On Nested Justification Systems

Marynissen, Simon; Heyninck, Jesse; Bogaerts, Bart; Denecker, Marc

Published in:
Theory and Practice of Logic Programming

DOI:
10.1017/S1471068422000266

Publication date:
2022

License:
CC BY

Document Version:
Accepted author manuscript

TPLP: Page 1–32.
doi:10.1017/xxxxx

On Nested Justification Systems[†]

MARYNISSEN SIMON[1]
KU Leuven
Vrije Universiteit Brussel

HEYNINCK JESSE
Vrije Universiteit Brussel
University of Cape Town and CAIR, South-Africa

BOGAERTS BART
Vrije Universiteit Brussel

DENECKER MARC
KU Leuven

Abstract

Justification theory is a general framework for the definition of semantics of rule-based languages that has a high explanatory potential. Nested justification systems, first introduced by Denecker et al. (2015), allow for the composition of justification systems. This notion of nesting thus enables the modular definition of semantics of rule-based languages, and increases the representational capacities of justification theory. As we show in this paper, the original characterization of semantics for nested justification systems leads to the loss of information relevant for explanations. In view of this problem, we provide an alternative characterization of their semantics and show that it is equivalent to the original one. Furthermore, we show how nested justification systems allow representing fixpoint definitions.

KEYWORDS: justification, modular, knowledge representation

1 Introduction

Justification theory (Denecker et al. 2015) is a general framework for the definition of semantics of rule-based languages that allows to design languages with high explanatory potential, as the justification-based semantics give an immediate explanation of the truth-value of a fact. In more detail, a justification is a tree of facts, where children of a given node occur in the body of a rule with this parent node as head. So-called *branch evaluations* specify how to evaluate justifications.

Justification theory is not just useful for defining new logics, but also for *unifying* existing ones: it captures many different KR-languages, including logic programming and

[1] This work is part of the PhD thesis of the first author (Marynissen 2022).

[†] This research was supported by the Flemish Government in the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme and by the FWO Flanders project G0B2221N.

abstract argumentation, and their various semantics. By establishing a precise correspondence between semantics of different logics, justification theory sheds light on the common semantic mechanisms underlying these different logics.

Denecker et al. (2015) also introduced *nested justification systems*, which allow justification systems to be composed by *nesting* them. Essentially, a nested justification system can be seen as a tree of justification systems, where the subsystems provide subdefinitions for their parent systems. Such nested systems have several benefits. Firstly, they allow for the modular definition of (the semantics of) rule-based languages. For instance, the problem of defining a suitable semantics for logic programs with aggregates is notoriously difficult (witnessed, e.g., by the lack of consensus on semantics for logic programs with aggregates (Pelov et al. 2007; Faber et al. 2011; Gelfond and Zhang 2019; Alviano et al. 2021; Vanbesien et al. 2021)). Nested justifications allow to separate distinct concerns in logic programs with aggregates: on the one hand, we can define what is a justification for an aggregate expression and when it is "acceptable" (branch evaluation for aggregate expressions). On the other hand, we can define a branch evaluation (e.g., stable or well-founded) for rules. By nesting the justification system for aggregate expressions inside the justification system for programs, we then obtain a semantics for logic programs with aggregates without any additional effort required. Thus, the modular semantics of nested justification systems allow for the definition of complex KR-languages using a "divide and conquer"-methodology. Secondly, as different branch evaluations can be used in different sub-systems, nested justification systems allow for the well-behaved combination of different semantics. This can be useful when modelling the combination of knowledge from different agents that use a different semantics to interpret their respective knowledge bases. Finally, nested justification systems allow to capture a richer class of logics than non-nested systems (e.g., fixpoint definitions, as we will show in this paper), and nesting thus increases the unifying power of justification theory.

Denecker et al. (2015) defined the semantics of nested justification systems by means of an operation called *compression*, which turns an entire justification of the subsystem into a set of facts to be pasted into the supersystem. However, properties of this characterisation of their semantics were never studied. Furthermore, as we argue in this paper, the compression-based characterisation of these semantics lead to the loss of explanatory potential, as information essential for explanations, such as the original rules, is lost. Therefore, in this work, we give an alternative characterisation of the semantics of nested justification systems in terms of a so-called *merging* operator. Merging retains the original rules in the evaluation of a nested justification system, and therefore brings the explanatory potential of justification theory to nested systems.

The contributions of this paper are as follows. (1) An expanded exposition and semantic study of *nested justification systems* and the *compression-based characterisation* of their semantics. (2) The introduction of the *merging-based characterisation* of their semantics. (3) A proof of *equivalence* between compression and merging. (4) An application of nested justification systems to the representation of *fixpoint definitions* (Hou et al. 2010).

The merging-based characterisation brings the explanatory potential of standard justification theory to *nested* justification systems. This is promising not just for nested fixpoint definitions, which we study in the current paper, but also for other applications of nesting, such as the modular definition of new language constructs. Furthermore, due

to the general nature of justification theory, as well as of our characterisations and results, our results also apply to any future branch evaluations.

Outline of the paper: The rest of this paper is structured as follows: necessary preliminaries on justification theory are given in Section 2. In Section 3, nested justification systems are defined. The first characterisation of semantics for nested justification systems, compression-based characterisation, is recalled and studied in Section 4. In Section 5, the merging-based characterisation is introduced. These two characterisations are shown equivalent in Section 6. Nested justification systems are shown to capture fixpoint definitions in Section 7. The paper is concluded in view of related work in Section 8.

2 Preliminaries

We use the formalisation of justification theory of Marynissen et al. (2020). We first give and explain all necessary definitions, and afterwards illustrate them with an example.

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

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

- $\mathcal{F}_d$ is a subset of $\mathcal{F} \setminus \mathcal{L}$ closed under $\sim$, i.e., $\sim\mathcal{F}_d = \mathcal{F}_d$; facts in $\mathcal{F}_d$ are called *defined*;[1]
- $R \subseteq \mathcal{F}_d \times 2^{\mathcal{F}}$;
- for each $x \in \mathcal{F}_d$, $(x, \emptyset) \notin R$ and there is an element $(x, A) \in R$ for $\emptyset \neq A \subseteq \mathcal{F}$.

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

In justification theory, defined facts are evaluated by constructing *justifications* for them. Justifications are directed graphs, where the set truth of the (labels of the) children of a node forms a reason (or argument, or cause, depending othe context) for the truth of the (label of the) node itself. Such reasons are not necessarily convincing: for example if they are based on a parameter that is not true, or might lead to cyclic argumentation. Therefore, branches in justification trees are evaluated using *branch evaluations*.

Definition 2.2

[1] Thus, no logical fact is defined, or, equivalently, the logical facts are opens.

Let $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R \rangle$ be a justification frame. A *justification* J in $\mathcal{JF}$ is a (possibly infinite) directed labelled graph $(N, \mathcal{F}_d, E, \ell)$, where N are the nodes, E the vertices, and $\ell : N \to \mathcal{F}_d$ is a labelling function, such that (1) the underlying undirected graph is a forest, i.e., is acyclic; and (2) for every internal node $n \in N$ it holds that $\ell(n) \leftarrow \{\ell(m) \mid (n, m) \in E\} \in R$.

