infinite number of elements is a child of $\mathcal{JS}^*$ and evaluating the restriction of this branch to these elements will produce this defined leaf. The former is not possible since J is locally complete and the latter is not possible by our assumptions. □

This property will also be needed when we do the reverse construction of shrinking. Let us provide another example.

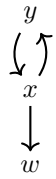
Example 6.3.8. Take the following nested system

$$\left\{ \begin{array}{l} z \leftarrow y \\ w \leftarrow x \\ \left\{ \begin{array}{l} y \leftarrow x \\ x \leftarrow w, y \end{array} \right\} \end{array} \right\},$$

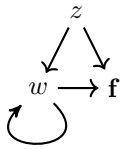
where every level is evaluated under $\mathcal{B}_{\mathrm{wf}}$. Take the following justification for z in the Merge:



The subjustifications for x and y are the same and equal to

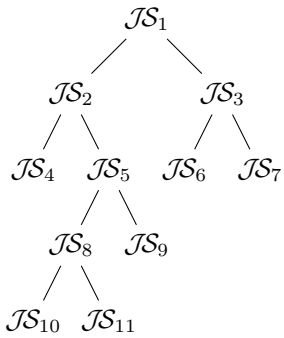


By evaluating the branches starting with y (respectively x), we get rules $y \leftarrow \mathbf{f}, w$ and $x \leftarrow \mathbf{f}, w$ in the flattening of the inner system. Shrinking the justification for z will produce (the rules for x and y are left out for clarity)

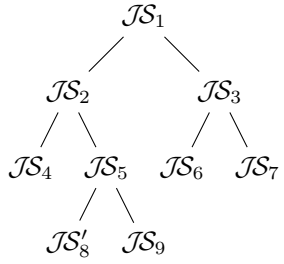


▲

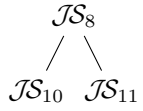
In both examples above, we had a single nesting. If multiple nesting occur, it is not directly clear in which system the shrunken justification is in. For example, if our nesting tree looks like



and we want to shrink with respect to $\mathcal{JS}_8$, then the shrinking brings us in the merge of the nested system with the nesting tree



where $\mathcal{JS}'_8$ is the compression of the system with the nesting tree



Shrinking a justification will not produce a justification with a worse value if we demand that every finite branch should be mapped to its last element by all branch evaluations involved.

Proposition 6.3.9. *Let $\mathcal{JS}$ be a nested justification system with all branch evaluations involved parametric, mapping infinite branches to logical facts, and finite branches to their last element. Let $\mathcal{JS}^*$ be a justification system in the nesting tree of $\mathcal{JS}$ for which we can shrink. Let* $\mathrm{Shrink}_{\mathcal{JS}^*}(\mathcal{JS})$ *be the nested justification system after shrinking. Take a justification J for x in* $\mathrm{Merge}(\mathcal{JS})$. *Then* $\mathrm{val}_{\mathrm{Merge}(\mathcal{JS})}(J, x, \mathcal{I}) = \mathrm{val}_{\mathrm{Merge}(\mathrm{Shrink}_{\mathcal{JS}^*}(\mathcal{JS}))}(\mathrm{Shrink}_{\mathcal{JS}^*}(J), x, \mathcal{I})$ *for all $\mathcal{F}$-interpretations $\mathcal{I}$.*

Proof. There is a correspondence between branches of J and $\text{Shrink}_{\mathcal{JS}^*}(J)$. Every J-branch $\mathbf{b}$ can be shrunk to a $\text{Shrink}_{\mathcal{JS}^*}(J)$-branch $\mathbf{b}^*$. Likewise, every $\text{Shrink}_{\mathcal{JS}^*}(J)$-branch $\mathbf{b}^*$ comes from a J-branch $\mathbf{b}$. Moreover, $\mathcal{B}(\mathbf{b}) = \mathcal{B}^*(\mathbf{b}^*)$. Indeed, take an arbitrary J-branch $\mathbf{b}$. For finite branches, this is obvious; so assume it is infinite. If it contains infinite facts from a system higher than the children of $\mathcal{JS}^*$, then equality is also obvious. Therefore, we can assume that $\mathbf{b}$ contains infinite facts from some child $\mathcal{JS}^i$ of $\mathcal{JS}^*$. This means $\mathbf{b}^*$ is finite and its evaluation under $\mathcal{B}^*$ is equal to $\mathcal{B}(\mathbf{b})$. □

So far, we only shrink a single level, but by applying the shrink operation iteratively, a justification in $\text{Merge}(\mathcal{JS})$ is transformed to a justification in $\text{Compress}(\mathcal{JS})$.

Definition 6.3.10. Let J be a justification in $\text{Merge}(\mathcal{JS})$. Choose a justification system $\mathcal{JS}^*$ with children, but not grandchildren in the nesting tree of $\mathcal{JS}$. Then $\text{Shrink}_{\mathcal{JS}^*}(\mathcal{JS})$ will have a smaller nesting tree than $\mathcal{JS}$. By iterating this procedure until the nesting tree is a single node, we get a justification in $\text{Compress}(\mathcal{JS})$, which we will denote by $\text{Shrink}(J)$.

The above definition is well-defined since the order of the local shrinking operations does not matter. This is because the local shrinking operation only affects the rules for facts defined in $\mathcal{JS}^*$ or its children. By Proposition 6.3.9, the value of J and $\text{Shrink}(J)$ are equal. This solves one side of the strong equivalence.

Corollary 6.3.11. $\text{SV}_{\text{Merge}(\mathcal{JS})}(x, \mathcal{I}) \leq_t \text{SV}_{\text{Compress}(\mathcal{JS})}(x, \mathcal{I})$ *for all defined facts* x *and interpretations* $\mathcal{I}$.

Proof. For every justification J in $\text{Merge}(\mathcal{JS})$, we can construct an equally good justification in $\text{Compress}(\mathcal{JS})$: $\text{Shrink}(J)$. □

6.3.2 Expanding Justifications

The other direction is not as straightforward since we have to start from a justification in Compress and we have to 'inflate' it to a justification of Merge. One way to achieve this, is by 'pasting' in subjustifications. However, this can lead to problems when working with graph-like justifications as illustrated by the next example.

Example 6.3.12. Take the nested system

$$\left\{ \begin{array}{l} x \leftarrow a \\ y \leftarrow b \\ \left\{ \begin{array}{l} a \leftarrow x \\ a \leftarrow \mathbf{t} \\ b \leftarrow a \end{array} \right\} \end{array} \right\}.$$

The compression is equal to

$$\left\{ \begin{array}{l} x \leftarrow x \\ x \leftarrow \mathbf{t} \\ y \leftarrow x \\ y \leftarrow \mathbf{t} \end{array} \right\}.$$

Take the following justification for y in the compression:

$$\begin{array}{c} y \\ \downarrow \\ x \\ \downarrow \\ \mathbf{t} \end{array}$$

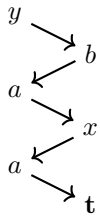
The rule for y in J comes from the rule $z \leftarrow b$ in the top level and the subjustification

$$\begin{array}{c} b \\ \downarrow \\ a \\ \downarrow \\ x \end{array}$$

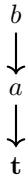
On the other hand, the rule for x in J comes from the rule $x \leftarrow a$ in the top level and the subjustification

$$\begin{array}{c} a \\ \downarrow \\ \mathbf{t} \end{array}$$

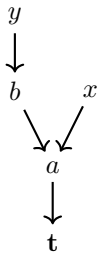
These subjustifications are not compatible: the rules for a are different. So putting the subjustifications into the justification for y in the compression produces a tree-like justification in the merge.



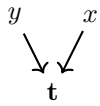
In this example, we can still produce an equally good justification by pasting the subjustifications together in the correct order to get the subjustification



Pasting this subjustification for the rule $y \leftarrow x$ we get the justification in the merge



Shrinking this justification produces an equally good justification as J

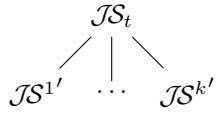


▲

As shown in the previous example, for a justification J in Merge($\mathcal{JS}$), it suffices to construct a justification J^* in Merge($\mathcal{JS}$) such that Shrink(J^*) = J. This

would imply that $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J, x, \mathcal{I}) = \mathrm{val}_{\mathrm{Merge}(\mathcal{JS})}(J^*, x, \mathcal{I})$ and solve the missing part of the strong equivalence between Compress($\mathcal{JS}$) and Merge($\mathcal{JS}$). We have illustrated how it can work in Example 6.3.12, but found that in general the construction only works for tree-like justifications. However, it is sufficient to construct a justification J^* such that $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J, x, \mathcal{I}) \leq_t \mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(\mathrm{Shrink}(J^*), x, \mathcal{I})$.

But let us first work with tree-like justifications. As with Shrink, we will first define the operation locally. So let $\mathcal{JS}$ be a nested justification system so that the unnested leaves of its nesting tree are compressions of nested justification systems.[3] Let $\mathcal{JS}^*$ be such a leaf which is a compression of a nested system with depth > 1. Take a justification J in Merge($\mathcal{JS}$). Suppose $\mathcal{JS}^*$ is the compression of $\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$. We define Expand($J$) to be a justification in Merge of the system obtained from $\mathcal{JS}$ with $\mathcal{JS}^*$ replaced by the nested system with nesting tree



where $\mathcal{JS}_t$ is the top level of $\mathcal{JS}^*$ and $\mathcal{JS}^{i'}$ is the compression of $\mathcal{JS}^i$.

Any $z \leftarrow A$ in J with z defined in $\mathcal{JS}^*$ corresponds to a rule $z \leftarrow B$ in $\mathcal{JS}_t$ such that for a subset X of B and all $x \in X$, there is a justification J_x in some $\mathcal{JS}^i$ for x such that $A = B \setminus X \cup \{\{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_{J_x}(x)\} \mid x \in X\}$. By replacing $z \leftarrow A$ with $z \leftarrow B$ and putting J_x under $x \in X$, we get a justification $\mathrm{Expand}_{\mathcal{JS}^*}(J)$.

Remark that the property that infinite branches are mapped to logical facts is necessary here. If not, then the construction above can put an infinite branch in between an infinite branch. This is will not create a branch, but a sequence with a higher ordinal number than ω.

Shrinking the obtained justification returns the original justification.

Proposition 6.3.13. *Let J be a tree-like justification and let $\mathcal{JS}_t$ and $\mathcal{JS}^*$ be defined as above. Then* $\mathrm{Shrink}_{\mathcal{JS}_t}(\mathrm{Expand}_{\mathcal{JS}^*}(J)) = J$.

Proof. Shrinking $\mathrm{Expand}_{\mathcal{JS}^*}(J)$ corresponds to removing the justifications J_x and replacing $z \leftarrow B$ with $z \leftarrow A$ as in the construction above. □

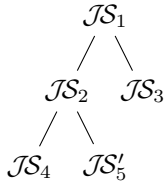
Corollary 6.3.14. *Let J be a tree-like justification. Then* $\mathrm{val}(J, x, \mathcal{I}) = \mathrm{val}(\mathrm{Expand}_{\mathcal{JS}^*}(J), x, \mathcal{I})$.

[3] These are exactly the systems one can get from shrinking a nested justification system.

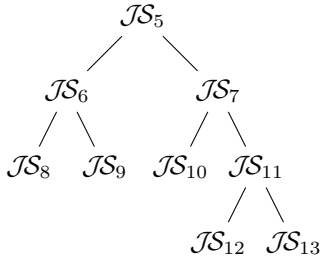
Proof. This follows directly from Propositions 6.3.9 and 6.3.13. □

If multiple nestings are present, it can become confusing in which system Expand(J) is a justification. The following example provides some clarification.

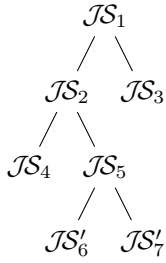
Example 6.3.15. Let $\mathcal{JS}$ be the nested justification system with nesting tree



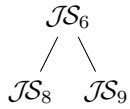
where $\mathcal{JS}'_5$ is the compression of the system with nesting tree



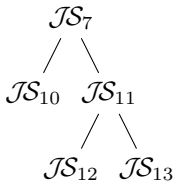
Expanding a justification in Merge($\mathcal{JS}$) gets us in the Merge of the system with nesting tree



where $\mathcal{JS}'_6$ is the compression of the system with nesting tree



and $\mathcal{JS}_7'$ is the compression of the system with nesting tree



▲

Similarly to Shrink, applying Expand iteratively, we can transform a justification in Compress($\mathcal{JS}$) to a justification in Merge($\mathcal{JS}$).

Definition 6.3.16. Let J be a justification in Compress($\mathcal{JS}$). By iteratively applying expand we get a justification in Merge($\mathcal{JS}$), which we will denote by Expand(J).

By Corollary 6.3.14, the value of J and Expand(J) are equal. This solves the other side of the strong equivalence between Merge($\mathcal{JS}$) and Compress($\mathcal{JS}$), but only for tree-like justifications.

Corollary 6.3.17. $\mathrm{SV}^t_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) \leq_t \mathrm{SV}^t_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I})$ *for all defined facts* x *and interpretations* $\mathcal{I}$.

This implies that we have solved our last research question for tree-like justifications.

Theorem 6.3.18. *For tree-like justifications,* Compress($\mathcal{JS}$) *and* Merge($\mathcal{JS}$) *are strongly equivalent.*

Proof. This is proven by combining Corollaries 6.3.11 and 6.3.17. □

In order for the expansion construction to work for graph-like justifications, we need a way to paste the subjustifications so that their values do not deteriorate. In previous chapters, we have seen splittable branch evaluations have good properties regarding this aspect. In combination of our other properties, splittability simplifies to transitivity.

Proposition 6.3.19. *If $\mathcal{B}$ is splittable and it maps finite branches to their last element, then $\mathcal{B}$ is transitive.*

Proof. No branch can be decided (except at the last element of a finite branch) since you can always add different finite branches after it because finite branches are mapped to their last element. By splittability, every branch is everywhere transitive; hence $\mathcal{B}$ is transitive. □

Together with the other demands for branch evaluations in nested system, we get a new branch evaluation type.

Definition 6.3.20. A branch evaluation $\mathcal{B}$ is *nestable* if it is parametric, transitive (splittable), maps finite branches to last element, and infinite branches into $\mathcal{L}$.

In Example 6.3.12, we have seen that we can paste together the subjustifications and then put this in the justification of the compression. This will produce a justification that if shrunken is as good as the original justification, but not equal. However, as we will see later, it is sufficient that $\mathrm{Shrink}(J^*)$ only adds branches that are better than the value of J.

To prove $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x,\mathcal{I}) \leq_t \mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x,\mathcal{I})$, we can split it up into three cases, depending on the value of $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x,\mathcal{I}) = \ell$. If $\ell = \mathbf{f}$, nothing has to be proved. If $\ell = \mathbf{u}$, then we need to prove that $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x,\mathcal{I}) \geq_t \mathbf{u}$. If $\ell = \mathbf{t}$, then we need to prove that $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x,\mathcal{I}) = \mathbf{t}$ as well. Before, we look more into the more general case, let us try to prove it for a few specific cases in system having a single nesting.

6.3.3 Kripke-Kleene Case

Let us start with $\mathcal{B}_{\mathrm{KK}}$, the simplest nestable branch evaluation. In the case that $\ell = \mathbf{t}$, it means that there exist a justification without infinite branches and all leaves interpreted $\mathbf{t}$ in $\mathcal{I}$. By pasting the subjustifications from the leaves up, we can get a good justification in Merge.

Proposition 6.3.21. *Take a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{KK}}$ at the top level, and nestable branch evaluations in the lower levels. If $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x,\mathcal{I}) = \mathbf{t}$, then $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x,\mathcal{I}) = \mathbf{t}$.*

Proof. By Lemma 2.6.16 and transitivity of $\mathcal{B}_{\mathrm{KK}}$, there is a locally complete justification J such that $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J,y,\mathcal{I}) = \mathbf{t}$ for all the internal nodes y of J and x is an internal node of J. This means that J has no infinite branches.

We define the relation $y \prec z$ if y is a descendant of z in J. This relation is well-founded since it has no infinite descending chains because J does not have infinite branches. By the axiom of choice, this relation can be extended to a well-order, i.e. the set of internal nodes of J is equal to $\{x_i \mid i \leq \beta\}$ for some ordinal β. For any x_i with $i \leq \beta$, the rule $x_i \leftarrow A_i$ in J corresponds to a rule $x_i \leftarrow B_i$ in the top level and a set $C_i \subseteq B_i$ such that for all $y \in C_i$, there is a justification of a lower level $K_{i,y}$ such that $A_i = B_i \setminus C_i \cup \bigcup_{y \in C_i} \left\{\mathcal{B}(\mathbf{b}) \;\middle|\; \mathbf{b} \in B_{K_{i,y}}(y)\right\}$. Let K_i be a pasting of the justifications $K_{i,y}$ for all $y \in C_i$. Any order of pasting is valid. This is possible by fixing a well-order on C_i. Define J_i inductively as follows.

