Argumentation Using Temporal Knowledge

Nicholas MANN [a], Anthony HUNTER [a]
[a] *Department of Computer Science, University College London, Gower Street, London, WC1E 6BT, UK.*

Abstract. Proposals for logic-based argumentation have the potential to be adapted for handling diverse kinds of knowledge. In this paper, a calculus for representing temporal knowledge is proposed, and defined in terms of propositional logic. This calculus is then considered with respect to argumentation, where an argument is pair $\langle \Phi, \alpha \rangle$ such that Φ a minimally consistent subset of a database entailing α. Two alternative definitions of an argument are considered and contrasted.

Keywords. Argumentation systems, Arguments, Temporal argumentation.

1. Introduction

Argumentation is a powerful tool for reasoning with inconsistent knowledge in a database. There are many examples and types of argumentation (see reviews [1,2,3]). These argumentation systems range in style from the abstract argumentation system of Dung [4], through defeasible argumentation of Garcia and Simari's DeLP [5] to the classical logic approach of Besnard and Hunter's argumentation system [6].

A common definition of an argument is a that of a minimally consistent subset of a database that is capable of proving a given conclusion (eg [6,7]).

Definition 1.1. An **argument** is a pair $\langle \Phi, \alpha \rangle$, such that:

(1) $\Phi \nvdash \bot$
(2) $\Phi \vdash \alpha$
(3) Φ is a minimal subset of a database Δ satisfying (2)

We say $\langle \Phi, \alpha \rangle$ is an argument for α, with Φ being the support of the argument, and α being the conclusion of the argument.

In this definition, classical propositional logic is usually used. However, to express temporal knowledge, we need to extend this definition, and replace propositional logic. One simple idea here is to use first order logic (eg [8]), but full first order logic is a large step to take in order to express temporal knowledge where a simpler definition may suffice. Hence, in this paper we consider some of the space between the established use of propositional logic and the use of first order logic within the above definition.

With the replacement of propositional logic in definition 1.1, $\vdash$ and $\bot$ may have different meanings. This may provide problems, since for example established theorems

may no longer hold, or perhaps an inconsistent subset of the knowledgebase may still prove to be useful as a support. However, these changes also present opportunities to expand upon the definition of an argument and broaden our horizons as to what an argument could be.

To be more specific, this paper uses a calculus capable of expressing temporal knowledge. However, this in turns brings more problems; types of temporal knowledge are diverse, as are the methods of formalising them, and many possible logics and calculuses are available, such as the event calculus of Kowalski and Sergot [9] or the tense logic of Prior [10].

Before picking a method of expressing temporal knowledge, it is useful to consider the type of knowledge we wish to express, and for this purpose we use the example of business news reports, which for example could include phrases such, "Cisco Systems announced year on year sales growth of 12% for Q1". News reports are interesting for several reasons; they can have conflicting information (usually in the form of several different sources providing different information), making them suitable for consideration in an argumentation system. They also have a practical application in terms of providing knowledge for which decisions need to be made; for example, the above sentence would provide useful information regarding a decision as to whether to buy Cisco stock. Most importantly with respect to this paper, though, is that they draw heavily on temporal knowledge due to their day-to-day nature - what is true today may well not be true tomorrow - as well as their inclusion of information concerning periods of time, such as profitability in the current or previous quarter.

Given the use of news reports, we can identify several possible features that we would wish to express in our database; most specifically that news reports concerning business are frequently about properties that change over an interval in time, such as recent sales growth or employment figures. We also require some measure of simplicity if we are to use this with argumentation; since little has been done in the way of temporal argumentation, we should choose a logic as close to the familiar as possible, and with as few new features as possible. However, we also require a certain amount of expressive power. Given these requirements, in this paper we propose a calculus built upon the ideas of Allen's interval logic [11]. Rather than using abstract intervals, and in keeping with the desire for a practical system, we restrict the system to using specific timepoints.

Before we go into the definition of our calculus, it should be noted that there are two other approaches to temporal argumentation can be found in the literature; that of Hunter [12] and Augusto and Simari [13]. Both of these have a differing standpoint to that presented here. Hunter's system is based on maximally consistent subsets of the knowledgebase, which are now not normally regarded as representative of arguments. Augusto and Simari's contribution is based upon a many sorted logic with defeasible formulae, and hence also falls into a different category of argumentation, and the use of many sorted logic raises similar concerns to that of using first order logic.

