TECHNISCHE UNIVERSITÄT DRESDEN
Dresden University of Technology
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

Pinpointing in Tableaus

Rafael Peñaloza

LTCS-Report 06-05

Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik
TU Dresden
http://lat.inf.tu-dresden.de

Nöthnitzer Str. 46
01187 Dresden
Germany

Pinpointing in Tableaus

Rafael Peñaloza
penaloza@tcs.inf.tu-dresden.de

December 15, 2006

Abstract

Tableau-based decision procedures have been successfully used for solving a wide variety of problems. For some applications, nonetheless, it is desirable not only to obtain a Boolean answer, but also to detect the causes for such a result. In this report, a method for finding explanations on tableau-based procedures is explored, generalizing previous results on the field. The importance and use of the method is shown by means of examples.

1 Introduction

A very important class of problems in Computer Science is that consisting of decision problems. In order to solve these problems, distinct decision procedures have been developed. One kind of decision procedure, called tableau, has become widely used in areas such as Description Logics (DLs), where one wants to reason with sets of axioms, or ontologies. Several examples of the use of tableaus in DLs are presented in [BS01].

In complex applications, the binary answer given by decision procedures is usually not enough, as one might want to get an explanation of such an answer, or some additional details. A good example is again given by Description Logics, where one could check for the satisfiability of some concept terms in order to find errors or inconsistencies in the Knowledge Base. If a concept term is supposed to be satisfiable, but turns out to be unsatisfiable, then there must have been a modelling error when constructing the ontology, and it should be checked to remove every such mistake. Unfortunately, the decision procedure states only if the term is satisfiable or not, but never gives an explanation for this, which would be helpful when trying to remove inconsistencies.

Starting from the World Wide Web Consortium's decision on using a DL as a standard for ontology representation, more and larger ontologies are being constructed withing this language. As a consequence, not only these ontologies are more difficult to read, due to their size, but also they are being created more by experts in the area to be modelled, and less by DL experts. This carries the inconvenience of raising the possibilities of errors in the model, motivating the search of methods that help detect mistakes, and correct them.

Some research has been done already in this area, leading to methods that work in specific instances of tableaus. Some examples of these can be found in [BH95], for finding maximal consistent ABoxes for the DL $\mathcal{ALCF}$; or [MLBP06], and [LMPB06] for finding maximal TBoxes for which a concept is satisfiable in $\mathcal{ALC}$. In all of these cases, the approach followed is simple: for every assertion inserted, keep track of the axioms that are responsible of its existence. In the end, when an inconsistency is found, one can track its cause back to the axioms that produced it. This information can be used afterwards to correct the error by, for example, removing just enough axioms to ensure that the causes of inconsistency cannot be produced anymore.

While this approach, which will be called *pinpointing*, had been shown to be useful, it had only be applied in very particular cases. In this report, the main ideas used in all these examples are generalized and shown to work in a wide class of tableaus. In order to do this, one needs to first formalize the notion of tableau. Once such a notion is stated, one can continue to state the way the pinpointing method will be applied to these tableaus. After all this process, one can finally show that the method indeed signals the relevant axioms, and how this information can be used to remove the inconsistencies.

The intuitive notion of tableau is very wide, and any formalization of it must be able to express notions such as the distinct blocking conditions that may be used. A previous formalization of this notion can be found in [BHLW03]. In that case, the aim was to show a relationship between tableaus and automata working over trees, and hence the definitions given were skewed towards the construction of tree-like models. In the present case, nonetheless, the tree-like shape is not necessary, and its use would only add more difficulties to the constructions for the pinpointing method.

For this reason the notion of tableau will be formalized once again in this report, making as little assumptions on the structure of its elements as possible. A definition general enough to include all the elements involved in the use of tableaus would be very difficult to state directly in a way that is readable, and also usable for developing the pinpointing technique on it. Hence, the formalization will be given step by step, starting with the simplest, most specific case, and gradually generalizing it, to capture as many of the aspects of these decision procedures as possible.

The starting point is given in Section 2, where the notion of *deterministic tableaus* is defined. In this notion are included only those tableaus for which there is one given path to reach the answer of the problem, given the input. Every rule in this framework has only positive applicability conditions, that is, the rules state when they can be applied, but not when they cannot be applied. Furthermore, the set of clashes is global in the sense that, regardless of the input given, the same clashes are always used. As it will be seen, some problems of interest can already be solved in this extremely restricted framework.

Section 3 generalizes the previous notion, adding non-determinism in the rules of the tableau. This way, problems including some decision for which the correct answer is not known forehand, can be solved. In these two frameworks, all the assertions are global, in the sense that there is no way to state that some elements in a domain satisfy some of them, while others satisfy some different ones. This restriction is removed in the next section, adding what are called *assertions with variables*.

The following generalizing step consists in allowing some negative applicability conditions. This way, a rule application may be blocked by the presence of some elements in the assertion set. In this framework, it will be possible to express the distinct blocking conditions. Finally, this is again further generalized in Section 6 to allow the use of different sets of clashes depending on the input given, defining this way the *dynamic tableaus*.

Dynamic tableaus are then used to reduce another widely used decision procedure, automata theory, to a tableau-based procedure. Given that the tableau that decides the emptiness problem for automata is in the class for which the pinpointing method works, one can then apply the pinpointing method to automata, and also to all those problems that can be reduced to the emptiness problem in order to be decided.

Along with the proofs of correctness of the pinpointing method in the different frameworks presented, some results of termination are also given. These state that, if a tableau can find an answer in finite time, then the pinpointing procedure will also get the answer in finite time. However, no further complexity results are stated in this work.

The last section states the conclusions of this work, along with some ideas regarding the limitations of the pinpointing method.

2 Deterministic framework

Intuitively, a tableau is a decision procedure that operates over a set of so called *assertions*, using rules that add, in each step, more assertions to the set until no rule is applicable anymore, at which point uses the resulting assertion set to decide the answer.

In this section, the simplest type of tableau, the *deterministic tableau*, will be dealt with. Deterministic tableaus have a very restricted definition, and for this reason work only for a small class of decision problems. It is still worth studying them, since they will form the base over which more complex kinds of tableaus will be defined. Additionally, the development given in this section should help to give an intuition on how tableaus work and why the different results hold.

After giving the formal notion of deterministic tableau, it will be shown how the pinpointing procedure can be applied to them, in order to obtain more information regarding the axioms used to reach the given answer.

2.1 Deterministic tableaus

As the name sugests, deterministic tableaus follow in certain sense a fixed path searching for the solution of the decision problem. As it will be shown later, even this simple setting includes some kind of non-determinism; this, nonetheless, will be a neglectable sort of non-determinism and will not affect the deterministic framework. The formal definition of a deterministic tableau is given now, where $\mathscr{P}$ denotes the power set constructor.

Definition 2.1 (Deterministic tableau,S-state) *Let $\mathfrak{I}$ be a set of* inputs, *and $\mathfrak{T}$ a set of* axioms*; an* axiomatized input *is an element of* $\mathfrak{I}\times\mathscr{P}(\mathfrak{T})$. *A* deterministic tableau for $\mathfrak{I}$,$\mathfrak{T}$ *is a tuple $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ where $\mathcal{A}$ is a set of* assertions*; the function $\cdot^{S_\mathfrak{I}}$ maps each $\mathcal{I} \in \mathfrak{I}$ to a set $A \subseteq \mathcal{A}$.*

A S-state *is an element of $\mathscr{P}(\mathcal{A}) \times \mathscr{P}(\mathfrak{T})$. $\mathcal{R} = \{\mathsf{R}_1, \dots, \mathsf{R}_n\}$ is a set of* rules *of the form $(A, \mathcal{T}) \xrightarrow{\mathsf{R}_i} A'$, where $(A, \mathcal{T})$ is a S-state and $A' \subseteq \mathcal{A}$. $\mathcal{C}$ is a subset of $\mathscr{P}(\mathcal{A})$ whose elements are called* clashes.

The function $\cdot^S$ extends $\cdot^{S_\mathfrak{I}}$ by mapping an axiomatized input $(I, \mathcal{T})$ to the S-state $(I, \mathcal{T})^S = (I^{S_\mathfrak{I}}, \mathcal{T})$.

As it was said before, the tableau is intended to apply the rules to the assertion set until no rule is applicable anymore. For this matter, a formal definition of applicability and the meaning of applying such a rule needs to be given.

Definition 2.2 (Applicability) *Let* $\mathfrak{S} = (A, \mathcal{T})$ *be a* S*-state and* $\mathsf{R} : (B, \mathcal{T}') \xrightarrow{\mathsf{R}} B'$ *a rule.* R *is* applicable *to* $\mathfrak{S}$ *if* $B \subseteq A, \mathcal{T}' \subseteq \mathcal{T}$*, and* $B' \nsubseteq A$*. The* result of applying R to $\mathfrak{S}$ *is the* S*-state* $\mathsf{R}(\mathfrak{S}) = (A \cup B', \mathcal{T})$*.*

For a given axiomatized input Γ, the tableau needs not know all the possible S-states in order to decide if this input satisfies certain property or not, and needs only know the S-states that are generated by applying rules to the state obtained by originally translating Γ. These will be called S-states *for* Γ.

Definition 2.3 (S-state for Γ,saturated,clash-free) *Given an axiomatized input* $\Gamma \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$*, the set of* S-states for Γ *is inductively defined as follows:*

- Γ^S *is a* S*-state for* Γ*,*
- *if* $\mathfrak{S}$ *is a* S*-state for* Γ *and* R *is applicable to* $\mathfrak{S}$*, then* $\mathsf{R}(\mathfrak{S})$ *is a* S*-state for* Γ*.*

Let $\mathfrak{S} = (A, \mathcal{T})$ *be a* S*-state for* Γ*.* $\mathfrak{S}$ contains a clash *if there is a* $C \in \mathcal{C}$ *such that* $C \subseteq A$*. In this case, it is said that* C *is* in $\mathfrak{S}$*.* $\mathfrak{S}$ *is* saturated *if no rule is applicable to it; it is* clash-free *if no clash is in it.*

It is now possible to distinguish the tableaus according to their capability for deciding a certain property. The following definition is a direct translation of the definition of *soundness* and *completeness* of general decision procedures.

Definition 2.4 (Soundness,completeness) *Let* $\mathcal{P} \subseteq \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$ *be a* property*. The tableau* S *is called* sound for $\mathcal{P}$ *if for any* $\Gamma \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$*, the existence of a saturated clash-free* S*-state for* Γ *implies that* $\Gamma \in \mathcal{P}$*. It is called* complete for $\mathcal{P}$ *if for any* $\Gamma \in \mathcal{P}$ *there exist a saturated and clash-free* S*-state for* Γ*.*

The following example should help the reader to grasp the intuitions behind the definitions given up to now, as well as show how they can be applied to create particular instances of the decision procedure.