- $J_0 = K_0$;
- $J_{i+1} = J_i \uparrow K_{i+1}$ for $i < \beta$;
- $J_\alpha = (\cup_{i<\alpha} J_i) \uparrow K_\alpha$

The justification J_β is a 'justification' of all the lower systems combined. By replacing $x_i \leftarrow A_i$ with $x_i \leftarrow B_i$ in J and pasting J_β on to it, we get a justification J^* in $\mathrm{Merge}(\mathcal{JS})$. Compared to J, $\mathrm{Shrink}(J^*)$ will have extra edges and some edges removed. The edges that are removed will have no detrimental effect on the value of the justification. For each ordinal i and $y \in C_i$, we have that $\left\{\mathcal{B}(\mathbf{b}) \;\middle|\; \mathbf{b} \in B_{J_\beta}(y)\right\} \subseteq \{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_{K_i}(y)\} \cup \bigcup_{j<i} \left\{\mathcal{B}(\mathbf{b}) \;\middle|\; \mathbf{b} \in B_{J_j}(y)\right\}$. Therefore, the new edges will be from x_i to x_j with $i > j$. Hence, $\mathrm{Shrink}(J^*)$ will also have no infinite branches, because that would imply the existence of a infinite descending chain. The justification $\mathrm{Shrink}(J^*)$ might have new finite branches compared to J, but they will have a final element equal to a final element of some finite branch in J. This means that $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(\mathrm{Shrink}(J^*), x, \mathcal{I}) = \mathbf{t}$. By Proposition 6.3.9, $\mathrm{val}_{\mathrm{Merge}(\mathcal{JS})}(J^*, x, \mathcal{I}) = \mathbf{t}$, which complete this proof. □

If the value of the justification is **u**, then it does not matter if infinite branches are added or not.

Proposition 6.3.22. *Take a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{KK}}$ at the top level, and nestable branch evaluations in the lower levels. If $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) = \mathbf{u}$, then $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I}) \geq_t \mathbf{u}$.*

Proof. The proof is almost the same as the proof of Proposition 6.3.21, except, here we can take any well-order on the internal nodes of J. The branches that are added are infinite branches or finite branches to a final element that was already in At most, some infinite branches are added after shrinking J^* compared to J; hence $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x, J^*, \mathcal{I}) \geq_t \mathbf{u}$. □

Corollary 6.3.23. *For a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{KK}}$ at the top level, and nestable branch evaluations in the lower levels, we have that* Compress($\mathcal{JS}$) *and* Merge($\mathcal{JS}$) *are strongly equivalent for graph-like justifications.*

Proof. This follows from Propositions 6.3.21 and 6.3.22. □

The general idea in the above two propositions, is to find an order for pasting the subjustifications.

6.3.4 Well-Founded Case

Let us try this idea for $\mathcal{B}_{\mathrm{wf}}$ now. Again, we do it in two steps, depending on if $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I})$ is $\mathbf{t}$ or $\mathbf{u}$.

Proposition 6.3.24. *Take a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{wf}}$ at the top level, and nestable branch evaluations in the lower levels. If $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) = \mathbf{u}$, then $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I}) \geq_t \mathbf{u}$.*

Proof. There is a locally complete $\mathbf{u}$-domain supporting justification J in Compress($\mathcal{JS}$) with x as internal node. Without loss of generality, we can assume that x is a root of J. For each node y in J we say that $val(y) = \mathrm{lub}_{z \in J(y) \cap \mathcal{F}_+}(val(z) + 1)$; so every node is assigned an ordinal number. Note that leaves y of J have $val(y) = 0$, but also internal nodes can have a value 0. This is well-defined since J does not contain an infinite positive branch since $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J, x, \mathcal{I}) = \mathbf{u}$. By taking the least upper bound of all the ordinals, we get an ordinal β. Let $X_\alpha = \{y \in J \mid val(y) = \alpha\}$ for $\alpha \leq \beta$.

For each ordinal $\alpha \leq \beta$ we define a graph J_α in the lowest level such that $J_i \subseteq J_j$ for $i \leq j \leq \beta$. We prove that the positive leaves of J_α are contained in $\cup_{i<\alpha} X_i$.

- Take $y \in X_0$. Either $y \in \mathcal{F}_o$ or not. If not, then the rule $y \leftarrow J(y)$ is a compressed rule, so it corresponds to a rule $y \leftarrow A$ in the highest level and a number of justifications in the lower level. Pasting all these justifications together gives the justification J_y (in any order). Then define J_0 to be the pasting-together of J_y for $y \in X_0 \setminus \mathcal{F}_o$ (in any order). The justification J_0 has no positive leaves.
- Take any non-zero ordinal α. If $X_\alpha = \emptyset$, then define $J_\alpha = \cup_{i<\alpha} J_i$. Of course, the positive leaves of J_α are contained in $\cup_{i<\alpha} X_i$ by induction. So assume X_α is not empty. Again, each $y \in X_\alpha$ has a compressed rule

$y \leftarrow J(y)$, which corresponds to a rule $y \leftarrow A$ in the highest level and a number of justifications in the lower level. Pasting all these justifications together gives the justification J_y. Pasting these J_y together produces J_α. Since the positive leaves of J_y have a lower *val* than $val(y)$ by definition, we get that the set of positive leaves of J_α is contained in $\cup_{i<\alpha} X_i$.

Now pasting into J and replacing the upper level edges with J_β produces the justification J^*. By Shrinking J^* and comparing with J we notice

- Some edges are removed (we don't care about them, removing them only increases the justification value)
- Some edges are added:
  - Edges to negative elements
  - Edges to positive elements of a lower *val*

Adding edges towards negative elements will not produce positive loops. Adding edges towards positive elements can produce positive loops, but it will give rise to an infinite descending chain of ordinal numbers, which is not possible due to the well-foundedness of ordinals.

Note that J_β does not have infinite branches evaluated to $\mathbf{f}$ (otherwise J would have a rule containing $\mathbf{f}$). Therefore, $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(\mathrm{Shrink}(J^*), x, \mathcal{I}) \geq_t \mathbf{u}$, which proves that $\mathrm{val}_{\mathrm{Merge}(\mathcal{JS})}(J^*, x, \mathcal{I}) \geq_t \mathbf{u}$. □

Similarly, we have the following result.

Proposition 6.3.25. *Take a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{wf}}$ at the top level, and nestable branch evaluations in the lower levels. If $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) = \mathbf{t}$, then $\mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I}) = \mathbf{t}$.*

Proof. The proof is very similar to the proof of the previous proposition, but with a different *val*. Let J be a locally complete $\mathbf{t}$-domain supporting justification with x as internal node. Without loss of generality, we can assume that x is a root of J. For each node y in J we say that $val(y) = \mathrm{lub}_{z \text{ is a descendant of } y, z \in \mathcal{F}_+}(val(z) + 1)$. The construction of J^* happens in the same way as before. Again, comparing J with $\mathrm{Shrink}(J^*)$ we remove edges and add edges. The added edges are towards negative elements or towards positive elements of a lower val. An edge $y \to z$ towards a positive element always decreases *val*. This is easy to see, because by definition $val(y) \geq_t val(z) + 1$. An edge $y \to \sim z$ towards a negative element will not increase *val*. This is a bit

more difficult. If this edge was already present in J, then $val(y) \geq_t val(\sim z)$ because every positive descendant of $\sim z$ is also a positive descendant of y. If this edge was added to the subjustification pasting, then there is a node v such that $v \to \sim z$ is in J and $val(v) \leq_t val(y)$ because it was pasted before. Therefore, we have that $val(\sim z) \leq_t val(v) \leq_t val(y)$. Hence, mixed loops and positive loops will give rise to an infinite decreasing sequence in val, which contradicts the well-foundedness of the ordinal numbers. □

Corollary 6.3.26. *For a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{wf}}$ at the top level, and nestable branch evaluations in the lower levels, we have that* Compress($\mathcal{JS}$) *and* Merge($\mathcal{JS}$) *are strongly equivalent for graph-like justifications.*

Proof. This follows from Propositions 6.3.24 and 6.3.25. □

The proofs of the above two propositions can also be transferred to $\mathcal{B}_{\mathrm{cwf}}$ by swapping the roles of $\mathcal{F}_+$ and $\mathcal{F}_-$.

Corollary 6.3.27. *For a nested system $\mathcal{JS}$ of depth 2 with $\mathcal{B}_{\mathrm{cwf}}$ at the top level, and nestable branch evaluations in the lower levels, we have that* Compress($\mathcal{JS}$) *and* Merge($\mathcal{JS}$) *are strongly equivalent for graph-like justifications.*

All the proofs above constructed a certain order on the nodes of a justification so that if the subjustifications are pasted in that order, we get a good justification: by shrinking again, we only add edges from nodes with a higher order to a lower order and that is not detrimental to the value of the justification. This can be put it in a new property.

Definition 6.3.28. Let $\mathcal{B}$ a branch evaluation. We call $\mathcal{B}$ *extendable* if for any (complementary) justification system $\mathcal{JS}$, interpretation $\mathcal{I}$, graph-like justification J and internal node x of J there is a well-order $\preceq$ (well-founded and total) on the nodes of J such that all branches $\mathbf{b}$ starting with x formed by

- $y \to z$ in J
- $y \to z$ for $z \prec y$

are at least as good as $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I})$ with respect to $\leq_t$.

In the proofs of previous results, we have seen that $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{wf}}$, and $\mathcal{B}_{\mathrm{cwf}}$ are extendable. In case of these branch evaluations, Compress($\mathcal{JS}$) and Merge($\mathcal{JS}$) are strongly equivalent for graph-like justifications. This can be generalised to any extendable branch evaluation.

Theorem 6.3.29. *For a nested system $\mathcal{JS}$ of depth 2 with an extendable and nestable branch evaluation $\mathcal{B}$ at the top level, and nestable (but not necessarily extendable) branch evaluations in the lower levels, we have that* $\mathrm{Compress}(\mathcal{JS})$ *and* $\mathrm{Merge}(\mathcal{JS})$ *are strongly equivalent for graph-like justifications.*

Proof. Take a justification J with $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J, y, \mathcal{I}) = \mathrm{SV}^g_{\mathrm{Compress}(\mathcal{JS})}(y, \mathcal{I})$ for all $y \in \mathcal{F}_d$. Paste the subjustifications in order of the well-order of the nodes of J to get the justification J^*. The justification $\mathrm{Shrink}(J^*)$ will have two types of edges exactly corresponding to the one in the condition of extendable. Therefore, $\mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(\mathrm{Shrink}(J^*), x, \mathcal{I}) \geq_t \mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J, x, \mathcal{I})$. By Proposition 6.3.9, $\mathrm{val}_{\mathrm{Merge}(\mathcal{JS})}(J^*, x, \mathcal{I}) \geq_t \mathrm{val}_{\mathrm{Compress}(\mathcal{JS})}(J, x, \mathcal{I}) = \mathrm{SV}^g_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I})$; hence $\mathrm{SV}_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) \leq \mathrm{SV}_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I})$. Combined with Corollary 6.3.11, the proof is finished. □

We have seen that both $\mathcal{B}_{\mathrm{wf}}$ and $\mathcal{B}_{\mathrm{cwf}}$ are extendable. Similarly, we find that the dual of an extendable branch evaluation is also extendable.

Proposition 6.3.30. *If $\mathcal{B}$ is extendable, then $\overline{\mathcal{B}}$ is extendable as well.*

Proof. A justification in $\mathcal{JF}$ is also a justification in $\overline{\mathcal{JF}}$ and $\mathrm{val}^{\overline{\mathcal{B}}}_{\mathcal{JF}}(J, x, \mathcal{I}) = \mathrm{val}^{\mathcal{B}}_{\overline{\mathcal{JF}}}(J, x, \mathcal{I})$. Since $\mathcal{B}$ is extendable, we have that there is a well-order $\prec$ on the nodes of J such that for branches $\mathbf{b}$ formed as in the definition of extendable, we have that $\mathcal{I}(\mathcal{B}(\mathbf{b})) \geq_t \mathrm{val}^{\mathcal{B}}_{\overline{\mathcal{JF}}}(J, x, \mathcal{I})$. This means that $\mathcal{I}(\overline{\mathcal{B}}(\mathbf{b})) \geq_t \mathrm{val}^{\overline{\mathcal{B}}}_{\mathcal{JF}}(J, x, \mathcal{I})$, proving that $\overline{\mathcal{B}}$ is extendable. □

So far, extendable is defined for graph-like justifications. The well-founded order constructed in the proofs for $\mathcal{B}_{\mathrm{wf}}$, $\mathcal{B}_{\mathrm{cwf}}$, and $\mathcal{B}_{\mathrm{KK}}$ also work for tree-like justifications, which we call *tree-like extendable*. This allows us to prove graph-reducibility, but we need another property.

Definition 6.3.31. A branch evaluation $\mathcal{B}$ is *double-resistant* if $\mathcal{B}(\mathbf{p} \to x \to x \to \mathbf{b}) = \mathcal{B}(\mathbf{p} \to x \to \mathbf{b})$ for any path $\mathbf{p}$ and branch $\mathbf{b}$.

Proposition 6.3.32. *If $\mathcal{B}$ is transitive and maps finite branches to their last element, then $\mathcal{B}$ is double-resistant.*

Proof. Since $\mathcal{B}$ maps finite branches to their last element, it is straightforward that $\mathcal{B}$ is double-resistant for finite branches. The result for infinite branches follows from transitivity. □

Corollary 6.3.33. *The branch evaluations $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{wf}}$, and $\mathcal{B}_{\mathrm{cwf}}$ are double-resistant*

The idea is to add edges between nodes with the same label and then use double-resistance to prove graph-reducibility.

Theorem 6.3.34. *If $\mathcal{B}$ is tree-like extendable and double-resistant, then $\mathcal{B}$ is graph-reducible.*

Proof. Take a locally complete tree-like justification T with root n with label x. For each $y \in \mathcal{F}_d$ internal in T, let N_y be the nodes of T labelled y. By tree-like extendable, N_y has a minimal element n_y since N_y is not empty.

Define the graph G obtained from T by adding edges from n to n_y for each $n \in N_y \setminus \{n_y\}$. Since $\mathcal{B}$ is tree-like extendable, the branches in G starting with x are at least as good as $\mathrm{val}_{\mathcal{B}}(T, x, \mathcal{I})$. The graph G is technically not a justification, but we can still speak of branches in G.

Define J to be the graph consisting of the rules for n_y in T. This is a locally complete graph-like justification and we can assume that x is a root of J (by removing the nodes not reachable from x). Take a branch $\mathbf{b} \in B_J(x)$. We prove it corresponds to a branch $\mathbf{b}^*$ in G with the same value. The start of $\mathbf{b}^*$ is a node n with label x. If n is equal to n_x, then we follow to the next element in $\mathbf{b}$ to get a node y. If n is not equal to n_x, then we first go through n_x and then to y. This process is repeated to get a branch $\mathbf{b}^*$ in G. By applying double-resistance at most ω times we get that $\mathcal{B}(\mathbf{b}) = \mathcal{B}(\mathbf{b}^*)$. Since $\mathbf{b}$ was taken arbitrarily, we proved that $\mathrm{val}_{\mathcal{B}}(J, x, \mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}}(T, x, \mathcal{I})$, completing the proof. □

This theorem provides a different way to prove that $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{wf}}$, and $\mathcal{B}_{\mathrm{cwf}}$ are graph-reducible.

6.4 Consistency of Nested Systems

So far, we have seen two ways to look at the semantics of nested systems: compression and merge. However, we have not discussed an important factor: the consistency of these systems. Of course, it makes sense to assume that every branch evaluation in the nesting tree is consistent. An unnested system is a nested system with depth one and the compression and merge are equal to that system. So if that system is not consistent, then the compression and merge are not consistent.

The branch evaluation of Compress($\mathcal{JS}$) is the branch evaluation of the top level. So if Compress($\mathcal{JS}$) is complementary, then it is consistent. However, proving that the compression is complementary turns out to be difficult. The

compression contains the rules of flattened systems. And thus in order to have complementarity, we need that each time flattening is performed during the construction of the compression, that the flattening is complementary. However, complementarity is a syntactical property, while the flattening is semantical. Recall that Proposition 2.3.8 gives a characterisation of complementarity in two parts. The following proposition illustrates that complementarity of the flattening implies consistency.