2. The calculus $\mathcal{T}$

In this section, we present the syntax of a calculus $\mathcal{T}$, and the machinery needed to define the consequence relation $\vdash_{\mathcal{T}}$.

2.1. Syntax

Since we are working with fixed timepoints, we can define a timeline as below.

Definition 2.1. A **timeline** is a tuple $\langle T, \preceq \rangle$, where T is a finite set of **timepoints**, and $\preceq$ is a total linear ordering over these timepoints. The symbol $\prec$ is used such that $t_1 \prec t_2$ iff $t_1 \preceq t_2 \wedge t_1 \neq t_2$.

Definition 2.2. An **interval** (on a timeline $\langle T, \preceq \rangle$) is a pair (t_1, t_2) where t_1 and t_2 are both timepoints in T, and $t_1 \prec t_2$.

Since a finite subset of $\mathbb{Z}$ and $\leq$ are perfectly adequate, in this paper we shall use this definition. Using this we can define relationships between fixed intervals (as noted by Hamblin [14], with the remaining six relationships being the inverses of the first six).

Definition 2.3. Relationships between intervals are defined as follows:

(a, b) **precedes** (c, d) iff $b < c$
(a, b) **meets** (c, d) iff $b = c$
(a, b) **overlaps** (c, d) iff $a < c < b < d$
(a, b) **during** (c, d) iff $c < a < b < d$
(a, b) **starts** (c, d) iff $a = c$ and $b < d$
(a, b) **ends** (c, d) iff $c < a$ and $b = d$
(a, b) **equals** (c, d) iff $a = c$ and $b = d$

For the letters of our calculus $\mathcal{T}$, we have a set of properties, given below as α, β etc, but from a practical standpoint would be more like *sales*, *profits*, to represent the sales or profits for the company in question rising. For each of these, we state whether that property holds over an interval using $Holds(\alpha, i)$ for some interval i, which may be a variable or fixed interval; for example, if we want to say that sales have risen during month 3, we could use $Holds(sales, (2, 3))$.

Definition 2.4. The syntax for the letters of $\mathcal{T}$ is defined as below.

$$
\begin{array}{lcl}
symbol & ::= & \alpha \mid \beta \mid \gamma \mid \delta \ldots \\
timepoint & ::= & 0 \mid 1 \mid 2 \mid 3 \ldots \\
varinterval & ::= & i \mid j \mid k \ldots \\
interval & ::= & (timepoint, timepoint) \\
 & \mid & varinterval \\
relation & ::= & \textbf{precedes} \mid \textbf{meets} \mid \textbf{overlaps} \mid \textbf{during} \\
 & & \mid \textbf{starts} \mid \textbf{ends} \mid \textbf{equals} \\
letter & ::= & Holds(symbol, interval) \\
 & \mid & (interval\ relation\ interval)
\end{array}
$$

We can combine these with the usual propositional connectives ($\wedge, \vee, \neg$, and also for convenience $\rightarrow$ and $\leftrightarrow$). This is not strictly a classical propositional language however, due to the presence of variable intervals, and in the remainder of this chapter, we provide a method of translation between this calculus and classical propositional logic.

Example 2.1. The following are formulae in $\mathcal{T}$:

$Holds(\alpha, i)$
$((Holds(\alpha, i) \wedge (i \textbf{ meets } j)) \vee \neg Holds(\beta, j))$
$(((0, 1) \textbf{ starts } (4, 7)) \wedge Holds(\alpha, (3, 4)))$

Note the third formula is acceptable according to the syntax, although it appears (and will prove to be) a contradiction.

Within $\mathcal{T}$ formulae, all variable intervals are universally quantified over the entire formula, hence a variable interval i refers to the same interval throughout a formula, although may be a different interval in a different formula.

2.2. Grounding

In order to work towards the conversion of formulae containing variable intervals into formulae in propositional logic, we consider the idea of a grounding.

