Games, Queries, and Argumentation Frameworks: Time for a Family Reunion!

Bertram Ludäscher[1,*], Shawn Bowers[2] and Yilin Xia[1]

[1]School of Information Sciences, University of Illinois, Urbana-Champaign, USA
[2]Department of Computer Science, Gonzaga University, USA

Abstract
Combinatorial game theory in the form of two-player games has played an important historical role in formal argumentation, logic programming, nonmonotonic reasoning, database query languages, and more recently in data provenance. While such game-based approaches played an integral role early on in formal argumentation, in the study of expressiveness of query languages, and in the quest to find well-behaved semantics for logic programs with recursion through negation, these areas seem to have largely separated from their historical connections, following their own, separate paths with distinct concepts, terminologies, and research results. We touch upon this history and highlight how the use of a single, unstratified logic rule continues to underly many of the approaches developed today within these different communities. We argue that a fruitful line of research exists by reconnecting the communities, in a kind of "family reunion", where results from one community may be transferable to the other (*mutatis mutandis*), leading to new insights in the neighboring fields. We describe some initial correspondences and connections and invite the community to join our exploration of additional ones.

Keywords
Argumentation frameworks, game theory, database theory, graph kernels

1. Introduction

Argumentation frameworks consist of abstract *arguments* and a binary *attacks* relation, specifying how arguments may defeat each other. Given an argumentation framework, a fundamental problem is to find admissible subsets of arguments, i.e., which (i) don't attack each other, and (ii) which are *accepted*, i.e., they defend themselves against attacks from outside arguments. Arguments attacked by an admissible subset (so-called *extensions*) are said to be *defeated*.

In Dung's seminal work on argumentation frameworks [1], the following two-line logic program (P_{AF2}) is introduced as an "*argument processing unit*" (APU), i.e., a "*general method for generating meta-interpreters for argumentation systems*":

$$\begin{aligned}\mathsf{defeated}(X) &\leftarrow \mathsf{attacks}(Y,X), \mathsf{accepted}(Y).\\ \mathsf{accepted}(X) &\leftarrow \neg\, \mathsf{defeated}(X).\end{aligned} \qquad (P_{\mathsf{AF2}})$$

The first rule states that an argument X is defeated in an argumentation framework if there exists an argument Y that attacks X and is accepted. The second rule states that an argument is

7th Workshop on Advances in Argumentation in Artificial Intelligence (AI^3), November 06–09, 2023, Rome, Italy
*Corresponding author.
ludaesch@illinois.edu (B. Ludäscher); bowers@gonzaga.edu (S. Bowers); yilinx2@illinois.edu (Y. Xia)

accepted if it is not defeated. Dung's work spawned a large body of research, including families of models, semantics, tools, and applications of abstract and structured argumentation [2, 3, 4].

Note that the body of the second rule in P_{AF2} can be placed directly inside the first rule, resulting in a single-rule APU that can be used to compute the defeated arguments (and thus, through complementation, also the accepted arguments):

$$\mathsf{defeated}(X) \leftarrow \mathsf{attacks}(Y, X), \neg\, \mathsf{defeated}(Y). \qquad (P_{\mathsf{AF}})$$

Now compare the single-rule program P_{AF} with the following single-rule program P_{WM}:

$$\mathsf{win}(X) \leftarrow \mathsf{move}(X, Y), \neg\, \mathsf{win}(Y). \qquad (P_{\mathsf{WM}})$$

We can view P_{WM} as a *game processing unit* (GPU) that specifies the solutions of a two-player game: A *position* x in the game is *winning* (short: a *win*) if there exists a *move* to a position y such that the new position y is lost for the opponent.[1] In particular, this means that a position x is lost, if there are no more outgoing moves left to play.[2]

Both logic rules can be seen as close relatives, even "identical twins", because they can be understood as syntactic variants of each other, i.e., one can obtain one rule from the other via a straightforward renaming of relation symbols. For a database theoretician this means that both rules specify the same *query* (i.e., the same input-output mapping) up to renaming, as long as the same semantics is applied (e.g., the three-valued well-founded model semantics [5]).

Going Separate Ways. Over the years, both logic rules and the argumentation-theoretic and game-theoretic frameworks they represent have received considerable attention from their respective communities. Somewhat surprisingly, however, there seems to be little or no work that discusses these rules *together* and spans across the different communities.

Consider, e.g., the second rule (P_{WM}): It has played a key role in the logic programming, nonmonotonic reasoning, and database communities in their quest to find the "right" semantics for unstratified rules with recursion through negation. Recall that a *stratified* logic program P can use both recursion and negation, but only in a "layered" manner, i.e., where the rule-goal graph of P must not contain negative cycles [6]. For unstratified programs such as P_{AF} and P_{WM}, two declarative semantics emerged as the most popular in the 1990s: The more expressive *stable-model semantics* [7] (used in *answer set programming*) and the more skeptical *well-founded semantics* [5]. For the latter, the win-move rule P_{WM} has been the poster-child example because its unique three-valued model assigns *True*, *False*, and *Undefined* to $\mathsf{win}(x)$ iff a position x in the given game graph is *won*, *lost*, or *drawn*, respectively. In other words, P_{WM} *solves games* and thus indeed is a GPU (game processing unit) when evaluated under the well-founded semantics.

Similarly, the twin rule P_{AF} is an APU that solves argumentation frameworks: Its well-founded model yields the *grounded extension* (or grounded *labeling*) [8, 9] where an argument x is *defeated*, *accepted*, or *undecided* iff $\mathsf{defeated}(x)$ is *True*, *False*, and *Undefined*, respectively.

Although close connections between formal argumentation on the one hand, and logic programming, database theory, and game-theory on the other have been known for a long

[1]In draw-free games the complement of winning is losing; otherwise the complement of winning is losing or drawing.
[2]For example, a *checkmate* position in chess!

time [1, 9, 10, 2], the overlap and cross-fertilization between them appears to be smaller than one might expect. In particular, we could not find works that discuss P_{AF} and P_{WM} together, despite (or maybe because of) the fact that these rules can be viewed as syntactic variants of the same underlying query.

In database theory, the win-move query expressed by P_{WM} has also been used to study the *expressive power* of query languages [11, 12] and to develop a unified *provenance model* that can explain the presence and absence of query answers [13, 14]. The game-theoretic notions and concepts developed in these and other database and game-theory papers [15, 16] seem to carry over to argumentation theory and may lead to new insights and results in formal argumentation. Conversely, related notions studied in argumentation theory may carry over to database theory and applications thereof.