Proposition 6.4.1. *Let $\mathcal{JS}$ be a complementary justification system with $\mathcal{B}$ respecting negation. If $\mathrm{Flat}_g(\mathcal{JS})$ satisfies (1) of Proposition 2.3.8, then $\mathcal{JS}$ is graph-like consistent. If $\mathrm{Flat}_t(\mathcal{JS})$ satisfies (1) of Proposition 2.3.8, then $\mathcal{JS}$ is tree-like consistent.*

Proof. The proof for graph-like and tree-like is the same. By Proposition 3.2.3 since $\mathcal{JS}$ is complementary, it suffices to prove that $\mathrm{SV}_{\mathcal{JS}}(x,\mathcal{I}) = \mathbf{f}$ implies that $\mathrm{SV}_{\mathcal{JS}}(\sim x,\mathcal{I}) = \mathbf{t}$. So take x with $\mathrm{SV}_{\mathcal{JS}}(x,\mathcal{I}) = \mathbf{f}$. This means that every locally complete justification J with x as root has a branch $\mathbf{b}_J$ starting with x with $\mathcal{I}(\mathcal{B}(\mathbf{b}_J)) = \mathbf{f}$. The function mapping such justifications J to $\mathcal{B}(\mathbf{b}_J)$ corresponds to a selection function for x in $\mathrm{Flat}(\mathcal{JS})$. By (1) of Proposition 2.3.8, there is a rule $\sim x \leftarrow A$ in $\mathrm{Flat}(\mathcal{JS})$ with $A \subseteq \{\sim\mathcal{B}(\mathbf{b}_J) \mid J\}$. The case A corresponds to a justification K; hence for every branch $\mathbf{b} \in B_K(\sim x)$ we have $\mathcal{B}(\mathbf{b}) = \sim\mathcal{B}(\mathbf{b}_J)$ for some justification J with x as root. This means that $\mathcal{I}(\mathcal{B}(\mathbf{b})) = \sim\mathcal{I}(\mathcal{B}(\mathbf{b}_J)) = \sim\mathbf{f} = \mathbf{t}$, which proves that $\mathrm{val}_{\mathcal{JS}}(J, \sim x, \mathcal{I}) = \mathbf{t}$. □

This implies that the complementarity of the flattening is at least as strong as the consistency.

However, consistency implies (2) of the complementarity characterisation for $\mathrm{Flat}(\mathcal{JS})$.

Proposition 6.4.2. *Let $\mathcal{JS}$ be a complementary justification system. If $\mathcal{JS}$ is graph-like consistent, then $\mathrm{Flat}_g(\mathcal{JS})$ satisfies (2) of Proposition 2.3.8. If $\mathcal{JS}$ is tree-like consistent, then $\mathrm{Flat}_t(\mathcal{JS})$ satisfies (2) of Proposition 2.3.8.*

Proof. Take $x \leftarrow A$ in $\mathrm{Flat}(\mathcal{JS})$, which corresponds to a locally complete justification J in $\mathcal{JS}$. We define a selection function $\mathcal{S}$ for $\sim x$. Take a rule $\sim x \leftarrow B$ in $\mathrm{Flat}(\mathcal{JS})$, which corresponds to a locally complete justification K in $\mathcal{JS}$. By Lemma 2.3.15 and complementarity of $\mathcal{JS}$, we have that there is a K-branch $\mathbf{b}$ starting with $\sim x$ such that $\sim\mathbf{b}$ is a J-branch starting with x. This means that $\mathcal{B}(\mathbf{b}) \in \sim A \cap B$. Define $\mathcal{S}(B) = \mathcal{B}(\mathbf{b})$. This implies that $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$. □

Consequently, to prove that $\mathrm{Flat}(\mathcal{JS})$ is complementary (and $\mathcal{JS}$ is consistent), you only need to prove (1) of the complementarity characterisation for $\mathrm{Flat}(\mathcal{JS})$.

Corollary 6.4.3. *Let $\mathcal{JS}$ be a complementary justification system. If $\mathrm{Flat}(\mathcal{JS})$ satisfies (1) of Proposition 2.3.8, then it is complementary.*

Proof. Follows from Propositions 6.4.1 and 6.4.2. □

There is one particular kind of justification system $\mathcal{JS}$ for which $\mathrm{Flat}(\mathcal{JS})$ is complementary.

Proposition 6.4.4. *Let $\mathcal{JS}$ be a parametric and complementary justification system. There is a well-founded relation $\preceq$ on $\mathcal{F}_d$ so that*

- *if $x \leftarrow A \in R$, then for all $y \in A$, $y \prec x$.*

Then $\mathrm{Flat}(\mathcal{JS})$ is complementary.

Proof. The well-founded relation can be extended to a well-order with the same properties. So assume $\preceq$ is a well-order from now on. Therefore, we can denote $\mathcal{F}_d = \{x_i \mid i \leq \beta\}$. By Corollary 6.4.3 it suffices to prove that for every selection function $\mathcal{S}$ of x in $\mathrm{Flat}(\mathcal{JS})$, there exists an $\sim x \leftarrow A$ in $\mathrm{Flat}(\mathcal{JS})$ such that $A \subseteq \sim\mathrm{Im}(\mathcal{S})$. We prove this by transfinite induction on $i \leq \beta$. The element x_0 only has rules $x_0 \leftarrow B$ with $B \subseteq \mathcal{F}_o$. And so the justifications for x_0 consist of a single such rule and the selection function $\mathcal{S}$ is a selection of x_0 in $\mathcal{JS}$. By complementarity of $\mathcal{JS}$, we have a rule $\sim x_0 \leftarrow A$ with $A \subseteq \sim\mathrm{Im}(\mathcal{S})$. Since $A \subseteq \mathcal{F}_o$, this is also a rule in $\mathrm{Flat}(\mathcal{JS})$. Take any ordinal i with $0 < i \leq \beta$ and assume that $\mathrm{Flat}(\mathcal{JS})$ is complementary for x_j with $j < i$.

For each $x \leftarrow B$ we will prove that there is a $y_B \in B$ such that either

- $y_B \in \mathcal{F}_o$ and $y \in \mathrm{Im}(\mathcal{S})$, or
- $y_B \in \mathcal{F}_d$ and there is a selection function $\mathcal{S}_y$ of y in $\mathrm{Flat}(\mathcal{JS})$ so that $\mathrm{Im}(\mathcal{S}_y) \subseteq \mathrm{Im}(\mathcal{S})$.

This would imply a selection function $\mathcal{T}$ for x in $\mathcal{JS}$ and by complementarity of $\mathcal{JS}$ would imply the existence of a rule $\sim x \leftarrow C$ with $C \subseteq \sim\mathrm{Im}(\mathcal{T})$. For each $y_B \in \mathcal{F}_d$, we have that $y_B < x$ and thus by induction there is a locally complete justification J_B with $\sim y$ as root and $\{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in \mathbf{b}_{J_B}(\sim y)\} \subseteq \sim\mathrm{Im}(\mathcal{S}_y) \subseteq \sim\mathrm{Im}(\mathcal{S})$. Combining the rule $\sim x \leftarrow C$ and the justifications y_B we get a justification for $\sim x$. The rule in $\mathrm{Flat}(\mathcal{JS})$ corresponding to this justification provides us our desired rule $\sim x \leftarrow A$ in $\mathrm{Flat}(\mathcal{JS})$.

So take a rule $x \leftarrow B$ in $\mathcal{JS}$. Let $\mathfrak{J}$ be the set of locally complete justifications with x as root and $x \leftarrow B$ the rule for x. Any justification J in $\mathfrak{J}$ is the rule $x \leftarrow B$ plus for all $y \in B \setminus \mathcal{F}_o$ a locally complete justification J_y with y as root. We can assume that there is no $y \in \mathcal{F}_o \cap \mathrm{Im}(\mathcal{S})$. Take $y \in B \cap \mathcal{F}_o$. If each locally complete justification J_y with y as root can be extended to a justification in $\mathfrak{J}$ such that $\mathcal{S}(J)$ chooses a path starting with $x \to y$, then $\mathcal{S}$ induces a selection function $\mathcal{S}_y$ of y in $\mathrm{Flat}(\mathcal{JS})$ so that $\mathrm{Im}(\mathcal{S}_y) \subseteq \mathrm{Im}(\mathcal{S})$. This exists by the axiom of choice.

So assume for each $y \in B \cap \mathcal{F}_o$, there is a locally complete justification J_y with y as root that cannot be extended to a justification in $\mathfrak{J}$ such that $\mathcal{S}(J)$ chooses a path starting with $x \to y$. The combination of these J_y together with $x \leftarrow B$ forms a justification J in $\mathfrak{J}$ so that all J-branches are mapped to an element outside of $\mathrm{Im}(\mathcal{S})$. This is a contradiction, which completes the proof that the rules for x_i in $\mathrm{Flat}(\mathcal{JS})$ are complementary. □

The conditions in the previous proposition are quite stringent, as the existence of such a well-founded relation implies that no justification can have infinite branches as that would imply the existence of an infinite decreasing chain.

As usual, the tree-like case is simpler.

Proposition 6.4.5. *If $\mathcal{JS}$ is complementary and $\mathcal{B}$ respects negation, then $\mathrm{Flat}_t(\mathcal{JS})$ is complementary.*

Proof. By Corollary 6.4.3, it suffices to prove (1) of Proposition 2.3.8. Take a selection function $\mathcal{S}$ of x in $\mathrm{Flat}_t(\mathcal{JS})$. This corresponds to a function $\mathcal{T}$ that chooses a branch $\mathbf{b}_J \in B_J(x)$ for each locally complete tree-like justification J with root x. Fix a rule $x \leftarrow A$ in $\mathcal{JS}$. Let $\mathfrak{J}_A$ be the set of locally complete tree-like justifications with x as root and $x \leftarrow A$ as the rule for the root. Every such justification is the combination of a rule $x \leftarrow A$ in $\mathcal{JS}$ and for each $y \in A \cap \mathcal{F}_d$ a locally complete tree-like justification J_y with root y. Let $\times_{y \in A \cap \mathcal{F}_d} J_y$ denote this justification. We prove that there is a $y \in A$ so that

- $y \in \mathcal{F}_o$ and for all $y \in A \cap \mathcal{F}_d$ there are locally complete tree-like justifications J_y with root y so that $\mathcal{T}$ chooses the branch $x \to y$ in the justification $\times_{y \in A \cap \mathcal{F}_d} J_y$.
- $y \in \mathcal{F}_d$ and for each locally complete tree-like justification J_y with root y, we have that for each $z \in (A \cap \mathcal{F}_d) \setminus \{y\}$, there are locally complete tree-like justifications J_z with root z so that $\mathcal{T}$ chooses a branch starting with $x \to y$ in the justification $\times_{z \in A \cap \mathcal{F}_d} J_z$.

Assume by contradiction that such a y does not exist. This means that there are justification J_y for $y \in A \cap \mathcal{F}_d$ so that $\mathcal{T}$ does not choose a branch in $\times_{y \in A \cap \mathcal{F}_d}$, which is a contradiction. Essentially, we chose a $y \in A$ with the above property. Since $x \leftarrow A$ is chosen arbitrarily, we get a selection function for x in $\mathcal{JS}$. By complementarity of $\mathcal{JS}$, there is a rule $\sim x \leftarrow A_x$ so that for all $y \in A_x$, we have that $\sim y$ has the above property. If $\sim y \in \mathcal{F}_d$, then this means that there is a selection function $\mathcal{S}_y$ in $\mathrm{Flat}_t(\mathcal{JS})$ for $\sim y$ so that $\mathrm{Im}(\mathcal{S}_y) \subseteq \mathrm{Im}(\mathcal{S})$. This means we can do the above all over again to construct a rule for y in $\mathcal{JS}$. Iterating this constructs a tree-like justification K with root x so that every branch starting with the root is equal to the negation of a branch chosen by $\mathcal{S}$. Since the construction replaces each defined leaf by a rule, the justification K is locally complete. This proves complementarity of $\mathrm{Flat}_t(\mathcal{JS})$ because $\mathcal{B}$ respects negation. □

This has the consequence that tree-like consistency can be considered to be completely solved.

Theorem 6.4.6. *Every $\mathcal{B}$ that respects negation is tree-like consistent.*

Proof. Follows from Propositions 6.4.1 and 6.4.5. □

This also has some ramifications for graph-like consistency by the link established in Corollary 2.4.7.

Corollary 6.4.7. *If $\mathcal{B}$ respects negation, then $\mathcal{B}$ is graph-like consistent if and only if $\mathcal{B}$ is graph-reducible.*

Proof. Follows from Proposition 6.4.5 and Corollary 2.4.7. □

Corollary 6.4.8. *If $\mathcal{B}$ respects negation, is double-resistant and tree-like extendable, then $\mathcal{B}$ is a graph-like consistent.*

Proof. Follows from Theorem 6.3.34 and Corollary 6.4.7. □

This provides an alternative way to prove that $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{wf}}$ and $\mathcal{B}_{\mathrm{cwf}}$ are graph-like consistent.

Proving complementarity of $\mathrm{Flat}_g(\mathcal{JS})$ can be done by showing the underlying justification frame is equivalent to the underlying justification frame of $\mathrm{Flat}_t(\mathcal{JS})$, which was proven to be complementary. However, this is equivalent to the following property, which we call strongly graph-reducibility.

Definition 6.4.9. A branch evaluation $\mathcal{B}$ is *strongly graph-reducible* if for each complementary justification frame $\mathcal{JF}$, $x \in \mathcal{F}_d$, and every locally complete tree-like justification T with root x, there is a locally complete graph-like justification with root x so that

$$\{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\} \subseteq \{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\}\,.$$

It is obvious that strong graph-reducibility implies graph-reducibility and by Corollary 6.4.7 graph-like consistency.

On the other hand, strong graph-reducibility implies that $\mathrm{Flat}_t(\mathcal{JS})$ and $\mathrm{Flat}_g(\mathcal{JS})$ are equivalent.

Proposition 6.4.10. *Let $\mathcal{JS}$ be a complementary justification system. If $\mathcal{B}$ respects negation and is strong graph-reducibility, then the underlying justification frames of $\mathrm{Flat}_t(\mathcal{JS})$ and $\mathrm{Flat}_g(\mathcal{JS})$ are equivalent.*

Proof. By Proposition 2.4.2, it suffices to prove that for each rule $x \leftarrow A$ in $\mathrm{Flat}_t(\mathcal{JS})$ there is a rule $x \leftarrow B$ in $\mathrm{Flat}_g(\mathcal{JS})$ such that $B \subseteq A$. The rule $x \leftarrow A$ corresponds to a locally complete tree-like justification T with x as root as follows: $A = \{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_T(x)\}$. Then, by strong graph-reducibility, there is a locally complete graph-like justification J with x as root with $\{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\} \subseteq A$, which completes the proof. □

Corollary 6.4.11. *Let $\mathcal{JS}$ be a complementary with $\mathcal{B}$ strong graph-reducible and respecting negation, then $\mathrm{Flat}_g(\mathcal{JS})$ is complementary.*

Proof. By Proposition 6.4.5, $\mathrm{Flat}_t(\mathcal{JS})$ is complementary. Then by Proposition 6.4.10, the result follows. □

Corollary 6.4.12. *Let $\mathcal{JS}$ be a complementary with $\mathcal{B}$ strong graph-reducible and respecting negation, then $\mathcal{JS}$ is graph-like consistent.*

Proof. Follows by Corollary 6.4.11 and Proposition 6.4.1. □

Since $\mathcal{B}_{\mathrm{KK}}$ is strong graph-reducible (see below), it provides yet another prove that $\mathcal{B}_{\mathrm{KK}}$ is graph-like consistent.

Contrast the requirements of Corollary 6.4.8 with that of Corollary 6.4.12. Tree-like extendable and double-resistant branch evaluations are not necessarily strong graph-reducible since the graph-like justification constructed in Theorem 6.3.34, might contain branches that are better. However, the evaluation of the finite branches will be the same as the evaluation of a finite branch in the tree-like

justification. The problem lies with the infinite branches. If the tree-like justification T has $\mathrm{val}(T, x, \mathcal{I}) = \mathbf{u}$, then the constructed graph-like justification might have an infinite branch evaluated to $\mathbf{t}$ under $\mathcal{I}$. We can assume that infinite branches are mapped to logical facts. Therefore, we get a branch $\mathbf{b}$ with $\mathcal{B}(\mathbf{b}) = \mathbf{t}$, while T does not. Hence, the branch evaluation might not satisfy strong graph-reducibility.

In case of $\mathcal{B}_{\mathrm{KK}}$, everything works out.

Lemma 6.4.13. *The branch evaluation $\mathcal{B}_{\mathrm{KK}}$ is strong graph-reducible.*

Proof. Take a locally complete tree-like justification T with root x. In the proof of Theorem 6.3.34, we construct a locally complete graph-like justification J with root x from T given an interpretation $\mathcal{I}$ so that $\mathrm{val}_{\mathcal{B}_{\mathrm{KK}}}(J, x, \mathcal{I}) \geq_t \mathrm{val}_{\mathcal{B}_{\mathrm{KK}}}(T, x, \mathcal{I})$. If J has an infinite branches, then T has an infinite branch as well, by doubling elements in the branch. Therefore, we have that $\{\mathcal{B}_{\mathrm{KK}}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\} \subseteq \{\mathcal{B}_{\mathrm{KK}}(\mathbf{b}) \mid \mathbf{b} \in B_T(x)\}$, completing the proof that $\mathcal{B}_{\mathrm{KK}}$ is strong graph-reducible. □

We are not yet able to prove the strong graph-reducibility of $\mathcal{B}_{\mathrm{wf}}$ (and $\mathcal{B}_{\mathrm{cwf}}$). This is because in the condition of extendable, we can have better branches, and suddenly, we can introduce negative loops where there were none before, see the discussion above. Further exploring whether $\mathcal{B}_{\mathrm{wf}}$ and $\mathcal{B}_{\mathrm{cwf}}$ are strong graph-reducible is important for applications such as fixpoint definitions (see Section 6.5.4).

So far, we have investigated the complementarity of $\mathrm{Flat}(\mathcal{JS})$, but we still need to prove that unfolding is also complementary to prove the complementarity of the compression. For systems $\mathcal{JS}$ of depth 2, we can prove that $\mathrm{Compress}(\mathcal{JS})$ is complementary. First, we need the following lemma.