Definition 2.5. A $\mathcal{T}$ **grounding** (also just a grounding) is pair $[\alpha, \Theta]$, where α is a formula in $\mathcal{T}$ and Θ is a set (possibly empty) of substitutions (the **substitution set**), in the form $i/(a, b)$, where i is a variable interval in α, (a, b) is any valid fixed interval (hence $a < b$), and there is no other substitution for i in Θ. If Θ contains a substitution for all variable intervals in α, then the pair is termed a **complete grounding**, otherwise it is a **partial grounding**.

Example 2.2. Valid groundings include

$[Holds(\alpha, i), \{i/(2, 3)\}]$
$[Holds(\alpha, i), \emptyset]$
$[Holds(\alpha, i) \wedge (i \textbf{ meets } j) \rightarrow Holds(\beta, j), \{i/(1, 2), j/(2, 5)\}]$

Given that we have a formula with a variable interval, we can consider the set of all possible groundings of that formula; to define this we use the complete grounding function.

Definition 2.6. The **complete grounding function** $G_C(\Phi)$, where Φ is a set of $\mathcal{T}$ formulae, is defined such that $G_C(\Phi) = \bigcup_{\phi \in \Phi} G_C(\phi)$, and $G_C(\phi)$ for a formula ϕ is defined such that $G_C(\phi)$ is the set containing all complete groundings $[\phi, \Theta]$.

On a similar note, we have the zero grounding function, which converts a formula into a grounding, with no actual substitution done.

Definition 2.7. The **zero grounding function** $G_0(\Phi)$, where Φ is a set of $\mathcal{T}$ formulae, is defined such that $G_0(\Phi) = \{[\phi, \emptyset] : \phi \in \Phi\}$.

For convenience, we extend the definition of complete grounding to be applicable to entire sets.

Definition 2.8. The notation for the grounding function $G_C(\Phi)$ can be extended to allow groundings of existing groundings. Specifically, we stipulate that, for the function G_C from definition 2.6 the following additional rules hold:

- For any grounding $[\phi, \Theta]$, $G_C([\phi, \Theta])$ is the set $\{[\phi, \Theta'] : [\phi, \Theta'] \in G_C(\phi), \Theta \subseteq \Theta'\}$. This is the set containing all complete groundings which are extensions of the existing grounding.
- For any set of groundings Ψ, $G_C(\Psi) = \bigcup_{\psi \in \Psi} G_C(\psi)$

Using this function, we can convert a set of formulae, or an entire database, into a set containing only completely ground formulae.

There are of course times when we wish to convert from a grounding back to a formula in $\mathcal{T}$, and we use the $Subst$ function, below. This simply takes each entry in the substitution set, and replaces every occurrence of the variable with the fixed interval.

Definition 2.9. The **substitution function,** $Subst([\alpha, \Theta])$, where $[\alpha, \Theta]$ is a grounding, returns a single formula α', such that, for each $i/(a, b) \in \Theta$, all occurrences of i in α are replaced by (a, b) in α'

Some useful results concerning the complete grounding function can be observed, showing that the application of G_C does not cause any problems with set membership; these results prove useful in later proofs. Firstly, we can apply G_C repeatedly with no effect.

Theorem 2.1. For any set of $\mathcal{T}$ formulae Φ, $G_C(G_C(\Phi)) = G_C(G_0(\Phi)) = G_C(\Phi)$

Secondly, we can distribute $\cup$ into G_C.

Theorem 2.2. $G_C(\Phi) \cup G_C(\Psi) = G_C(\Phi \cup \Psi)$.

And as a consequence of this, we can show that the subset relationship is maintained through complete grounding.

Theorem 2.3. Suppose $\Phi \subseteq \Psi$. Then $G_C(\Phi) \subseteq G_C(\Psi)$.

This demonstrates a useful property with respect to arguments: if Φ is a minimal subset of a database Δ such that Φ entails α, then we wish there to be a minimal subset $\Psi \subseteq G_C(\Phi) \subseteq G_C(\Delta)$ such that Ψ also entails α.

2.3. The consequence relation $\vdash_{\mathcal{T}}$

For the consequence relation in $\mathcal{T}$, we define a translation into a classical propositional language $\mathcal{T}'$, and then define $\vdash_{\mathcal{T}}$ in terms of $\vdash$ (i.e. the classical consequence relation). We also need to add some predefined knowledge about interval relationships into $\vdash_{\mathcal{T}}$, so that, for example, $(0, 1)$ $\mathbf{meets}$ $(1, 2)$ is a tautology.