Contributions and Game Plan. The purpose of this paper is to foster a "family reunion" of sorts with the goal of developing new insights and findings through cross-fertilization, i.e., by transferring concepts, ideas, and results between communities. Our game plan is as follows:

- In Section 2 we recall some standard definitions of argumentation frameworks and introduce basics notions from two-player games, i.e., the win-move game defined by P_{WM}.
- We kick off the family reunion in Section 3 by describing the nature of the correspondence between the two rules P_{AF} and P_{WM} and their associated semantics: We show how the grounded extension of an argumentation framework AF corresponds to the well-founded model of an associated win-move game, the *Defeatist's Game* DG. In that game, the *attack* edges of AF are reversed and interpreted as *defeated_by* moves in DG. We illustrate this correspondence with a running example.
- We then introduce another "lost sibling" of the family, i.e., a rule P_{ker} that can be used to compute the *graph kernels* of the move-relation. This gives rise to another correspondence, this time between the *stable extensions* of AF (which coincide with the stable models of P_{AF}) and the graph kernels of DG, which one obtains from the stable models of P_{ker}.
- The correspondence to graph kernels allows us to transfer an important *Decomposition Theorem* [15] from game-theory to argumentation frameworks. It turns out that the well-founded model of P_{WM} yields this decomposition—and thus further insights into an argumentation framework AF—"for free".
- In Section 4 we introduce *query evaluation games*, which are a means to reduce query evaluation in databases to solving games. We also briefly compare this to related notions in argumentation, e.g., assumption-based argumentation.
- Finally, in Section 5 we discuss how game-based notions of *data provenance* [13] lend themselves to reinterpretation in argumentation frameworks and thus to new insights and applications in AF. In particular the *provenance structure* of a solved game graph consists of different *move types*, i.e., *winning* moves, *delaying* moves, and *bad* moves. This edge structure seems to be unexplored in argumentation frameworks so far and thus constitutes another example of cross-fertilization between the different communities.

2. Preliminaries

This section provides the basic graph-theoretic definitions and results of abstract argumentation frameworks and win-move games. The definitions are based on those given in [17, 1] and [13, 12], respectively. In this section, we also introduce a running example of an argumentation framework and a win-move game, shown in Figure 1.

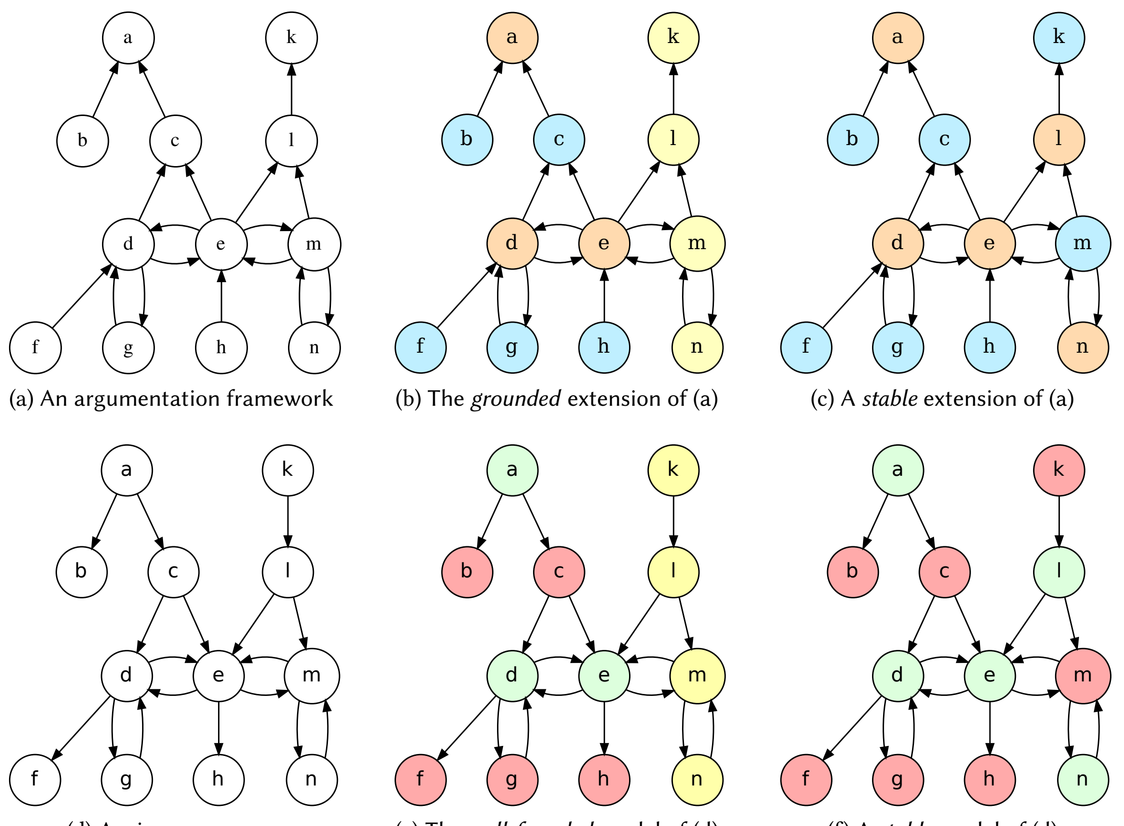
Argumentation Frameworks. An argumentation framework AF is a finite, directed graph $G_{\mathsf{AF}} = (V, E)$, whose vertices V denote atomic *arguments* and whose edges $E \subseteq V \times V$ denote a binary *attacks* relation. An edge $(x, y) \in E$ states that argument x *attacks* argument y. An example AF is shown in Figure 1a, consisting of arguments a, b, c, . . . , and their attack relation $\mathsf{attacks(b, a)}, \mathsf{attacks(c, a)}, \ldots$ A subset $S \subseteq V$ of acceptable arguments is called an *extension*, provided S satisfies certain conditions. An extension S is said to *attack* an argument x if an argument $y \in S$ attacks x. The *attackers* of S are the arguments that attack at least one argument in S. An extension S is *conflict-free* if no argument in S attacks another argument in S. Conversely, an extension S *defends* an argument x if it attacks all attackers of x. The arguments *defended by* S are those that S defends; this is often described via the *characteristic function* of an argumentation framework.