Lemma 6.4.14. *Let $\mathcal{JS}$ be a nested system of depth 2 with all systems in its nesting tree complementary and the flattening of all leaf systems complementary. Then $\mathrm{Compress}(\mathcal{JS})$ satisfies (2) of Proposition 2.3.8.*

Proof. Take $x \leftarrow A$ in $\mathrm{Compress}(\mathcal{JS})$. This corresponds to a rule $x \leftarrow B$ in the top system and a bunch of rules $y \leftarrow C_y$ for $y \in Y$ in flattening of lowest systems so that $A = B \setminus Y \cup \bigcup_{y \in Y} C_y$. Similarly, each $\sim x \leftarrow D$ in $\mathrm{Compress}(\mathcal{JS})$ corresponds to $\sim x \leftarrow E$ in top and $z \leftarrow F_z$ for $z \in Z$ in flattening of lowest systems so that $D = E \setminus Z \cup \bigcup_{z \in Z} F_z$. By complementarity of the top system, $\sim B \cap E \neq \emptyset$, so take $w \in \sim B \cap E$. If $\sim w \in Y$, then $w \in Z$; hence by complementarity of lowest systems, we get that $\sim C_{\sim w} \cap F_w \neq \emptyset$. So take $v \in \sim C_{\sim w} \cap F_w$. Now define the selection function $\mathcal{S}$ mapping D to

w if $\sim w \notin Y$ and to v otherwise. We have that $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$. Indeed, if $\sim w \notin Y$, then $\sim w \in B \setminus Y \subseteq A$. If $\sim w \in Y$, then $\sim v \in C_{\sim w} \subseteq A$. Therefore, $\sim \mathrm{Im}(\mathcal{S}) \subseteq A$. □

Proposition 6.4.15. *Let $\mathcal{JS}$ be a nested system of depth 2 such that*

1. *all systems in its nesting tree are complementary;*
2. *the flattening of all leaf systems are complementary;*
3. *each defined fact x in the top level has a single rule.*

Then Compress($\mathcal{JS}$) *is complementary.*

Proof. Note that if every fact has a single rule, then the body of every rule is a single element due to complementarity. By Lemma 6.4.14, it suffices to prove that Compress($\mathcal{JS}$) satisfies (1) of Proposition 2.3.8. If x is defined in a leaf, then this is immediate from the complementarity of the flattening of that leaf. So assume x is defined in the top level. Take a selection function $\mathcal{S}$ of x in Compress($\mathcal{JS}$). Any rule $x \leftarrow B$ in Compress($\mathcal{JS}$) is constructed by unfolding the single rule $x \leftarrow y$ in the top level. We distinguish two cases. The first case is when y is open in Compress($\mathcal{JS}$) or defined in the top level. Then by complementarity of the top level, the rule $\sim x \leftarrow \sim y$ is a rule in the top level; hence $\sim x \leftarrow \sim y$ is a rule in Compress($\mathcal{JS}$) and $\mathrm{Im}(\mathcal{S}) = \{y\}$. The second case is when y is defined in the lowest level. Then for every rule $x \leftarrow B$ in Compress($\mathcal{JS}$), $y \leftarrow B$ is a rule in the flattening of the lowest level. This means that $\mathcal{S}$ is also a selection function of x in the flattening of the lowest level. By complementarity of the flattening of the lowest level, there is a rule $\sim y \leftarrow C$ with $C \subseteq \sim \mathrm{Im}(\mathcal{S})$. By unfolding, $\sim x \leftarrow C$ is also a rule in Compress($\mathcal{JS}$); hence Compress($\mathcal{JS}$) is complementary. □

Apart from proving that Compress($\mathcal{JS}$) is complementary, we can also try to prove that Merge($\mathcal{JS}$) is consistent. Merging complementary systems provides a complementary system as shown by the following proposition.

Proposition 6.4.16. *If every system in the nesting tree of $\mathcal{JS}$ is complementary, then* Merge($\mathcal{JS}$) *is complementary as well.*

Proof. Combining rules for different facts does not affect complementarity. □

Proposition 6.4.17. *The merge branch evaluation is tree-like consistent if every branch evaluation respects negation.*

Proof. It is straightforward that the merge branch evaluation respects negation. The result then follows from Theorem 6.4.6. □

This has also a consequence for the compression.

Proposition 6.4.18. *If every system in the nesting tree of $\mathcal{JS}$ is complementary and every branch evaluation involved respects negation, then compression is tree-like consistent.*

Proof. By Theorem 6.3.18, we have for all $x \in \mathcal{F}_d$ that $\mathrm{SV}^t_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) = \mathrm{SV}^t_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I})$ and $\sim \mathrm{SV}^t_{\mathrm{Compress}(\mathcal{JS})}(\sim x, \mathcal{I}) = \sim \mathrm{SV}^t_{\mathrm{Merge}(\mathcal{JS})}(\sim x, \mathcal{I})$. Then by Proposition 6.4.17, we have $\mathrm{SV}^t_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I}) = \sim \mathrm{SV}^t_{\mathrm{Merge}(\mathcal{JS})}(\sim x, \mathcal{I})$. This proves that $\mathrm{SV}^t_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) = \sim \mathrm{SV}^t_{\mathrm{Compress}(\mathcal{JS})}(\sim x, \mathcal{I})$. □

Proving the graph-like consistency of Merge($\mathcal{JS}$) is left for possible future work.

All the results in this chapter around the connection between Merge($\mathcal{JS}$) and Compress($\mathcal{JS}$) and their consistency are shown in Fig. 6.1.

6.5 Applications

Having a modular design in a knowledge representation language is invaluable. Such a design allows to extend the language with various new language constructs without the need to overhaul the existing language and corresponding semantics. If you do not have a modular design, then everytime some new construct is added, everything has to be proven again, perhaps in similar manners, but most of the times it would take a full paper to do this. One line of research is especially messy in this regards: aggregates in logic programming. This is observation becomes apparant by the vast number of semantics for aggregates in logic progrogramming (Faber et al., 2011; Gelfond and Zhang, 2019; Liu et al., 2010; Marek and Remmel, 2004; Pelov et al., 2007; Son et al., 2007).

We argue that nested justification systems have such a modular design. In principle, to add a new language constructs, e.g., aggregates or compound formulae into justification theory, one would only need to come up with an unnested system describing the new language construct. Once that has been worked out, then the nesting will take care of the integration of the feature into the rest of the language. This section illustrate this with first-order logical formulae and aggregates. We must, however, note that these applications only just scratch the surface of what is possible with nesting.

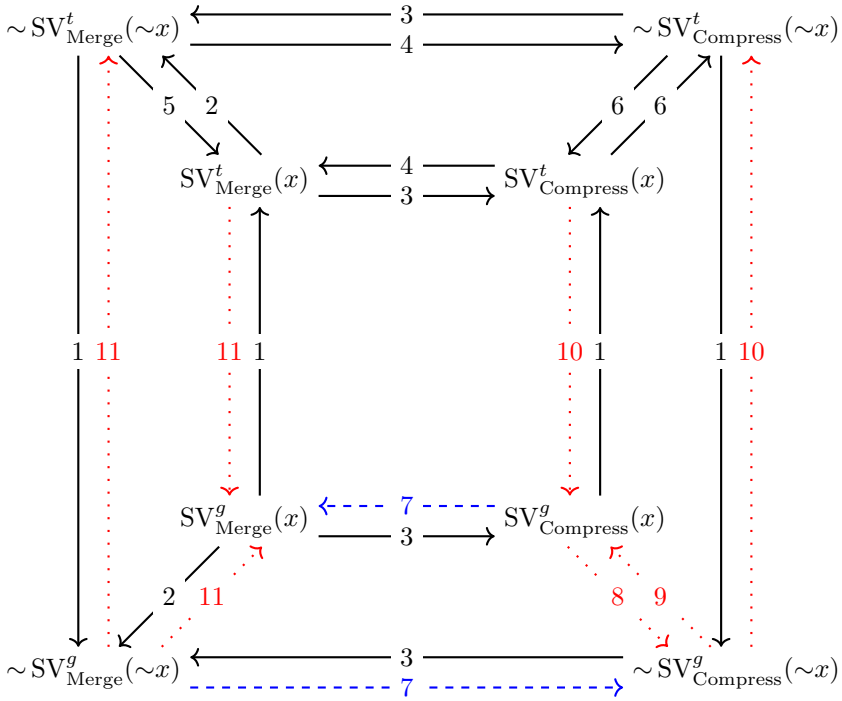


Figure 6.1: This diagram shows the relations between merge and compression, where an arrow from x to y means $x \leq_t y$. The arguments $\mathcal{I}$ and $\mathcal{JS}$ are left out for better presentation. Full lines are proven, dashed (blue) lines are proven for depth 2, all other lines (red) are unproven.

1. Tree-like is stronger than graph-like, see Corollary 2.4.4.
2. The easy side of consistency since merge is complementary, see Proposition 6.4.16 and Theorem 2.3.18.
3. Compression is stronger than merge, see Corollary 6.3.11.
4. Merge is stronger than compression for tree-like justifications, see Corollary 6.3.17.
5. Tree-like consistency of merge, see Proposition 6.4.17.
6. Tree-like consistency of compression, see Proposition 6.4.18.
7. Merge is stronger than compression for graph-like justifications in case of depth 2, see Theorem 6.3.29.
8. This holds if compress is complementary by Theorem 2.3.18.
9. This holds if compress is complementary and the top branch evaluation is consistent.
10. This holds if compress is complementary and top branch evaluation is graph-reducible.
11. This holds if merge is graph-reducible by complementarity of merge, see Proposition 6.4.16 and Corollary 6.4.7.

6.5.1 FO system

This section is not an application of nested systems, but is used as a subsystem in the other applications in the chapter. So far, only propositional facts are used in justification theory, but it is not difficult to add first-order logic formulae. If for example, we have a formula $\phi \wedge \psi$, then we add a rule $\phi \wedge \psi \leftarrow \phi, \psi$, while for $\psi \vee \psi$ we add two rules $\phi \vee \psi \leftarrow \phi$ and $\phi \vee \psi \leftarrow \psi$. This idea can be extended to any formula by looking at its decomposition into subformulae.

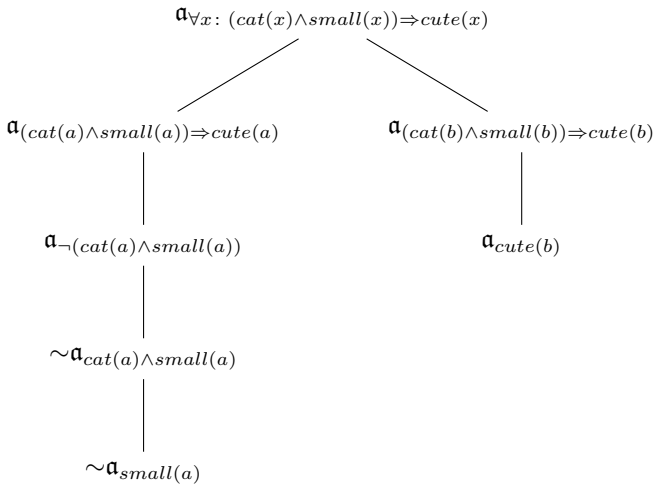
Definition 6.5.1. Let V be a set of first-order predicate symbols over a domain D and let $FOL(V)$ be the set of first-order formulae without free variables over V. To each formula ϕ in $FOL(V)$, we associate a fact $\mathfrak{a}_\phi$. The justification frame $\mathcal{JF}_{FOL(V),D}$ consists of $\mathcal{F}_+ = \{\mathfrak{a}_\phi \mid \phi \in FOL(V)\}$, $\mathcal{F}_d \cap \mathcal{F}_+ = \{\mathfrak{a}_\phi \mid \phi \in FOL(V)$ and ϕ is not an atom$\}$, and R is the complementation of the inductive rule set defined as follows.

- $\mathfrak{a}_{\phi_1 \wedge \cdots \wedge \phi_n} \leftarrow \mathfrak{a}_{\phi_1}, \ldots, \mathfrak{a}_{\phi_n}$;
- $\mathfrak{a}_{\phi_1 \vee \cdots \vee \phi_n} \leftarrow \mathfrak{a}_{\phi_i}$ for all i;
- $\mathfrak{a}_{\neg\phi} \leftarrow {\sim}\mathfrak{a}_\phi$;
- $\mathfrak{a}_{\forall x\colon \phi(x)} \leftarrow \left\{\mathfrak{a}_{\phi[x/d]} \;\middle|\; d \in D\right\}$;
- $\mathfrak{a}_{\exists x\colon \phi(x)} \leftarrow \mathfrak{a}_{\phi[x/d]}$ for all $d \in D$.

Note that we use $\neg$ to denote negation in first-order formulae to distinguish between $\sim$ in the justification frame.

This justification frame can capture first-order logic by letting the facts $\mathfrak{a}_{P(\overline{d})}$ and ${\sim}\mathfrak{a}_{P(\overline{d})}$ for $P \in V$ and $d \in D$ be the set of opens. Usually, we identify $\mathfrak{a}_{P(\overline{d})}$ with $P(\overline{d})$.

Example 6.5.2. Suppose *cat*, *small*, and *cute* are unary predicate symbols and we have a domain $\{a, b\}$ Take the following first-order formula $\forall x\colon (cat(x) \wedge small(x)) \Rightarrow cute(x)$, where $\phi \Rightarrow \psi$ is just a shorthand for $\neg\phi \vee \psi$. A locally complete justification for this formula can look as follows.



Note that the fact $\sim\mathfrak{a}_{cat(a)\wedge small(a)}$ has two cases: $\{\sim\mathfrak{a}_{cat(a)}\}$ and $\{\sim\mathfrak{a}_{small(a)}\}$. This is due to complementation since $\mathfrak{a}_{cat(a)\wedge small(a)}$ has only one case: $\{\mathfrak{a}_{cat(a)}, \mathfrak{a}_{small(a)}\}$. This justification shows that if a is not small and b is cute, then the formula $\forall x\colon (cat(x) \wedge small(x)) \Rightarrow cute(x)$ is true. ▲

The idea in the example above works for any branch evaluation that maps finite branches to their last element. This includes $\mathcal{B}_{\mathrm{KK}}$, $\mathcal{B}_{\mathrm{wf}}$, $\mathcal{B}_{\mathrm{wf}}$, and the alternative versions of $\mathcal{B}_{\mathrm{sp}}$ and $\mathcal{B}_{\mathrm{st}}$. However, $\mathcal{B}_{\mathrm{KK}}$ is the better candidate in this case, since it says nothing about infinite branches ($\mathbf{u}$) and it is the least precise such branch evaluation: $\mathcal{I}(\mathcal{B}_{\mathrm{KK}}(\mathbf{b})) \leq_p \mathcal{I}(\mathcal{B}(\mathbf{b}))$.

Definition 6.5.3. Let $\mathcal{JS}_{FOL(V),D}$ be the justification system with justification frame $\mathcal{JF}_{FOL(V),D}$ and branch evaluation $\mathcal{B}_{\mathrm{KK}}$.

We can simulate first-order logic with justification theory, however note that justification theory is three-valued and thus will also be able to cope with unknown facts. For instance in Example 6.5.2, if we do not know anything of a, but b is a small cat, but not cute, then our formula is false.

Proposition 6.5.4. *For an interpretation of the atoms in $FOL(V)$, the single model $\mathcal{I}$ of $\mathcal{JS}_{FOL(V),D}$ extending this interpretation captures (three-valued) first-order logic: $\phi^I = \mathrm{SV}_{\mathcal{JS}}(\mathfrak{a}_\phi, \mathcal{I})$.*

Proof. Follows by induction on the depth of the formula. □

This system will be used extensively as a subsystem in a nested system and will essentially enable first-order logic formulae in justification systems. Therefore, it is important to know that $\mathrm{Flat}(\mathcal{JS}_{FOL(V),D})$ is complementary.

Proposition 6.5.5. *The system* $\mathrm{Flat}(\mathcal{JS}_{FOL(V),D})$ *is complementary.*

Proof. Let $\phi \preceq \psi$ if ϕ is a subformula of ψ. This is a well-founded relation, then Proposition 6.4.4 finishes the proof. □

If you are only interested in a finite set of formulae, then $\mathcal{JS}_{FOL(V),D}$ can be restricted to the subformulae of these formulae.

6.5.2 Aggregates

In Section 2.7.1, we have seen how to construct a justification frame for normal logic programs. If you want to extend logic programs with aggregates, we need to add extra rules for aggregate atoms. However, now we have nested justification systems at our disposal, we can define the aggregate atoms in a nested system. This allows for a more modular representation of the rules. For simplicity, let us add weight constraints of the form $i \leq \{x_1, \ldots, x_n\} \leq j$ meaning that at least i and at most j xs hold. Let $L = \{x_1, \ldots, x_n\}$. Then we add facts $w_{i,L,j}$ and $\sim w_{i,L,j}$ representing the weight constraints. Then we have the following rules

$$w_{i,L,j} \leftarrow L^+ \cup \sim L^-,$$

where $L^+, L^- \subseteq L$ and $|L^+| = i$ and $|L^-| = n-j$. If $L^+ \cup \sim L^- = \emptyset$, then we have a rule $w_{i,L,j} \leftarrow \mathbf{t}$. The rules for $\sim w_{i,L,j}$ come from taking the complementation of the rules for $w_{i,L,j}$. By adding these rules in a nested system we added weight constraints to our justification frame. The branch evaluation of this inner system does not matter as long as finite branches are mapped to their last element, so we can take $\mathcal{B}_{\mathrm{KK}}$ as we did for the first-order logic systems.