Definition 2.10. The language $\mathcal{T}'$ is a classical propositional language with the following syntax for a letter:

$$
\begin{array}{lll}
symbol & ::= & \alpha \mid \beta \mid \gamma \mid \delta \ldots \\
index & ::= & 0 \mid 1 \mid 2 \mid 3 \ldots \\
relation & ::= & \textbf{precedes} \mid \textbf{meets} \mid \textbf{overlaps} \mid \textbf{during} \\
 & & \quad \mid \textbf{starts} \mid \textbf{ends} \mid \textbf{equals} \\
letter & ::= & symbol_{(index,index)} \\
 & & \mid relation_{(index,index),(index,index)}
\end{array}
$$

Where $symbol$ is any $symbol$ in $\mathcal{T}$, and $index$ is any $timepoint$ in $\mathcal{T}$ (and thus both are of finite size).

Combined with the usual connectives, it is fairly simple to define a function which converts from a completely ground formula in $\mathcal{T}$ to $\mathcal{T}'$.

Definition 2.11. The function $Prop$ applied to a complete grounding results in a formula in $\mathcal{T}'$, and can be defined recursively as below:

- $Prop([\alpha, \Theta]) = PropSub(Subst(\alpha, \Theta))$
- $PropSub(Holds(\alpha, (a, b))) = \alpha_{(a,b)}$
- For any relation (eg **meets**, from definition 2.10), $PropSub((a, b)$ **relation** $(c, d)) =$ **relation**$_{(a,b),(c,d)}$
- $PropSub(\neg\alpha) = \neg PropSub(\alpha)$
- $PropSub(\alpha \vee \beta) = PropSub(\alpha) \vee PropSub(\beta)$
- $PropSub(\alpha \wedge \beta) = PropSub(\alpha) \wedge PropSub(\beta)$

Definition 2.12. When $Prop$ is applied to a set Φ of groundings in $\mathcal{T}$, the result is as if $Prop$ was applied to each member in turn, ie $Prop(\Phi) = \bigcup_{\phi \in \Phi} Prop(\phi)$

The second part of $\vdash_{\mathcal{T}}$ requires knowledge of which interval relationships are "true" and which are not. This is done by the set I, which contains this information.

Definition 2.13. The set I of interval relations is $I^{\top} \cup I^{\bot}$, where:

- $I^{\top}$ is the set of all formulae $((a, b)$ **relation** $(c, d))$ such that (a, b) and (c, d) are intervals, **relation** is a relation from definition 2.3 and the conditions on a, b, c and d from definition 2.3 hold.
- $I^{\bot}$ is the set of all formulae $\neg((a, b)$ **relation** $(c, d))$ such that (a, b) and (c, d) are intervals, **relation** is a relation from definition 2.3 and the conditions on a, b, c and d from definition 2.3 do not hold.

Example 2.3. The set I will contain formulae such as $((0, 3)\ \mathbf{meets}\ (3, 5))$ (in $I^{\top}$) and $\neg((0, 7)\ \mathbf{before}\ (2, 4))$ (in $I^{\bot}$).

With the $Prop$ and I, we can define $\vdash_{\mathcal{T}}$.

Definition 2.14. Let Φ be a set of formulae in $\mathcal{T}$ and α a formula in $\mathcal{T}$. Then $\Phi \vdash_{\mathcal{T}} \alpha$ iff

$$Prop(G_C(\Phi) \cup I) \vdash Prop(\bigwedge G_C(\{\alpha\}))$$

where I is the set of interval relations given in definition 2.13 and $\vdash$ is the classical consequence relation.

The use of $\bigwedge G_C(\{\alpha\})$ here is necessary in order to allow the function $Prop$ to be applied should α contain intervals; effectively, we say that we can conclude $Holds(sales\_rising, i)$ only if we can prove that $sales\_rising$ holds for every possible interval (a, b).