We write $\mathfrak{J}(x)$ for the set of justifications that have a node labelled x. A justification is *locally complete* if it has no leaves with label in $\mathcal{F}_d$. We call $x \in \mathcal{F}_d$ a *root* of a justification J if there is a node n labelled x such that every node is reachable from n in J.

Remark 2.3
In some works, justifications are formalized as *graphs* and the justifications as defined here are then called *tree-like justifications* (Marynissen et al. 2020). Since we restrict attention to the latter, we shall just use the term *justification*.

Definition 2.4
Let $\mathcal{JF}$ be a justification frame. A *$\mathcal{JF}$-branch* is either an infinite sequence in $\mathcal{F}_d$ or a finite non-empty sequence in $\mathcal{F}_d$ followed by an element in $\mathcal{F}_o$. For a justification J in $\mathcal{JF}$, a *J-branch* starting in $x \in \mathcal{F}_d$ is a path in J starting in x that is either infinite or ends in a leaf of J. We write $B_J(x)$ to denote the set of J-branches starting in x.

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

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

Definition 2.5
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 frame $\mathcal{JF}$ together with a branch evaluation $\mathcal{B}$ form a *justification system* $\mathcal{JS}$, which is presented as a quadruple $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$.

A branch evaluation is *parametric* if every branch is mapped to an open fact. A justification system $\langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ is *parametric* if $\mathcal{B}$ is parametric.

The *supported* (completion) branch evaluation $\mathcal{B}_{\text{sp}}$ maps $x_0 \to x_1 \to \cdots$ to x_1. The *Kripke-Kleene* branch evaluation $\mathcal{B}_{\text{KK}}$ maps finite branches to their last element and infinite branches to $\mathbf{u}$. Let $\mathcal{JF}$ be a justification frame. A *sign function* on $\mathcal{JF}$ is a map $\text{sgn} : \mathcal{F}_d \to \{-, +\}$ such that $\text{sgn}(x) \neq \text{sgn}(\sim x)$ for all $x \in \mathcal{F}_d$. We denote $\mathcal{F}_- := \text{sgn}^{-1}(\{-\})$ and $\mathcal{F}_+ := \text{sgn}^{-1}(\{+\})$. From now on, we fix a sign function on $\mathcal{JF}$. We say that an infinite branch has a positive (respectively negative) tail if from some point onwards all elements are in $\mathcal{F}_+$ (respectively $\mathcal{F}_-$). The *well-founded* branch evaluation $\mathcal{B}_{\text{wf}}$ maps finite branches to their last element. It maps infinite branches to $\mathbf{t}$ if they have a negative tail, to $\mathbf{f}$ if they have a positive tail and to $\mathbf{u}$ otherwise. The *co-well-founded branch evaluation* $\mathcal{B}_{\text{cwf}}$ maps finite branches to their last element, infinite branches with a positive tail to $\mathbf{t}$, infinite branches with a negative tail to $\mathbf{f}$, and all other infinite branches to $\mathbf{u}$. The *stable* (answer set) branch evaluation $\mathcal{B}_{\text{st}}$ maps a branch $x_0 \to x_1 \to \cdots$ to the first element with a different sign than x_0 if it exists; otherwise $\mathbf{b}$ is mapped to $\mathcal{B}_{\text{wf}}(\mathbf{b})$.

The final ingredient of the semantics of justification systems are *interpretations*, which are abstractions of possible states of affairs, formalized as an assignment of a truth value to each fact. A *(three-valued) interpretation* of $\mathcal{F}$ is a function $\mathcal{I} : \mathcal{F} \to \mathcal{L}$ such that

$\mathcal{I}(\sim x) = \sim\mathcal{I}(x)$ and $\mathcal{I}(l) = l$ for all $l \in \mathcal{L}$. The set of interpretations of $\mathcal{F}$ is denoted by $\mathfrak{I}_\mathcal{F}$. We will call an interpretation *two-valued* if $\mathcal{I}(x) \neq \mathbf{u}$ for all $x \neq \mathbf{u}$. Given an interpretation $\mathcal{I}$ and a justification system $\mathcal{JS}$, we can now evaluate the quality of justifications: the value assigned to a justification will be the least (in the truth order) value of its branches. The rationale behind this definition is that for a justification to be "good", all of the arguments it contains should be good as well. On top of this definition, we will define a notion of supported value of a fact, which is the value of its best justification. Indeed, in general we will not be interested in the existence of *bad* arguments why a fact holds, but only in the existence of its best arguments.

Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B}\rangle$ be a justification system, $\mathcal{I}$ an interpretation of $\mathcal{F}$, and J a locally complete justification in $\mathcal{JS}$. Let $x \in \mathcal{F}_d$ be a label of a node in J. The *value* of $x \in \mathcal{F}_d$ by J under I is defined as $\mathrm{val}(J, x, \mathcal{I}) = \min_{\mathbf{b}\in B_J(x)} \mathcal{I}(\mathcal{B}(\mathbf{b}))$, where the minimum is taken with respect to $\leq_t$. The *supported value* of $x \in \mathcal{F}$ in $\mathcal{JS}$ under $\mathcal{I}$ is defined as

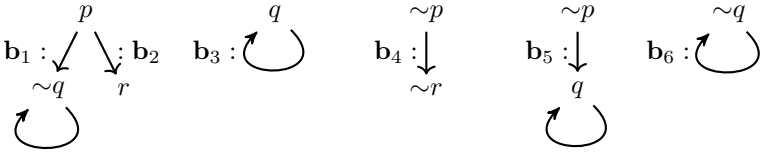
$$\mathrm{SV}(x, \mathcal{I}) = \max_{J\in\mathfrak{J}(x)} \mathrm{val}(J, x, \mathcal{I}) \text{ for } x \in \mathcal{F}_d \qquad \mathrm{SV}(x, \mathcal{I}) = \mathcal{I}(x) \text{ for } x \in \mathcal{F}_o.$$

Definition 2.6
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 $\mathcal{JF}$.

Definition 2.7
Two justification systems $\mathcal{JS}_1$ and $\mathcal{JS}_2$ are *equivalent* if $\mathrm{SV}_{\mathcal{JS}_1} = \mathrm{SV}_{\mathcal{JS}_2}$.

Example 2.8
Consider the justification system $\langle \mathcal{F}, \mathcal{F}_d, R\rangle$ with $\mathcal{F}_d = \{p, \sim p, q, \sim q\}$, $\mathcal{F} = \mathcal{F}_d \cup \{r, \sim r\} \cup \mathcal{L}$ and $R = \{p \leftarrow \sim q, r; q \leftarrow q; \sim p \leftarrow q; \sim p \leftarrow \sim r; \sim q \leftarrow \sim q\}$.