Example 2.5 *Let* $\mathcal{V}$ *be a set of propositional variables, and* $\perp \notin \mathcal{V}$*. A* complex Horn clause *over* $\mathcal{V}$ *is of the form* $q \leftarrow p_1, \ldots, p_n$ *with* $n > 0$*, where* $p_i \in \mathcal{V}$ *for all* i *and* $q \in \mathcal{V} \cup \{\perp\}$*. The set of all complex Horn clauses will be denoted as* cHorn*. A* fact *is of the form* $q\leftarrow$ *where* $q \in \mathcal{V}$*. A* Horn clause *is either a complex Horn clause or a fact. A set of Horn clauses over* $\mathcal{V}$ *is* satisfiable *if there is a valuation that maps every clause in the set to* true*.*

A deterministic tableau for verifying whether a set of Horn clauses over $\mathcal{V}$ is satisfiable is given by $S_{\mathsf{H}} = (\mathcal{A}, \cdot^{S_{\mathsf{H}}\mathfrak{I}}, \mathcal{R}, \mathcal{C})$, where $\mathcal{A} = \mathsf{cHorn} \cup \mathcal{V} \cup \{\bot\}$; for a set of Horn clauses over $\mathcal{V}$, $\Gamma = (\mathcal{I}, \mathcal{T})$, where $\mathcal{I} \subseteq \mathsf{cHorn}$ and $\mathcal{T}$ is a set of facts, $\Gamma^{S_{\mathsf{H}}} = \Gamma$; $\mathcal{C} = \{\{\bot\}\}$; and $\mathcal{R}$ contains the following two rules:

$$\begin{aligned}
\mathsf{R}{\leftarrow} &: (\{p_1, \ldots, p_n, (q \leftarrow p_1, \ldots, p_n)\}, \emptyset) \xrightarrow{\mathsf{R}\leftarrow} \{q\} \\
\mathsf{Rf} &: (\emptyset, \{p{\leftarrow}\}) \xrightarrow{\mathsf{Rf}} \{p\}
\end{aligned}$$

Intuitively, given a S_{H}-state $(A, \mathcal{T})$, the set A contains, additionally from the complex Horn clauses given in the input, all the elements that must be mapped to true *by any valuation that maps every clause in the set given as input to* true*. If a fact $p{\leftarrow}$ is in $\mathcal{T}$, then any valuation that will satisfy this condition must map p to true; and rule* Rf *states exactly that. On the other hand, if there is a complex Horn clause $q \leftarrow p_1, \ldots, p_n$ and each of the elements p_1 to p_n must be mapped by the valuation to* true*, then also the valuation of q must be set to* true*, as stated by rule* R←*.*

If, after applying all the rules, the element $\bot$ appears in the set A, it means that, in order to satisfy all the Horn clauses in the input, then also $\bot$ must be satisfied, which is impossible, and thus, the input must be unsatisfiable. Conversely, if the input given is unsatisfiable, then every valuation that maps every Horn clause in it to true*, must also satisfy $\bot$. Thus, S_{H} is sound and complete for the specified property.*

The soundness and completeness of the tableau does not depend on the division of the Horn clauses in complex Horn clauses, stored in the assertion set, and facts, stated as axioms. The only difference given by this partition is what is considered an axiom within the tableau; in this case, the axioms are the facts. Notice that application of rules in a tableau can never modify the set of axioms given at the beginning, while the set of assertions can be extended by such applications. The reason for using in this case the facts as the only possible axioms is because Example 2.5 will be later extended to find information regarding the facts that were needed to make the whole set of Horn clauses unsatisfiable. This information can later be used to produce maximal subsets of facts with which set of complex Horn clauses satisfiable. The method for doing so is the goal of the last part of this section.

The reason for the name 'deterministic tableau', is that all the rules in each of them are deterministic; that is, given one set of assertions to which the rule is applicable, there is a unique set of assertions that can be obtained from such an application. Nonetheless, the order of application of the rules adds a kind of non-determinism to the procedure, since at one particular S-state, there might be several applicable rules and the definition states no order in which the applications must be made. On the other hand, the way the tableau decides whether an input belongs to a given property is by finding out if there is a saturated and clash-free S-state for that input. For that reason, the main interest centers only in saturated S-states. It will now be shown that the non-determinism added by the rule application order is a kind of *don't care* non-determinism, since all saturated S-states for each input are equal.

Definition 2.6 (Substate,equal) *Let S be a deterministic tableau and $\mathfrak{S} = (A, \mathcal{T})$, $\mathfrak{S}' = (A', \mathcal{T}')$ two S-states. Then $\mathfrak{S}$ is a* substate *of $\mathfrak{S}'$, denoted as $\mathfrak{S} \subseteq \mathfrak{S}'$, iff $A \subseteq A'$ and $\mathcal{T} \subseteq \mathcal{T}'$. $\mathfrak{S}$ and $\mathfrak{S}'$ are* equal, *denoted $\mathfrak{S} = \mathfrak{S}'$, iff $\mathfrak{S} \subseteq \mathfrak{S}'$ and $\mathfrak{S}' \subseteq \mathfrak{S}$.*

The following lemma shows that all the elements that would be added to any S-state $\mathfrak{S}$ by application of a rule to it have to be present in any saturated S-state of which $\mathfrak{S}$ is a substate.

Lemma 2.7 *Let S be a deterministic tableau, $\mathfrak{S}_0 = (A_0, \mathcal{T}_0)$ be a saturated S-state, $\mathfrak{S} = (A, \mathcal{T})$ a S-state and $\mathsf{R} : (B, \mathcal{T}') \xrightarrow{\mathsf{R}} B'$ such that $\mathfrak{S} \subseteq \mathfrak{S}_0$ and R is applicable to $\mathfrak{S}$. Then $B' \subseteq A_0$.*

Proof. As $\mathfrak{S}_0$ is saturated, R is not applicable to it. But since $B \subseteq A \subseteq A_0$ and $\mathcal{T}' \subseteq \mathcal{T} \subseteq \mathcal{T}_0$, the only way R is not applicable to $\mathfrak{S}_0$ is that $B' \subseteq A_0$, by definition of applicability. ■

With this lemma, it is now easy to show that, given an axiomatized input Γ, there is only one saturated S-state for Γ, and hence, the rule application order is irrelevant.

Theorem 2.8 *Let S be a deterministic tableau and Γ an input. If $\mathfrak{S}$ and $\mathfrak{S}'$ are two saturated S-states for Γ, then $\mathfrak{S} = \mathfrak{S}'$.*

Proof. As $\mathfrak{S}'$ is a saturated S-state for Γ, there must be a sequence of rules $\mathsf{R}_1, \ldots, \mathsf{R}_n$ that were applied to reach $\mathfrak{S}'$ from Γ^S. As $\mathfrak{S}$ is a S-state for Γ, and an application of a rule can only add elements to the assertion set, it must be the case that $\Gamma^S \subseteq \mathfrak{S}$. Then, by Lemma 2.7, $\mathsf{R}_1(\Gamma^S) \subseteq \mathfrak{S}$. Multiple applications of the same lemma yield $\mathfrak{S}' = \mathsf{R}_n(\mathsf{R}_{n-1}(\cdots(\mathsf{R}_1(\Gamma^S)\cdots)) \subseteq \mathfrak{S}$. The same argument can be used symmetrically to show that $\mathfrak{S} \subseteq \mathfrak{S}'$. Thus, $\mathfrak{S} = \mathfrak{S}'$. ■

This theorem impliest that, in order to check if a property $\mathcal{P}$ holds for an axiomatized input Γ using a deterministic tableau S that is sound and complete for $\mathcal{P}$, it is sufficient to find one saturated S-state for Γ and check if it contains a clash or not. Since all saturated S-states are equal, the presence of a clash in the found S-state is enough to ensure that there is not saturated and clash-free S-state for Γ.

Let now be S a sound and complete tableau for a property $\mathcal{P}$. In the case in which an axiomatized input $(\mathcal{I}, \mathcal{T})$ is not in the property, it is sometimes desirable to find those *maximum subsets of axioms* for which the property is satisfied. In other words, to find a $\Theta \subset \mathcal{T}$ such that $(\mathcal{I}, \Theta) \in \mathcal{P}$ and for every $\Theta \subset \mathcal{T}' \subseteq \mathcal{T}$ it is the case that $(\mathcal{I}, \mathcal{T}') \notin \mathcal{P}$. For example, if a set of Horn clauses is not satisfiable, one might want to compute the largest subsets of facts, for which adding the original complex Horn clauses leads to a satisfiable set.

When a tableau S answers that an axiomatized input Γ does not satisfy the property decided by S, it must be the case that every saturated S-state for Γ contains a clash. In order to find a maximum subset of axioms for which the property is satisfied, one must remove just enough of the axioms in the original input to obtain one saturated state without any clash, but not more. One possible way to proceed toward this goal is to *pinpoint* the axioms that are used to obtain each element appearing in the assertion set of any given S-state; this way, when a clash is found, one can easily identify the axiomatic cause of it and, afterwards, decide the remotion of which would lead to a clash-free state.

In this paper, the pinpointing method consists in labelind every axiom with a unique propositional variable that will help to identify it. While the decision procedure works, whenever any set of labeled elements are used to include a new element to a state, the newly generated element receives as label the conjunction of the labels found in the elements that were necessary to create it. It might also be the case, though, that a single element could be created in many different ways; since all these ways must be taken into account when a clash is found, the label of the element must specify this, hence the disjunction of the labels of all paths leading to this element will be used to label it.

The pinpointing procedure is done by means of *jalals*, which are decision procedures based on tableaus that work in a fashion very similar to them, but also take into account the use of labels in the assertion and axioms sets. These decision procedures are formally defined, and explained in more detail, within the deterministic framework, in the following subsection.

2.2 Deterministic jalals

Before defining formally the notion of deterministic jalals, it is necessary to state some notation for the labeled elements, and their respective labels, which will be given by monotonic propositional formulas.

Notation 2.9 *Let Φ be a set of propositional formulas, and A an arbitrary set. $\mathbb{B}^+(\Phi)$ denotes the set of all formulas formed by conjunction and disjuntion of elements in Φ. $A^\Phi = \{a^\phi \mid a \in A, \phi \in \mathbb{B}^+(\Phi)\}$ is the set obtained by labeling elements in A with formulas in $\mathbb{B}^+(\Phi)$. For a set of labeled elements B, the set of elements of B without its labels is denoted as $\mathsf{unl}(B) = \{b \mid b^\phi \in B\}$.*

If ϕ is a propositional formula, then A^ϕ denotes the set $A^{\{\phi\}}$. The symbol $\top$ expresses any tautology.

Definition 2.10 (Deterministic jalal) *Let $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ be a deterministic tableau for $\mathfrak{I}$,$\mathfrak{T}$. Label every element of $\mathfrak{T}$ with a unique propositional variable and let lab be the set of all those variables. For a set $\mathcal{T} \subseteq \mathfrak{T}$, let $\hat{\mathcal{T}}$ denote the set containing all the elements of $\mathcal{T}$ with their respective variables. The* deterministic jalal judging S *is a tuple $S_\mathsf{j} = (\mathcal{A}^{\mathsf{lab}}, \cdot^{(S_\mathsf{j})_\mathfrak{I}}, \mathcal{R}_\mathsf{j}, \mathcal{C}_\mathsf{j})$, where*

- *for every $\Gamma \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$, if $\Gamma^S = (A, \mathcal{T})$, then $\Gamma^{S_\mathsf{j}} = (A^\top, \hat{\mathcal{T}})$;*
- *for every rule $\mathsf{R} \in \mathcal{R}$ of the form*

$$(\{a_1, \ldots, a_n\}, \{t_1, \ldots, t_m\}) \xrightarrow{\mathsf{R}} B$$

  the rule R' is given by

$$(\{a_1^{\phi_1}, \ldots, a_n^{\phi_n}\}, \{t_1^{\varphi_1}, \ldots, t_m^{\varphi_m}\}) \xrightarrow{\mathsf{R}} B^\psi$$

  where $\psi = \bigwedge_{i=1}^n \phi_i \wedge \bigwedge_{i=1}^m \varphi_i$;
- *$\mathcal{R}_\mathsf{j} = \{\mathsf{R}' \mid \mathsf{R} \in \mathcal{R}\}$;*
- *for every $C = \{a_1, \ldots, a_n\} \in \mathcal{C}$ define*

$$\bar{C} = \{\{a_1^{\phi_1}, \ldots, a_n^{\phi_n}\} \mid \phi_i \in \mathbb{B}^+(\mathsf{lab})\}$$

- *$\mathcal{C}_\mathsf{j} = \bigcup_{C \in \mathcal{C}} \bar{C}$.*

*A S_j-*state *is an element of $\mathscr{P}(\mathcal{A}^{\mathsf{lab}}) \times \mathscr{P}(\hat{\mathfrak{T}})$.*

Notice that a jalal is very similar to a tableau, only that assertions, rules and clashes have additional labels in them. In the case of the rules R', the propositional formulas ϕ_i in the labels are considered as parameters additional to the original ones in R, and are instantiated when deciding the applicability of a rule.