3. Arguments in $\mathcal{T}$

Before we begin, we make two simple assumptions about databases in $\mathcal{T}$. Firstly, for a database Δ we assume that each subset of $G_C(\Delta)$ is given an enumeration $\langle \alpha_1, \ldots \alpha_n \rangle$, known as the canonical enumeration. Also, in a database, each entry is assumed to have a unique numerical label, and using this label, $(1)_{i=(0,1)}$ is shorthand for the database entry 1 ground with $[i/(0,1)]$. (1) is shorthand for the zero grounding of database entry 1.

3.1. Argument Definitions

We can alter the established definition of an argument (eg Amgoud and Cayrol [7], Besnard and Hunter [6]) to work with $\mathcal{T}$ easily enough, with the definition of an unground argument below.

Definition 3.1. An **unground $\mathcal{T}$ argument** (or just an unground argument) is a pair $\langle \Phi, \alpha \rangle$, assuming a database Δ, such that:

(1) $\Phi \nvdash_{\mathcal{T}} \bot$
(2) $\Phi \vdash_{\mathcal{T}} \alpha$
(3) Φ is a minimal subset of Δ satisfying (1) and (2)

Example 3.1. Suppose we have the database (where $sales$ denotes high sales):

(1) $Holds(sales, i) \wedge (j \ \mathbf{during} \ i) \rightarrow Holds(sales, j)$
(2) $Holds(sales, (0, 8))$

Then there is an unground argument $\langle \{(1), (2)\}, Holds(sales, (3, 4)) \rangle$

The example above is a case where we can exhibit the characteristics of Shoham's downward hereditary propositions [15], in that if we have high sales over an interval, we can say that we have high sales in the subintervals of that interval.

However, as is indicated by the name unground above, there are alternate approaches to the definition of an argument we can use, such as the definition of a database ground argument below, so called because the database itself is ground before the argument is taken.

Definition 3.2. A **database ground $\mathcal{T}$ argument** (or simply database ground argument) is a pair $\langle \Phi, \alpha \rangle$, assuming a database Δ, such that:

(1) $\Phi \nvdash_{\mathcal{T}} \bot$
(2) $\Phi \vdash_{\mathcal{T}} \alpha$
(3) Φ is a minimal subset of $G_C(\Delta)$ satisfying (1) and (2)

This gives two advantages, the first simply being that of a more informative support, since we can state exactly which intervals we are using in place of the variables in the support.

Example 3.2. Using the database of example 3.1, we have the following database ground argument:

$$\langle \{(1)_{i=(0,8),j=(3,4)}, (2)\}, Holds(sales, (3,4))\rangle$$

However, there is another possible advantage over the simpler definition above, as in the example below, where we can produce arguments that are not possible with the above definition.

Example 3.3. Suppose we have the database:

(1) $Holds(sales, i) \rightarrow Holds(profits, i)$
(2) $Holds(sales, (0,1)) \wedge Holds(sales(1,2)) \wedge \neg Holds(profits(1,2))$

Then, despite the fact that (1) and (2) are mutually inconsistent, there is a database ground argument $\langle \{(1)_{i=(0,1)}, (2)\}, Holds(profits, (0,1))\rangle$

This example is perhaps best considered with respect to news reports; the first formula in the database would be part of background knowledge, and the second a news report saying that a company has had good sales last quarter and this quarter, but profits appear to be down.

We could also use a different grounding function; the support ground argument (so called because the support is ground) uses the zero grounding function.

Definition 3.3. A **support ground $\mathcal{T}$ argument** (or just support ground argument) is a pair $\langle \Phi, \alpha \rangle$, assuming a database Δ, such that:

(1) $\Phi \nvdash_{\mathcal{T}} \bot$
(2) $\Phi \vdash_{\mathcal{T}} \alpha$
(3) Φ is a minimal subset of $G_0(\Delta)$ satisfying (1) and (2)

Example 3.4. Suppose we have the database:

(1) $Holds(sales, i) \rightarrow Holds(profits, i)$
(2) $Holds(sales, (0,1))$

Then there is a support ground argument $\langle \{(1), (2)\}, Holds(profits, (0,1))\rangle$

In fact, a support ground argument is no different from an unground argument, as formalized in the theorem below. Nevertheless, we use support ground arguments rather than unground arguments as a comparison to database ground arguments, simply because the support of both support and database ground arguments consists of a set of groundings, and so they are more directly comparable.