We have the following $\mathcal{JF}$-branches (we denote branches compactly in a graph-like fashion, i.e. a loop like $\mathbf{b}_3$ denotes the infinite branch $q \to q \to \ldots$):



These branches are evaluated as follows:

i	1	2	3	4	5	6
$\mathcal{B}_{\mathrm{sp}}(\mathbf{b}_i)$	$\sim q$	r	q	$\sim r$	q	$\sim q$
$\mathcal{B}_{\mathrm{wf}}(\mathbf{b}_i)$	$\mathbf{t}$	r	$\mathbf{f}$	$\sim r$	$\mathbf{f}$	$\mathbf{t}$
$\mathcal{B}_{\mathrm{cwf}}(\mathbf{b}_i)$	$\mathbf{f}$	r	$\mathbf{t}$	$\sim r$	$\mathbf{t}$	$\mathbf{f}$

i	1	2	3	4	5	6
$\mathcal{B}_{\mathrm{KK}}(\mathbf{b}_i)$	$\mathbf{u}$	r	$\mathbf{u}$	$\sim r$	$\mathbf{u}$	$\mathbf{u}$
$\mathcal{B}_{\mathrm{st}}(\mathbf{b}_i)$	$\sim q$	r	$\mathbf{f}$	$\sim r$	q	$\mathbf{t}$

Let $\mathcal{I}$ be the interpretation with $\mathcal{I}(r) = \mathcal{I}(p) = \mathcal{I}(\sim q) = \mathbf{t}$, J the justification for p made up of $\mathbf{b}_1$ and $\mathbf{b}_2$, and $\mathcal{JF} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B}_{\mathrm{wf}}\rangle$. We see that, e.g., $\mathrm{val}(J, p, \mathcal{I}) = \min_{\mathbf{b}\in B_J(p)} \mathcal{I}(\mathcal{B}_{\mathrm{wf}}(\mathbf{b})) = \mathbf{t}$, and thus that $\mathrm{SV}(p, \mathcal{I}) = \mathcal{I}(p)$. In fact, it can be verified that this holds for every literal, i.e., $\mathcal{I}$ is a $\mathcal{JF}$-model. In $\mathcal{I}$ (still, under the well-founded semantics), J serves as an explanation as to why p is true: because $\sim q$ and r are true. Simultaneously, J also explains why the facts r and $\sim q$ are true. In the case of r, this is

simply because it is an open fact (with value true). For $\sim q$, this explanation looks self-supporting: the reason why $\sim q$ holds is because $\sim q$ holds. For negative facts, stable and well-founded semantics accept such cyclic branches: the reason is that this cycle actually represents the fact that the positive fact (q) can only be justified by cyclic justifications, and well-founded and stable semantics reject cyclic justifications for positive facts.

3 Nested Justification Systems

As outlined in the introduction, nested justifications, originally introduced by Denecker et al. (2015), allow for the modular definition of semantics, the representation of richer classes of logics and the combination of different semantics in different modules. In this section, we introduce nested justification systems formally.

A nested justification system is essentially a tree structure of justification systems, meaning, that some systems are local to certain others, i.e. some facts occurring as an open fact in one component system are defined in another component system.

Definition 3.1
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 justification system;
2. for each i, $\mathcal{JS}^i$ is a nested justification system $\langle \mathcal{F}^i, \mathcal{F}^i_d, \mathcal{F}^i_{dl}, R^i, \mathcal{B}^i, \ldots\rangle$;
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. $\mathcal{F}^i_o \subseteq \mathcal{F}_o \cup \mathcal{F}_{dl}$ (where $\mathcal{F}^i_o = \mathcal{F}^i \setminus \mathcal{F}^i_d$ as usual)

A nested justification system is called *parametric* if $\mathcal{B}$ is parametric and all of its subsystems are parametric. We call a nested justification system *compressible* if for each i, $\mathcal{JS}^i$ is parametric.

Given $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\}\rangle$, we call (for $i = 1, \ldots, k$) $\mathcal{JS}$ the *parent system of* $\mathcal{JS}_i$ and $\mathcal{JS}_i$ a *child system of* $\mathcal{JS}$. Ancestor and descendant systems are defined analogously by transitively closing the parent, respectively, child relation. This defines a tree of nested justification systems, where the leaves have $\mathcal{F}_{dl} = \mathcal{F}_d$ and $k = 0$, and thus correspond directly to an unnested justification system.

The factspace $\mathcal{F}$ consists of all the facts used in (some component system of) $\mathcal{JS}$. The facts in $\mathcal{F}_{dl}$ are those that are defined *locally* in the justification system, i.e., in the rules R. The facts in $\mathcal{F}_d$ are those that are either defined locally, or in some component system of $\mathcal{JS}$. Every defined fact is either defined locally in the top system ($\mathcal{F}_{dl}$), or in one of the subsystems ($\mathcal{F}^i_d$). Each child system $\mathcal{JS}^i$ can use as opens only the opens of the root and the facts defined locally in the root. This has the consequence that facts defined in $\mathcal{JS}^i$ do not appear as opens in $\mathcal{JS}^j$ if $i \neq j$.

Lemma 3.2
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 justification system. If $i \neq j$, then $\mathcal{F}^i_d \cap \mathcal{F}^j = \emptyset$.

Example 3.3

Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}^1\} \rangle$ be a nested system with $\mathcal{F} = \{p, q, r, \sim p, \sim q, \sim r\} \cup \mathcal{L}$, $\mathcal{F}_{dl} = \{r, \sim r\}$, $\mathcal{B} = \mathcal{B}_{\text{KK}}$, and

$$R = \{\ r \leftarrow p, q; \quad \sim r \leftarrow \sim p; \quad \sim r \leftarrow \sim q\ \}.$$

The inner system $\mathcal{JS}^1$ is equal to the unnested system with $\mathcal{F}^1_{dl} = \{p, \sim p, q, \sim q\}$, $\mathcal{B}^1 = \mathcal{B}_{\text{wf}}$, and

$$R^1 = \{\ p \leftarrow \sim q, r; \quad \sim p \leftarrow q; \quad \sim p \leftarrow \sim r; \quad q \leftarrow q; \quad \sim q \leftarrow \sim q\ \}.$$

We summarize this nested justification system graphically as follows:

$$\mathcal{B}_{\text{KK}} : \left\{ \begin{array}{l} r \leftarrow p, q; \quad \sim r \leftarrow \sim p; \quad \sim r \leftarrow \sim q \\ \mathcal{B}_{\text{wf}} \{\ p \leftarrow \sim q, r; \quad \sim p \leftarrow q; \quad \sim p \leftarrow \sim r; \quad q \leftarrow q; \quad \sim q \leftarrow \sim q\ \} \end{array} \right\}.$$

This example also illustrates how different branch evaluations (e.g., $\mathcal{B}_{\text{KK}}$ and $\mathcal{B}_{\text{cwf}}$) can be combined in nested justification systems.

One notoriously difficult problem is the integration of aggregates in non-monotonic semantics of logic programming (Pelov et al. 2007; Faber et al. 2011; Gelfond and Zhang 2019; Alviano et al. 2021; Vanbesien et al. 2021). While we do not develop the full theory here, we conjecture that nested justification systems can shed light on the relations between different semantics. As a first step towards this goal, we will show on a single example how different semantics for aggregates can be obtained by plugging in a different nested system defining the aggregate atoms in question. The outer system will always simply be evaluated under the stable semantics.