As it was said before, the idea behind a jalal is to mark the axioms that were necessary to obtain each assertion, and thus, also each of the clashes that could be found. The applicability conditions for tableau rules ensure that, whenever all the elements that would be added by the rule appear already in the assertion set, such a rule would not be applied. This is sufficient for the purpose of the tableau, in which it is only important to know whether there *is* a clash. For jalals, nonetheless, other applicability conditions must be used, since many different sets of axioms may produce the same clash, and it is necessary to distinguish them all. The following example shows why using the same applicability conditions is not enough for recognizing all the causes for certain assertions to be produced.

Example 2.11 *Let S_{Hj} be the deterministic jalal judging the tableau S_{H} of Example 2.5, and let $\Gamma = (\{\bot \leftarrow p, \bot \leftarrow q\}, \{p \leftarrow, q \leftarrow, r \leftarrow\})$ be the axiomatized input with u, v, w their respective axiom variables. If the rules in S_{Hj} had the same applicability conditions and results as the ones in tableaus, a possible sequence obtained from applying such rules to $\Gamma^{S_{\mathsf{Hj}}}$ is the following, where $\hat{\mathcal{T}} = \{p \leftarrow^u, q \leftarrow^v, r \leftarrow^w\}$ and $\mathcal{I} = \{\bot \leftarrow p^\top, \bot \leftarrow q^\top\}$:*

$$
\begin{aligned}
(\mathcal{I}, \mathcal{T}) &\xrightarrow{\mathsf{Rf}'} (\{p^u\} \cup \mathcal{I}, \mathcal{T}) \\
&\xrightarrow{\mathsf{R}\leftarrow'} (\{p^u, \bot^u\} \cup \mathcal{I}, \mathcal{T}) \\
&\xrightarrow{\mathsf{Rf}'} (\{p^u, \bot^u, q^v\} \cup \mathcal{I}, \mathcal{T}) \\
&\xrightarrow{\mathsf{Rf}'} (\{p^u, \bot^u, q^v, r^w\} \cup \mathcal{I}, \mathcal{T})
\end{aligned}
$$

At this point, no rule is further applicable. For the Rf' rule, there are no more facts in the axiom set for which the rule could be applied, and for the $\mathsf{R}\leftarrow'$ rule, the element $\bot$ is already in the assertion set and hence it cannot be applied for any of the complex Horn clauses available. The labeling of the clash found states that it was produced by the axiom labeled with u, that is, $p \leftarrow$. Nonetheless, the same clash could also be produced by axiom $q \leftarrow$. This is not represented because the rule $\mathsf{R}\leftarrow$ could not be applied to $\bot \leftarrow q$ and $q \leftarrow$.

The applicability conditions for jalal rules must allow for application even if the elements that will be added are already present in the assertion set, under some conditions that ensure that the same rule will not be applied once and again, producing no progress in the execution of the jalal. These conditions basically ensure that new information is added to the assertion set every time a rule is applied; either new elements are added, or the labels in previouly present elements are changed to a more relaxed formula. The following definition states this in a more formal way.

Definition 2.12 (Insertable,applicability) *Let A be a labeled set and B an unlabeled set. The* set of ψ-insertable elements of B to A, *denoted as* $\mathsf{ins}_\psi(B, A)$, *is the set of all* $b \in B$ *such that either there is no* ϕ *such that* $b^\phi \in A$, *or if there exists such a* ϕ, *then* $\psi \not\models \phi$.

The set $A \Cup B^\psi$ *is defined recursively as follows:*

- $A \Cup \{b^\psi\} = (A \setminus \{b^\phi\}) \cup \{b^{\phi \vee \psi}\}$ *if there exists a* ϕ *such that* $b^\phi \in A$ *and* $A \cup \{b^\psi\}$ *otherwise;*
- $A \Cup \{b_1^\psi, \ldots, b_n^\psi\} = (A \Cup \{b_1^\psi\}) \Cup \{b_2^\psi, \ldots, b_n^\psi\}$, *for* $n > 1$.

Let $\mathfrak{S} = (A, \mathcal{T})$ *be a* S_j*-state and* $\mathsf{R} : (B, \mathcal{T}') \xrightarrow{\mathsf{R}} (B')^\psi$ *a rule of* S_j. R *is* applicable *to* $\mathfrak{S}$ *if* $B \subseteq A, \mathcal{T}' \subseteq \mathcal{T}$, *and* $\mathsf{ins}_\psi(B', A) \neq \emptyset$. *The* result of applying R to $\mathfrak{S}$ *is the* S_j*-state* $\mathsf{R}(\mathfrak{S}) = (A \Cup (B')^\psi, \mathcal{T})$.

The following example continues the application of rules where Example 2.11 stoped, using the previous definition of applicability.

Example 2.13 *At the last* S_j*-state in Example 2.11, the rule* $\mathsf{R}\!\leftarrow'$ *is further applicable, leading to the following* S_j*-state:*

$$(\{p^u, \bot^u, q^v, r^w\} \cup \mathcal{I}, \mathcal{T}) \quad \xrightarrow{\mathsf{R}\leftarrow'} \quad (\{p^u, \bot^{u \vee v}, q^v, r^w\} \cup \mathcal{I}, \mathcal{T})$$

The label on the clash states now that it can be produced either by the axiom labeled with u, *or with the one labeled with* v, *which is the expected result.*

Before going on to show that jalals really do what has been claimed up to now, pinpointing the relevants axioms that produce clashes, it is necessary to define other notions inherited from tableaus.

Definition 2.14 (Sound,complete) *Let* S_j *be a deterministic jalal. For an axiomatized input* Γ, *the set of* S_j-states for Γ *is inductively defined as follows:*

- Γ^{S_j} *is a* S_j*-state for* Γ,
- *if* $\mathfrak{S}$ *is a* S_j*-state for* Γ *and* R *is applicable to* $\mathfrak{S}$, *then* $\mathsf{R}(\mathfrak{S})$ *is a* S_j*-state for* Γ.

Let $\mathfrak{S} = (A, \mathcal{T})$ *be a* S_j*-state.* $\mathfrak{S}$ contains a clash *if there is a* $C \in \mathcal{C}_\mathsf{j}$ *such that* $C \subseteq A$; *in this case, it is also said that* C *is* in $\mathfrak{S}$. $\mathfrak{S}$ *is* saturated *if no rule is applicable to it and is* clash-free *if it does not contain a clash.*

Let $\mathcal{P} \subseteq \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$ *be a property.* S_j *is* sound for $\mathcal{P}$ *if for any input* Γ, *the existence of a saturated and clash-free* S_j*-state for* Γ *implies that* $\Gamma \in \mathcal{P}$; *it is* complete for $\mathcal{P}$ *if for any* $\Gamma \in \mathcal{P}$ *there is a saturated and clash-free* S_j*-state for* Γ.

A notion of equivalence between the states of a jalal is also necessary. This definition needs to take care of the labels in the assertions, which need not to be equal for the equivalence to hold, but only represent equivalent propositional formulas.

Definition 2.15 (Substate,equal) *Let S_{j} be a deterministic jalal, $\mathfrak{S} = (A, \mathcal{T})$ and $\mathfrak{S}' = (A', \mathcal{T}')$ be two S_{j}-states. $\mathfrak{S}$ is a* substate *of $\mathfrak{S}'$, denoted as $\mathfrak{S} \subseteq \mathfrak{S}'$ iff $\mathcal{T} \subseteq \mathcal{T}'$ and for every $a^{\phi} \in A$, there exists φ such that $a^{\varphi} \in A'$ and $\phi \models \varphi$. $\mathfrak{S}$ and $\mathfrak{S}'$ are* equal*, denoted $\mathfrak{S} \subseteq \mathfrak{S}'$, iff $\mathfrak{S} \subseteq \mathfrak{S}'$ and $\mathfrak{S}' \subseteq \mathfrak{S}$.*

Notice that this definition entails that, whenever two states $\mathfrak{S}$ and $\mathfrak{S}'$ are equal, then every assertion appearing in $\mathfrak{S}$ will also appear in $\mathfrak{S}'$, but they might have different labels. The only requirement over these labels for the equality to hold is that they are equivalent propositional formulas. The reason for this, as should become clear later in this section, is that what is used to detect the axioms necessary to generate an assertion is in fact the *meaning* of the formula, not its actual shape.

For deterministic jalals it is also the case that the non-determinism obtained from the order of rule application is a kind of *don't care* non-determinism. This means that the S_{j} states that are saturated contain all the same elements, and their labels are equivalent propositional formulas. The proof of this fact will rely on the following lemma.

Lemma 2.16 *Let S_{j} be a deterministic jalal, $\mathfrak{S}_0 = (A_0, \mathcal{T}_0)$ a saturated S_{j}-state, $\mathfrak{S} = (A, \mathcal{T})$ a S_{j}-state and $\mathsf{R} : (B, \mathcal{T}') \xrightarrow{\mathsf{R}} (B')^{\psi}$ a rule such that $\mathfrak{S} \subseteq \mathfrak{S}_0$ and R is applicable to $\mathfrak{S}$. Then $\mathsf{R}(\mathfrak{S}) \subseteq \mathfrak{S}_0$.*

Proof. Since $\mathfrak{S} \subseteq \mathfrak{S}_0$ and R is applicable to $\mathfrak{S}$, there is a valuation of the parameters in R such that $\mathfrak{S}_0$ satisfies the first condition of applicability of R on it, with a ψ' such that $\psi \models \psi'$. It is then sufficient to show that, for every $b \in B'$, there is a φ such that $b^{\varphi} \in A_0$ and $\psi \models \phi$. As R is not applicable to $\mathfrak{S}_0$, for every $b \in B'$ ther must be a φ such that $b^{\varphi} \in A_0$ and $\psi' \models \varphi$. But then, $\psi \models \varphi$. Hence, it holds that $\mathsf{R}(\mathfrak{S}) \subseteq \mathfrak{S}_0$. ■

This lemma states that any saturated S_{j}-state for which a S_{j}-state $\mathfrak{S}$ is a substate must contain all the elements that would be added by any of the rules that are applicable to $\mathfrak{S}$. Thus, if there are two saturated S_{j}-states obtained by rule application from a common substate, they must be equivalent.