An added benefit of using nested justification systems means that we can have nested weight constraints. The maximum nested depth of these weight constraints will dictate the nesting depth of the nested system. So even though, the definition of these weight constraints is very easy, nesting will allow arbitrary nesting of the constructs.

Example 6.5.6. The logic program

$$\left\{ \; p \leftarrow w_{0,\left\{w_{1,\{x_1,x_2\},2}, w_{0,\{x_1\},1}\right\},1} \; \right\}$$

will be translated to the nested system

$$\left\{ \begin{array}{l} p \leftarrow w \\ \\ \mathcal{B}_{\mathrm{KK}}: \left\{ \begin{array}{l} w \leftarrow \sim v_1 \\ w \leftarrow \sim v_2 \\ \\ \mathcal{B}_{\mathrm{KK}}: \left\{ \begin{array}{l} v_1 \leftarrow x_1 \\ v_1 \leftarrow x_2 \\ v_2 \leftarrow \mathbf{t} \end{array} \right\} \end{array} \right\} \end{array} \right\}$$

The fact p will be $\mathbf{t}$ in a model only if both x_1 and x_2 are false. ▲

This application illustrates that we can define new language constructs locally and use them in a modular fashion.

6.5.3 First-Order Definitions: FO(ID)

In this section, we capture First-order logic extended with Inductive Definitions (FO(ID)) definitions (Denecker and Vennekens, 2007). Let Σ be a vocabulary consisting of a set of predicate symbols. A *definitional rule* over Σ is an expression of the form:

$$\forall \overline{x}(P(\overline{x}) \leftarrow \phi),$$

where P is a predicate symbol in Σ, $\overline{x}$ a tuple of n variables, and ϕ an first-order logic (FO) formula over Σ such that the free variables of ϕ all occur in x. Similar to rules in justification frame, $P(\overline{x})$ is the head and ϕ the body of the rule.

Definition 6.5.7. A *first-order definition* Δ over Σ is a set of definitional rules. The set $\mathrm{Def}(\Delta)$ is the set of predicate symbols occurring as a head in some definitional rule of Δ. The set $\mathrm{Open}(\Delta) = \Sigma \setminus \mathrm{Def}(\Delta)$ is the set of *open* symbols.

As with justification frames, we will denote them with curly braces:

$$\left\{ \begin{array}{l} \mathit{Prime}(2) \leftarrow \mathbf{t} \\ \forall x(\mathit{Prime}(x) \leftarrow 2 < x \wedge \neg \exists y\colon y < x \wedge \mathit{Prime}(y) \wedge \mathit{Divisible}(x, y)) \end{array} \right\}$$

Here Prime is in $\mathrm{Def}(\Delta)$ and $\mathit{Divisible}$ and $<$ (viewed as a binary predicate) are in $\mathrm{Open}(\Delta)$.

For a given domain D, a value for an n-ary predicate symbol is a function from D^n to $\mathcal{L} = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. A Σ-interpretation $\mathcal{I}$ consists of a domain $D^{\mathcal{I}}$ and a value $\sigma^{\mathcal{I}}$ for each symbol σ in Σ. Let Δ be a definition over Σ and O a two-valued $\mathrm{Open}(\Delta)$-interpretation. Let V_O^{Σ} be the collection of three-valued

Σ-interpretations extending O. On this set, we can define the three-valued immediate consequence operator Ψ_Δ^O which maps any $\mathcal{I} \in V_O^\Sigma$ to the O-extension $\mathcal{J}$ such that for each defined domain atom $P(\overline{a})$,

$$P(\overline{a})^{\mathcal{J}} = \max_{\leq_t} \left\{ \phi(\overline{a})^{\mathcal{J}} \;\middle|\; \forall \overline{x}(P(\overline{x}) \leftarrow \phi) \in \Delta \right\}.$$

The set V_O^Σ is isomorphic to the set L^c for the lattice L of two-valued Σ-interpretations extending O. Therefore, Ψ_Δ^O is a consistent approximator and has a well-founded fixpoint $\mathcal{I}_O^\Delta$, which in general is three-valued.

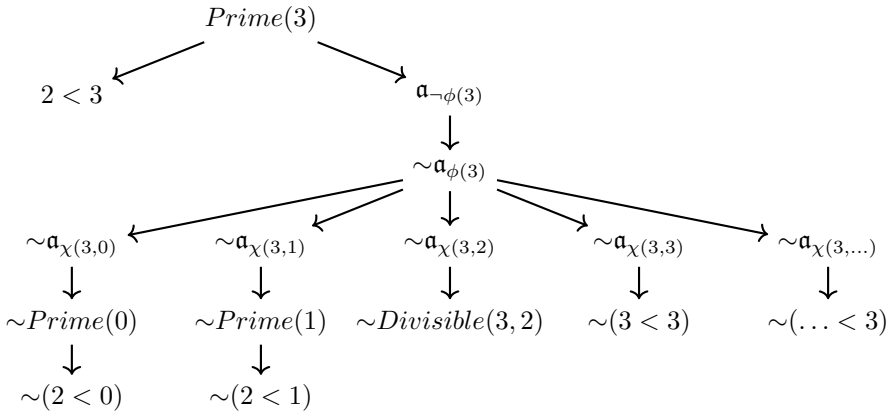
Definition 6.5.8. The *model* of Δ with respect to a two-valued interpretation O of $\text{Open}(\Delta)$ is the two-valued well-founded model of Ψ_Δ^O.

In order to capture first-order definitions with justification theory, we should use the well-founded semantics and the AFT correspondence from Chapter 5. Previously in this chapter, we have seen that introducing first-order formulas into bodies of rule can change the semantics. Fortunately, nested systems solved this predicate introduction problem. These compound formulae in bodies can be decomposed into their subformulae in a subsystem.

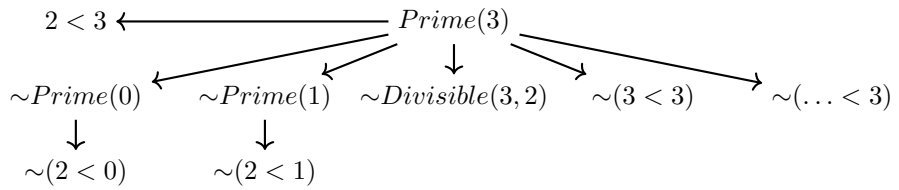
Example 6.5.9. The above definition for *Prime* could be transformed into the following nested system.

$$\mathcal{B}_{\text{wf}}\colon \left\{ \begin{array}{l} Prime(2) \leftarrow \mathbf{t} \\ Prime(x) \leftarrow \mathfrak{a}_{\phi(x)} \text{ for all } x \in \mathbb{N} \\ \mathcal{B}_{\text{KK}}\colon \left\{ \begin{array}{l} \mathfrak{a}_{\phi(x)} \leftarrow x > 2, \mathfrak{a}_{\neg\psi(x)} \text{ for all } x \in \mathbb{N} \\ \mathfrak{a}_{\neg\psi(x)} \leftarrow {\sim}\mathfrak{a}_{\psi(x)} \text{ for all } x \in \mathbb{N} \\ \mathfrak{a}_{\psi(x)} \leftarrow \mathfrak{a}_{\chi(x,y)} \text{ for all } x, y \in \mathbb{N} \\ \mathfrak{a}_{\chi(x,y)} \leftarrow y < x, Prime(y), Divisible(x,y) \text{ for all } x, y \in \mathbb{N} \end{array} \right\} \end{array} \right\}$$

A justification for $Prime(3)$ in the merge would look like



Shrinking this justification provides the following justification in the compression



The shrinking essentially removes the auxiliary symbols. In both justifications, there are a lot of superfluous facts of the form $\sim(y < 3)$ for $y \geq 3$. This is a shortcoming of the propositional nature of justification theory. We will come back to this in the conclusion chapter. ▲

In general, FO(ID) definitions can be evaluated similarly with justification theory.

Definition 6.5.10. Let Δ be an FO(ID) definition. The justification system $\mathcal{JS}_\Delta$ associated with Δ is the nested system with nesting tree of depth two where

- The single leaf system is the restriction of $\mathcal{JS}_{FOL(V),D}$ to the set of formulae $\{\phi \mid \forall \overline{x}(P(\overline{x}) \leftarrow \phi) \in \Delta\}$.
- The root system is the system with $\mathcal{B}_{\text{wf}}$ as branch evaluation and rules equal to the complementation of the set

$$\left\{ \; P(\overline{d}) \leftarrow \phi[\overline{x}/\overline{d}] \;\mid\; \overline{d} \in D^n, \forall \overline{x}(P(\overline{x}) \leftarrow \phi) \in \Delta \; \right\}.$$

Proposition 6.5.11. *The system* Compress($\mathcal{JS}_\Delta$) *is complementary if every* $P(x)$ *has at most one definitional rule.*

Proof. This follows from Propositions 6.5.5 and 6.4.15 . □

Multiple definitional rules for the same $P(x)$ can be merged by taking the disjunction of the bodies.

Since $\mathcal{JS}_\Delta$ has two levels, we know that Merge($\mathcal{JS}_\Delta$) and Compress($\mathcal{JS}_\Delta$) are strongly equivalent. Therefore, we can speak of the model of $\mathcal{JS}_\Delta$: the model of Compress($\mathcal{JS}_\Delta$) or Merge($\mathcal{JS}_\Delta$).

Theorem 6.5.12. *The model of* Δ *corresponds to the model of* $\mathcal{JS}_\Delta$.

Proof. Assume O is a fixed interpretation of the open symbols. The operator $A_{\text{Compress}(\mathcal{JS}_\Delta)}$ is equal to the three-valued immediate consequence operator Ψ^O_Δ. Then by complementarity of $\mathcal{JS}_\Delta$ and the correspondence from Chapter 5, we have that the well-founded model of Compress($\mathcal{JS}_\Delta$) is equal to the well-founded fixpoint of Ψ^O_Δ, which concludes the proof. □

Many different types of definitions found in mathematics can be formalised with well-founded semantics in logic programming (Denecker, 1998; Denecker and Ternovska, 2008; Denecker and Vennekens, 2014; Denecker et al., 2001) and thus by extension in justification theory. One can argue that, in general, mathematical language is first-order in nature. This is evident from the fact that a large part of mathematics can be expressed in Zermelo–Fraenkel set theory, which in turn is expressed with first-order logic. Denecker and Vennekens (2007) state

> The fact that the inability to express inductive definitions is a well-known weakness of first order logic, has subsequently motivated an extension of FO with a new construct for representing definitions, whose semantics is based on the well-founded semantics.

FO(ID) is exactly such an extension of FO. Justification theory can capture the definitional part of FO(ID). Very similar, one can capture coinductive definitions by swapping $\mathcal{B}_{\text{wf}}$ with $\mathcal{B}_{\text{cwf}}$, see the next section for more details.

6.5.4 Fixpoint Definitions: FO(FD)

In this section, we will capture fixpoint definitions (Hou and Denecker, 2009; Hou et al., 2010) with justification theory. Fixpoint definitions are inspired by first-order definitions and in fact nested justification systems are inspired by fixpoint definitions as done by Hou and Denecker (2009).

Definition 6.5.13 (Hou and Denecker, 2009, Definition 1)**.** A *least fixpoint definition*, respectively *greatest fixpoint definition* over a vocabulary Σ is defined by simultaneous induction as an expression $\mathcal{D}$ of the form

$$\lfloor \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n \rfloor \quad \text{respectively} \quad \lceil \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n \rceil$$

with $0 \leq n, m$ such that:

1. $\mathcal{R}$ is a set of definitional rules over Σ.
2. Each Δ_i is a least fixpoint definition over Σ and each ∇_j is a greatest fixpoint definition over Σ.

3. Every defined symbol in $\mathcal{D}$ has only positive occurrences in bodies of rules in $\mathcal{D}$.

4. Each defined symbol $P \in \mathrm{Def}(\mathcal{D})$ has exactly one local definition, i.e. formally $\{\mathrm{Def}(\mathcal{R}), \mathrm{Def}(\Delta_1), \ldots, \mathrm{Def}(\nabla_n)\}$ is a partition of $\mathrm{Def}(\mathcal{D})$. Formally: either locally defined or in some sub definitions.

5. For every subdefinition $\mathcal{D}'$ of $\mathcal{D}$, $\mathrm{Open}(\mathcal{D}') \subseteq \mathrm{Open}(\mathcal{D}) \cup \mathrm{Def}(\mathcal{R})$.

A *fixpoint definition* is either a least or greatest fixpoint definition. The sets $\mathrm{Def}(\mathcal{D})$ and $\mathrm{Open}(\mathcal{D})$ are defined similarly as for inductive definitions.

We assume that there is a single rule $\forall \overline{x}(P(\overline{x}) \leftarrow \phi_P)$ with $P \in \mathrm{Def}(\mathcal{D})$.

Given two disjoint first-order vocabularies Σ and Σ', and Σ-interpretation I and Σ'-interpretation I. The $\Sigma \cup \Sigma'$-interpretation mapping each element $\sigma \in \Sigma$ to σ^I and each $\sigma \in \Sigma'$ to $\sigma^{I'}$ is denoted by $I + I'$. When $\Sigma' \subseteq \Sigma$, we denote the restriction of a Σ-interpretation I to Σ' by $I|_{\Sigma'}$.

Fix a domain D. Let $\mathcal{R}$ be a set of definitional rules and O a two-valued $\mathrm{Open}(\mathcal{R})$-interpretation. We can associate an operator $\Gamma^{\mathcal{R}}_O$ on the set of $\mathrm{Def}(\mathcal{R})$-interpretations.

We define $\Gamma^{\mathcal{R}}_O(I_1) = I_2$ if for every $\forall \overline{x}(P(\overline{x}) \leftarrow \phi_P) \in \mathcal{R}$, $P^{I_2} = \left\{\overline{d} \in D^n \;\middle|\; \phi_P[\overline{x}/\overline{d}]^{I_1} = \mathbf{t}\right\}$.

Since each defined symbol in $\mathrm{Def}(\mathcal{R})$ has only positive occurrences in the body of a rule in $\mathcal{R}$, we have that $\Gamma^{\mathcal{R}}_O$ is monotone with respect to $\leq_t$. Therefore, it has least and greatest fixpoints denoted by $\mathrm{lfp}(\Gamma^{\mathcal{R}}_O)$ and $\mathrm{gfp}(\Gamma^{\mathcal{R}}_O)$.

Let $\mathcal{D}$ be a fixpoint definition and let O be a two-valued $\mathrm{Open}(\mathcal{D})$-interpretation. We define an operator $\Gamma^{\mathcal{D}}_O$ on the set of $\mathrm{Def}(\mathcal{D})$-interpretations.

$\Gamma^{\mathcal{D}}_O(I)$ is defined inductively as the interpretation $K + K'$ where

- K is the $(\mathrm{Def}(\mathcal{D}) \setminus \mathrm{Def}(\mathcal{R}))$-interpretation such that for $I' = O + I|_{\mathrm{Def}(\mathcal{R})}$:
  - $K|_{\mathrm{Def}(\Delta_i)} = \mathrm{lfp}(\Gamma^{\Delta_i}_{I'})$ for all $i = 1, \ldots, m$.
  - $K|_{\mathrm{Def}(\nabla_j)} = \mathrm{gfp}(\Gamma^{\nabla_j}_{I'})$ for all $j = 1, \ldots, n$.
- K' is $\mathrm{Def}(\mathcal{R})$-interpretation $\Gamma^{\mathcal{R}}_{O+K}(I|_{\mathrm{Def}(\mathcal{R})})$.

Definition 6.5.14. Let $\mathcal{D}$ be a fixpoint definition and I a two-valued interpretation of the symbols in $\mathcal{D}$. The interpretation I is a *model* of $\mathcal{D}$ if either

- $\mathcal{D}$ is a least fixpoint and $I|_{\mathrm{Def}(\mathcal{D})} = \mathrm{lfp}\left(\Gamma^{\mathcal{D}}_{I|_{\mathrm{Open}(\mathcal{D})}}\right)$.
- $\mathcal{D}$ is a greatest fixpoint and $I|_{\mathrm{Def}(\mathcal{D})} = \mathrm{gfp}\left(\Gamma^{\mathcal{D}}_{I|_{\mathrm{Open}(\mathcal{D})}}\right)$.

The similarities between fixpoint definitions and nested justification systems are abundant. Fixpoint definitions have an additional restriction: every defined symbol has only positive occurrences in bodies of rules.

Definition 6.5.15. A justification frame $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R\rangle$ is *positive* if for every $x \in \mathcal{F}_+$ and $x \leftarrow A \in R$ we have that $A \cap \mathcal{F}_d \cap \mathcal{F}_- = \emptyset$.