Example 3.4
We consider a representation of an aggregate `atLeastTwo` expressing that at least two atoms among p, q and s are true.

$$\mathcal{JS}_{FLP} = \mathcal{B}_{\text{st}} : \left\{ \begin{array}{l} p \leftarrow \mathbf{t}; \quad q \leftarrow \mathbf{t}; \quad s \leftarrow p, \texttt{atLeastTwo}; \\ \mathcal{B}_{\text{KK}} : \{\ \texttt{atLeastTwo} \leftarrow p, q; \quad \texttt{atLeastTwo} \leftarrow s, q; \quad \texttt{atLeastTwo} \leftarrow p, s\ \} \end{array} \right\}$$

$$\mathcal{JS}_{GZ} = \mathcal{B}_{\text{st}} : \left\{ \begin{array}{l} p \leftarrow \mathbf{t}; \quad q \leftarrow \mathbf{t}; \quad s \leftarrow p, \texttt{atLeastTwo}; \\ \mathcal{B}_{\text{KK}} : \left\{ \begin{array}{l} \texttt{atLeastTwo} \leftarrow p, q, \sim s; \quad \texttt{atLeastTwo} \leftarrow s, q, \sim p; \\ \texttt{atLeastTwo} \leftarrow p, s, \sim q; \quad \texttt{atLeastTwo} \leftarrow p, q, s \end{array} \right\} \end{array} \right\}$$

For brevity and for emphasizing the relation with logic programming, we have here only written down the rules for *positive* facts. The rules for negative facts can be obtained automatically by means of a technique called *complementation*; more details can be found in Appendix A of the full version of this article (Marynissen et al. 2022). These systems differ only in their inner justification system, i.e., in which justifications for the atom `atLeastTwo` they accept. The inner system is evaluated here (in both cases) under Kripke-Kleene semantics; however, since it is a non-recursive system, all major branch evaluations we discussed coincide. These two justification systems are inspired by the logic program

$$p. \qquad q. \qquad s \leftarrow p,\ \#\{p, q, s\} \geq 2.$$

which has one (two-valued) stable model, namely $\{p, q, s\}$, according to the FLP semantics (Faber et al. 2011) but no (two-valued) stable models according to the GZ semantics (Gelfond and Zhang 2019). We will show later that this is indeed what the semantics for nested justifications for the first respectively the second system.

In the following two sections, two different characterisations of the semantics for nested justification systems are given. In a nutshell, these characterisations both describe how to convert a nested justification system in a non-nested justification system. The first characterisation, called the *compression-based* characterisation, keeps the branch evaluation of the top system, but manipulates the rules to "squeeze in" all information about subsystems (their branch evaluation as well as their rules), while the second, *merging-based* characterisation, keeps the rules of the original system intact, but creates a new branch evaluation based on all branch evaluations present in the system.

4 Compression-based Characterisation of Semantics for Nested Systems

The semantics for nested justification systems was originally introduced by Denecker et al. (2015). The basic idea is to *compress* a nested justification system in a regular justification system, starting from the leaves of the nesting tree and iteratively moving up. In a *compressible* nested system, the branch evaluations of subsystems are parametric and hence we know (by Proposition 3 of Marynissen et al. (2021)) that, if the interpretation of the open facts is fixed, there is a unique model in which the value of a fact depends solely on the values of the opens. Therefore, the model can be represented by a set of rules for which the body contains only open facts. This representation is formed by transforming each justification into a single rule, by an operation called *flattening*.

Definition 4.1
Let $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, R, \mathcal{B} \rangle$ be a justification system. The *flattening* $\text{Flat}(\mathcal{JS})$ is the justification system $\langle \mathcal{F}, \mathcal{F}_d, R^f, \mathcal{B} \rangle$, where

$$R^f = \{x \leftarrow A \mid x \in \mathcal{F}_d, J \in \mathfrak{J}_x, A = \{\mathcal{B}(\mathbf{b}) \mid \mathbf{b} \in B_J(x)\}\}.$$

Intuitively, $\text{Flat}(\mathcal{JS})$ is obtained by constructing a rule $x \leftarrow A$ for every justification J for x, where A is obtained by simply taking the facts to which the branches in J starting in x are mapped by the branch evaluation $\mathcal{B}$.

Example 4.2 (*Example 3.3 ctd.*)
In view of Example 2.8, the flattening of the inner justification system $\mathcal{JS}^1$ has the following rules:

$$(R^1)^f = \{\ p \leftarrow \mathbf{t}, r; \quad \sim p \leftarrow \mathbf{f}; \quad \sim p \leftarrow \sim r; \quad q \leftarrow \mathbf{f}; \quad \sim q \leftarrow \mathbf{t}\ \}$$

In more detail, $p \leftarrow \mathbf{t}, r$ is obtained from the justification made up of $\mathbf{b}_1$ and $\mathbf{b}_2$, and the fact that $\mathcal{B}_{\text{KK}}(\mathbf{b}_1) = \mathbf{t}$ and $\mathcal{B}_{\text{KK}}(\mathbf{b}_2) = r$.

This is called flattening because every locally complete justification is reduced to a single rule. If $\mathcal{B}$ is parametric, justifications in this new system have only branches of length two. If these simple branches are mapped to their last element, then the flattening is equivalent to the original system. For parametric systems, flattening thus preserves the meaning of the original system:

Proposition 4.3
If $\mathcal{JS}_1$ and $\mathcal{JS}_2$ are parametric justification systems mapping a branch $x \to y$ to y, then:

1. $\mathcal{JS}_1$ is equivalent to $\text{Flat}(\mathcal{JS}_1)$.

2. $\mathcal{JS}_1$ and $\mathcal{JS}_2$ are equivalent if and only if $\text{Flat}(\mathcal{JS}_1)$ and $\text{Flat}(\mathcal{JS}_2)$ are equivalent.

Notice that it immediately follows that if the preconditions of the above proposition are satisfied, then the $\mathcal{JS}_1$ and $\text{Flat}(\mathcal{JS}_1)$-models coincide. 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.

Flattening provides us with a way to evaluate inner definitions. The second operation needed for compression is *unfolding*, which allows to eliminate inner symbols from the outside definitions, by replacing facts at a lower level by a (flattened) case for that fact.

Definition 4.4
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

$$\text{Unf}_f\,(x \leftarrow A) = x \leftarrow (A \setminus X) \cup \bigcup_{y \in A \cap X} f(y).$$

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

$$\text{Unf}_{R_\ell}\,(x \leftarrow A) = \{\text{Unf}_f\,(x \leftarrow A) \mid f \in F_{x \leftarrow A}\}\,.$$

The *unfolding* of R with respect to R_ℓ is

$$\text{Unfold}_{(X,R_\ell)}\,(R) = \bigcup_{x \leftarrow A \in R} \text{Unf}_{R_\ell}\,(x \leftarrow A)\,.$$

In a nutshell, $\text{Unfold}_{(X,R_\ell)}\,(R)$ is obtained by replacing, in each $x \leftarrow y_1, \ldots, y_n \in R$, each fact y_i defined in R_ℓ by the body facts A of a rule $y_i \leftarrow A$ defining y_i.