Theorem 2.17 *Let S_{j} be a deterministic jalal and Γ an input. If $\mathfrak{S}$ and $\mathfrak{S}'$ are two saturated S_{j}-states for Γ, then $\mathfrak{S} = \mathfrak{S}'$.*

Proof. As $\mathfrak{S}'$ is a S_{j}-state for Γ, there must be a sequence of rules $\mathsf{R}_1, \ldots, \mathsf{R}_n$ that lead, by means of its application, from $\Gamma^{S_{\mathsf{j}}}$ to $\mathfrak{S}'$. Since $\mathfrak{S}$ is a S_{j}-state for Γ, it is the case that $\Gamma^{S_{\mathsf{j}}} \subseteq \mathfrak{S}$. By Lemma 2.16, $\mathsf{R}_1(\Gamma^{S_{\mathsf{j}}}) \subseteq \mathfrak{S}$. Repeating the same argument leads to $\mathfrak{S}' \subseteq \mathfrak{S}$. Analogously, $\mathfrak{S} \subseteq \mathfrak{S}'$. Hence, $\mathfrak{S} = \mathfrak{S}'$. ■

This theorem, as in the tableau case, implies that in order to verify whether a property is satisfied by an input Γ, it is sufficient to find one saturated S_{j}-state for Γ, since any other will have the same elements, labeled with equivalent propositional formulas and so, if one contains a clash, all of them will.

After executing the jalal on an axiomatized input, one obtains a set of assertions labeled with monotonic propositional formulas. These formulas are meant to explain why each assertion is there and, given the case, why were the clashes produced. The next step is to use these explanations to find the maximal subsets of axioms that avoid the appearances of clashes in the assertion set, when used with the same input element. This will be done with the help of the clash formula.

Definition 2.18 (Clash formula) *Let $\mathfrak{S}$ be a saturated S_{j}-state for Γ, where S is a sound and complete tableau for a property $\mathcal{P}$. A particular clash $C = \{a_1^{\phi_1}, \ldots, a_n^{\phi_n}\} \in \mathcal{C}$ is* expressed *by the formula $\bigwedge_{i=1}^{n} \phi_i$. Let $\varphi_1, \ldots, \varphi_m$ be the formulas expressing all the clashes in $\mathfrak{S}$. The* clash formula for Γ *is*

$$\bigvee_{j=1}^{m} \varphi_j$$

The following proposition shows how the clash formula can be used to find subsets of axioms for which the jalal – or, in that case, the tableau – would produce no clashes, when applied with the same input.

Proposition 2.19 *Let S be a sound and complete tableau for a property $P \subseteq \mathfrak{I} \times \mathscr{P}(\mathfrak{T}), \Gamma = (\mathcal{I}, \mathcal{T})$ be an axiomatized input and ψ the clash formula for Γ. Let $\Theta \subseteq \mathcal{T}$, and ω be the valuation that maps the propositional variables corresponding to elements of Θ to* true*, and the rest to* false*. Then $(\mathcal{I}, \Theta) \notin \mathcal{P}$ if and only if ψ evaluates to* true *under ω.*

In order to make the proof of this proposition more clear, two lemmas will be shown first. These lemmas rely on the concept of ω-projection, which has not yet been introduced.

Definition 2.20 (ω-projection) *Let $\mathfrak{S}$ be a S_{j}-state and ω a valuation. The* ω-projection of $\mathfrak{S}$, *denoted as $\omega(\mathfrak{S})$, is an unlabeled set obtained from $\mathfrak{S}$ by removing all elements whose labels evaluate to* false *under ω, and then removing the labels of the remaining elements.*

A simple example of a ω-projection is the remotion of all the labels from the states. This is done using the valuation that maps every propositional variable to true, thus no label will ever evaluate to false– since they are monotonic propositional formulas – and only the remotion of labels will be done.

Lemma 2.21 *Let S_{j} be a deterministic jalal, $\mathfrak{S}_0$ a S_{j}-state, $\mathfrak{S}_1 = \mathsf{R}'(\mathfrak{S}_0)$ for some jalal rule R' constructed from the tableau rule R, and ω a valuation. Then either $\omega(\mathfrak{S}_1) = \omega(\mathfrak{S}_0)$, or $\omega(\mathfrak{S}_1) = \mathsf{R}(\omega(\mathfrak{S}_0))$.*

Proof. Let R' be the rule $(\{a_1^{\phi_1}, \ldots, a_n^{\phi_n}\}, \{t_1^{\varphi_1}, \ldots, t_m^{\varphi_m}\}) \xrightarrow{\mathsf{R}'} B^{\psi}$. If $\psi = \bigwedge_{i=1}^{n} \phi_i \wedge \bigwedge_{i=1}^{m} \varphi_i$ evaluates to false under ω, then for every assertion $b \in B$, either b is not present in $\mathfrak{S}_0$ and hence the rule will add it, with the label ψ which evaluates to false, or it was already present, and hence its previous label will be modified to add a disjunction with ψ; and thus, the new label evaluates to the same truth value as the previous one did, under ω. This implies that no new elements are added to the ω-projection of the state after the application of the rule, and so $\omega(\mathfrak{S}_1) = \omega(\mathfrak{S}_0)$.

If, on the contrary, ψ evaluates to true under ω, then every element added to the assertion set will also be labeled with a formula that evaluates to true under ω, and hence will be also added to the ω-projection. The elements that were already present in the assertion set, will have their labels modified to be disjuncted with ψ, and for that reason, regardless of their previous label, they will be included in the ω-projection. Thus, every element obtained by the application of the rule will be now in $\omega(\mathfrak{S}_1)$. ■

This lemma entails that it makes no difference for the result if one first applies a rule to a S_{j}-state and then calculates its ω-projection or if one uses the opposite order, obtaining first the ω-projection, and then applying the same rule, regardless of the valuation that is used for that. One of the consequences of this is that if one executes a jalal over an axiomatized input $(\mathcal{I}, \mathcal{T})$ and then computes its ω-projection, for a valuation ω, the result would be the same as applying the same rules on the input $(\mathcal{I}, \Theta)$, where Θ is the set of all the axioms whose labels are mapped to true by ω.

Lemma 2.22 *Let S be a deterministic tableau, $\mathfrak{S}$ be a saturated S_{j}-state and ω a valuation; then $\omega(\mathfrak{S})$ is a saturated S-state.*

Proof. Let $\mathfrak{S} = (A, \mathcal{T})$ and $\mathsf{R} : (B, \mathcal{T}') \xrightarrow{\mathsf{R}} B'$ be a rule of S such that $B \subseteq \omega(A)$ and $\mathcal{T}' \subseteq \omega(\mathcal{T})$, with jalal version R' having laft-hand-side labels ϕ_i, for $i \in \{1, \ldots, n\}$ and φ_i, for $i \in \{1, \ldots, m\}$. Since $B \subseteq \omega(A)$ and $\mathcal{T}' \subseteq \omega(\mathcal{T})$, every formula ϕ_i and φ_i must evaluate to true under ω, and hence, the formula $\psi = \bigwedge_{i=1}^{n} \phi_i \wedge \bigwedge_{i=1}^{m} \varphi_i$ must evaluate to true too.

To show that $\omega(\mathfrak{S})$ is saturated, it is sufficient to show that $B' \subseteq \omega(A)$. Since $\mathfrak{S}$ is saturated, R' is not applicable to it. This means that, for every $b \in B'$, there is a ϕ such that $b^{\phi} \in A$ and $\psi \models \phi$. As ψ evaluates to true under ω, so does ϕ; hence, $b \in \omega(A)$. ∎

With the help of these lemmas, it is now easy to prove Proposition 2.19.

Proof of Proposition 2.19 Let $\mathfrak{S}$ be a saturated S_{j}-state for Γ; then, by Lemma 2.22, $\omega(\mathfrak{S})$ is a saturated S-state. Since $\omega(\Gamma^{S_{\mathsf{j}}}) = (\mathcal{I}, \Theta)^{S}$, then by Lemma 2.21 $\omega(\mathfrak{S})$ is a S-state for $(\mathcal{I}, \Theta)$. As S is sound and complete for $\mathcal{P}$, $(\mathcal{I}, \Theta)$ $/\!\!\mathcal{P}$ iff $\omega(\mathfrak{S})$ contains a clash. A particular clash C is present in $\omega(\mathfrak{S})$ iff for every element $c^{\vartheta} \in C$, it holds that $c^{\vartheta} \in \mathfrak{S}$ and ϑ evaluates to true under ω. Let now $\psi_1, \ldots, \psi_m$ be the formulas expressing all the clashes in $\mathfrak{S}$, as in Definition 2.18.

Obviously, $\omega(\mathfrak{S})$ contains a clash iff ω evaluates any of these ψ_i's to true. In other words, $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff ω evaluates $\bigvee_{j=1}^{m} \psi_j$ to true. ∎

A simple consequence of this proposition is the soundness and completeness of a deterministic jalal, under the assumption that the deterministic tableau from which it was constructed is sound and complete for the property being decided.

Corollary 2.23 *Let S be a sound and complete deterministic tableau for a property $\mathcal{P}$, then S_{j} is sound and complete for $\mathcal{P}$.*

Proof. By Proposition 2.19, $(\mathcal{I}, \mathcal{T}) \notin \mathcal{P}$ iff the clash formula evaluates to true under the valuation that maps all the propositional variables in $\mathcal{T}$ to true. Hence, all the formulas appearing as labels in the elements are mapped to true under this valuation. Hence, every formula representing a particular clash is mapped to true, and so is the clash-formula if at least one clash exists. Thus, $(\mathcal{I}, \mathcal{T}) \in \mathcal{P}$ iff there is a saturated and clash-free S_{j}-state for $(\mathcal{I}, \mathcal{T})$. ∎

The previous proposition and corollary show that deterministic jalals may be used instead of the deterministic tableaus from which they were constructed to not only decide the property, but if it is the case that the property is not held, to find out explanations for this, and with the help of these explanations, obtain sub-sets of axioms for which the property holds, keeping the rest of the input unchanged. To do this, one only needs to find valuations that map the clash formula obtained at the end of the application of the jalal to false. In fact, the interesting valuations would be those that map the maximum amount possible of propositional variables to true, since, as the labels are all monotonic propositional formulas, any subset of them would also set the valuation of the clash formula to false. The problem of finding such maximal valuations is NP-complete [BH95]; it can, nonetheless be sometimes optimized, for example by the method described in [Rym92].

Before one can replace the use of tableaus by jalals to solve the mentioned problem, it is still necessary to show that the execution of a deterministic jalal will give a result after a finite number of steps, if the tableau from which it was constructed also does so; otherwise, it would have no sense to apply a jalal without knowing if an answer will be ever given.

Definition 2.24 (Termination) *Let S (S_{j}) be a deterministic tableau (jalal), and Γ an axiomatized input. S (S_{j})* terminates for Γ *if, independently of the order of application of rules, a saturated S-state (S_{j}-state) for Γ is reached after finitely many rule applications.*

Theorem 2.25 *Let S be a deterministic tableau such that, for every rule of the form $\mathfrak{R} \xrightarrow{\mathsf{R}} B$, B is finite, and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input where $\mathcal{I}^{S_{\mathsf{j}}}$ and $\mathcal{T}$ are both finite. If S terminates for Γ, then also S_{j} terminates for Γ.*

Proof. Since S terminates, Γ^S is finite, and every rule application adds only finitely many new elements to a S-state, any saturated S-state for Γ must be finite. Let $\mathfrak{S}$ be a saturated S_{j}-state for Γ and unl the valuation that maps every propositional variable in $\mathcal{T}$ to true. Then $\mathsf{unl}(\mathfrak{S})$ is a saturated S-state for Γ, and hence finite.