Dung [1] and others have defined a variety of classes of extensions each of which are referred to as different *extension semantics*. In the following, we focus on two popular extension semantics, i.e., the skeptical *grounded extension* and the more expressive *stable extensions*. While every AF has a unique grounded extension (corresponding to the unique 3-valued well-founded model of P_{AF}), it may have many stable extensions (including none). An extension $S \subseteq V$ is called *stable* if it is conflict-free and attacks every argument not in S (i.e., all nodes in $V \setminus S$). It was shown by Dung [1] that stable extensions correspond to the stable models of logic programs, and similarly, that the grounded extension corresponds to the well-founded model.

Similar to an extension, a (reinstatement) *labeling* [17] assigns each argument in an AF one of three labels, `in`, `out`, or `undec`, such that an argument is labeled `in` if all its attackers are labeled `out`, an argument is labeled `out` if it has an attacker that is labeled `in`, and an argument is labeled `undec` otherwise. Caminada showed [17] that a labeling without any `undec` arguments corresponds to a stable extension, and similarly, a labeling that maximizes `undec` arguments corresponds to the grounded extension. Figure 1b shows the *grounded* extension of the argumentation framework in Figure 1a using colors for labels (blue for `in`, orange for `out`, yellow for `undec`). Similarly, Figure 1c shows one of two possible *stable* extensions of Figure 1a using the same coloring scheme. Note that *accepted* and *defeated* correspond to labels `in` and `out`, respectively.

In [1], Dung noted that logic programming corresponds to a form of argumentation and vice versa: An AF can be evaluated via the logic program (meta-interpreter) P_{AF}. In particular, he shows that S is a stable extension of an AF iff S corresponds to a stable model of P_{AF}. Similarly, S is a grounded extension of an AF iff S corresponds to the well-founded model of P_{AF}.

Win-Move Games. A (win-move) game can be defined as a graph $G_{\mathsf{WM}} = (V, E)$ such that two players move alternately between a finite set of *positions* V along *move* edges $E \subseteq V \times V$. Each position $p_0 \in V$ defines a game over the graph G_{WM} when starting from position p_0.


(a) An argumentation framework (b) The *grounded* extension of (a) (c) A *stable* extension of (a)

(d) A win-move game (e) The *well-founded* model of (d) (f) A *stable* model of (d)

Figure 1: (a) The digraph in the upper left defines an argumentation framework $G_{\text{AF}} = (V, E)$ with arguments V and attacks E. (b) Its *grounded extension* is shown in the upper middle using a color-based labeling where "`in`" arguments are blue, "`out`" arguments are orange, and "`undec`" arguments are yellow. (c) G_{AF} has two *stable extensions*, one of which is shown in the upper right. (d) The graph in the lower left defines a win-move game $G_{\text{WM}} = (V, E)$ with moves E between positions V. (e) The *solved* game under the well-founded semantics is shown in the lower middle using a color-labeling where *won* positions are green, *lost* positions are red, and *drawn* positions are yellow. (f) G_{WM} has two *stable models* of rule P_{WM}, one of which is shown in the lower right.

A *play* $\pi = p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \cdots$ is a finite or infinite sequence of edges from V where for all $i = 0, 1, 2, \ldots$, the edge $p_i \rightarrow p_{i+1}$ is a move $(p_i, p_{i+1}) \in E$. A play π is *complete* if it is infinite or if it ends after $n = |\pi|$ moves in a sink of the game graph. The player who cannot move *loses* the play π, while the previous player who made the last move *wins* π. A play π of infinite length is a *draw*, which in finite game graphs G_{WM} means that E must have a cycle. Figure 1d shows an example game graph (similar but different to the AF graph in Figure 1a). Assume e is the start position for Player I. In the play $\pi_1 = \text{e} \rightarrow \text{d} \rightarrow \text{f}$, Player I moves to d and Player II moves to f. Since Player I cannot move from f, Player II wins. However, in $\pi_2 = \text{e} \rightarrow \text{h}$, Player II cannot move, so π_2 is won for Player I. Thus, from position e, the "best" move is e $\rightarrow$ h, whereas the other moves are "bad": e $\rightarrow$ d loses, while e $\rightarrow$ m only draws (if Player II sticks to m $\rightarrow$ n). To determine the true value of a position $p \in V$, bad moves are not considered. Instead, only plays are considered where both players play optimally (or at least "good enough") so that the best possible outcome is guaranteed for both.

The *solution* of a game consists of assigning one of the labels `win`, `lose`, or `draw` to each position in the game graph G_{WM}. Specifically, a position $p \in V$ is assigned the label `win` if Player I has a *winning strategy*, i.e., can force a win, no matter how Player II moves. Conversely, p is labeled `lose`, if Player II can force a win, no matter how Player I moves. If neither player can force a win, a position is a `draw`. Figure 1e shows the labeled solution for the game graph of Figure 1d where colors are used for labels (green $\sim$ `win`, red $\sim$ `lose`, yellow $\sim$ `draw`). A game can be solved by evaluating the rule P_{WM} from Section 1 under the well-founded semantics. Similar to the AF in Figure 1a, the win-move rule under the stable-model semantics results in two stable models. However, the interpretation of the stable models (e.g., Fig. 1f) is less intuitive or natural for win-move games, and it is the well-founded model of P_{WM} that correctly identifies all won, lost, and drawn positions.

3. A Family Reunion: Argumentation, Games, and Kernels

Starting point for our "family reunion" is the observation that the argumentation framework meta-interpreter[3] given by the single unstratified rule P_{AF} and the "game engine" (or GPU) given by P_{WM} are syntactic variants of each other and thus specify the same *query* in the sense of database theory. In particular, this means that for a given logic programming semantics such as the well-founded or stable semantics, the resulting models of P_{AF} and P_{WM} are *isomorphic*.