Example 4.5 (*Examples 3.3 & 4.2 ctd.*)
With all systems as defined previously, we see that

$$\text{Unfold}_{(\mathcal{F}_{dl},\text{Flat}(R^1))}\,(R) = \left\{\ r \leftarrow \mathbf{t}, r, \mathbf{f};\quad \sim r \leftarrow \sim r;\quad \sim r \leftarrow \mathbf{f};\quad \sim r \leftarrow \mathbf{t}\ \right\} \cup R^1.$$

On the basis of the unfolding of a set of rules we can define the unfolding of a justification system, obtained by unfolding the rules of the parent system with respect to the rules of the children systems, and keeping the rules for the lower-level facts (R^s):

Definition 4.6
Let $\mathcal{JS} = \left\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \left\{\mathcal{JS}^1, \ldots, \mathcal{JS}^k\right\}\right\rangle$ be a two-level nested compressible justification system. The *unfolding* of $\mathcal{JS}$ is

$$\text{Unfold}\,(\mathcal{JS}) = \langle \mathcal{F}, \mathcal{F}_d, R^s \cup \text{Unfold}_{(\mathcal{F}_d \setminus \mathcal{F}_{dl}, R^s)}\,(R)\,, \mathcal{B}\rangle,$$

where $R^s = \cup_{i=1}^{k} R^i$.

The unfold-operation reduces the depth of the nesting tree and thus serves as a basis for the compression of a nested justification system (notice that since the subsystems $\mathcal{JS}^i$ will have a smaller depth than $\mathcal{JS}$, $\text{Compress}(\mathcal{JS})$ is well-defined).

Definition 4.7 (*Denecker et al. 2015*)

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 compressible nested justification system. The compression $\text{Compress}(\mathcal{JS})$ is defined inductively to be the justification system

$$\text{Unfold}\left(\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\text{Flat}(\text{Compress}(\mathcal{JS}^1)), \ldots, \text{Flat}(\text{Compress}(\mathcal{JS}^k))\}\rangle\right).$$

Remark 4.8
Intuitively, the compression works as follows, described here for a two-level system $\mathcal{JS}$ with only one subsystem $\mathcal{JS}^1 = \langle \mathcal{F}^1, \mathcal{F}^1_d, R^1, \mathcal{B}^1\rangle$. First of all, $\mathcal{JS}^1$ is flattened. We know from Proposition 4.3 that this preserves the meaning of the system, and in fact, it no longer matters which branch evaluation is used, provided that it maps branches of length 2 onto their last element. In other words, the original system and branch evaluation are consolidated into the flattened frame. The next step is to replace each occurrence (in $\mathcal{JS}$) of each fact defined in $\mathcal{JS}^1$ by all possible cases for it in the flattened system. If $\mathcal{B}^1$ is parametric, each such case in the flattened system contains only fact that are either locally defined or open in $\mathcal{JS}$. The result can then be evaluated without any knowledge of the branch evaluation or rules of $\mathcal{JS}^1$. While technically, the compressed system also contains a copy of R^1, they are only there for recovering the value of facts in $\mathcal{F}^1_d$; they play no role for determining the value of facts in $\mathcal{JS}$. In case $\mathcal{B}^1$ is not parametric, this construction no longer works: the cases in the flattening can then contain facts that are locally defined in $\mathcal{JF}^1$. For this reason, compression was only defined for parametric branch evaluations by Denecker et al. (2015). We here slightly relax this condition, by allowing (in a compressible system) the outer branch evaluation to be non-parametric.

Example 4.9 (*Example 4.5 ctd.*)
We have already calculated $\text{Unfold}_{\left(\mathcal{F}^1_{dl}, R^2\right)}\left(R^1\right)$ in Example 4.5. Since $\text{Unfold}_{(\mathcal{F}_{dl}, R^1)}(R)$ contains only rule bodies with open facts, we see that $\text{Flat}(\text{Unfold}_{(\mathcal{F}_{dl}, R^1)}(R)) = \text{Unfold}_{(\mathcal{F}_{dl}, R^1)}(R)$. We therefore obtain:

$$\text{Compress}(\mathcal{JS}) = \langle \mathcal{F}, \mathcal{F}_d, \text{Unfold}_{(\mathcal{F}_{dl}, R^1)}(R), \mathcal{B}_{\text{KK}}\rangle$$

and thus end up with an unnested justification system. It can be verified that the unique $\text{Compress}(\mathcal{JS})$-model is $\mathcal{I}$ with $\mathcal{I}(\sim r) = \mathcal{I}(p) = \mathcal{I}(q) = \mathbf{f}$.

Example 4.10
We return to the nested justification system of Example 3.4. It can be verified that $\text{Compress}(\mathcal{JS}_{FLP})$ is equal to (the complementation of)

$$\mathcal{B}_{\text{st}} : \left\{ \begin{array}{l} p \leftarrow \mathbf{t}; \quad q \leftarrow \mathbf{t}; \quad s \leftarrow p, s \quad s \leftarrow p, q, s; \quad s \leftarrow p, q; \\ \texttt{atLeastTwo} \leftarrow p, q; \quad \texttt{atLeastTwo} \leftarrow s, q; \quad \texttt{atLeastTwo} \leftarrow p, s; \end{array} \right\}$$

The interpretation $\mathcal{I}$ with $\mathcal{I}(s) = \mathcal{I}(p) = \mathcal{I}(q) = \mathbf{t}$ is the only two-valued $\text{Compress}(\mathcal{JS}_{FLP})$. This interpretation indeed corresponds to the only stable model under the FLP semantics (mentioned above).

On the other hand, there are no two-valued $\text{Compress}(\mathcal{JS}_{GZ})$-models. This can be seen by observing that $\text{Compress}(\mathcal{JS}_{GZ})$ equals (the complementation of):

$$\mathcal{B}_{\text{st}} : \left\{ \begin{array}{l} p \leftarrow \mathbf{t}; \quad q \leftarrow \mathbf{t}; \quad s \leftarrow p, q, \sim s; \quad s \leftarrow p, s, \sim q; \quad s \leftarrow p, q, s, \sim p; \quad s \leftarrow p, q, s; \\ \texttt{atLeastTwo} \leftarrow p, q, \sim s; \quad \texttt{atLeastTwo} \leftarrow s, q, \sim p; \\ \texttt{atLeastTwo} \leftarrow p, s, \sim q; \quad \texttt{atLeastTwo} \leftarrow p, q, s \end{array} \right\}$$

Suppose $\mathcal{I}$ were a Compress($\mathcal{JS}_{GZ}$)-model, then clearly $\mathcal{I}(p) = \mathcal{I}(q) = \mathbf{t}$. If $\mathcal{I}(s) = \mathbf{t}$, then $\mathrm{SV}(s, \mathcal{I}) = \mathbf{f}$ (because each justification of s has a branch $s \to \sim x \to \ldots$ (with x either p, q, or s). On the other hand, if $\mathcal{I}(s) = \mathbf{f}$, then $\mathrm{SV}(s, \mathcal{I}) = \mathbf{t}$, and it can be verified that the justification that selects the rule $s \leftarrow p, q, \neg s$ supports s in this case. Hence, such an $\mathcal{I}$ can indeed not exist.