Let $\mathfrak{S}_0$ be a S_{j}-state for Γ, R a rule applicable to it, and $\mathfrak{S}_1 = \mathsf{R}(\mathfrak{S}_0)$. By Lemma 2.21, either $\mathsf{unl}(\mathfrak{S}_0) = \mathsf{unl}(\mathfrak{S}_1)$ or $\mathsf{unl}(\mathfrak{S}_0) \subset \mathsf{unl}(\mathfrak{S}_1)$. In the latter case, the application of rule R adds at least one new element to $\mathsf{unl}(\mathfrak{S}_0)$. The finiteness of $\mathsf{unl}(\mathfrak{S})$ entails that rules that satisfy this condition can be applied only finitely many times.

Now for the other case, when $\mathsf{unl}(\mathfrak{S}_0) = \mathsf{unl}(\mathfrak{S}_1)$. As $\mathcal{T}$ is finite, there are only finitely many different valuations with respect to $\hat{\mathcal{T}}$. For each of these valuations ω, let $|\mathfrak{S}_0|_\omega$ be the number of elements in $\mathfrak{S}_0$ whose label is mapped to true by ω. For each possible ω, as all the labels are monotonic propositional formulas, $|\mathfrak{S}_0|_\omega \leq |\mathfrak{S}_0|_{\mathsf{unl}}$; hence, it is finite. Let val be the number of all different valuations. The sum of $|\mathfrak{S}_0|_\omega$ over all possible valuations ω is then bounded by $\mathsf{val}\cdot|\mathfrak{S}_0|_{\mathsf{unl}}$; call this sum $\Sigma(\mathfrak{S}_0)$. Since $\mathsf{unl}(\mathfrak{S}_0) = \mathsf{unl}(\mathfrak{S}_1)$, and the rule R is applicable to $\mathfrak{S}_0$, there must exist a b^ϕ in $\mathfrak{S}_0$ appearing in the right-hand-side of R such that $\psi \not\models \phi$, where ψ is the label added to the new elements. Hence, there must be a valuation ω that maps ψ to true, but ϕ to false. Since $b^{\phi\vee\psi}$ will appear in $\mathfrak{S}_1$, $\Sigma(\mathfrak{S}_0) < \Sigma(\mathfrak{S}_1)$. Thus, rules that do not add new elements to a S_j-state and only modify the labels can only be applied finitely many times before applying one that adds a new element. Since the rules that add new elements can also be applied only finitely many times, S_j must terminate for Γ. ∎

The finiteness assumptions given in the previous theorem are necessary for the termination. The next example shows a case when they are violated and termination of the jalal does not hold, even when the original tableau does terminate.

Example 2.26 *Let $\mathfrak{I} = \{a\}$ and $\mathfrak{T} = \{t_1, t_2, \ldots\}$ be an infinite set. Let $S = (\{a, \bot\}, \mathsf{id}, \mathcal{R}, \{\{\bot\}\})$ be a deterministic tableau for $\mathfrak{I}$,$\mathfrak{T}$ where* id *is the identity function and $\mathcal{R}$ contains only the rule:*

$$\mathsf{R} : (\{a\}, \{p\}) \xrightarrow{\mathsf{R}} \{\bot\}$$

S terminates for the axiomatized input $\Gamma = (\{a\}, \mathfrak{T})$ since once the rule R *has been applied, the element $\bot$ is in the assertion set, and hence* R *is not applicable anymore. Nonetheless, S_j does not terminate for Γ as the rule* R' *can always be applied using an axiom that has not been used before, since the input contains infinitely many of them. Thus, the jalal cannot reach a saturated S_j-state after any finite amount of applications of the rule* R.

The conditions of requiring finiteness in the distinct elements of a tableau to ensure termination of the jalal judging it are not too restrictive, since in most practical cases these conditions are naturally held.

This section has dealt in detail with deterministic tableaus. As it has been mentioned several times before, these tableaus define only a very basic kind of decision procedure. In the next section, it will be extended to add non-deterministic rules in order to be able to solve a wider class of decision problems. It will be shown that the pinpointing method can be used also when non-deterministic rules are used.

3 Adding non-determinism

For the tableaus defined in the previous section, given one input there is only one state with which the decision procedure begins, and the application of a rule leads always to one unique state; that is the reason of the name *deterministic* given to them. As it was said before, even in this simple framework a kind of non-determinism arises, produced by the different application orders that can be used when more than one rule is applicable on the same state. For this case, Theorem 2.8 shows that this non-determinism can be neglected as the result will never depend on the order chosen; in other words, is a *don't care* non-determinism.

There exist another kind of non-determinism, *don't know* non-determinism, which is sometimes desirable to state within a decision procedure. The intuitive meaning of it is that, if there are several possible search routes, one of them should lead to the positive solution, but it is not know in advance exactly which one does. When this happens, it must be the case that all the routes lead to a negative solution in order to state a negative result, but the moment one route is found yielding a positive one, the positive answer can be stated.

In this section, the concept of tableau will be generalized to include this kind of non-determinism. It will be shown that the pinpointing method using jalals can also be extended for this kind of tableaus.

Definition 3.1 (Non-deterministic tableau) *Let $\mathfrak{I}$ be a set of inputs and $\mathcal{T}$ a set of axioms. A* non-deterministic tableau for $\mathfrak{I}$,$\mathfrak{T}$ *is a tuple $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{R})$, where $\mathcal{A}$ and $\mathcal{C}$ are as in Definition 2.1; the function $\cdot^{S_\mathfrak{I}}$ maps each $\mathcal{I} \in \mathfrak{I}$ to a finite set of sets of assertions $\{A_1, \dots, A_n\}$; the function $\cdot^{S}$ extends $\cdot^{S_\mathfrak{I}}$ by mapping an input $(\mathcal{I}, \mathcal{T}) \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$ to the set of S-states $\{(A_1, \mathcal{T}), \dots, (A_n, \mathcal{T})\}$, where $A_i \in \mathcal{I}^{S_\mathfrak{I}}$.*

$\mathcal{R}$ is a set of rules $\{\mathsf{R}_1, \dots, \mathsf{R}_n\}$ of the form

$$(A, \mathcal{T}) \xrightarrow{\mathsf{R}_i} \{B_1, \dots, B_{m_i}\}.$$

For a S-state $\mathfrak{S} = (A, \mathcal{T})$ and a rule $(B, \mathcal{T}') \xrightarrow{\mathsf{R}} \{B_1, \dots, B_m\}$, R *is* applicable to $\mathfrak{S}$ *if $B \subseteq A$ and for every $1 \leq i \leq m$ it holds that $B_i \not\subseteq A$. The* result of applying R to $\mathfrak{S}$ *is the set of S-states* $\mathsf{R}(\mathfrak{S}) = \{(A \cup B_i, \mathcal{T}) \mid 1 \leq i \leq m\}$.

Notice that this is indeed a generalization of deterministic tableaus, since if when applying the function $\cdot^{S}$ to any input, the result consists of just one set of assertions, and the rules have on the right-hand-side only one set of assertions too, this definition is exactly the same as the one given in the previous section. The notions of *saturated*, *clash-free*, *sound*, and *complete* are defined exactly as in Section 2.

Example 3.2 *Let $\mathcal{V}$ be a set of propositional variables. A* propositional definition *is of the form $p \doteq \phi$ where $p \in \mathcal{V}$ and ϕ is a propositional formula over $\mathcal{V}$; in this case, p is called* defined, *and ϕ is its* definition. *Without loss of generality, suppose that every propositional formula is given in negation normal form. A set of propositional definitions is* acyclic *if there are no multiple definitions; that is, no two propositional definitions $p \doteq \phi_1$ and $p \doteq \phi_2$; and no cyclic definitions; that is, a sequence of concept definitions $p_1 \doteq \phi_1, \dots p_k \doteq \phi_k$ such that p_{i+1} appears in ϕ_i, and p_1 appears in ϕ_k.*

A propositional formula is satisfiable with respect to an acyclic set of propositional definitions *if the propositional formula obtained by replacing every defined propositional variable by its definition leads to a satisfiable propositional formula.*

A tableau for verifying whether a propositional formula is satisfiable with respect to an acyclic set of propositional definitions is given by $S_{\mathsf{Sat}} = (\mathcal{A}, \cdot^{S_{\mathsf{Sat}}\mathfrak{I}}, \mathcal{R}, \mathcal{C})$, where $\mathcal{A} = \mathbb{B}(\mathcal{V})$; for an axiomatized input $\Gamma = (\phi, \mathcal{T}) \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$ where ϕ is a propositional formula and $\mathcal{T}$ is a set of propositional definitions, $\Gamma^S = \{(\phi, \mathcal{T})\}$; $\mathcal{C} = \{\{p, \neg p\} \mid p \in \mathcal{V}\}$, and $\mathcal{R}$ consists of the following rules:

$$
\begin{aligned}
\mathsf{R}\vee &: (\{\phi \vee \varphi\}, \emptyset) \xrightarrow{\mathsf{R}\vee} \{\{\phi\}, \{\varphi\}\} \\
\mathsf{R}\wedge &: (\{\phi \wedge \varphi\}, \emptyset) \xrightarrow{\mathsf{R}\wedge} \{\{\phi, \varphi\}\} \\
\mathsf{R}\doteq^{+} &: (\{p\}, \{p \doteq \phi\}) \xrightarrow{\mathsf{R}\doteq^{+}} \{\{\phi\}\} \\
\mathsf{R}\doteq^{-} &: (\{\neg p\}, \{p \doteq \phi\}) \xrightarrow{\mathsf{R}\doteq^{-}} \{\{\mathsf{nnf}(\neg\phi)\}\}
\end{aligned}
$$

The tableau S_{Sat} is sound and complete for the property $\mathcal{P} = \{(\mathcal{I}, \mathcal{T}) \mid \mathcal{I}$ is satisfiable with respect to $\mathcal{T}\}$. Intuitively, the tableau breaks the formula appart in its conjuncts, and is capable of selecting the disjuncts that will be satisfied. If there is no possible selection of this disjuncts which is satisfiable, that is, if for every breaking out of the formula it turns out that both p and $\neg p$ must be satisfied, then the answer is that the formula is not satisfiable with respect to the set of propositional definitions.

Figure 1 shows the assertion sets belonging to some S_{Sat}-states for the axiomatized input $(\{p \wedge q\}, \{q \doteq \neg p \vee r\})$ obtained by applying the rule $\mathsf{R}\wedge$, followed by $\mathsf{R}\doteq^{+}$ and then by $\mathsf{R}\vee$, where the branching is caused by the non-determinism of this rule, having two successors from its application. From the assertion set, the S_{Sat}-state is completely defined, since the axiom part remains always unchanged.

$$
\begin{tikzcd}[column sep=small]
 & \{p \wedge q\} \arrow[d, dash] & \\
 & \{p \wedge q, p, q\} \arrow[d, dash] & \\
 & \{p \wedge q, p, q, \neg p \vee r\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{p \wedge q, p, q, \neg p \vee r, \neg p\} & & \{p \wedge q, p, q, \neg p \vee r, r\}
\end{tikzcd}
$$

Figure 1: Assertional sets of S_{Sat}-states for the input $(\{p \wedge q\}, \{q \doteq \neg p \vee r\})$

The S_{Sat}-state on the right-hand side of the branching, defined by the assertion set $\{p \wedge q, p, q, \neg p \vee r, r\}$, is saturated and clash-free. Hence, the propositional formula $p \wedge q$ is satisfiable with respect to the propositional definitions $q \doteq \neg p \vee q$.

Notice that the reason why the state used in the previous example is saturated is that the applicability of a rule requires that, for each of the assertion sets in the right-hand-side of the rule, there is at least one element in it that is not in the assertion set of the S-state to which it is being applied. In this particular case, the state contains the assertion r, which is the only element of one of the sets and hence the rule $\mathsf{R}\vee$ is not applicable even when $\neg p$ is not in it.