One minor twist in this correspondence is that the two different interpretations of a given graph $G = (V, E)$, i.e., either as an argumentation framework G_{AF} or as a win-move game G_{WM} have *reversed* edge directions (cf. Fig. 1): In the argumentation framework in Fig. 1a, e.g., the edge b $\rightarrow$ a in E means that argument b *attacks* a. Since the argument b has no attackers, it follows that b is accepted (color-labeled blue in Fig. 1b and Fig. 1c). In the corresponding win-move graph, however, there is a *reversed* edge a $\rightarrow$ b, i.e., indicating that a player can move from position a to b in the game. Since there are no further moves possible from b, it follows that in the canonical game semantics, b is lost (red in Fig. 1e and Fig. 1f).

We can understand this edge reversal better by rewriting the APU rule P_{AF} as follows:

$$\mathsf{defeated}(X) \leftarrow \mathsf{attacked\_by}(X, Y), \neg\, \mathsf{defeated}(Y). \tag{$P_{\mathsf{AF}'}$}$$

Here we replaced the "forward attack" relation $\mathsf{attacks}(Y, X)$ (cf. the three AF graphs in the top row of Fig. 1) by a "backward reasoning" relation $\mathsf{attacked\_by}(X, Y)$ analogously to the game relation $\mathsf{move}(X, Y)$ in P_{WM} (see the three game graphs in the bottom row of Fig. 1).

With this renaming in place, the following correspondences between the well-founded model of $P_{\mathsf{AF}'}$ and P_{WM} are immediate: Arguments in the AF correspond to positions in the game. The reverse of the attacks relation, $\mathsf{attacks}^{-1}(Y, X)$ is the relation $\mathsf{attacked\_by}(X, Y)$ and corresponds to $\mathsf{move}(X, Y)$, i.e., the moves of the game.

[3]or *argument processing unit* (APU) as Dung [1] calls it

The Defeatist's Game. Despite the natural correspondence between grounded extensions of argumentation frameworks and solved win-move games in Fig. 1, this isomorphism under the well-founded semantics can appear unintuitive at first: How is it that *accepted* arguments (e.g., the blue nodes b and c in Fig. 1b) correspond to *lost* positions (the red nodes b and c in Fig. 1e) in the game, while *defeated* arguments (e.g., the orange nodes d and e in Fig. 1b) correspond to *won* positions (the green nodes in Fig. 1e)?

The answer is that the win-move game G_{WM} we are playing to solve an argumentation framework is in fact a "defeatist's game" G_{DG}: In this game, when Player I (the defeatist) starts to move from a position (e.g., a in Fig. 1), the claim made for this position is: a is *defeated*! The opponent, Player II, begs to differ and tries to demonstrate that argument a is *accepted*. By design, the defeatist's game DG from a position x is *won* (for Player I) if x is *defeated* in the grounded extension. Similarly it is *lost* (for Player I) from position x, i.e., argument x is *accepted*, if Player II has a winning strategy, i.e., can force a win. Finally, a *drawn* position x in this game means that argument x is undecided in the grounded semantics and no player can force a win.

What is gained by viewing an argumentation framework as a game? After all, game-theoretic treatments have been part of formal argumentation from the very beginning [1]. First, it appears that our particular rendering, i.e., the defeatist's game G_{DG} hasn't been considered before, despite the fact that it is the most immediate path from AF to game theory since both formalizations are equivalent, as sketched above. Therefore, concepts and results from game theory can be *directly* applied to argumentation frameworks. One such natural notion, discussed below, is the *length* of a position; another one is the *provenance structure* of a game, which can be obtained from an (implied) edge coloring of solved games (cf. Fig. 4 and Fig. 5).

Digraph Kernels: Another Lost Twin Rejoins. One of the earliest mathematical tools devised to study games are *kernels* of directed graphs [18, 19, 15, 20, 21]. As it turns out, by studying kernels of game graphs, additional results for argumentation frameworks can be (re-)discovered, further strengthening the family ties between the different communities. For example, the concepts and results about kernels described by Fraenkel in [15] demonstrate that at the core, the three formalisms (argumentation frameworks, win-move games, and digraph kernels) are intimately related and based on a common underlying graph-theoretic machinery.

A *kernel* of a directed graph $G = (V, E)$ is a subset $K \subseteq V$ that is both independent and dominating [15]. Here, *independent* means that no edges exists between vertices in K, and *dominating* means that every vertex in $V \setminus K$ has a follower (successor) in K, i.e., an edge into the kernel K. If we view G as an argumentation framework G_{AF}, but with edges representing the *reversed* attacked_by relation (as in Fig. 1d), then each kernel K corresponds to a *stable extension* of G_{AF}. Similarly, if G is viewed as a win-move game G_{WM} (again Fig. 1d), then K is the set of *lost* nodes in a stable model of P_{WM} (e.g., Fig. 1f depicts one such kernel).

The following unstratified rules P_{ker2} state that if there is an edge from x to y, where y is in the kernel ker, then x cannot be in the kernel but instead is in the kernel *complement* $\mathsf{ker}^{\mathsf{c}}$:

$$\begin{aligned} \mathsf{ker}^{\mathsf{c}}(X) &\leftarrow \mathsf{edge}(X, Y), \mathsf{ker}(Y). \\ \mathsf{ker}(X) &\leftarrow \neg\, \mathsf{ker}^{\mathsf{c}}(X). \end{aligned} \tag{P_{ker2}}$$

As in the case of Dung's APU rules, we can "plug in" the body of the second rule into the first, to obtain another syntactic variant of P_{AF} and P_{WM} as follows:

$$\mathsf{ker}^{\mathsf{c}}(X) \leftarrow \mathsf{edge}(X,Y), \neg \mathsf{ker}^{\mathsf{c}}(Y). \qquad (P_{\mathsf{ker}})$$

It is easy to see that the stable models of this program can be used to compute all kernels of a digraph: For a given stable model, the kernel consists of all vertices that are not in the kernel complement $\mathsf{ker}^{\mathsf{c}}$. Note further that P_{ker} is a syntactic variant of P_{WM} (and thus also of P_{AF}):

$$\mathsf{win}(X) \leftarrow \mathsf{move}(X,Y), \neg\, \mathsf{win}(Y). \qquad (P_{\mathsf{WM}})$$

In other words, the *winning* positions of a stable model of P_{WM} (green in Fig. 1) are precisely the nodes *not* in the graph kernel of the move relation, while the *lost* positions (red in Fig. 1) are the nodes *in* the kernel. Summarizing, we have three syntactic variants $P_{\mathsf{AF'}}$, P_{WM}, and P_{ker} of a single underlying query "$\mathsf{q}(X) \leftarrow \mathsf{e}(X,Y), \neg\mathsf{q}(Y)$" that encapsulates the common core of all three frameworks: Using the well-founded semantics, we can solve win-move games and compute grounded extensions. Via stable models, we can compute the stable extensions of argumentation frameworks and the kernels of digraphs. There are other results from game theory that carry over to argumentation frameworks as well, as shown next.