In case a justification frame is positive, then the unique $\mathcal{B}_{\mathrm{wf}}$-model corresponds to the least fixpoint of $O_{\mathcal{JF}}$ and similarly the unique $\mathcal{B}_{\mathrm{cwf}}$-model corresponds to the greatest fixpoint of $O_{\mathcal{JF}}$.

Proposition 6.5.16. *Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R\rangle$ be a positive complementary justification frame. If the interpretation of the opens is fixed, then*

- *the unique $\mathcal{B}_{\mathrm{wf}}$-model $\mathcal{I}$ of $\mathcal{JF}$ corresponds (as in Section 5.3.1) to (I, I) with $I = \mathrm{lfp}(O_{\mathcal{JF}})$.*
- *the unique $\mathcal{B}_{\mathrm{cwf}}$-model $\mathcal{I}$ of $\mathcal{JF}$ corresponds to (I, I) with $I = \mathrm{gfp}(O_{\mathcal{JF}})$.*

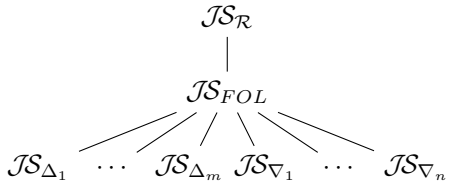
Proof. Since $\mathcal{JF}$ is positive, we have that $O_{\mathcal{JF}}$ is monotone with respect to $\subseteq$, hence $O_{\mathcal{JF}}$ has least and greatest fixpoints with respect to $\subseteq$. This also means that there is a $\leq_t$-least $\mathcal{B}_{\mathrm{sp}}$-model and $\leq_t$-greatest $\mathcal{B}_{\mathrm{sp}}$-models and they correspond with $\mathrm{lfp}(O_{\mathcal{JF}})$ and $\mathrm{gfp}(O_{\mathcal{JF}})$. Since $\mathcal{JF}$ is positive, the $\leq_t$-least $\mathcal{B}_{\mathrm{sp}}$-model is the unique $\mathcal{B}_{\mathrm{st}}$-model. Since there is a single $\mathcal{B}_{\mathrm{st}}$-model, it is equal to the unique $\mathcal{B}_{\mathrm{wf}}$-model. Similarly, the $\leq_t$-greatest $\mathcal{B}_{\mathrm{sp}}$-model is the unique $\mathcal{B}_{\mathrm{cst}}$-model, which is equal to the unique $\mathcal{B}_{\mathrm{cwf}}$-model. □

We now are going to construct a nested system corresponding to a fixpoint definition. Take a fixpoint definition $\mathcal{D}$.

Definition 6.5.17. For a set of definitional rules, the justification frame $\mathcal{JF}_{\mathcal{R}}$ contains the complementation of the rules

$$\left\{ P(\overline{d}) \leftarrow \mathfrak{a}_{\phi_P[\overline{x}/\overline{d}]} \;\middle|\; \forall \overline{x}(P(\overline{x} \leftarrow \phi_P) \in \mathcal{R} \right\}.$$

The nested system $\mathcal{JS}_{\mathcal{D}}$ is defined inductively as the nested system with nesting tree (the leaves represent the nesting trees of those systems)



where $\mathcal{JS}_{\mathcal{R}}$ is $\mathcal{JF}_{\mathcal{R}}$ together with $\mathcal{B}_{\text{wf}}$ if $\mathcal{D}$ is a least fixpoint and $\mathcal{B}_{\text{cwf}}$ is $\mathcal{D}$ is a greatest fixpoint. The system $\mathcal{JS}_{FOL}$ is the restriction of the system from Definition 6.5.3 to the formula ϕ_P in $\mathcal{R}$.

There is a slight hiccup in the definition above: it is possible that ϕ_P and ϕ_Q have subformulae in common. In that case, some facts are defined in multiple levels. We can avoid this by doubling the facts: every $\mathcal{JS}_{FOL}$ system uses a different symbol, for example $\mathfrak{a}_\phi$ and $\mathfrak{b}_\phi$.

Example 6.5.18. Let T be a binary predicate symbol denoting a transition graph and R a unary predicate on the states. The set of states that have a path passing infinitely many times through a state satisfying R is captured by the predicate P defined using the following fixpoint definition.

$$\left[\begin{array}{l} \forall x(P(x) \leftarrow Q(x)) \\ \left[\begin{array}{l} \forall x(Q(x) \leftarrow R(x) \wedge \exists y\colon T(x,y) \wedge P(y)) \\ \forall x(Q(x) \leftarrow \exists y\colon T(x,y) \wedge Q(y)) \end{array}\right] \end{array}\right]$$

The corresponding justification system is in Fig. 6.2. ▲

Lemma 6.5.19. *Let $\mathcal{D}$ be a fixpoint definition. Then* $\text{Compress}_t(\mathcal{JS}_{\mathcal{D}})$ *is complementary.*

Proof. Follows by induction using Propositions 6.5.5, 6.4.5 and 6.4.15. □

Theorem 6.5.20. *Let $\mathcal{D}$ be a fixpoint definition. Then the model of $\mathcal{D}$ is equal to the model of* $\text{Compress}(\mathcal{JS}_{\mathcal{D}})$

Proof. We do this by induction of the depth of $\mathcal{D}$. If no nesting occurs, then $O_{\mathcal{JF}}$ associated to $\text{Compress}(\mathcal{JS}_{\mathcal{D}})$ is equal to $\Gamma^{\mathcal{R}}_{O}$. Then by Proposition 6.5.16 the result follows. So assume nesting occur and the theorem holds for all fixpoint definitions with smaller depth. The operator $\Gamma^{\Delta_i}_{J'}$ correspond to $O_{\mathcal{JF}}$ associated to $\text{Compress}(\mathcal{JS}_{\Delta_i})$ and $\Gamma^{\nabla_j}_{J'}$ correspond to $O_{\mathcal{JF}}$ associated to $\text{Compress}(\mathcal{JS}_{\nabla_j})$. This means that $O_{\mathcal{JF}}$ associated to $\text{Compress}(\mathcal{JS}_{\mathcal{D}})$ is equal to $\Gamma^{\mathcal{D}}_{O}$. Then by Proposition 6.5.16 the proof is completed. □

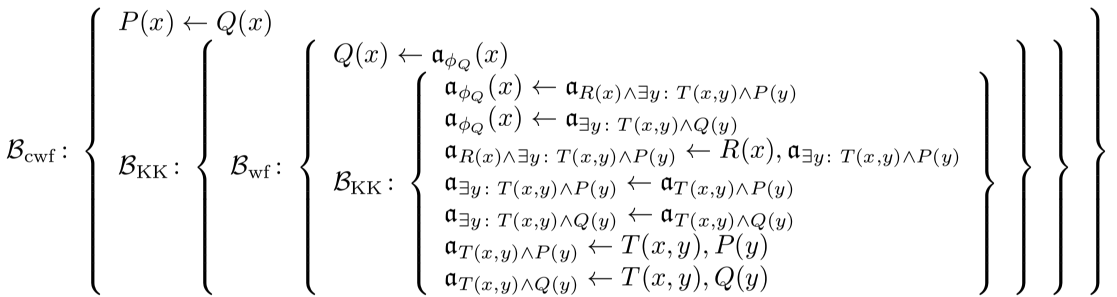


Figure 6.2: The corresponding justification system to the fixpoint definition of Example 6.5.18.

6.6 Conclusion

In this chapter, we gave a new view on nested justification systems that retains the justification quality by merging the nested systems into a single justification system with an adapted branch evaluation. Our new approach is actually more general, as it allows for branch evaluations that are not parametric, while parametricity is essential for compression. We identified a sufficient condition such that the compression and merge are strongly equivalent for tree-like justifications. As usual, graph-like justifications tend to be more difficult. But with more restrictions, we are able to prove it for systems of depth 2. This property, extendable, is however not really properly understood. Future work could try to understand what it means to be extendable: what is the significance of the well-founded order and what is its intuitive meaning. We have seen that extendable branch evaluation are also graph-reducible if double-resistance is also required. We discussed the consistency of the compression and merge and found that this is intricately related to the complementarity of flat systems. We proved that tree-like flat systems are complementary and thus we solved tree-like consistency for branch evaluations that respect negation. This means that graph-like consistency is equivalent to graph-reducibility, which could help with future research. Nested justification systems are a really powerful tool to design modular knowledge representation, as shown by the applications at the end of this chapter.

Chapter 7

Conclusion

7.1 Contributions

In this thesis we investigated two main properties. The first one is the consistency of justification semantics. Intuitively, it says that if a fact does not have a good justification, then its negation has a good justification and vice versa. The second is the difference between graph-like and tree-like justifications, or as we called it, graph-reducibility. These seemingly unrelated properties are actually intricately related: graph-like consistency holds if and only if tree-like consistency and graph-reducibility hold. Because of this relation, our attention was mainly focused on graph-like consistency. In Chapter 3, we proved the graph-like consistency for the main branch evaluations. At the end of this thesis, we managed to prove tree-like consistency for any branch evaluation that respects negation. Therefore, graph-like consistency is equivalent to just graph-reducibility.

Beside these two properties, we have three other research questions each handled in a separate chapter. First, we showed in Chapter 4 that justification systems can be seen as two-player games. This allowed us to solve the consistency problem in case we have a finite system. The second result is the embedding of justification theory into AFT. This embedding is an important one as it links a logical framework (justification theory) to an algebraic one (AFT). We managed to transfer ultimate semantics, a concept from AFT, to justification theory. The last line of results is discussed in Chapter 6: a different view on nested justification systems. Even though, it is not completely solved for graph-like justifications, nested justification systems can be used to define modular

semantics as shown by the various applications at the end of Chapter 6. In that chapter, we showed that nested justification systems can be viewed in a different light, at least for tree-like justifications. At the end of this chapter, by happenstance, we also solved tree-like consistency.

This work advances the state of justification theory quite a lot compared to the works of Denecker (1993); Denecker et al. (2015). We established a number of properties that can be checked for new branch evaluations in order to check their soundness.

7.2 Future Directions

Conducting science usually raises more questions than it can answer. This thesis is no exception. We distinguish two future research directions: applications and extensions of justification theory. Let us first discuss the challenges and open questions.

7.2.1 Challenges and Open Questions

In Chapter 3, we discussed the connection between different justification semantics. In particular, we have that every $\mathcal{B}_{\text{wf}}$-model is a $\mathcal{B}_{\text{st}}$-model, every $\mathcal{B}_{\text{KK}}$-model and every $\mathcal{B}_{\text{st}}$-model are a $\mathcal{B}_{\text{sp}}$-model, and similar for $\mathcal{B}_{\text{cwf}}$ and $\mathcal{B}_{\text{cst}}$. Determining such a relation between two branch evaluations happens on a case-by-case basis. Crucial for proving these relations are the splittability property and the associated pasting theorem (Theorem 2.6.20). Determining a general property on two branch evaluations $\mathcal{B}_1$ and $\mathcal{B}_2$ so that every $\mathcal{B}_1$-model is a $\mathcal{B}_2$-model might give more insight between the relation of various branch evaluations. Similarly, when fixing the interpretation of the open facts, we have that the $\mathcal{B}_{\text{KK}}$-model is the $\leq_p$-least $\mathcal{B}_{\text{sp}}$-model, the $\mathcal{B}_{\text{wf}}$-model is the $\leq_p$-least $\mathcal{B}_{\text{st}}$-model, and similarly for $\mathcal{B}_{\text{cwf}}$ and $\mathcal{B}_{\text{cst}}$. Also, for these types of result, having an easy to check property might come in handy.

Chapter 4 establishes justification theory in a game theoretic setting. This connection is only used to prove consistency in a finite context. Future work should exploit it further by bringing results from game theory to justification theory. For example, results on memoryless determinacy of parity games (Emerson and Jutla, 1991; Roux, 2019; Zielonka, 1998) and of ω-automata (Jutla, 1997) might be useful to prove graph-like consistency. If a game associated to a justification system is equivalent to a parity game or an ω-automata, then by our connection, the justification system is graph-like consistent. For a given

branch evaluation, not every justification system will have that form. Therefore, this line of work can be useful to determine consistency of certain systems instead of the complete branch evaluation at hand.

The correspondence between AFT and justification theory outlined in Chapter 5 is a one-way street: for every justification frame, we can associate an AFT-approximator, but not the other way around. Therefore, it begs the question if the reverse is true. That is, does every approximator on a power set lattice correspond to a justification frame? If the answer is positive, then all applications of AFT on the power set lattice can be captured using justification theory. Let L be a power set lattice on the set F. In our connection, F is the set of positive (defined) facts. So given F, we can define $\mathcal{F}_+ \coloneqq F$. The difficult part is to come up with the correct rules. Every I in L corresponds to a conjunction $\bigwedge_{x\in I} x \wedge \bigwedge_{x\notin I} {\sim}x$. A naive rule set might have rules $y \leftarrow I_1 \cup {\sim}(L \setminus I_2)$ for $y \in A(I_1, I_2)_1$. This would however constructs a lot of rules with big bodies. Perhaps some resolution similar to ultimate justification frames can be performed. Solving this might allow us to bring over results in AFT to the main justification semantics. One example is stratification (Bogaerts and Cruz-Filipe, 2021; Vennekens et al., 2006, 2007a). Perhaps, we can characterise this more generally: in which cases is a justification frame stratifiable.

Apart from the main fixpoint types, there is another important type of fixpoint: the grounded fixpoints (Bogaerts, 2015; Bogaerts et al., 2015a,b). This raises the question whether there is a branch evaluation for which the models correspond to the grounded fixpoints. Perhaps, the justification frame has to be manipulated to be able to capture grounded fixpoints, just as we did for ultimate semantics.

We have added new type of fixpoints for co-well-founded and co-stable semantics on the AFT side. This begs the question if we can find corresponding algebraic fixpoints for an arbitrary branch evaluation.

The nested systems of Chapter 6 have a few open questions. At first, the graph-like strong equivalence between merge and compress is only proved for systems of depth two in case the branch evaluation is extendable. We conjecture this is true for any depth.

Conjecture 7.2.1. *If all branch evaluations are extendable and nestable, then* Merge($\mathcal{JS}$) *and* Compress($\mathcal{JS}$) *are strongly equivalent for graph-like justifications.*

Here, two possibilities arise to prove this. First, we can prove it directly by using the following conjecture.

Conjecture 7.2.2. *Let* $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$ *be a nested system of depth two and for each* $1 \leq i \leq k$*, let* $\mathcal{JS}^{i'}$ *be a system strongly*

equivalent to $\mathcal{JS}^i$*. Then* Compress($\mathcal{JS}$) *is strongly equivalent to*

$$\text{Compress}\left(\left\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \left\{ \mathcal{JS}^{1\prime}, \ldots, \mathcal{JS}^{k\prime} \right\} \right\rangle\right).$$

Proposition 7.2.3. *If Conjecture 7.2.2 holds, then Conjecture 7.2.1 holds.*

Proof. The proof is done by induction of the depth of $\mathcal{JS}$. The cases depth 1 and 2 are already proven. So assume a depth $n > 2$ and the result holds for $m < n$. Therefore by induction, we know that Merge($\mathcal{JS}^i$) is strongly equivalent to Compress($\mathcal{JS}^i$). Since compression gives an unnested system, we have that Compress($\mathcal{JS}^i$) = Compress(Compress($\mathcal{JS}^i$)). By Conjecture 7.2.2, we have that Compress($\mathcal{JS}$) and

$$\text{Compress}(\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\text{Merge}(\mathcal{JS}^1), \ldots, \text{Merge}(\mathcal{JS}^k)\}\rangle)$$

are strongly equivalent. The latter is strongly equivalent to

$$\text{Merge}(\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\text{Merge}(\mathcal{JS}^1), \ldots, \text{Merge}(\mathcal{JS}^k)\}\rangle)$$

by the result of depth two, which is equal to Merge($\mathcal{JS}$) by Lemma 6.3.3. □

A second possibility to prove Conjecture 7.2.1 is by proving graph-reducibility of merge.

Proposition 7.2.4. *If* Merge($\mathcal{JS}$) *is graph-reducible, then Conjecture 7.2.1 holds.*

Proof. In Fig. 6.1, the condition amount to the arrows with label 11, making a loop in the middle to get the following chain of inequalities:

$$\text{SV}^g_{\text{Merge}}(x) \leq_t \text{SV}^g_{\text{Compress}}(x) \leq_t \text{SV}^t_{\text{Compress}}(x) = \text{SV}^t_{\text{Merge}}(x) \leq_t \text{SV}^g_{\text{Merge}}(x).$$

Therefore, this collapses to all equalities, thus completing the proof. □

This brings us to the second missing result in Chapter 6: the consistency of compress and merge. The consistency of the compression is related to the complementarity of compress.

Proposition 7.2.5. *If* Compress($\mathcal{JS}$) *is complementary and the top-level branch evaluation is consistent, then* Compress($\mathcal{JS}$) *is consistent.*

By lack of counterexamples, we conjecture that Compress($\mathcal{JS}$) is complementary if all branch evaluations are consistent.