In this section, the semantics of nested justifications in terms of compressions, originally proposed by Denecker et al. (2015) was described and studied. This characterisation of semantics of nested justification systems is the first one given in the literature. However, it does have a significant downside: the explanatory potential of justification systems is partially lost, since rules in lower levels of the justification system are compressed to their evaluation. For example, the definition of r in Example 3.3 is compressed to $\{r \leftarrow \mathbf{t}, r, \mathbf{f}\}$, and thus the explanatory potential of justification theory is completely lost. Indeed, intuitively, we have a justification in the original nested justification system that looks like the justification on the left of the figure below, and is obtained by constructing a justification on the basis of all (relevant) rules in $\mathcal{JS}$:

However, under the compression-based semantics, we obtain the justification on the right, where most information on the justification of r has been lost. Thus, from the point of view of explainable reasoning, it is preferable to be able to somehow evaluate the justification on the left while retaining semantic equivalence with the compression-based semantics. This is exactly what will be done in the next section.

5 Merging-based Characterisation of Semantics for Nested Systems

To avoid the loss of information with explanatory potential suffered by the compression-based characterisation, the merging-based characterisation of semantics is now proposed. The basic idea of the merging-based characterisation is to consolidate all component systems of the nested justification system in a justification system $\langle \mathcal{F}, \mathcal{F}_d, R^*, \mathcal{B}^* \rangle$, where all rules of all component justification systems are simply gathered in R^*. This also means that justification branches are now constructed on the basis of *all* rules occurring somewhere in the nested justification system, and not simply on the basis of one component justification system. The information about the nesting structure of the original system is not discarded, but taken into account in the *merge branch evaluation* $\mathcal{B}^*$. In more detail, for infinite branches $\mathbf{b}$ in R^*, one looks for the highest component justification system $\mathcal{JS}'$ in the nested justification system s.t. $\mathbf{b}$ contains infinitely many elements of locally defined facts of $\mathcal{JS}'$, and then applies the original branch evaluation $\mathcal{B}'$ of the component system in question to the $\mathcal{JS}'$-branch $\mathbf{b}'$ that corresponds to $\mathbf{b}$. This formalizes the intuition that priority is given to the outermost semantics.

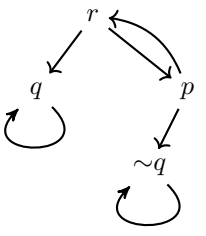
Definition 5.1
The *merge* Merge($\mathcal{JS}$) of $\mathcal{JS}$ is the justification system $\langle \mathcal{F}, \mathcal{F}_d, R^*, \mathcal{B}^* \rangle$, where:

- R^* is the union of all the rules in $\mathcal{JS}$.
- the *merge branch evaluation* $\mathcal{B}^*$w.r.t. $\mathcal{JS}$ is defined as follows:
  1. if $\mathbf{b}$ is finite, then $\mathcal{B}^*(\mathbf{b})$ is equal to the last element of $\mathbf{b}$.
  2. if $\mathbf{b}$ is infinite, then let $\mathcal{JS}'$ be the unique[2] system equal to either $\mathcal{JS}$ or one of its descendant systems such that:
     (a) $\mathbf{b}$ has infinitely many occurrences of facts defined locally in $\mathcal{JS}'$.[3]
     (b) No ancestor system of $\mathcal{JS}'$ has property (a), i.e. for all ancestor systems of $\mathcal{JS}$, $\mathbf{b}$ has only finitely many occurrences of facts defined locally in that system.

     Let $\mathbf{b}'$ be the branch obtained from $\mathbf{b}$ by removing all facts not defined locally in $\mathcal{JS}'$. Then $\mathcal{B}^*(\mathbf{b}) = \mathcal{B}'(\mathbf{b}')$, where $\mathcal{B}'$ is the branch evaluation of $\mathcal{JS}'$.

The conditions in Definition 3.1 guarantee that branches in Merge($\mathcal{JS}$) are sequences of facts $x_0 \to x_1 \to \ldots$ s.t. x_i is defined locally in a descendant or ancestor of x_{i+1}. This guarantees that the system $\mathcal{JS}'$ indeed exists and is unique: if a branch has infinitely many occurrences of facts defined (locally) in two systems, then it must also go infinitely often through a common ancestor.

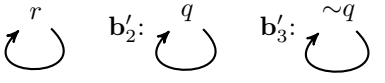
Example 5.2 (*Example 3.3 ctd.*)
Consider again $\mathcal{JS}$ as in Example 3.3. We have, among others, the justification for r in Merge($\mathcal{JS}$) on the left side of the following picture[4]; on the right side, the individual branches have been unravelled and the corresponding "primed" branches are drawn:



$\mathbf{b}_1$: $r \longrightarrow p \longrightarrow r \longrightarrow p \longrightarrow \cdots$

$\mathbf{b}_2$: $r \longrightarrow q \longrightarrow q \longrightarrow \cdots$

$\mathbf{b}_3$: $r \longrightarrow p \longrightarrow {\sim}q \to {\sim}q \to \cdots$



Notice that all literals in $\mathbf{b}_1$ occur infinitely many times in $\mathbf{b}_1$. Since r is defined locally in $\mathcal{JS}$, i.e. it occurs in $\mathcal{F}_{dl}$, we have to look at $\mathcal{JS}$ to evaluate $\mathbf{b}_1$ with $\mathcal{B}^\star$. On the right side, the branch $\mathbf{b}'_1 = r \to r \to \ldots$ obtained from $\mathbf{b}_1$ by removing all facts defined outside of $\mathcal{JS}$ is given. We now obtain $\mathcal{B}^\star(\mathbf{b}_1)$ by evaluating $\mathbf{b}'_1$ according to $\mathbf{b}_1$, i.e., $\mathcal{B}^\star(\mathbf{b}_1) = \mathcal{B}_{\mathrm{KK}}(\mathbf{b}'_1) = \mathbf{u}$.

For $\mathbf{b}_2$ and $\mathbf{b}_3$, we see that the highest justification system in which q respectively ${\sim}q$ are defined is $\mathcal{JS}^1$. We therefore obtain the branches $\mathbf{b}'_2 = q \to q \to \ldots$ and

[2] Existence and uniqueness of this system are argued below the definition.
[3] In more formal detail, there is some $a \in \mathcal{F}'_d$ s.t. $\mathbf{b}$ contains infinitely many occurrences of a.
[4] The justification is visualized here as a graph; the actual justification is the tree-unravelling of this graph which will for instance have infinitely many nodes labeled r. For instance, the edge from p to r symbolises the fact that every node labelled p has a child node labelled r.

$\mathbf{b}'_3 = \sim q \rightarrow \sim q \rightarrow \ldots$. Since $\mathcal{JS}^1$ uses the well-founded branch evaluation $\mathcal{B}_{\mathrm{wf}}$, we obtain $\mathcal{B}^\star(\mathbf{b}_2) = \mathcal{B}_{\mathrm{wf}}(\mathbf{b}'_2) = \mathbf{f}$ and $\mathcal{B}^\star(\mathbf{b}_3) = \mathcal{B}_{\mathrm{wf}}(\mathbf{b}'_3) = \mathbf{t}$. Here we can thus see the different branch evaluations of the component systems at work: for example, branches with infinite occurrences of positive literals are treated differently in view of the component justification system in which these literals are defined, and the branch evaluations used in these justification systems. In more detail, $\mathbf{b}_1$ is evaluated to $\mathbf{u}$ since it is defined locally in $\mathcal{JS}$, which uses the Kripke-Kleene evaluation, whereas $\mathbf{b}_3$ is evaluated to $\mathbf{f}$ since it is defined in $\mathcal{JS}^2$, which uses the well-founded evaluation. Altogether, we see that for $\mathcal{I}(r) = \mathbf{t}$, $\mathcal{I}(q) = \mathcal{I}(p) = \mathbf{f}$, it holds that $\mathrm{SV}(r, \mathcal{I}) = \mathbf{t}$. In fact, it can be checked that $\mathcal{I}$ is the unique Compress($\mathcal{JS}$)-model, which is no coincidence as we will see next.