Theorem 3.1. $\langle \Psi, \alpha \rangle$ is a support ground argument iff $\langle \Phi, \alpha \rangle$ is an unground argument, where Ψ is the set $\{[\phi, \emptyset] : \phi \in \Phi\}$

While there are obvious similarities between the above definitions of an argument, there is a tangible difference between database ground and support ground arguments, as in the examples above; in example 3.3 there is no support ground argument. However, we can show that that there are always at least as many database ground arguments as support ground ones.

Theorem 3.2. Given a finite database Δ, for every support ground argument $\langle \Phi, \alpha \rangle$, there is at least one database ground argument $\langle \Psi, \alpha \rangle$, such that $\Psi \subseteq G_C(\Phi)$

3.2. Conservatism and Canonical Undercuts

The following section, and the theorems therein are adapted from Besnard and Hunter's argumentation system [6], and indeed there is little difference; the addition of either grounding function does not affect conservatism.

Conservatism is a way of finding relationships between arguments, and is used to reduce the number of similar arguments we consider when making a point; a more conservative argument has a less specific conclusion and/or uses less information. The aim of conservatism and canonical undercuts is to reduce the number of similar but valid arguments we can use to attack another argument; for $\langle\{\alpha, \alpha \leftrightarrow \beta\}, \beta\rangle$ we could attack with both $\langle\{\neg\alpha\}, \neg(\alpha \wedge \alpha \leftrightarrow \beta)\rangle$ and $\langle\{\neg\alpha, \alpha \leftrightarrow \beta\}, \neg\beta\rangle$, but these effectively make the same point in their attack, so we prefer one over the other (in this case, the former over the latter).

Definition 3.4. A $\mathcal{T}$ argument $\langle\Phi, \alpha\rangle$ is **more conservative than** a $\mathcal{T}$ argument $\langle\Psi, \beta\rangle$ iff $\Phi \subseteq \Psi$ and $\beta \vdash_{\mathcal{T}} \alpha$.

Theorem 3.3. Being more conservative forms a pre-order on arguments in $\mathcal{T}$. Maximally conservative arguments exist, these are $\langle\emptyset, \top\rangle$, where $\top$ is any tautology.

Undercuts are arguments which undercut the support of another argument by contradicting some part of their support.

Definition 3.5. An **undercut** for a $\mathcal{T}$ argument $\langle\Phi, \alpha\rangle$ is a $\mathcal{T}$ argument $\langle\Psi, \neg(\phi_1 \wedge \ldots \wedge \phi_n)\rangle$ where $\{\phi_1, \ldots, \phi_n\} \subseteq \Phi$.

Example 3.5. Given the formula

(3) $\quad Holds(sales(1, 2)) \wedge \neg Holds(profits(1, 2))$

Then $\langle\{(3)\}, \neg(Holds(sales, i) \rightarrow Holds(profits, i))\rangle$ is an undercut for the support ground argument of example 3.4. Note this is not an undercut for the database ground argument of example 3.3.

While other types of defeater exist, such as the rebuttal and the more general defeater, undercuts can be shown, as in Besnard and Hunter's paper [6], to be representative of other defeaters, and so in this paper we simply consider undercuts.

Definition 3.6. A $\mathcal{T}$ argument $\langle\Psi, \beta\rangle$ is a **maximally conservative undercut** for another $\mathcal{T}$ argument $\langle\Phi, \alpha\rangle$ iff $\langle\Psi, \beta\rangle$ is a undercut of $\langle\Phi, \alpha\rangle$ such that there is no undercut of $\langle\Phi, \alpha\rangle$ which is strictly more conservative than $\langle\Psi, \beta\rangle$.

While maximally conservative undercuts are a good representative of many similar undercuts, a canonical undercut solves the problem of both $\langle\{\neg\alpha, \neg\beta\}, \neg(\alpha \wedge \beta)\rangle$ and $\langle\{\neg\alpha, \neg\beta\}, \neg(\beta \wedge \alpha)\rangle$ being maximally conservative.

Definition 3.7. A **canonical undercut** for a $\mathcal{T}$ argument $\langle\Phi, \alpha\rangle$ is a maximally conservative $\mathcal{T}$ undercut $\langle\Psi, \neg(\phi_1 \wedge \ldots \wedge \phi_n)\rangle$ such that $\langle\phi_1 \ldots \phi_n\rangle$ is the canonical enumeration of Φ.