The *non-deterministic jalal judging a non-deterministic tableau* can be defined using a straight-forward adaptation of Definition 2.10 for deterministic jalals. The only important step is that, when defining the labeled versions of the rules, every set in the right-hand-side of the rule must be labeled with the same formula, which is the conjunction of the formulas in the left-hand-side. The applicability condition for non-deterministic jalal rules is also a simple adaptation of the applicability conditions of deterministic jalal and non-deterministic tableau rules.

Since the initial function and the application of rules for non-deterministic tableaus lead to sets of states instead of single states, some notions defined in the previous section need to be reformulated to handle this generalization.

Definition 3.3 (S-state (S_{j}-state) for Γ) *Let S (S_{j}) be a non-deterministic tableau (jalal) and Γ an axiomatized input. The* set of S-states (S_{j}-states) for Γ *is defined inductively as follows:*

- *every S-state (S_{j}-state) $\mathfrak{S} \in \Gamma^S$ ($\mathfrak{S} \in \Gamma^{S_{\mathsf{j}}}$) is a S-state (S_{j}-state) for Γ;*

- *if* $\mathfrak{S}$ *is a* S*-state* (S_{j}*-state) for* Γ *and* R *is applicable to* $\mathfrak{S}$*, then every* S*-state* (S_{j}*-state) in* $\mathsf{R}(\mathfrak{S})$ *is a* S*-state* (S_{j}*-state) for* Γ.

As it was seen in Example 3.2, in non-deterministic tableaus it is possible to obtain several different saturated S-states for a single axiomatized input Γ, some of which may contain clashes while others are clash-free, and hence Theorem 2.8 does not hold anymore in the non-deterministic framework. It will be shown, nonetheless, that even for non-deterministic tableaus, the order of rule application is again a *don't care* non-determinism since, if a saturated S-state is reached after applying a sequence of rules then, independently on the order in which such rules are applied, the same S-state can always be reached. In particular, if there exists a saturated and clash-free S-state for Γ, the order in which the rules are applied will be irrelevant for producing it.

Lemma 3.4 *Let* S *be a non-deterministic tableau,* $\mathfrak{S}_0 = (A_0, \mathcal{T}_0)$*, and* $\mathfrak{S} = (A, \mathcal{T})$ *two* S*-states such that* $\mathfrak{S} \subseteq \mathfrak{S}_0$ *and* $\mathfrak{S}_0$ *is saturated, and* R *the rule* $(B, \mathcal{T}') \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$ *applicable to* $\mathfrak{S}$*. Then, there is a* S*-state* $\mathfrak{S}' \in \mathsf{R}(\mathfrak{S})$ *such that* $\mathfrak{S}' \subseteq \mathfrak{S}_0$.

Proof. Since $\mathfrak{S}_0$ is saturated, R is not applicable to it. But it is the case that $B \subseteq A \subseteq A_0$ and $\mathcal{T}' \subseteq \mathcal{T} \subseteq \mathcal{T}_0$; hence, the only way that R is not applicable to $\mathfrak{S}_0$ is that there is a $1 \leq i \leq n$ such that $B_i \subseteq A_0$. But then, the S-state $\mathfrak{S}' = (A \cup B_i, \mathcal{T}) \in \mathsf{R}(\mathfrak{S})$ is such that $\mathfrak{S}' \subseteq \mathfrak{S}_0$. ∎

An analogous proof can be used to show this same result for non-deterministic jalals.

Now, if for a given axiomatized input a tableau answers that the property it decides does not hold, it must be the case that every saturated S-state for it contains a clash. As it has been already said, in the non-deterministic framework there might be several different saturated S-states for that input. If one wants then to find the maximal subset of axioms for which the property holds, it is not sufficient to look into only one of those saturated states, as was done in the previous section, but it is necessary to look into all of them at the same time and remove axioms such that at least one of those states becomes clash-free. Is for this reason that the clash formula has to be redefined to fit this framework.

Recall that the clash formula for the deterministic case had a disjunction of all the formulas expressing clashes in the saturated state. To falsify it, it was necessary that none of the clashes in it was held anymore. In the non-deterministic case, it is necessary to ensure that there is at least one saturated state for which none of the clashes holds anymore. For that reason, the new clash formula will be formed by the conjunction over all the saturated states, of the formula formed by the disjunction of all the formulas representing clashes in them, similar to the deterministic clash formula.

Definition 3.5 (Clash formula) *Let S be a sound and complete non-deterministic tableau for a property $\mathcal{P}$. Let $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ be all the saturated S_{j}-states for an axiomatized input Γ. For each $1 \leq i \leq n$, let $\psi_{i,1}, \ldots, \psi_{i,k_i}$ be the formulas expressing all the clashes in $\mathfrak{S}_i$. The* clash formula associated with Γ and $\mathcal{P}$ *is:*

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k_i} \psi_{i,j}$$

This clash formula will be used in the exact same way the deterministic one was used in the previous section. A valuation that makes the formula false can be used to find a set of axioms for which the property holds.

Proposition 3.6 *Let S be a sound and complete non-deterministic tableau for a property $\mathcal{P}$; $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input and ψ the clash formula associated with Γ and $\mathcal{P}$. Let $\Theta \subseteq \mathcal{T}$ and ω be the valuation that maps the propositional variables corresponding to the elements of Θ to true and the rest to false. Then $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff ψ evaluates to true under ω.*

To prove this proposition an adaptation of Lemma 2.21 will be used, the proof of which will be ommited since is almost identical to the proof given in the previous section. Lemma 2.22 can be used in this framework without any modification.

Lemma 3.7 *Let $\mathfrak{S}_0$ be a S_{j}-state, $\{\mathfrak{S}_1, \ldots, \mathfrak{S}_n\} = \mathsf{R}'(\mathfrak{S}_0)$ and ω a valuation. Then either $\omega(\mathfrak{S}_i) = \omega(\mathfrak{S}_0)$ for all $1 \leq i \leq n$ or $\{\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)\} = \mathsf{R}(\omega(\mathfrak{S}_0))$, where R is the tableau rule from which R' was constructed.*

This adaptation of Lemma 2.21 changes only the fact that there is only one possible state after applying the rule, to the set of states induced by non-deterministic rules. Nonetheless, the proof of that lemma, given in Section 2 does not assume anything on the result of applying the rule, and hence, a direct translation of that proof can also be used in the non-deterministic framework.

Proof of Proposition 3.6 Let $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ be all the saturated S_{j}-states for Γ. Then, for every $1 \leq i \leq n$, $\omega(\mathfrak{S}_i)$ is a saturated S-state. Since $\{\omega(\mathfrak{S}) \mid \mathfrak{S} \in \Gamma^{S_{\mathsf{j}}}\} = (\mathcal{I}, \Theta)^S$, every $\omega(\mathfrak{S}_i)$ is a saturated S-state for $(\mathcal{I}, \Theta)$ – see Lemma 3.7.

Let now $\mathfrak{S}$ be a saturated S-state for $(\mathcal{I}, \Theta)$. Then there is a sequence of S-states $\mathfrak{Q}_0, \ldots, \mathfrak{Q}_m$ such that $\mathfrak{Q}_0 \in (\mathcal{I}, \Theta)^S$, $\mathfrak{Q}_m = \mathfrak{S}$, and for every $0 \leq i < m$ there is a rule R_i of S such that $\mathfrak{Q}_{i+1} \in \mathsf{R}_i(\mathfrak{Q}_i)$. It is easy to see, by Lemma 3.7, that the S_{j}-state $\mathfrak{S}'$ obtained by applying the corresponding jalal rule R'_i and selecting the corresponding element in the resulting set is such that $\omega(\mathfrak{S}') = \mathfrak{S}$. Further application of the same lemma shows that there is a saturated S_{j}-state $\mathfrak{S}''$ with the same property, that is $\omega(\mathfrak{S}'') = \mathfrak{S}$. Hence, $\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)$ are all the saturated S-states for $(\mathcal{I}, \Theta)$.

Then, $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff every $\omega(\mathfrak{S}_i)$ contains a clash. A particular clash C is present in $\omega(\mathfrak{S}_i)$ iff for every element c^ϕ in C, $c^\phi \in \mathfrak{S}_i$ and ω evaluates ϕ to true. Let now $\psi_{i,1}, \ldots, \psi_{i,k_i}$ be the formulas expressing clashes in $\mathfrak{S}_i$. Obviously, $\omega(\mathfrak{S}_i)$ contains a clash iff $\bigvee_{j=1}^{k_i} \psi_{i,j}$ evaluates to true under ω. Thus, all the saturated S-states for Γ $\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)$ contain a clash iff the clash formula evaluates to true under ω. ∎

The transference of termination for non-deterministic tableaus can easily be shown adapting Theorem 2.25; if a non-deterministic tableau terminates, then the non-deterministic jalal judging it terminates too.

Although non-determinitic tableaus generalize the concept of deterministic tableaus to include some uncertainty in the selection of a path to find a saturated and clash-free state, it turns out that they are still too restricted for some decision problems. To keep general assertions in the set is sometimes insufficient to represent the information that is necessary to solve the problem. For example, one may need to express that certain assertions hold only for some specific elements, while other assertions are satisfied by completely different elements in the same domain. The next section introduces a further generalization to the concept of tableau, aimed to deal with this lack of expresivity, by adding the use of variables within the assertions.

4 Use of variables

One way to generalize the notion of tableau introduced in the previous sections is to allow the use of variables in the sets that form the states. Assertions will then be able to express that some information holds for all the elements in a certain domain, or just for one of them, or that a pair of elements keep certain relationship between them. For the framework presented in this section, the variables will be divided into two sets: the *assertion variables* and the *tableau variables*, which range over the set of assertion variables. The tableau variables are used when defining the rules or clashes to state the presence of a particular element; assertion variables represent those particular elements, held inside the assertion set.

4.1 Variables in assertions

One first generalization consists in including the use of variables only in the assertion set, and leaving the rest of the elements forming a tableau unchanged. Later in this section, this will be further generalized to allow also the use of variables in the axiom set.

Definition 4.1 (Variable tableau) *Let $\mathcal{V}$,$\mathcal{W}$ be two disjoint sets of* assertion variables *and* tableau variables*, respectively, $\mathfrak{I}$ a set of inputs and $\mathfrak{T}$ a set of axioms. A* variable tableau *for $\mathfrak{I}$,$\mathfrak{T}$ is a tuple of the form $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$, where $\mathcal{A} = \bigcup_{i\geq 0} \mathcal{A}^{(i)}$, with each $\mathcal{A}^{(i)}$ a* set of assertions of arity *i; if $\mathcal{U}$ is a set of variables, then $\mathcal{A}(\mathcal{U}) = \{(u_1, \ldots, u_i) : A \mid A \in \mathcal{A}^{(i)}, u_j \in \mathcal{U}\}$ denotes the set of* assertions with variables over *$\mathcal{U}$; a S-state is an element of $\mathscr{P}(\mathcal{A}(\mathcal{V})) \times \mathscr{P}(\mathfrak{T})$.*

The function $\cdot^{S_\mathfrak{I}}$ maps each $\mathcal{I} \in \mathfrak{I}$ to a set $\{A_1, \ldots, A_n\}$, where each $A_i \subseteq \mathcal{A}(\mathcal{V})$; $\cdot^S$ extends $\cdot^{S_\mathfrak{I}}$ by mapping an axiomatized input $(\mathcal{I}, \mathcal{T}) \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$ to the set of S-states $\{(A_i, \mathcal{T}) \mid A_i \in \mathcal{I}^{S_\mathfrak{I}}, 1 \leq i \leq n\}$.