Conjecture 7.2.6. *Let $\mathcal{JS}$ be a nested system such that all branch evaluations are graph-like consistent, then* Compress($\mathcal{JS}$) *is complementary.*

For the consistency of merge, we can either try to prove it directly, or by using Conjecture 7.2.1 if it is true.

Proposition 7.2.7. *If* Compress($\mathcal{JS}$) *is complementary and Conjecture 7.2.1 holds, then* Merge($\mathcal{JS}$) *is graph-like consistent.*

For nested systems, we do not have the AFT connection from Chapter 5, apart from the approximator corresponding to the compression. Researching this further will allow to bring nestings to AFT. Bogaerts (2015) conjectured that fixpoint definitions correspond to the unique grounded fixpoint (or unique stable fixpoint) of some operator. The idea sketched is reminiscent of the merge system. Exploring this further might bring about nestings to AFT.

7.2.2 Applications of Justification Theory

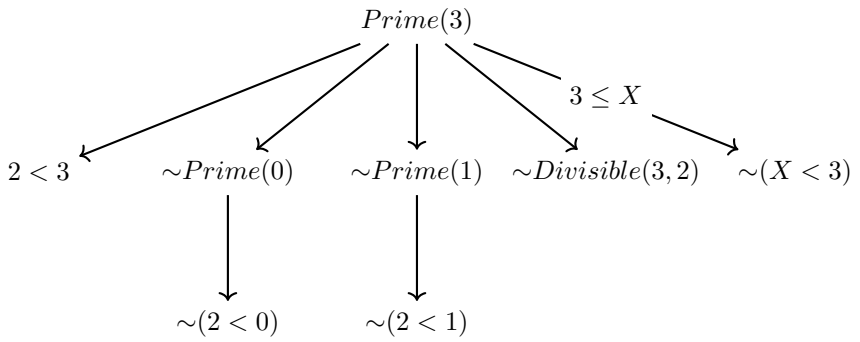
So far, justification theory has not as many applications covered as AFT, and a future challenge is to cover more applications. One way to do this is to exploit the connection between AFT and justification theory set about in Chapter 5. Two important fields captured by AFT, but not by justification theory are autoepistemic logic and default logic. Other applications include extensions of logic programming (Antic et al., 2013; Charalambidis et al., 2018; Pelov et al., 2007), description logics (Liu et al., 2016) and active integrity constraints (Bogaerts and Cruz-Filipe, 2018).

By extending the AFT connection in the other direction, we can construct a justification frame corresponding to an approximator (see discussion above). If such a construction is possible, most applications of AFT can be captured. We should start to investigate what justifications mean and what they can teach us in this setting. We might also investigate whether the automated way of translating an approximator in a justification frame corresponds to the "natural" justifications arising in these settings.

7.2.3 Extensions of Justification Theory

We have seen in Example 6.5.9 that the propositional nature of justification theory can be hampering to get succinct explanations from justifications. Perhaps, extending justification theory to a first-order system might alleviate this problem.

Example 7.2.8. The second justification from Example 6.5.9 can expressed as follows.



The rightmost edge has a condition $3 \leq X$ and it means that the fact $\sim(X < 3)$ is a child of $Prime(3)$ for each such X. ▲

By allowing first-order logic formulae with free variables as facts, we can extend justification theory. However, justifications now have arbitrary first-order logic conditions on the edges. For tree-like justifications, we can take the conjunction of the conditions of the edges leading up to the fact to resolve the actual set of facts. However, you still need to resolve possible unifications of the variables occurring in the conditions. For graph-like justifications, this becomes more difficult due the existence of loops. It is not clear yet how to resolve this problem. One line of research is expanding on this idea to bring true first-order logic to justification theory. A benefit of researching this type of justification theory is that redundant and duplicate information of a justification can be represented compactly. Related to this is the work of Bogaerts and Weinzierl (2018), where they construct justifications to learn new clauses. However, if these clauses are of a first-order nature, the learned clause could be more generally applicable and more succinctly represented. Some preliminary work by Bogaerts, Marynissen, and Weinzierl (2020) on completion formulas for non-ground programs might give insight on how to resolve unifications between the conditions in a first-order graph-like justification.

Bibliography

Christian Antic, Thomas Eiter, and Michael Fink. Hex semantics via approximation fixpoint theory. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2013. doi: 10.1007/978-3-642-40564-8_11. URL https://doi.org/10.1007/978-3-642-40564-8_11.

Grigoris Antoniou and Frank van Harmelen. Web ontology language: OWL. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 91–110. Springer, 2009. doi: 10.1007/978-3-540-92673-3_4. URL https://doi.org/10.1007/978-3-540-92673-3_4.

Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. Constraint answer set programming without grounding. *Theory and Practice of Logic Programming*, 18(3-4):337–354, 2018. doi: 10.1017/S1471068418000285. URL https://doi.org/10.1017/S1471068418000285.

Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. Justifications for goal-directed constraint answer set programming. In Francesco Ricca, Alessandra Russo, Sergio Greco, Nicola Leone, Alexander Artikis, Gerhard Friedrich, Paul Fodor, Angelika Kimmig, Francesca A. Lisi, Marco Maratea, Alessandra Mileo, and Fabrizio Riguzzi, editors, *Proceedings 36th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2020, (Technical Communications) UNICAL, Rende (CS), Italy, 18-24th September 2020*, volume 325 of *EPTCS*, pages 59–72, 2020. doi: 10.4204/EPTCS.325.12. URL https://doi.org/10.4204/EPTCS.325.12.

Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan,

editors, *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *Lecture Notes in Computer Science*, pages 228–248. Springer, 2005. doi: 10.1007/978-3-540-32254-2_14. URL https://doi.org/10.1007/978-3-540-32254-2_14.

Christopher Béatrix, Claire Lefèvre, Laurent Garcia, and Igor Stéphan. Justifications and blocking sets in a rule-based answer set computation. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, volume 52 of *OASICS*, pages 6:1–6:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/OASIcs.ICLP.2016.6. URL https://doi.org/10.4230/OASIcs.ICLP.2016.6.

Harald Beck, Thomas Eiter, and Christian Folie. Ticker: A system for incremental asp-based stream reasoning. *Theory and Practice of Logic Programming*, 17(5-6):744–763, 2017. doi: 10.1017/S1471068417000370. URL https://doi.org/10.1017/S1471068417000370.

Nuel D. Belnap. A useful four-valued logic. In J. Michael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. Reidel, Dordrecht, 1977. Invited papers from the Fifth International Symposium on Multiple-Valued Logic, held at Indiana University, Bloomington, Indiana, May 13-16, 1975.

Yi Bi, Jia-Huai You, and Zhiyong Feng. A generalization of approximation fixpoint theory and application. In Roman Kontchakov and Marie-Laure Mugnier, editors, *Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, volume 8741 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 2014. doi: 10.1007/978-3-319-11113-1_4. URL https://doi.org/10.1007/978-3-319-11113-1_4.

Howard A. Blair. Game characterizations of logic program properties. In V. Wiktor Marek and Anil Nerode, editors, *Logic Programming and Nonmonotonic Reasoning, Third International Conference, LPNMR'95, Lexington, KY, USA, June 26-28, 1995, Proceedings*, volume 928 of *Lecture Notes in Computer Science*, pages 99–112. Springer, 1995. doi: 10.1007/3-540-59487-6_8. URL https://doi.org/10.1007/3-540-59487-6_8.

Alexander Bochman. Here and there among logics for logic programming. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2012.

doi: 10.1007/978-3-642-30743-0_7. URL https://doi.org/10.1007/978-3-642-30743-0_7.

Bart Bogaerts. *Groundedness in logics with a fixpoint semantics*. PhD thesis, Department of Computer Science, KU Leuven, Jun 2015. Denecker, Marc (supervisor), Vennekens, Joost and Van den Bussche, Jan (cosupervisors).

Bart Bogaerts and Luís Cruz-Filipe. Fixpoint semantics for active integrity constraints. *Artificial Intelligence*, 255:43–70, 2018. doi: 10.1016/j.artint.2017.11.003. URL https://doi.org/10.1016/j.artint.2017.11.003.

Bart Bogaerts and Luís Cruz-Filipe. Stratification in approximation fixpoint theory and its application to active integrity constraints. *ACM Transactions on Computational Logic*, 22(1):6:1–6:19, 2021. doi: 10.1145/3430750. URL https://doi.org/10.1145/3430750.

Bart Bogaerts and Antonius Weinzierl. Exploiting justifications for lazy grounding of answer set programs. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1737–1745. ijcai.org, 2018. doi: 10.24963/ijcai.2018/240. URL https://doi.org/10.24963/ijcai.2018/240.

Bart Bogaerts, Joost Vennekens, and Marc Denecker. Grounded fixpoints and their applications in knowledge representation. *Artificial Intelligence*, 224: 51–71, 2015a. doi: 10.1016/j.artint.2015.03.006. URL https://doi.org/10.1016/j.artint.2015.03.006.

Bart Bogaerts, Joost Vennekens, and Marc Denecker. Partial grounded fixpoints. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2784–2790. AAAI Press, 2015b. URL http://ijcai.org/Abstract/15/394.

Bart Bogaerts, Simon Marynissen, and Antonius Weinzierl. Towards lightweight completion formulas for lazy grounding in answer set programming. In Maria V. Martínez and Ivan Varzinczak, editors, *18th International Workshop on Non-monotonic Reasoning NMR 2020 Workshop Notes*, pages 58–66, 2020. URL https://nmr2020.dc.uba.ar/WorkshopNotes.pdf.

Wilfried Buchholz, Solomon Feferman, Wolfram Pohlers, and Wilfried Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*, volume 897 of *Lecture Notes in Mathematics*. Springer, 1981.

Pedro Cabalar and Jorge Fandinno. Justifications for programs with disjunctive and causal-choice rules. *Theory and Practice of Logic Programming*, 16(5-6): 587–603, 2016. doi: 10.1017/S1471068416000454. URL https://doi.org/10.1017/S1471068416000454.

Pedro Cabalar and Jorge Fandinno. Enablers and inhibitors in causal justifications of logic programs. *Theory and Practice of Logic Programming*, 17(1):49–74, 2017. doi: 10.1017/S1471068416000107. URL https://doi.org/10.1017/S1471068416000107.

Pedro Cabalar and Paolo Ferraris. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming*, 7(6):745–759, 2007. doi: 10.1017/S1471068407003110. URL https://doi.org/10.1017/S1471068407003110.

Pedro Cabalar, Jorge Fandinno, and Michael Fink. Causal graph justifications of logic programs. *Theory and Practice of Logic Programming*, 14(4-5):603–618, 2014. doi: 10.1017/S1471068414000234. URL https://doi.org/10.1017/S1471068414000234.

Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Predicate logic as a modelling language: The IDP system. *CoRR*, abs/1401.6312, 2014. URL http://arxiv.org/abs/1401.6312.

Angelos Charalambidis, Panos Rondogiannis, and Ioanna Symeonidou. Approximation fixpoint theory and the well-founded semantics of higher-order logic programs. *Theory and Practice of Logic Programming*, 18(3-4): 421–437, 2018. doi: 10.1017/S1471068418000108. URL https://doi.org/10.1017/S1471068418000108.

Roberto Di Cosmo, Jean-Vincent Loddo, and Stephane Nicolet. A game semantics foundation for logic programming (extended abstract). In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *Principles of Declarative Programming, 10th International Symposium, PLILP'98 Held Jointly with the 7th International Conference, ALP'98, Pisa, Italy, September 16-18, 1998, Proceedings*, volume 1490 of *Lecture Notes in Computer Science*, pages 355–373. Springer, 1998. doi: 10.1007/BFb0056626. URL https://doi.org/10.1007/BFb0056626.

Patrick Cousot and Radhia Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43 – 57, 1979. doi: pjm/1102785059. URL https://doi.org/.

Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. Justifications for logic programming. In Pedro Cabalar and Tran Cao Son, editors, *Logic*

Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings, volume 8148 of *Lecture Notes in Computer Science*, pages 530–542. Springer, 2013. doi: 10.1007/978-3-642-40564-8_53. URL https://doi.org/10.1007/978-3-642-40564-8_53.

Adnan Darwiche. Human-level intelligence or animal-like abilities? *Communications of the ACM*, 61(10):56–67, 2018. doi: 10.1145/3271625. URL https://doi.org/10.1145/3271625.

Marc Denecker. *Knowledge representation and reasoning in incomplete logic programming.* PhD thesis, K.U. Leuven, Leuven, Belgium, September 1993.

Marc Denecker. The well-founded semantics is the principle of inductive definition. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *Logics in Artificial Intelligence, European Workshop, JELIA '98, Dagstuhl, Germany, October 12-15, 1998, Proceedings*, volume 1489 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998. doi: 10.1007/3-540-49545-2_1. URL https://doi.org/10.1007/3-540-49545-2_1.

Marc Denecker and Danny De Schreye. Justification semantics: A unifiying framework for the semantics of logic programs. In Luís Moniz Pereira and Anil Nerode, editors, *Logic Programming and Non-monotonic Reasoning, Proceedings of the Second International Workshop, Lisbon, Portugal, June 1993*, pages 365–379. MIT Press, 1993.

Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic*, 9(2):14:1–14:52, 2008. doi: 10.1145/1342991.1342998. URL https://doi.org/10.1145/1342991.1342998.

Marc Denecker and Joost Vennekens. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 84–96. Springer, 2007. doi: 10.1007/978-3-540-72200-7_9. URL https://doi.org/10.1007/978-3-540-72200-7_9.

Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 71–76. Springer, 2008. doi: 10.1007/978-3-540-89982-2_12. URL https://doi.org/10.1007/978-3-540-89982-2_12.

Marc Denecker and Joost Vennekens. The well-founded semantics is the principle of inductive definition, revisited. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7957.

Marc Denecker, Victor Marek, and Mirosław Truszczyński. *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning*, pages 127–144. Springer US, Boston, MA, 2000. ISBN 978-1-4615-1567-8. doi: 10.1007/978-1-4615-1567-8_6. URL https://doi.org/10.1007/978-1-4615-1567-8_6.

Marc Denecker, Maurice Bruynooghe, and V. Wiktor Marek. Logic programming revisited: Logic programs as inductive definitions. *ACM Transactions on Computational Logic*, 2(4):623–654, 2001. doi: 10.1145/383779.383789. URL https://doi.org/10.1145/383779.383789.

Marc Denecker, V. Wiktor Marek, and Miroslaw Truszczynski. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence*, 143 (1):79–122, 2003. doi: 10.1016/S0004-3702(02)00293-X. URL https://doi.org/10.1016/S0004-3702(02)00293-X.

Marc Denecker, Victor W. Marek, and Miroslaw Truszczynski. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, 192(1):84–121, 2004. doi: 10.1016/j.ic.2004.02.004. URL https://doi.org/10.1016/j.ic.2004.02.004.

Marc Denecker, Maurice Bruynooghe, and Joost Vennekens. Approximation fixpoint theory and the semantics of logic and answers set programs. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2012. doi: 10.1007/978-3-642-30743-0_13. URL https://doi.org/10.1007/978-3-642-30743-0_13.

Marc Denecker, Gerhard Brewka, and Hannes Strass. A formal theory of justifications. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, volume 9345 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2015. doi: 10.1007/978-3-319-23264-5_22. URL https://doi.org/10.1007/978-3-319-23264-5_22.

Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995. doi: 10.1016/0004-3702(94)00041-X. URL https://doi.org/10.1016/0004-3702(94)00041-X.

E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi: 10.1109/SFCS.1991.185392. URL https://doi.org/10.1109/SFCS.1991.185392.

Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011. doi: 10.1016/j.artint.2010.04.002. URL https://doi.org/10.1016/j.artint.2010.04.002.

François Fages. A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. In David H. D. Warren and Péter Szeredi, editors, *Logic Programming, Proceedings of the Seventh International Conference, Jerusalem, Israel, June 18-20, 1990*, pages 441–458. MIT Press, 1990.

Jorge Fandinno and Claudia Schulz. Answering the "why" in answer set programming - A survey of explanation approaches. *Theory Practice of Logic Programming*, 19(2):114–203, 2019. doi: 10.1017/S1471068418000534. URL https://doi.org/10.1017/S1471068418000534.

Melvin Fitting. A kripke-kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985. doi: 10.1016/S0743-1066(85)80005-4. URL https://doi.org/10.1016/S0743-1066(85)80005-4.

Melvin Fitting. Partial models and logic programming. *Theoretical Computer Science*, 48(3):229–255, 1986. doi: 10.1016/0304-3975(86)90096-4. URL https://doi.org/10.1016/0304-3975(86)90096-4.

Melvin Fitting. Kleene's logic, generalized. *Journal of Logic and Computation*, 1(6):797–810, 1991. doi: 10.1093/logcom/1.6.797. URL https://doi.org/10.1093/logcom/1.6.797.

Melvin Fitting. Tableaux for logic programming. *Journal of Automated Reasoning*, 13(2):175–188, 1994. doi: 10.1007/BF00881954. URL https://doi.org/10.1007/BF00881954.

Melvin Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1-2):25–51, 2002. doi: 10.1016/S0304-3975(00)00330-3. URL https://doi.org/10.1016/S0304-3975(00)00330-3.