Example 3.6. Continuing example 3.5, the following is a canonical undercut for the support ground argument of example 3.4:

$$\langle\{(3)\}, \neg(Holds(sales,(0,1) \wedge (Holds(sales,i) \rightarrow Holds(profits,i))\rangle$$

Given the fact that the conclusion of a canonical undercut is always the negation of the conjunction of the support (shown in [6]), we use the symbol $\Diamond$ in place of this, as a shorthand.

4. Argument Trees

As in Besnard and Hunter's argumentation system [6,8], we can put sets of arguments into argument trees, according to the definition below.

Definition 4.1. An **argument tree** for α is a tree where the nodes are $\mathcal{T}$ arguments, such that:

(1) The root node is an argument for α
(2) For no node $\langle\Phi,\beta\rangle$ with ancestor nodes $\langle\Phi_1,\beta_1\rangle \ldots \langle\Phi_n,\beta_n\rangle$ is Φ a subset of $\Phi_1 \cup \ldots \cup \Phi_n$.
(3) The children of a node N consist of all canonical undercuts which obey (2).

While argument trees may seem superficially similar for database ground and support ground arguments, and indeed the definition is identical apart from the type of $\mathcal{T}$ argument, the trees themselves are not always similar, and the example below sums up many of the differences in both shape and size between database ground and support ground argument trees.

Example 4.1. Considering the following database:

(1) $Holds(\beta,i) \rightarrow Holds(\alpha,i)$
(2) $Holds(\alpha,(1,2)) \wedge Holds(\beta,(2,3)) \wedge \neg Holds(\alpha,(2,3))$
(3) $Holds(\beta,(1,2)) \wedge Holds(\beta,(3,4)) \wedge \neg Holds(\alpha,(3,4))$
(4) $Holds(\beta,(5,6)) \wedge \neg Holds(\alpha,(5,6))$

Suppose we wish to argue for $Holds(\alpha,(1,2))$. We have the following support ground argument tree:

$$\begin{array}{ccc} & \langle\{(2)\}, Holds(\alpha,(1,2))\rangle & \\ & \uparrow & \\ & \langle\{(1)\},\Diamond\rangle & \\ \nearrow & & \nwarrow \\ \langle\{(3)\},\Diamond\rangle & & \langle\{(4)\},\Diamond\rangle \end{array}$$

We also have the following pair of database ground argument trees, the first with a root corresponding to the support ground argument above, and the second having a root with no corresponding support ground argument.

$$\begin{array}{cc} \langle\{(2)\}, Holds(\alpha,(1,2))\rangle & \langle\{(3),(1)_{i=(1,2)}\}, Holds(\alpha,(1,2))\rangle \\ \uparrow & \uparrow \\ \langle\{(1)_{i=(2,3)}\},\Diamond\rangle & \langle\{(1)_{i=(3,4)}\},\Diamond\rangle \end{array}$$

There is a very limited relationship between these trees; since for every database ground argument there is a support ground argument, we know that there is at least one database ground tree with the same root (or a related root) for each support ground tree. We can take this slightly further, with the theorem below.

Theorem 4.1. For every support ground argument tree, there is a database ground argument tree, such that for every branch $\langle \Phi_1, \alpha \rangle, \langle \Phi_2, \Diamond \rangle, \ldots \langle \Phi_m, \Diamond \rangle$ in the support ground argument tree, there is a branch in the database ground argument tree $\langle \Psi_1, \alpha \rangle, \langle \Psi_2, \Diamond \rangle, \ldots \langle \Psi_n, \Diamond \rangle$ (where $n > 0$) such that, for some p where $0 \leq p \leq m$ and $p \leq n$, the following holds:

(1) Φ_p is the last support (if any) in $\Phi_1 \ldots \Phi_m$ such that, for all i such that $0 < i \leq p$, Φ_i is a set of only completely ground formulae.
(2) For all i such that $1 \leq i \leq p$, $\Psi_i = \Phi_i$.
(3) if $p < n$ and $p < m$, then $\Psi_{p+1} \subseteq G_C(\Phi_{p+1})$.