A Decomposition Theorem. In [15], Fraenkel proves various results regarding structural properties of digraphs kernels. This enables another route to transfer results and insights from games and kernel theory to argumentation frameworks. In [15], Fraenkel shows that:

(i) Any digraph $G = (V, E)$ can be partitioned in $O(|E|)$ time into subsets $S_1, S_2, S_3 \subseteq V$ such that S_1 lies in all of the kernels (= *lost* in the well-founded model of P_{WM}), S_2 lies in the complements of all the kernels (= *won* in the well-founded model), and on S_3 the kernels may be non-unique (= *drawn* in the well-founded model of P_{WM})—e.g., see Fig. 1e;

(ii) G can be decomposed into two subgraphs: G_1 with vertex set $S_1 \cup S_2$, which has a unique kernel (= the *won* and *lost* positions of the well-founded model), and G_2 with vertex set S_3 (the *drawn* positions), such that any kernel K of G is the union of the unique kernel of G_1 and some kernel of G_2 (if it exists);[4] and . . .

(iii) . . . since G_1 is unique, the total number of kernels in G (= number of stable extensions of P_{AF}) is determined by the number of kernels of G_2.

These and other results [15, 19, 16] reinforce the fundamental connections that exist between win-move games and argumentation frameworks, e.g., via well-founded and stable models. These connections may also shed new light on the intricate connections between forms of *skeptical* and *credulous* acceptance, and may lead to new, efficient inference algorithms.

4. On Query Evaluation Games and Structured Argumentation

In connecting logic programming and AF, Dung [1] states that an argumentation system consists of two essential components: an *argument generation unit* (AGU) to generate arguments and their

[4]In Fig. 1e, the G_1 nodes are red and green, while the G_2 nodes are yellow.

attack relationships (e.g., from a logic program), and an APU (P_{AF}) that finds the acceptability of arguments produced by the AGU. We describe prior work [12, 13] (similar to an AGU) on translating database queries into graphs in a *game normal form.* We also briefly highlight similarities of this translation with assumption-based approaches [22] and with approaches for showing correspondences between logic programming and argumentation semantics [10].

Games vs Stratified Rules. During the late 1980s and through the 1990s, the logic-programming and non-monotonic reasoning community developed and studied a number of proposals for a canonical semantics for rules with recursion through negation. Proponents of the stratified semantics [6] simply ruled out such unstratified programs. An earlier paper [23] claimed that stratified rules express all of FIXPOINT [24], which is a large class of database queries with PTIME data complexity. As shown in [11], however, the FIXPOINT query that computes the game positions for which a player has a *winning strategy* is *not* expressible by stratified rules, therefore demonstrating that stratified Datalog is strictly less expressive than FIXPOINT.

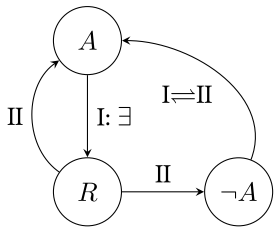
Win-Move: A Universal Query Engine. The rule P_{WM} turns out to also be a *universal query engine* in that every n-ary FIXPOINT query with answer $Q(D)$ of a query Q over database D can be expressed in *game normal form* P_{WM}: $\mathsf{win}(\bar{X}) \leftarrow \mathsf{move}(\bar{X}, \bar{Y}), \neg\mathsf{win}(\bar{Y})$, where $\bar{X}$ and $\bar{Y}$ are n-tuples of variables, P_{WM} is the only recursive rule, and $\mathsf{move}(\bar{X}, \bar{Y})$ is definable via a quantifier-free formula over the input database D [12]. Positions of the game correspond to (ground-instantiated) rules, head atoms, and body literals, and playing this query evaluation game mimics a form of SLD(NF) resolution. Fig. 2 summarizes the translation[5]: Player I tries to show that an atom $A \in Q(D)$ is in the answer by selecting a rule R that derives A. Player II then tries to refute this by selecting a subgoal A' of R that is not satisfied, after which Player I tries to prove A' and so on. To further illustrate the basic idea of this game, let

$$\begin{aligned} r_{1_X}: &\quad \mathsf{q}(X) \leftarrow \mathsf{s}(X), \neg\mathsf{t}(X). \\ r_2: &\quad \mathsf{s}(\mathsf{a}). \end{aligned}$$

be two rules consisting of a query q and a single fact $\mathsf{s}(\mathsf{a})$. Assume Player I wants to show that $\mathsf{q}(\mathsf{a})$ is an answer and so starts the query game by moving from position $\mathsf{q}(\mathsf{a})$ to the rule position r_{1_a}. Player II then selects a subgoal of r_{1_a}, either $\mathsf{s}(\mathsf{a})$ or $\neg\mathsf{t}(\mathsf{a})$, to refute. Assuming Player II picks $\mathsf{s}(\mathsf{a})$ to refute, Player I then selects the rule r_2, denoting the fact $\mathsf{s}(\mathsf{a})$, ending the game immediately, as there are no additional moves from r_2 (facts have an empty rule body). Thus Player II loses and Player I wins. If Player II had instead chosen to refute $\neg\mathsf{t}(\mathsf{a})$, Player I could then have moved to position $\mathsf{t}(\mathsf{a})$ (forcing Player II to justify $\mathsf{t}(\mathsf{a})$). Since $\mathsf{t}(\mathsf{a})$ is not supported by any rules, this play also ends, and again Player I wins. Since Player I can force a win, no matter how Player II moves, $\mathsf{q}(\mathsf{a})$ is an answer to the query.

Fig. 3 gives another example of the translation to game form for a propositional logic program, similar to the examples used for translating logic programs to argumentation frameworks in [22, 10]. The program P in Fig. 3a consists of four rules labeled r_1 through r_4. Using the translation in Fig. 2, the game for P is shown as a solved game graph in Fig. 3b. Note that under the well-founded semantics, P has a single model where a and b are *Undefined*, c is *True*, and d is *False*. This model exactly corresponds to the solved game graph in Fig. 3b where c is a

[5]This is a simplified version of the translation given in [13].