$\mathcal{R}$ is a set of rules *of the form $(A, \mathcal{T}) \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$, where $A, B_i \subseteq \mathcal{A}(\mathcal{W})$ and $\mathcal{T} \subseteq \mathfrak{T}$. $\mathcal{C}$ is a subset of $\mathscr{P}(\mathcal{A}(\mathcal{W}))$.*

For this tableau, the previous definitions of applicability become useless, since it is unclear what must be done with the variables in the assertions. Furthermore, the way the two different sets of variables interact has not been stated. Their use should become clear with the following definition.

Definition 4.2 ($\mathcal{V}$-valuation,applicability) *Given a set $W \in \mathcal{W}$, a* $\mathcal{V}$-valuation *of W is a function $\varrho : W \to \mathcal{V}$. Given two $\mathcal{V}$-valuations $\varrho : W \to \mathcal{V}$ and $\varrho' : W' \to \mathcal{W}$, it is said that ϱ'* extends *ϱ if it holds that $W \subseteq W'$ and for every $w \in W$, $\varrho(w) = \varrho'(w)$. For a set of assertions with variables A, the set of variables appearing in A is denoted as* $\mathsf{var}(A)$.

Given a set $A \subseteq \mathcal{A}(\mathcal{W})$ and a $\mathcal{V}$-valuation ϱ for $\mathsf{var}(A)$, A^ϱ is the set obtained by replacing every variable $x \in \mathsf{var}(A)$ by $\varrho(x)$; that is, $A^\varrho = \{(\varrho(x_1), \ldots, \varrho(x_i)) : B \mid (x_1, \ldots, x_i) : B \in A\} \subseteq \mathcal{A}(\mathcal{V})$.

$$
\begin{aligned}
\mathsf{R}\sqcap &: (\{y_0 : D_1 \sqcap D_2\}, \emptyset) \xrightarrow{\mathsf{R}\sqcap} \{\{y_0 : D_1, y_0 : D_2\}\} \\
\mathsf{R}\sqcup &: (\{y_0 : D_1 \sqcup D_2\}, \emptyset) \xrightarrow{\mathsf{R}\sqcup} \{\{y_0 : D_1\}, \{y_0 : D_2\}\} \\
\mathsf{R}\exists &: (\{y_0 : \exists r.D\}, \emptyset) \xrightarrow{\mathsf{R}\exists} \{\{y_1 : D, (y_0, y_1) : r\}\} \\
\mathsf{R}\forall &: (\{y_0 : \forall r.D, (y_0, y_1) : r\}, \emptyset) \xrightarrow{\mathsf{R}\forall} \{\{y_1 : D\}\} \\
\mathsf{R}\doteq^{+} &: (\{y_0 : A\}, \{A \doteq D\}) \xrightarrow{\mathsf{R}\doteq^{+}} \{\{y_0 : D\}\} \\
\mathsf{R}\doteq^{-} &: (\{y_0 : \neg A\}, \{A \doteq D\}) \xrightarrow{\mathsf{R}\doteq^{-}} \{\{y_0 : \mathsf{nnf}(\neg D)\}\}
\end{aligned}
$$

Figure 2: Rules for a tableau checking satisfiability of $\mathcal{ALC}$-concept terms

Given a S-state $\mathfrak{S} = (A, \mathcal{T})$ and a rule $(B, \mathcal{T}') \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$, R *is* applicable to $\mathfrak{S}$ *if there is a $\mathcal{V}$-valuation ϱ for $\mathsf{var}(B)$ such that $B^\varrho \subseteq A, \mathcal{T}' \subseteq \mathcal{T}$, and for every $1 \leq i \leq n$ and every $\mathcal{V}$-valuation σ for $\mathsf{var}(B) \cup \mathsf{var}(B_i)$ extending ϱ, it holds that $B_i^\sigma \not\subseteq A$.*

The result of applying R to $\mathfrak{S}$ *is the set of S-states $\mathsf{R}(\mathfrak{S}) = \{(A \cup B_i^\sigma, \mathcal{T}) \mid 1 \leq i \leq n\}$, where σ is a $\mathcal{V}$-valuation for $\mathsf{var}(B) \cup \bigcup_{i=1}^n \mathsf{var}(B_i)$ extending ϱ such that for every $x, y \in (\bigcup_{i=1}^n \mathsf{var}(B_i)) \setminus \mathsf{var}(B)$, $\sigma(x) \in \mathcal{V} \setminus \mathsf{var}(A)$, and if $x \neq y$, then $\sigma(x) \neq \sigma(y)$.*

All the notions of S-state for Γ, saturated, clash-free, soundness and completeness are defined exactly as in the previous section.

Example 4.3 *Let $\mathcal{V} = \{x_i \mid i \in \mathbb{N}\}$ and $\mathcal{W} = \{y_i \mid i \in \mathbb{N}\}$. A tableau for checking satisfiability of an $\mathcal{ALC}$-concept term with respect to an acyclic TBox is given by $S_{\mathcal{ALC}} = (\mathcal{A}, \cdot^{S_{\mathcal{ALC}}}, \mathcal{R}, \mathcal{C})$, where $\mathcal{A}^{(1)}$ is the set of all $\mathcal{ALC}$-concept terms, assuming without loss of generality that these are always given in negation normal form, and $\mathcal{A}^{(2)}$ is the set of all role names; given an input $\Gamma = (C, \mathcal{T})$ with C a concept term and $\mathcal{T}$ an acyclic TBox, $\Gamma^S = \{(\{x_0 : C\}, \mathcal{T})\}$; $\mathcal{R}$ is given by the rules in Figure 2; and $\mathcal{C} = \{\{A, \neg A\} \mid A$ is a concept name$\}$.*

This tableau is a straightforward translation of the method given in [Lut99] for deciding $\mathcal{ALC}$ satisfiability with respect to acyclic TBoxes, and hence is sound and complete for this property.

The rule $\mathsf{R}\exists$ is applicable to the $S_{\mathcal{ALC}}$-state $\mathfrak{S} = (\{x_0 : \exists r.D\}, \emptyset)$, and the resulting state obtained after of such a rule application is $\mathsf{R}\exists(\mathfrak{S}) = \{(x_0 : \exists r.D, x_1 : D, (x_0, x_1) : r\}, \emptyset)$.

As in the case of deterministic and non-deterministic tableaus, the order in which the rules are applied is irrelevant, in the sense that it has no influence to whether a saturated and clash-free state is found or not. In this tableau there is also the possibility to chose many distinct $\mathcal{V}$-valuations by which the rules are applied. As will be shown next, the choice of $\mathcal{V}$-valuation includes also a kind of *don't care* non-determinism, since choosing a different one will only give different names to the assertion variables, but will not modify the underlying structure they form.

Definition 4.4 (Substate,equal) *Let S be a variable tableau and $\mathfrak{S} = (A, \mathcal{T}), \mathfrak{S}' = (A', \mathcal{T}')$ two S-states. $\mathfrak{S}$ is a* substate *of $\mathfrak{S}'$, denoted as $\mathfrak{S} \subseteq \mathfrak{S}'$, if $\mathcal{T} \subseteq \mathcal{T}'$ and there exists a function $f : \mathsf{var}(A) \to \mathsf{var}(A')$ such that if $(x_1, \ldots, x_i) : a \in A$, then $(f(x_1), \ldots, f(x_i)) : a \in A'$.*

$\mathfrak{S}$ and $\mathfrak{S}'$ are equal*, denoted as $\mathfrak{S} = \mathfrak{S}'$, if $\mathfrak{S} \subseteq \mathfrak{S}'$ and $\mathfrak{S}' \subseteq \mathfrak{S}$.*

The definition of a substate, or equal S-states is analogous to the previous ones, with the difference that having two equal S-states does not imply that their elements are exactly the same; they may have different variable names in the assertion sets, but there is a bijection between those variable names.

Lemma 4.5 *Let S be a variable tableau, $\mathfrak{S}_0 = (A_0, \mathcal{T}_0)$ a saturated S-state and $\mathfrak{S} = (A, \mathcal{T})$ a S-state such that $\mathfrak{S} \subseteq \mathfrak{S}_0$; and let R be the rule $(B, \mathcal{T}') \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$ applicable to $\mathfrak{S}$. Then, there is a S-state $\mathfrak{S}' \in \mathsf{R}(\mathfrak{S})$ such that $\mathfrak{S}' \subseteq \mathfrak{S}_0$.*

Proof. Since R is applicable to $\mathfrak{S}$, there is a $\mathcal{V}$-valuation ϱ of $\mathsf{var}(B)$ such that $B^\varrho \subseteq A$ and $\mathcal{T}' \subseteq \mathcal{T}$. As $\mathfrak{S} \subseteq \mathfrak{S}_0$, there is a function $f : \mathsf{var}(A) \to \mathsf{var}(A')$ having the property of Definition 4.4; thus, the $\mathcal{V}$-valuation ϱ' for $\mathsf{var}(B)$ given by $\varrho'(x) = f(\varrho(x))$ is such that $B^{\varrho'} \subseteq A_0$, and it also holds that $\mathcal{T}' \subseteq \mathcal{T}_0$. Let σ be the $\mathcal{V}$-valuation chosen when R was applied to $\mathfrak{S}$.

Since $\mathfrak{S}_0$ is saturated, R is not applicable to it, and hence there must exist a $1 \leq i \leq n$ and a $\mathcal{V}$-valuation σ' for $\mathsf{var}(B) \cup \mathsf{var}(B_i)$ extending ϱ such that $B_i^{\sigma'} \subseteq A_0$. With the help of this $\mathcal{V}$-valuation, extend the function f by setting $f(\sigma(x)) = \sigma'(x)$ for every $x \in \mathsf{var}(B_i) \setminus \mathsf{var}(B)$. Then, if $(x_1, \ldots, x_m) : a \in B_i^\sigma$, it must be the case that $(f(x_1, \ldots, f(x_m)) : a \in A_0$ because $B_i^{\sigma'} \subseteq A_0$. But then, the S-state $\mathfrak{S}' = (A \cup B_i^\sigma, \mathcal{T}) \in \mathsf{R}(\mathfrak{S})$ is such that $\mathfrak{S}' \subseteq \mathfrak{S}_0$. ■

The pinpointing method can also be applied to variable tableaus. The notion of jalal as given in the previous section needs to be modified to allow the use of variables in the assertions, as has been done for variable tableaus. The following definition states such a generalization in a straightforward manner.