Note that because of theorem 3.2, it follows directly that the number of trees with the same conclusion in the support ground case can never exceed the number of database ground trees.

Unfortunately, while database ground arguments have their advantages, they do have additional problems. Consider the following example:

Example 4.2. Consider the database:

(1) $Holds(\alpha, i)$
(2) $Holds(\beta, i)$
(3) $Holds(\beta, i) \rightarrow \neg Holds(\alpha, i)$

In this database, suppose we form arguments with the conclusion $Holds(\alpha, i)$. The support ground argument tree is very simple:

$$\begin{array}{c} \langle \{(1)\}, Holds(\alpha, i) \rangle \\ \uparrow \\ \langle \{(2), (3)\}, \Diamond \rangle \end{array}$$

However, the database ground argument tree has a very large number of leaves; in fact the leaf nodes are all nodes of the form $\langle \{(2)_{i=(a,b)}, (3)_{i=(a,b)}\}, \Diamond \rangle$ where $a \in T, b \in T, a \prec b$ and T is the set of timepoints; part of this tree is as below. The syntax $G_C((1))$ is used to represent the set containing every possible complete grounding of (1).

$$\begin{array}{ccccc} & & \langle G_C((1)), Holds(\alpha, i) \rangle & & \\ & \nearrow & & \uparrow & \\ \langle \{(2)_{i=(1,2)}, (3)_{i=(1,2)}\}, \Diamond \rangle & & \langle \{(2)_{i=(2,3)}, (3)_{i=(2,3)}\}, \Diamond \rangle & & \ldots \end{array}$$

Here, we can see an example of a tree with a potentially infinite width (it is not actually infinitely wide, since there are a finite number of timepoints, and so a finite number of possible groundings). This is clearly an undesirable feature, and further work, beyond the scope of this paper, examines possible solutions to problems such as these.

5. Discussion

In this paper we have discussed a way of encoding temporal information into propositional logic, and examined its impact in a coherence argumentation system.

We have shown that it is possible to use the standard definition of an argument, but there are other possible argument definitions that arise through the use of grounding in the definition of $\vdash_{\mathcal{T}}$, the database ground and support ground arguments, and we have shown that while the definitions of these are similar, there are differences, in particular in the case of situations such as example 3.3 where there are no support ground arguments.

Although theorem 4.1 shows there is some comparison between the support ground and database ground cases, when we put these arguments into the argument trees of Besnard and Hunter, there are some significant differences, including different depths and widths of tree or different numbers of trees.

While database ground arguments and support ground/unground arguments both have their advantages, we do not consider that either is the "best" type, and each has their own uses. In particular, example 4.2 shows that while there are benefits that database ground arguments may provide over support ground or unground arguments, they also have disadvantages. Nevertheless, we consider that, particularly with further work, the use of grounding within the argument definition provides definite advantages and a useful avenue of study for further work in temporal argumentation.

References

[1] H. Prakken and G. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer, 2000.

[2] C. I. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32(4):337–383, 2000.

[3] Ph. Besnard and A. Hunter. *Elements of Argumentation*. MIT Press, 2008.

[4] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic progamming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[5] A. Garcia and G. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.

[6] Ph. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128:203–235, 2001.

[7] L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34:197–216, 2002.

[8] Ph. Besnard and A. Hunter. Practical first order argumentation. In *Procceedings of the 20th American National Conference on Artificial Intelligence (AAAI'2005)*. MIT Press, 2005.

[9] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[10] A. Prior. *Past, Present and Future*. Clarenden Press, Oxford, 1967.

[11] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[12] A. Hunter. Ramification analysis with structured news reports using temporal argumentation. In *Proceedings of the Adventures in Argumentation Workshop (part of the Sixth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty)*. Institut de Recherche en Informatique de Toulouse, 2001.

[13] J. C. Augusto and G. Simari. A temporal argumentation system. *AI Communications*, 12(4):237–257, 1999.

[14] C. Hamblin. Instants and intervals. In J. F. H. Fraser and G. Muller, editors, *The Study of Time*, pages 324–328. Springer, 1972.

[15] Y. Shoham. Temporal logics in ai: Semantic and ontological considerations. *Artificial Intelligence*, 33:89–104, 1987.