Chrysida Galanaki, Panos Rondogiannis, and William W. Wadge. An infinite-game semantics for well-founded negation in logic programming. *Annals of Pure and Applied Logic*, 151(2-3):70–88, 2008. doi: 10.1016/j.apal.2007.10.004. URL https://doi.org/10.1016/j.apal.2007.10.004.

Chrysida Galanaki, Christos Nomikos, and Panos Rondogiannis. Game semantics for non-monotonic intensional logic programming. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 329–341. Springer, 2013. doi: 10.1007/978-3-642-40564-8_33. URL https://doi.org/10.1007/978-3-642-40564-8_33.

Chrysida Galanaki, Christos Nomikos, and Panos Rondogiannis. Game semantics for non-monotonic intensional logic programming. *Annals of Pure and Applied Logic*, 168(2):234–253, 2017. doi: 10.1016/j.apal.2016.10.005. URL https://doi.org/10.1016/j.apal.2016.10.005.

David Gale and F. M. Stewart. *Infinite Games with Perfect Information*, volume 2, pages 245–266. Princeton University Press, 1953. doi: doi:10.1515/9781400881970-014. URL https://doi.org/10.1515/9781400881970-014.

Robin O. Gandy. Inductive definitions. In J.E. Fenstad and P.G. Hinman, editors, *Generalized Recursion Theory*, volume 79 of *Studies in Logic and the Foundations of Mathematics*, pages 265–299. Elsevier, 1974. doi: https://doi.org/10.1016/S0049-237X(08)70591-3. URL https://www.sciencedirect.com/science/article/pii/S0049237X08705913.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. On the implementation of weight constraint rules in conflict-driven ASP solvers. In Patricia M. Hill and David Scott Warren, editors, *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, volume 5649 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2009. doi: 10.1007/978-3-642-02846-5_23. URL https://doi.org/10.1007/978-3-642-02846-5_23.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

Michael Gelfond and Yuanlin Zhang. Vicious circle principle, aggregates, and formation of sets in ASP based languages. *Artificial Intelligence*, 275:28–77, 2019. doi: 10.1016/j.artint.2019.04.004. URL https://doi.org/10.1016/j.artint.2019.04.004.

Hugo Gimbert and Wieslaw Zielonka. When can you play positionnaly? In I. Fiala, V. Koubek, and J. Kratochvil, editors, *Mathematical Foundations of Computer Science 2004*, Lecture Notes in Comp. Sci. 3153, pages 686–697, Prague, Czech Republic, 2004. Springer. URL https://hal.archives-ouvertes.fr/hal-00160436.

Hugo Gimbert and Wieslaw Zielonka. Games where you can play optimally without any memory. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi: 10.1007/11539452_33. URL https://doi.org/10.1007/11539452_33.

Kurt Gödel. Zum intuitionistischen aussagenkalkül. *Anzeiger der Akademie der Wissenschaften in Wien*, 69:65–66, 1932.

Gopal Gupta, Ajay Bansal, Richard Min, Luke Simon, and Ajay Mallya. Coinductive logic programming and its applications. In Verónica Dahl and Ilkka Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007, Proceedings*, volume 4670 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2007. doi: 10.1007/978-3-540-74610-2_4. URL https://doi.org/10.1007/978-3-540-74610-2_4.

Ping Hou and Marc Denecker. A logic of fixpoint definitions. In Wolfgang Faber and Joohyung Lee, editors, *Second Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*, pages 1–15, 2009.

Ping Hou, Broes De Cat, and Marc Denecker. FO(FD): extending classical logic with rule-based fixpoint definitions. *Theory and Practice of Logic Programming*, 10(4-6):581–596, 2010. doi: 10.1017/S1471068410000293. URL https://doi.org/10.1017/S1471068410000293.

Matti Järvisalo, Tommi A. Junttila, and Ilkka Niemelä. Justification-based non-clausal local search for SAT. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 535–539. IOS Press, 2008. doi: 10.3233/978-1-58603-891-5-535. URL https://doi.org/10.3233/978-1-58603-891-5-535.

Andrew J. I. Jones. Deontic logic and legal knowledge representation*. *Ratio Juris*, 3(2):237–244, 1990. doi: 10.1111/j.1467-9337.1990.tb00060.x. URL https://doi.org/10.1111/j.1467-9337.1990.tb00060.x.

Charanjit S. Jutla. Determinization and memoryless winning strategies. *Information and Computation*, 133(2):117–134, 1997. doi: 10.1006/inco.1997.2624. URL https://doi.org/10.1006/inco.1997.2624.

Alexander S. Kechris, Benedikt Löwe, and John R. Steel, editors. *Wadge degrees and projective ordinals. The Cabal Seminar. Volume II*, volume 37 of *Lecture Notes in Logic*. Association for Symbolic Logic, La Jolla, CA; Cambridge University Press, Cambridge, 2012. ISBN 978-0-521-76203-8.

Stephen Cole Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

Ruben Lapauw, Maurice Bruynooghe, and Marc Denecker. Improving parity game solvers with justifications. In Dirk Beyer and Damien Zufferey, editors, *Verification, Model Checking, and Abstract Interpretation - 21st International Conference, VMCAI 2020, New Orleans, LA, USA, January 16-21, 2020, Proceedings*, volume 11990 of *Lecture Notes in Computer Science*, pages 449–470. Springer, 2020. doi: 10.1007/978-3-030-39322-9_21. URL https://doi.org/10.1007/978-3-030-39322-9_21.

Ruben Lapauw, Maurice Bruynooghe, and Marc Denecker. Justifications and a reconstruction of parity game solving algorithms. *CoRR*, abs/2102.01440, 2021. URL https://arxiv.org/abs/2102.01440.

Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia. Asperix, a first-order forward chaining approach for answer set computing. *Theory and Practice of Logic Programming*, 17(3):266–310, 2017. doi: 10.1017/S1471068416000569. URL https://doi.org/10.1017/S1471068416000569.

Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999. doi: 10.1023/A:1018978005636. URL https://doi.org/10.1023/A:1018978005636.

Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001. doi: 10.1145/383779.383783. URL https://doi.org/10.1145/383779.383783.

Fangfang Liu, Yi Bi, Md. Solimul Chowdhury, Jia-Huai You, and Zhiyong Feng. Flexible approximators for approximating fixpoint theory. In Richard Khoury and Christopher Drummond, editors, *Advances in Artificial Intelligence - 29th Canadian Conference on Artificial Intelligence, Canadian AI 2016, Victoria, BC, Canada, May 31 - June 3, 2016. Proceedings*, volume 9673 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 2016. doi: 10.1007/978-3-319-34111-8_28. URL https://doi.org/10.1007/978-3-319-34111-8_28.

Lengning Liu, Enrico Pontelli, Tran Cao Son, and Miroslaw Truszczynski. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence*, 174(3-4):295–315, 2010. doi: 10.1016/j.artint.2009.11.016. URL https://doi.org/10.1016/j.artint.2009.11.016.

John W. Lloyd. Declarative error diagnosis. *New Generation Computing*, 5(2):133–154, 1987. ISSN 1882-7055. doi: 10.1007/BF03037396. URL https://doi.org/10.1007/BF03037396.

Jean-Vincent Loddo and Roberto Di Cosmo. Playing logic programs with the alpha-beta algorithm. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, volume 1955 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2000. doi: 10.1007/3-540-44404-1_14. URL https://doi.org/10.1007/3-540-44404-1_14.

V. Wiktor Marek and Jeffrey B. Remmel. Set constraints in logic programming. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2004. doi: 10.1007/978-3-540-24609-1_16. URL https://doi.org/10.1007/978-3-540-24609-1_16.

Maarten Mariën. *Model Generation for ID-Logic*. PhD thesis, Department of Computer Science, KU Leuven, Belgium, February 2009.

Maarten Mariën, Rudradeb Mitra, Marc Denecker, and Maurice Bruynooghe. Satisfiability checking for PC(ID). In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*, pages 565–579. Springer, 2005. doi: 10.1007/11591191_39. URL https://doi.org/10.1007/11591191_39.

Maarten Mariën, Johan Wittocx, and Marc Denecker. Integrating inductive definitions in SAT. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pages 378–392. Springer, 2007. doi: 10.1007/978-3-540-75560-9_28. URL https://doi.org/10.1007/978-3-540-75560-9_28.

Maarten Mariën, Johan Wittocx, Marc Denecker, and Maurice Bruynooghe. SAT(ID): satisfiability of propositional logic extended with inductive definitions. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2008. doi: 10.1007/978-3-540-79719-7_20. URL https://doi.org/10.1007/978-3-540-79719-7_20.

Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. ISSN 0003486X. URL http://www.jstor.org/stable/1971035.

Donald A. Martin. A purely inductive proof of Borel determinacy. In *Recursion theory (Ithaca, N.Y., 1982)*, volume 42 of *Proceedings of Symposia in Pure Mathematics*, pages 303–308. Amer. Math. Soc., Providence, RI, 1985. doi: 10.1090/pspum/042/791065. URL https://doi-org/10.1090/pspum/042/791065.

Simon Marynissen, Niko Passchyn, Bart Bogaerts, and Marc Denecker. Consistency in justification theory. In *13th Workshop on Logic and Semantic Frameworks with Applications LSFA 2018, Fortaleza, Brazil, Sept 26-28, 2018*, pages 173–188, 2018a. URL https://lia.ufc.br/~lsfa2018/wp-content/uploads/2018/09/LSFA18.pdf.

Simon Marynissen, Niko Passchyn, Bart Bogaerts, and Marc Denecker. Consistency in justification theory. In *Proceedings of 17th International Workshop on Non-Monotonic Reasoning (NMR 2018), Tempe, Arizona, USA, Oct. 27-29, 2018*, pages 41–52. AAAI Press 2018, 2018b. URL http://www4.uma.pt/nmr2018/NMR2018Proceedings.pdf.

Simon Marynissen, Bart Bogaerts, and Marc Denecker. Exploiting game theory for analysing justifications. *Theory and Practice of Logic Programming*, 20(6): 880–894, 2020. doi: 10.1017/S1471068420000186. URL https://doi.org/10.1017/S1471068420000186.

Simon Marynissen, Bart Bogaerts, and Marc Denecker. On the relation between approximation fixpoint theory and justification theory. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1973–1980. ijcai.org, 2021. doi: 10.24963/ijcai.2021/272. URL https://doi.org/10.24963/ijcai.2021/272.

Richard Min and Gopal Gupta. Coinductive logic programming with negation. In Danny De Schreye, editor, *Logic-Based Program Synthesis and Transformation, 19th International Symposium, LOPSTR 2009, Coimbra,*

Portugal, September 2009, Revised Selected Papers, volume 6037 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2009. doi: 10.1007/978-3-642-12592-8_8. URL https://doi.org/10.1007/978-3-642-12592-8_8.

Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985. doi: 10.1016/0004-3702(85)90042-6. URL https://doi.org/10.1016/0004-3702(85)90042-6.

David Paulius and Yu Sun. A survey of knowledge representation in service robotics. *Robotics and Autonomous Systems*, 118:13–30, 2019. doi: 10.1016/j.robot.2019.03.005. URL https://doi.org/10.1016/j.robot.2019.03.005.

Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming*, 7(3):301–353, 2007. doi: 10.1017/S1471068406002973. URL https://doi.org/10.1017/S1471068406002973.

Luís Moniz Pereira, José Júlio Alferes, and Joaquim Nunes Aparício. Contradiction removal semantics with explicit negation. In Michael Masuch and László Pólos, editors, *Knowledge Representation and Reasoning Under Uncertainty, Logic at Work [International Conference Logic at Work, Amsterdam, The Netherlands, December 17-19, 1992]*, volume 808 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 1992. doi: 10.1007/3-540-58095-6_7. URL https://doi.org/10.1007/3-540-58095-6_7.

Luís Moniz Pereira, Joaquim Nunes Aparício, and José Júlio Alferes. Non-monotonic reasoning with logic programming. *Journal of Logic Programming*, 17(2/3&4):227–263, 1993. doi: 10.1016/0743-1066(93)90032-C. URL https://doi.org/10.1016/0743-1066(93)90032-C.

Enrico Pontelli and Tran Cao Son. *Justifications* for logic programs under answer set semantics. In Sandro Etalle and Miroslaw Truszczynski, editors, *Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4079 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2006. doi: 10.1007/11799573_16. URL https://doi.org/10.1007/11799573_16.

Teodor C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990.

Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980. doi: 10.1016/0004-3702(80)90014-4. URL https://doi.org/10.1016/0004-3702(80)90014-4.

Stéphane Le Roux. Memoryless determinacy of infinite parity games: Another simple proof. *Information Processing Letters*, 143:8–13, 2019. doi: 10.1016/j.ipl.2018.10.015. URL https://doi.org/10.1016/j.ipl.2018.10.015.

Abhik Roychoudhury, C. R. Ramakrishnan, and I. V. Ramakrishnan. Justifying proofs using memo tables. In Maurizio Gabbrielli and Frank Pfenning, editors, *Proceedings of the 2nd international ACM SIGPLAN conference on on Principles and practice of declarative programming, Montreal, Canada, September 20-23, 2000*, pages 178–189. ACM, 2000. doi: 10.1145/351268.351290. URL https://doi.org/10.1145/351268.351290.

Claudia Schulz and Francesca Toni. Justifying answer sets using argumentation. *Theory and Practice of Logic Programming*, 16(1):59–110, 2016. doi: 10.1017/S1471068414000702. URL https://doi.org/10.1017/S1471068414000702.

Ehud Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, USA, 1983. ISBN 0262192187.

Robert I. Soare. *Gale-Stewart Games*, pages 217–219. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. ISBN 978-3-642-31933-4. doi: 10.1007/978-3-642-31933-4_15. URL https://doi.org/10.1007/978-3-642-31933-4_15.

Tran Cao Son, Enrico Pontelli, and Phan Huy Tu. Answer sets for logic programs with arbitrary abstract constraint atoms. *Journal of Artificial Intelligence Research*, 29:353–389, 2007. doi: 10.1613/jair.2171. URL https://doi.org/10.1613/jair.2171.

Leon Sterling and Marucha Lalee. An explanation shell for expert systems. *Computational Intelligence*, 2:136–141, 1986. doi: 10.1111/j.1467-8640.1986.tb00079.x. URL https://doi.org/10.1111/j.1467-8640.1986.tb00079.x.

Leon Sterling and L. Ümit Yalçinalp. Explaining prolog based expert systems using a layered meta-interpreter. In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 66–71. Morgan Kaufmann, 1989. URL http://ijcai.org/Proceedings/89-1/Papers/011.pdf.

Hannes Strass. Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence*, 205:39–70, 2013. doi: 10.1016/j.artint.2013.09.004. URL https://doi.org/10.1016/j.artint.2013.09.004.

Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285 – 309, 1955.

Thanos Tsouanas. A game semantics for disjunctive logic programming. *Annals of Pure and Applied Logic*, 164(11):1144–1175, 2013. doi: 10.1016/j.apal.2013.05.008. URL https://doi.org/10.1016/j.apal.2013.05.008.

Maarten H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986. doi: 10.1016/0743-1066(86)90003-8. URL https://doi.org/10.1016/0743-1066(86)90003-8.

Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976. doi: 10.1145/321978.321991. URL https://doi.org/10.1145/321978.321991.

Bas C. van Fraassen. Singular terms, truth-value gaps and free logic. *Journal of Philosophy*, 63(17):481–495, 1966. ISSN 0022362X. URL http://www.jstor.org/stable/2024549.

Joost Vennekens, David Gilis, and Marc Denecker. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic*, 7(4):765–797, 2006. doi: 10.1145/1183278.1183284. URL https://doi.org/10.1145/1183278.1183284.

Joost Vennekens, David Gilis, and Marc Denecker. Erratum to splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic*, 8(1):7, 2007a. doi: 10.1145/1182613.1189735. URL https://doi.org/10.1145/1182613.1189735.

Joost Vennekens, Johan Wittocx, Maarten Mariën, and Marc Denecker. Predicate introduction for logics with a fixpoint semantics. part I: logic programming. *Fundamenta Informaticae*, 79(1-2):187–208, 2007b. URL http://content.iospress.com/articles/fundamenta-informaticae/fi79-1-2-09.

Marina De Vos and Dirk Vermeir. Choice logic programs and nash equilibria in strategic games. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 266–276. Springer, 1999. doi: 10.1007/3-540-48168-0_19. URL https://doi.org/10.1007/3-540-48168-0_19.

Marina De Vos and Dirk Vermeir. Logic programming agents and game theory. In Alessandro Provetti and Tran Cao Son, editors, *Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning, Proceedings of the 1st Intl. ASP'01 Workshop, Stanford, CA, USA, March 26-28, 2001*, 2001. URL http://www.cs.nmsu.edu/%7Etson/ASP2001/8.ps.

Jia-Huai You and Li-Yan Yuan. Three-valued formalization of logic programming: Is it needed? In Daniel J. Rosenkrantz and Yehoshua Sagiv, editors, *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, pages 172–182. ACM Press, 1990. doi: 10.1145/298514.298559. URL https://doi.org/10.1145/298514.298559.

Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. doi: 10.1016/S0304-3975(98)00009-7. URL https://doi.org/10.1016/S0304-3975(98)00009-7.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
https://dtai.cs.kuleuven.be/