6 Equivalence of Compression- and Merging-Based Characterisations

Even though the compression-based and merging-based characterisation of semantics are based on quite different mechanisms, they give rise to the same evaluations for nested justification systems, i.e., they are equivalent. Showing this is rather involved, and due to spatial limitations, the details are given in Appendix D of the full version of this article (Marynissen et al. 2022) and illustrated schematically in Figure 1. The crucial idea for the proof of equivalence is the definition of two operations, shrinking and expanding, that allow converting justifications in Compress($\mathcal{JS}$) to justifications in Merge($\mathcal{JS}$) and vice versa. On the basis of these operations, it is then shown that the supported value for defined facts are the same under Merge($\mathcal{JS}$) and Compress($\mathcal{JS}$) (for all defined facts x and interpretations $\mathcal{I}$):

$$\mathrm{SV}^t_{\mathrm{Compress}(\mathcal{JS})}(x, \mathcal{I}) = \mathrm{SV}^t_{\mathrm{Merge}(\mathcal{JS})}(x, \mathcal{I})$$

To guarantee that shrink and expand behave as expected, two minor assumptions are necessary, namely (1) that finite branches are mapped to their last element (Marynissen et al. (2021, Propositions 1 and 2) have shown how to modify branch evaluations to satisfy this condition) and (2) that all parametric branch evaluations map infinite branches to logical facts (in fact, unless additional structure on the fact space is assumed, this will always be satisfied). We obtain the following theorem.

Theorem 6.1
Let a compressible nested justification system $\mathcal{JS}$ in which every branch evaluation maps finite branches to their last element, and infinite branches to logical facts. Then Compress($\mathcal{JS}$) and Merge($\mathcal{JS}$) are equivalent.

It immediately follows that for a justification system that satisfies all of the preconditions of Theorem 6.1, the Compress($\mathcal{JS}$)-models coincide with the Merge($\mathcal{JS}$)-models.

Remark 6.2
As mentioned in Remark 4.8, compression was only defined for so-called compressible branch evaluations to allow evaluating the different involved systems separately. For merging, this restriction does not play a role: here all branch evaluations are taken into account simultaneously, in the definition of $\mathcal{B}^*$. As such, we have not just given a novel characterization of the semantics of nested systems, but also significantly expanded its scope, by allowing non-parametric systems, such as stable or supported, to be nested.

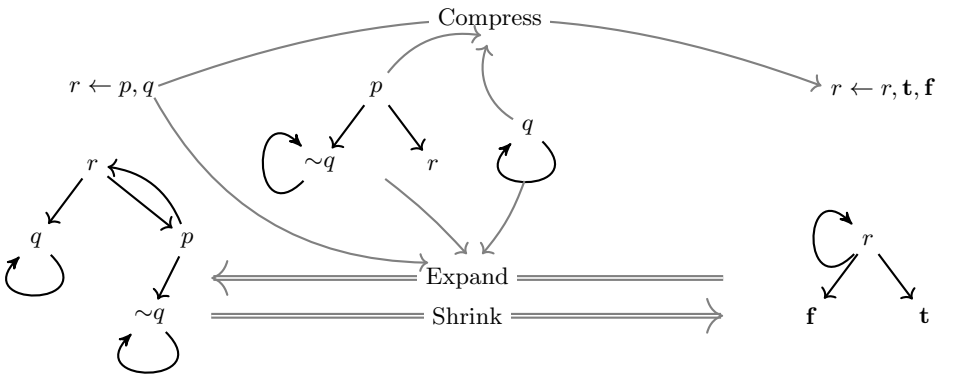


Fig. 1. Schematic illustration (based on Example 3.3) of the *shrinking*-operation, which converts a justification in Merge($\mathcal{JS}$) into a justification in Compress($\mathcal{JS}$) (also illustrated schematically), and the *expanding*-operation, which converts a justification in Compress($\mathcal{JS}$) (taking into account the relevant justifications in $\mathcal{JS}$) into a justification in Merge($\mathcal{JS}$).

7 FO(FD): Application to Fixpoint Definitions

FO(FD) (Hou et al. 2010) is a logic that integrates fixpoint definitions based on rules with first-order logic. Essentially, a *least fixpoint definition* (respectively *greatest fixpoint definition*) is defined inductively as an expression of the form $\mathcal{D} = \lfloor \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n \rfloor$ (respectively $\lceil \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n \rceil$), where $\mathcal{R}$ is a set of rules and each Δ_i is a least fixpoint definition and each ∇_j is a greatest fixpoint definition, s.t. every symbol is defined in at most one of $\mathcal{R}$, Δ_1, …, Δ_m, ∇_1, …, ∇_n. Additionally, defined symbols are only allowed to occur positively in rule bodies. The semantics of FO(FD), explained in full detail in Appendix E of the full version of this article (Marynissen et al. 2022), is given in terms of (two-valued) interpretations, and is defined iteratively, by means of an immediate consequence operator $\Gamma^{\mathcal{D}}(I)$. Consistent with the idea of capturing a definition, a unique model according to these semantics is guaranteed to exist. The nested nature of fixpoint definitions comes into play when requirements of different levels conflict with each-other. In that case, intuitively, the highest level (the outermost definition) will be given priority.

As an example, restricted to the propositional case, consider:

$$\mathcal{D} = \left\lfloor \begin{array}{llll} p \leftarrow q \vee r; & q \leftarrow p; & u \leftarrow s & \\ \lceil\ r \leftarrow p; & s \leftarrow t \vee q; & t \leftarrow s & \rceil \end{array} \right\rfloor$$

This fixpoint definition consists of two levels. The uppermost or global level is a least fixpoint definition of p, q and u, whereas the lower or inner level contains a greatest fixpoint definition of the atoms r, s and t. Intuitively, the semantics will ensure that the model of this definition corresponds to a greatest fixpoint of the immediate consequence operator based on the inner system, and a least fixpoint of the immediate consequence operator based on the outer system, together with the respective interpretation for the inner system. For example, the interpretation that assigns **t** to s, t and u, and **f** to p, q and r to **f** is a model of this fixpoint definition. We see that the cycle between $p \leftarrow q \vee r$ and $r \leftarrow p$ gives rise to a conflict between the two levels: the greatest fixpoint nature of the lowest level would require us to make p and r true, whereas the least fixpoint nature

of the highest level would require us to make p and r false. Since priority is given to the highest level, p and r are false in the defined interpretation.

Nested justifications were partially inspired by FO(FD), and is therefore not surprising that FO(FD) can be captured using nested justifications. This way of assigning priority to higher levels in case of conflicting demands also shows up in the merging-based characterisation, where the highest component system is used to evaluate infinite branches.