(a) Translation pattern

Move	Claim made by making the move
$A \overset{\exists}{\rightsquigarrow} R$	"Atom A holds because of rule R!"
$R \rightsquigarrow A$	"Not satisfied because a subgoal A is false!"
$R \rightsquigarrow \neg A$	"Not satisfied because a subgoal $\neg A$ is false!"
$\neg A \rightsquigarrow A$	"Not true, subgoal A is false! (Prove me wrong)"

(b) Claims made with a move

Figure 2: (a) Move types of the query evaluation game. (b) The claims made when playing the game. Moving along an edge, a player aims to verify a claim, thereby refuting the opponent. Initially, Player I is a verifier, trying to prove A, while Player II tries to spoil this and refute Player I.

winning position, d is a losing position, and both a and b are drawn. The solved game graph (via position labels and move edges) also contains explanations for the truth-values of atoms relative to the rules of a program. For instance, c is true (winning) because it is derived from rule r_4 (i.e., there is a move from position c to position r_4), whose only subgoal $\neg d$ holds (i.e., d is false) since d is not supported by any rules.

Similarities to LP $\rightsquigarrow$ AF Translations. One of the commonly used transformations from logic programs to argumentation frameworks [22, 10] shares some similarities with the query-to-game translation above. In these transformations, an argumentation framework for a propositional logic program consists of arguments representing the rules of the program such that an argument A attacks an argument B if A's corresponding rule derives an atom that contradicts a premise of B. The rules of Fig. 3a would be translated to the following AF attacks graph:

$$\begin{tikzcd} r_2 \arrow[r, bend left] & r_3 \arrow[l, bend left] & r_1 \arrow[l] & r_4 \arrow[l] \end{tikzcd}$$

In [10], an argument is associated with each rule's conclusion (e.g., a is the conclusion of rule r_1), and a procedure is given that relates the conclusions associated with an extension of the argumentation framework under a given semantics to the set of answers of the corresponding logic program under an equivalent semantics (e.g., grounded extensions with well-founded semantics, and stable extensions with stable models). The grounded extension of the AF above, using the approach in [10], yields the well-founded model of P. Similar approaches have been developed, e.g., in [22] for assumption-based argumentation frameworks.

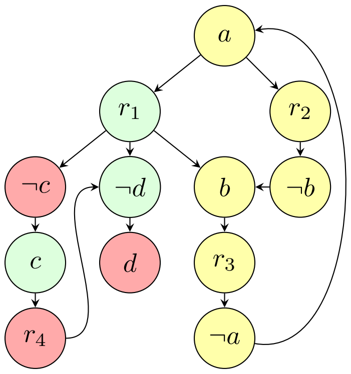
5. Provenance and Explanations: Solved Games to the Rescue!

Provenance-based annotations can be added to game graphs [13] and then help to explain the value of positions. We show how these annotations can be adopted directly to explain why arguments are accepted or defeated in grounded AF extensions.

The Length of Plays. Consider the solved game from Fig. 1e, which is shown with additional annotations in Fig. 4a (explained below). Games can be solved in stages. Positions b, f, and h are immediately lost (red nodes): No moves are possible from sink nodes. Next we can infer that

$$\begin{aligned} r_1: &\quad a \leftarrow \neg c, \neg d, b \\ r_2: &\quad a \leftarrow \neg b \\ r_3: &\quad b \leftarrow \neg a \\ r_4: &\quad c \leftarrow \neg d \end{aligned}$$

(a) Example propositional program P.



(b) Corresponding solved win-move game of P.

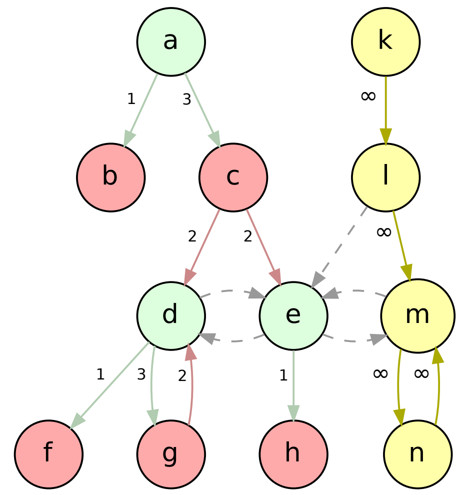
Figure 3: (a) Example of a simple propositional program P. (b) The corresponding translation of P to a solved win-move graph according to the win-move normal form of Fig. 2.

positions that have an outgoing move to a lost position (for the opponent) are definitely won (green). Based on our initial determination that b, f, and h are lost, it then follows that a, d, and e are won. What is the status of the remaining positions? The status of c is now determined since *all* outgoing moves from c definitely end in a node that is won for the opponent (d and e are already green), so c is objectively lost. Solving a game can thus proceed by *iterating* the following two labeling rules in stages:[6]

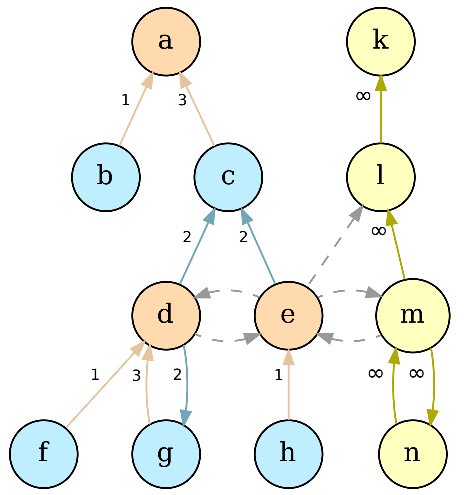
- Position x is *won* (green) if $\exists$ move $x \to y$ and position y is known to be lost (red)
- Position x is *lost* (red) if $\forall$ moves $x \to y$, position y is known to be won (green)

With each position x we can associate its *length* [13], i.e., the stage number when its label first became known. Similarly, we can associate a length with each move, indicating at what stage its *type* (i.e., edge color) became known. In Fig. 4a, edges into (red) sinks are winning moves (colored green) and labeled with length = 1, so a, d, e and those edges to sink nodes all have length = 1. In the next stage, all successors of c are won, so c itself must be lost, and its length is 1 + the *maximal* length of any of its succcessors. Similarly, for won x, length(x) = 1 + the *minimal* length of any lost successor, etc. After a fixpoint is reached, all remaining unlabeled nodes correspond to *drawn* positions (colored yellow). We set length = ∞ for drawn positions, since neither player can force a win, but both can avoid losing by repeating moves indefinitely. The length ℓ of an edge $x \xrightarrow{\ell} y$ indicates how *quickly* a player can force a win, or how long a player can delay a loss: In Fig. 4a that position a can be won in as few as one move (to position b), whereas all moves from c delay for only two moves. Similarly, while there is a loop between d and g, position g is lost in only two moves.