For the propositional case, the translation is rather straightforward. Given a fixpoint definition $\mathcal{D}$, $\mathcal{JS}_\mathcal{D}$ is defined as $\langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \mathcal{B}, \{\mathcal{JS}_{\Delta_1}, \ldots, \mathcal{JS}_{\Delta_m}, \mathcal{JS}_{\nabla_1}, \ldots, \mathcal{JS}_{\nabla_n}\}\rangle$, where $\mathcal{B} = \mathcal{B}_{\text{wf}}$ if $\mathcal{D}$ is a least fixpoint definition and $\mathcal{B} = \mathcal{B}_{\text{cwf}}$ if $\mathcal{D}$ is a greatest fixpoint definition. We can thus represent the nested justification system $\mathcal{JS}_\mathcal{D}$ as:

$$\mathcal{B}_{\text{wf}} : \left\{ \begin{array}{l} p \leftarrow q \vee r; \quad q \leftarrow p; \quad u \leftarrow s \\ \mathcal{B}_{\text{cwf}} : \{ \; r \leftarrow p; \quad s \leftarrow t; \quad s \leftarrow q; \quad t \leftarrow s \; \} \end{array} \right\}$$

This translation is adequate, in the sense that a unique $\mathcal{B}_{\text{wf}}$-model (respectively $\mathcal{B}_{\text{cwf}}$-model) that leaves no atoms $\mathbf{u}$ is guaranteed to exist and corresponds to the least (respectively greatest) fixpoint of the fixpoint definition. Notice that we replaced $s \leftarrow t \vee q$ by $s \leftarrow t$ and $s \leftarrow q$. For more complex formulae, more work is required. For the full first-order case, the idea is essentially the same, but an intermediary system is included between $\mathcal{JS}_\mathcal{D}$ and $\mathcal{JS}_{\Delta_1}, \ldots, \mathcal{JS}_{\Delta_m}, \mathcal{JS}_{\nabla_1}, \ldots, \mathcal{JS}_{\nabla_n}$ to ensure that first-order formulae are treated adequately. This translation, as well as the correspondence results, are detailed in Appendix E of the full version of this article (Marynissen et al. 2022).

8 Conclusion, in View of Related Work

In this paper, we gave two characterisations of the semantics of nested justification systems, showed that these characterisations are equivalent and demonstrated how nested justifications can capture nested fixpoint definitions. The potential applications of nested justification systems are extensive, as they allow for the modular, and therefore straightforward, design of rule-based languages. A prime example is the definition of semantics for aggregates. Nested justification systems allow to separate the definition of semantics aggregates from that of a logic program, by adding rules defining the aggregates at the lowest level of a justification system (cf. Example 3.4). In future work, we will investigate exact translations from semantics for logic programs with aggregates (e.g., (Pelov et al. 2007; Faber et al. 2011; Gelfond and Zhang 2019; Alviano et al. 2021; Vanbesien et al. 2021) in nested justification systems, bringing justification theory's explanatory potential to such semantics.

Some aspects of nested justifications, in particular the fact that different modules using different semantics can be combined, are reminiscent of *multi-context systems* (in short, MCSs) (Brewka et al. 2018). There are, however, significant differences between the two formalisms. On the one hand, MCSs allow for a broader range of knowledge bases to be used as modules. For example, defaults or autoepistemic belief bases can be used in MCSs, but cannot be modelled straightforwardly in justification theory. On the other hand, nested justification systems allow for more sophisticated interactions and structures. Indeed, a multi-context system can be seen as a set of contexts $\{C_1, \ldots, C_n\}$ and a set of bridge rules $\mathcal{R}$ allow the flow of information between contexts. I.e. it corresponds, at least structurally, to a nested justification system $\mathcal{JS} = \langle \mathcal{F}, \mathcal{F}_d, \mathcal{F}_{dl}, R, \{\mathcal{JS}^1, \ldots, \mathcal{JS}^n\}\rangle$

where every $\mathcal{JS}^i$ (for $i = 1, \ldots, n$) is a non-nested justification system. It remains an open question whether a sub-class of MCSs based on rule-based contexts can be translated into nested justification systems.

In this paper, we gave a new view on nested justification systems that retains justification quality and showed (under mild restrictions) the two views are equivalent for tree-like justifications. Whether this holds for graph-like justifications is an open question.

Another question that shows up in several papers on justification theory (Denecker 1993; Marynissen et al. 2018; 2020) is the *consistency* question: is it always so that $\mathrm{SV}(\sim x, \mathcal{I}) = \sim \mathrm{SV}(x, \mathcal{I})$? The same question is relevant for nested systems. In a companion paper (Marynissen and Bogaerts 2022), we show that in the tree-like system, all branch evaluations (and hence also the *merge* evaluation) are consistent. And because of our equivalence, we have that compression is also consistent.

References

Alviano, M., Faber, W., and Gebser, M. 2021. Aggregate semantics for propositional answer set programs. *CoRR*, *abs/2109.08662*.

Brewka, G., Ellmauthaler, S., Gonçalves, R., Knorr, M., Leite, J., and Pührer, J. 2018. Reactive multi-context systems: Heterogeneous reasoning in dynamic environments. *Artificial Intelligence*, *256*, 68–104.

Denecker, M. 1993. *Knowledge representation and reasoning in incomplete logic programming.* PhD thesis, K.U.Leuven, Leuven, Belgium.

Denecker, M., Brewka, G., and Strass, H. A formal theory of justifications. In *Proceedings of LPNMR* 2015, pp. 250–264.

Faber, W., Pfeifer, G., and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, *175*, 1, 278–298.

Gelfond, M. and Zhang, Y. 2019. Vicious circle principle, aggregates, and formation of sets in asp based languages. *Artificial Intelligence*, *275*, 28–77.

Hou, P., De Cat, B., and Denecker, M. 2010. FO(FD): Extending classical logic with rule-based fixpoint definitions. *TPLP*, *10*, 4-6, 581–596.

Marynissen, S. 2022. *Advances in Justification Theory.* PhD thesis, Department of Computer Science, KU Leuven. Denecker, Marc and Bart Bogaerts (supervisors).

Marynissen, S. and Bogaerts, B. Tree-like justifications are consistent. In *Technical Communications of ICLP* 2022. To appear.

Marynissen, S., Bogaerts, B., and Denecker, M. 2020. Exploiting game theory for analysing justifications. *Theory Pract. Log. Program.*, *20*, 6, 880–894.

Marynissen, S., Bogaerts, B., and Denecker, M. On the relation between approximation fixpoint theory and justification theory. In *Proceedings of IJCAI* 2021, pp. 1973–1980.

Marynissen, S., Heyninck, J., Bogaerts, B., and Denecker, M. 2022. On nested justification systems (full version).

Marynissen, S., Passchyn, N., Bogaerts, B., and Denecker, M. Consistency in justification theory. In *Proceedings of NMR* 2018, pp. 41–52.

Pelov, N., Denecker, M., and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *TPLP*, *7*, 3, 301–353.

Vanbesien, L., Bruynooghe, M., and Denecker, M. 2021. Analyzing semantics of aggregate answer set programming using approximation fixpoint theory. *arXiv preprint arXiv:2104.14789,*.