[6]This method corresponds to the *alternating fixpoint* procedure [25] and to Algorithm 6.1 for computing the *grounded labeling* of an argumentation framework in [9].



(a) The solved win-move game with move-edge lengths and corresponding (colored) edge types.



(b) The grounded argumentation framework with attack-edge lengths and corresponding (colored) edge types.

Figure 4: (a) The *solved game* G_{WM} from Fig. 1e with the length ℓ of an edge $x \xrightarrow{\ell} y$ indicating how quickly one can force a win, or how long one can delay a loss, with that move, together with color-labeled edges denoting the type of move being made [13]. (b) The *grounded extension* of G_{AF} from Fig. 1b with similar lengths and colors added to attacks edges [26].

Solved Games Explain it All! Solved games have an intriguing property: Node labels (colors) induce different *edge types*, which in turn can be used to *explain why* a position is won, lost, or drawn, respectively. Fig. 5 shows how edge types are determined from the color-labels of incident vertices. These types, in turn, induce a downstream *provenance (explanation) subgraph* that provides the *justification* or *explanation* for the status of any $x \in V$.[7] The provenance of position x in the solved game is the subgraph reachable from x via certain *regular path queries* (RPQs), where an RPQ is a regular expression R over the labels of an edge-labeled directed graph G. The answer to an RPQ R, given a start node x, is the set of nodes y reachable along simple paths from x whose (concatenated) labels match R. Assuming edges are labeled with their edge-type colors, the provenance of a won position x matches the RPQ x.green.(red.green)*, lost positions match x.(red.green)*, and drawn positions match x.yellow$^+$. The following examples are drawn from Fig. 4a.

- The provenance of e consists of the single path e $\rightarrow$ h: e wins because h is lost.
- The provenance of d consists of paths d $\rightarrow$ f and d $\rightarrow$ g: d wins because f and g are lost.
- The provenance of g consists of the path g $\rightarrow$ d $\rightarrow$ f: g is lost ultimately because of f.

Interpreting Paths in AF. The correspondence described in Section 3 allows us to apply edge lengths and typed edges, e.g., by playing the defeatist's game, directly to grounded AFs. An

[7]These explanations are similar in spirit to dialog trees as described in [27].

	y ***won*** (W)	y ***drawn*** (D)	y ***lost*** (L)
x ***won*** (W)	*bad*	*bad*	g: ***winning***
x ***drawn*** (D)	*bad*	y: ***drawing***	*n/a*
x ***lost*** (L)	r: ***delaying***	*n/a*	*n/a*

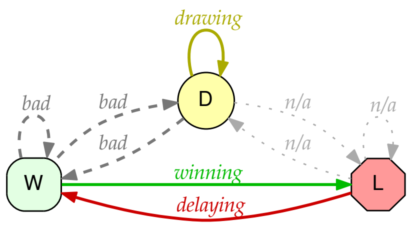


Figure 5: Depending on node labels, moves $x \to y$ are either *winning* (green) (W $\overset{g}{\rightsquigarrow}$ L), *delaying* (red) (L $\overset{r}{\rightsquigarrow}$ W), or *drawing* (yellow) (D $\overset{y}{\rightsquigarrow}$ D). All other moves are either "*bad*" (allowing the opponent to improve the outcome), or cannot exist (*n/a*) due to the nature of the game: e.g., if x is lost, then there are only delaying moves (i.e., ending in won positions y for the opponent) [13].

example of the grounded AF in Fig. 1b is shown with its *provenance* information in Fig. 4b. Using this approach, the lengths assigned to attack edges in Fig. 4b correspond to stages of the alternating fixpoint computation applied to attacked_by edges. Thus, edge lengths have a similar interpretation in grounded extensions as in win-move graphs in that they represent the stages in which argument labels become known. Another interpretation of attack-edge lengths is as follows. The length ℓ of an attack edge $x \xrightarrow{\ell} y$ states that argument x is the ℓ-th argument along an *argument attack chain*, i.e., a path composed of only alternating orange (accepted to defeated) and blue (defeated to accepted) edges starting from an accepted argument without any attackers and ending at the last possible defeated argument. Similar to lengths in win-move games, the path (on the attack chain) leading into x is assumed to be the shortest such path. For instance, in Fig. 4b, argument b is the first argument in the argument attack chain that terminates at argument a, and thus, the attack edge b $\to$ a has the value $\ell = 1$. A similar situation exists for arguments f and h. Argument d is the second argument along the (shortest) argument attack chain f $\to$ d $\to$ c $\to$ a, and thus, the edge d $\to$ c has the value $\ell = 2$. Note that argument g lies on a non-simple argument attack chain where d is the second argument (hence, $\ell = 2$ for the edge d $\to$ g), making g the third argument on the chain (hence, $\ell = 3$ for the edge g $\to$ d). Similarly, c is the third argument on two separate argument attack chains of the same path length, and so $\ell = 3$ for the edge c $\to$ a. Thus, the notion of edge lengths, along with argument labels, can help to clarify the role (i.e., a part of the impact) played by each argument in a grounded argumentation framework.

Explaining Acceptance and Defeat. Similarly to win-move games, an argument's status in the grounded extension in Fig. 4b can be explained by an RPQ-definable subgraph. To obtain the provenance of defeated and accepted arguments, we adjust the RPQ examples by fixing the end vertex (as opposed to the start vertex) of each RPQ. Assuming edges are labeled with their edge-type colors, the provenance of a defeated position x matches the RPQ orange.(blue.orange)$^*.x$, accepted positions match (orange.blue)$^*.x$, and drawn positions match yellow$^+.x$. For Fig. 4b:

- The provenance of e consists of the single path h $\to$ e: e is defeated since h is accepted.
- The provenance of d consists of f $\to$ d and g $\to$ d: d is defeated since f and g are accepted.
- The provenance of g consists of f $\to$ d $\to$ g: g is accepted ultimately because of f.

As in Fig. 4a, the edge-types of Fig. 4b filter out *non-relevant* attacks within the AF to focus specifcally on those arguments that contribute to acceptance or non-acceptance of an argument.

6. Summary and Conclusion

Games have played an important role in the study of database query languages and in finding acceptable arguments in argumentation frameworks. While these approaches share a common history, it appears the communities have largely separated (or possibly haven't had much overlap to begin with). The goal of this paper was to demonstrate some of the deep underlying connections that exist, exemplified by the win-move rule P_{WM} and its twin-rules[8] P_{AF} and P_{ker}, and to use these to reestablish the lost family ties. We believe that a fruitful line of research can be established by reconnecting the communities and transfering concepts, tools, and results between them, leading to further cross-fertilization and new insights.

The results from database theory for the query P_{WM} presented here should look familiar to researchers in formal argumentation. Our starting point was the straightforward link between P_{AF} and P_{WM}: Twin rules that have their distinct histories and applications in separate communities, but that haven't been studied together, at least to the best of our knowledge. Under the well-founded semantics, the solved win-move game G_{WM}—with its additional structure and "built-in" provenance–corresponds to the grounded labeling of an argumentation framework [17, 9]. The additional provenance structure induced by edge types ("not all edges are created equal") [13] and the decomposition results about graph kernels [15] immediately suggest corresponding structures for argumentation frameworks, both of which appear to be new results in abstract argumentation.

Finally, we invite feedback and welcome collaboration opportunities on these and similar questions. An open source demonstration using Jupyter notebooks, including the example from Fig. 4, is available [28]. We plan to evolve and expand these notebooks as teaching materials for some of our undergraduate and graduate courses, covering knowledge representation and reasoning, information modeling, and database theory.

References

[1] P. M. Dung, On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games, AI 77 (1995) 321–357.

[2] P. Baroni, D. Gabbay, M. Giacomin, L. v. d. Torre, Handbook of Formal Argumentation, London, England: College Publications, 2018.

[3] P. Baroni, F. Toni, B. Verheij, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games: 25 years later, Argument & Computation 11 (2020) 1–14.

[4] P. Besnard, C. Cayrol, M.-C. Lagasquie-Schiex, Logical theories and abstract argumentation: A survey of existing works, Argument & Computation 11 (2020) 41–102.

[5] A. Van Gelder, K. A. Ross, J. S. Schlipf, The Well-founded Semantics for General Logic Programs, Journal of the ACM 38 (1991) 619–649.

[8] Identical triplets actually.

[6] K. R. Apt, H. A. Blair, A. Walker, Towards a Theory of Declarative Knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, 1988, pp. 89–148.

[7] M. Gelfond, V. Lifschitz, The Stable Model Semantics for Logic Programming, in: ILPS, 1988, pp. 1070–1080.

[8] M. W. A. Caminada, D. M. Gabbay, A Logical Account of Formal Argumentation, Studia Logica 93 (2009).

[9] S. Modgil, M. Caminada, Proof Theories and Algorithms for Abstract Argumentation Frameworks, in: Argumentation in Artificial Intelligence, 2009, pp. 105–129.

[10] M. Caminada, S. Sá, J. Alcântara, W. Dvořák, On the Equivalence between Logic Programming Semantics and Argumentation Semantics, Approx. Reasoning 58 (2015) 87–111.

[11] P. G. Kolaitis, The expressive power of stratified logic programs, Information and Computation 90 (1991) 50–66.

[12] J. Flum, M. Kubierschky, B. Ludäscher, Total and Partial Well-Founded Datalog Coincide, in: ICDT, LNCS 1186, Springer, 1997, pp. 113–124.

[13] S. Köhler, B. Ludäscher, D. Zinn, First-order provenance games, in: In Search of Elegance in the Theory and Practice of Computation, LNCS 8000, 2013, pp. 382–399.

[14] S. Lee, S. Köhler, B. Ludäscher, B. Glavic, A SQL-Middleware Unifying Why and Why-Not Provenance for First-Order Queries, in: ICDE, 2017, pp. 485–496.

[15] A. Fraenkel, Combinatorial Game Theory Foundations Applied to Digraph Kernels, Electronic Journal of Combinatorics 4 (1997) 1–17.

[16] J. Flum, Games, Kernels, and Antitone Operations, Order 17 (2000) 61–73.

[17] M. Caminada, On the Issue of Reinstatement in Argumentation, in: Logics in Artificial Intelligence, LNAI 4160, Springer, 2006, pp. 111–123. doi:10.1007/11853886_11.

[18] J. von Neumann, O. Morgenstern, A. Rubinstein, Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition), Princeton University Press, 1944.

[19] B. v. Karger, R. Berghammer, Computing kernels in directed bichromatic graphs, Information Processing Letters 62 (1997) 5–11. doi:10.1016/S0020-0190(97)00035-5.

[20] Y. Dimopoulos, A. Torres, Graph theoretical structures in logic programs and default theories, Theoretical Computer Science 170 (1996) 209–244.

[21] E. Boros, V. Gurvich, Perfect graphs, kernels, and cores of cooperative games, Discrete Mathematics 306 (2006) 2336–2354.

[22] P. M. Dung, R. A. Kowalski, F. Toni, Dialectic proof procedures for assumption-based, admissible argumentation, AI 170 (2006) 114–159.

[23] A. K. Chandra, D. Harel, Horn clause queries and generalizations, The Journal of Logic Programming 2 (1985) 1–15. doi:10.1016/0743-1066(85)90002-0.

[24] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[25] A. Van Gelder, The alternating fixpoint of logic programs with negation, Journal of Computer and System Sciences 47 (1993) 185–221.

[26] B. Ludäscher, Y. Xia, Games and argumentation: Time for a family reunion!, in: Fourth Workshop on Explainable Logic-Based Knowledge Representation (XLoKR), 2023.

[27] P. M. Dung, P. Mancarella, F. Toni, Computing ideal sceptical argumentation, AI 171 (2007) 642–674.

[28] Y. Xia, S. Bowers, B. Ludäscher, Games and argumentation demo repository, 2023. github.com/idaks/Games-and-Argumentation-AI3.