Definition 4.6 (Variable jalal) *Let $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ be a variable tableau for $\mathfrak{I}$,$\mathfrak{T}$. Label each element of $\mathfrak{T}$ with a unique propositional variable and let* lab *be the set of all those variables. Given a set of axioms $\mathcal{T} \subseteq \mathfrak{T}$, let $\hat{\mathcal{T}}$ denote the set containing all the elements of $\mathcal{T}$ with their respective variable. The* variable jalal judging S *is given by $S_\mathsf{j} = (\mathcal{A}^\mathsf{lab}, \cdot^{(S_\mathsf{j})_\mathfrak{I}}, \mathcal{R}_\mathsf{j}, \mathcal{C}_\mathsf{j})$, where*

- *for every $\Gamma \in \mathfrak{I} \times \mathscr{P}(\mathfrak{T})$, if $\Gamma^S = \{(A_1, \mathcal{T}), \ldots, (A_n, \mathcal{T})\}$, then $\Gamma^{S_\mathsf{j}} = \{(A_1^\top, \hat{\mathcal{T}}), \ldots, (A_n^\top, \hat{\mathcal{T}})\}$,*
- *for every rule $\mathsf{R} \in \mathcal{R}$ of the form*

$$(\{a_1, \ldots, a_k\}, \{t_1, \ldots, t_l\}) \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_m\}$$

construct the rule

$$(\{a_1^{\phi_1}, \ldots, a_k^{\phi_k}\}, \{t_1^{\varphi_1}, \ldots, t_l^{\varphi_l}\}) \xrightarrow{\mathsf{R}'} \{B_1^\psi, \ldots, B_m^\psi\}$$

where $\psi = \bigwedge_{i=1}^k \phi_i \wedge \bigwedge_{i=1}^l \varphi_i$,

- *$\mathcal{R}_\mathsf{j} = \{\mathsf{R}' \mid \mathsf{R} \in \mathcal{R}\}$, and $\mathcal{C}_\mathsf{j}$ is constructed as in Definition 2.10.*

A S_j-state is an element of $\mathscr{P}(\mathcal{A}^\mathsf{lab}) \times \mathscr{P}(\hat{\mathfrak{T}})$.

For this kind of jalal it is again necessary to define under which conditions will a rule be applicable to a S_j-state. This definition is once again a simple adaptation of the ones that have been given before, to fit into the variable framework, which brings no additional problems to the pinpointing method.

Definition 4.7 (Applicability) *A rule $(B, \mathcal{T}') \xrightarrow{\mathsf{R}} \{B_1^\psi, \ldots, B_n^\psi\}$ of a variable jalal S_j is* applicable *to a S_j-state $\mathfrak{S} = (A, \mathcal{T})$ if there is a $\mathcal{V}$-valuation ϱ for $\mathsf{var}(B)$ such that $B^\varrho \subseteq A, \mathcal{T}' \subseteq \mathcal{T}$, and for every $1 \leq i \leq n$ and every $\mathcal{V}$-valuation σ for $\mathsf{var}(B) \cup \mathsf{var}(B_i)$ extending ϱ, it holds that $\mathsf{ins}_\psi(B_i, A) \neq \emptyset$.*

The result of applying R to $\mathfrak{S}$ *is the set of S_j-states $\mathsf{R}(\mathfrak{S})$ given by $\mathsf{R}(\mathfrak{S}) = \{(A \uplus (B_j^\psi)\sigma, \mathcal{T}) \mid 1 \leq j \leq n\}$, where σ is a $\mathcal{V}$-valuation for $\mathsf{var}(B) \cup \bigcup_{j=1}^n \mathsf{var}(B_j)$ extending ϱ such that for every pair of elements $x, y \in \bigcup_{j=1}^n \mathsf{var}(B_j) \setminus \mathsf{var}(B)$ it holds that $\sigma(x) \in \mathcal{V} \setminus \mathsf{var}(A)$, and whenever $x \neq y$, then also $\sigma(x) \neq \sigma(y)$.*

The next part of this section will follow the same path as in the previous sections, showing how the jalals can be used to find the axiomatic causes of the presence of clashes, from which maximal subsets of axioms for which a property holds can be derived. Although the proofs follow basically the same pattern as the ones for deterministic jalals, the use of variables adds some difficulties in their development;

for this reason, the results will be proven in detail. In the following, some concepts may be mentioned which have not been formally defined for the variable framework; these are straightforward adaptations of the same concepts given in the non-deterministic case.

Lemma 4.8 *Let S be a variable tableau, $\mathfrak{S}_0$ a S_{j}-state, R' a jalal rule with $\mathsf{R}'(\mathfrak{S}_0) = \{\mathfrak{S}_1, \ldots, \mathfrak{S}_n\}$, and ω a valuation of the propositional variables in* lab. *Then, either $\omega(\mathfrak{S}_i) = \omega(\mathfrak{S}_0)$ for all $1 \leq i \leq n$, or $\mathsf{R}(\omega(\mathfrak{S}_0)) = \{\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)\}$, where R is the tableau rule from which R' is constructed, if the same $\mathcal{V}$-valuation for the variables appearing in the rule is used.*

Proof. Let ψ be the conjunction of all the labels appearing in the left-hand-side of R'. After the rule is applied, some new elements are added to the assertion set, labeled with the formula ψ, and some, that were already present in the assertion set of $\mathfrak{S}_0$, will have their labels changed to be disjointed with ψ, to create each of the S_{j}-states in $\mathsf{R}'(\mathfrak{S}_0)$.

If ψ evaluates to false under ω, then none of the new elements will be added to any of the $\omega(\mathfrak{S}_0)$ to produce any $\omega(\mathfrak{S}_i)$, since their label evaluates to false under ω; and the labels of all those elements that were already present will evaluate to the same truth value as did before being conjuncted with ψ under ω. Hence, the application of the rule does not add or remove any element from $\omega(\mathfrak{S}_0)$. So, $\omega(\mathfrak{S}_i) = \omega(\mathfrak{S}_0)$.

If, on the contrary, ψ evaluates to true under ω, then all the new elements will be added to $\omega(\mathfrak{S}_0)$ when producing each $\omega(\mathfrak{S}_i)$ as their label evaluates to true under ω. Furthermore, all the elements which were already present in the assertion set, will now have their label disjointed with ψ and hence, this label will evaluate to true under ω; thus, all the elements in the sets on the righ-hand-side of R will be added to $\omega(\mathfrak{S}_0)$ to form the $\omega(\mathfrak{S}_i)$'s. Nonetheless, since the application of the rule allows for the selection on any $\mathcal{V}$-valuation for the new elements to be added, if a different valuation is used, the elements will not be the same, since they would contain distinct assertion variables. If the same $\mathcal{V}$-valuation is used, then the exact same sets are obtained, which is what was to be proven. ∎

Notice that the restriction of using the same valuations when applying the rules is only necessary to obtain the exact same set of S-states. Nonetheless, this lemma could be relaxed to not need the use of the same valuations, if one wants only to obtain S-states that are equal. The proof given for Lemma 4.8 proofs also this claim.

Lemma 4.9 *Let S be a variable tableau, $\mathfrak{S}$ a saturated S_{j}-state, and ω a valuation of propositional variables. Then $\omega(\mathfrak{S})$ is a saturated S-state.*

Proof. Let $\mathfrak{S} = (A, \mathcal{T})$, $\mathsf{R} : (B, \mathcal{T}') \xrightarrow{\mathsf{R}} \{B_1, \dots, B_n\}$ be a tableau rule of S with jalal version R', ψ the conjunction of the labels appearing in the left-hand-side of R', and ϱ a valuation of $\mathsf{var}(B)$ such that $B^{\varrho} \subseteq \omega(A)$ and $\mathcal{T}' \subseteq \omega(\mathcal{T})$. Since all the elements appearing in the left-hand-side of the rule are present in $\omega(\mathfrak{S})$, all their labels evaluate to true under ω, and hence also does ψ.

Since $\mathfrak{S}$ is saturated, then R' must not be applicable to it. This means that there must be a $1 \leq j \leq n$ and a $\mathcal{V}$-valuation σ for $\mathsf{var}(B) \cup \mathsf{var}(B_j)$ extending ϱ with the property that for every $b \in B_j^{\sigma}$, there is be a ϕ such that $b^{\phi} \in A$ and $\psi \models \phi$. As ψ evaluates to true under ω, so does ϕ, and thus $b \in \omega(\mathfrak{S})$. Hence, R is not applicable to $\omega(\mathfrak{S})$. ∎

These two lemmas will help in the proof of the following proposition. The proposition, along with its proof, follows the lines of Proposition 3.6, without big modifications; it is nonetheless stated in full detail to show clearly the role of the assertion variables in the result.

Proposition 4.10 *Let S be a sound and complete variable tableau for a property $\mathcal{P}$, and ψ the clash formula associated to an input $\Gamma = (\mathcal{I}, \mathcal{T})$. Let $\Theta \subseteq \mathcal{T}$ and ω be the valuation mapping the propositional variables corresponding to elements of Θ to* true *and the rest to* false. *Then $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff ψ evaluates to* true *under ω.*

Proof. Let $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ be all the saturated S_{j}-states for Γ. Since $\{\omega(\mathfrak{S}) \mid \mathfrak{S} \in \Gamma^{S_{\mathsf{j}}}\} = (\mathcal{I}, \Theta)^S$, by means of Lemmas 4.8 and 4.9, every $\omega(\mathfrak{S}_i)$ is a saturated S-state for $(\mathfrak{I}, \Theta)$. It will now be shown that these are all the saturated S-states for that input, up to assertional variable renaming.

Let $\mathfrak{S}$ be a saturated S-state for $(\mathcal{I}, \Theta)$; then, there must be a sequence $\mathfrak{Q}_0, \dots, \mathfrak{Q}_m$ of S-states such that $\mathfrak{Q}_0 \in (\mathcal{I}, \Theta)^S$, $\mathfrak{Q}_m = \mathfrak{S}$, and for every $0 \leq i < m$ there exists a rule R_i of S such that $\mathfrak{Q}_{i+1} \in \mathsf{R}_i(\mathfrak{Q}_i)$. Using Lemma 4.8, one can deduce that the S_{j}-state $\mathfrak{S}'$ obtained by applying the corresponding jalal rules R_i' with the same $\mathcal{V}$-valuation on the tableau variables, and selecting the corresponding element in the set obtained after that application is such that $\omega(\mathfrak{S}') = \mathfrak{S}$. Further application of the same lemma yields the existence of a saturated S_{j}-state $\mathfrak{S}''$ such that $\omega(\mathfrak{S}'') = \mathfrak{S}$. If instead other $\mathcal{V}$-valuations were chosen, one would obtain the same S_{j}-states, but using different assertion variables. Hence, the S-states $\omega(\mathfrak{S}_1), \dots, \omega(\mathfrak{S}_n)$ are all the saturated S-states for $(\mathcal{I}, \Theta)$, up to assertional variable renaming.

All the previous implies that $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff every $\omega(\mathfrak{S}_i)$ contains a clash. A particular clash C is present in $\omega(\mathfrak{S}_i)$ iff for every element c^ϕ in C, it holds that $c^\phi \in \mathfrak{S}_i$ and ϕ evaluates to true under ω. Let now $\psi_{i,1}, \ldots, \psi_{i,k_i}$ be the formulas expressing all the clashes in $\mathfrak{S}_i$. It then holds that $\omega(\mathfrak{S}_i)$ contains a clash iff $\bigvee_{j=1}^{k_i} \psi_{i,j}$ evaluates to true under ω. Thus, every S-state $\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)$ contains a clash iff the clash formula evaluates to true under ω. ∎

As it was the case in the two previous sections, termination of the decision procedure transfers to its jalal under some finiteness assumptions. In other words, if a variable tableau is terminating, every rule is finite and the initial function is finite, then the variable jalal judging it is also terminating.

In the rest of this section, the use of variables will be generalized to allow the fact that axioms are, in a sense, not global facts, but rather facts over specific elements in the domain. To do that, these axioms will be allowed to include the use of assertional variables in them.

4.2 Axioms with variables

Up to now, the variables can appear only in the assertion sets of a state. As it was said before, this approach can be further generalized to allow axioms which include variables themselves. With this generalization it becomes then possible to *translate* directly an axiom into the assertion set. Axioms will only be able to use assertion variables since, if tableau variables were used in them, the $\mathcal{V}$-valuation approach would lead to a behaviour equivalent to that of unlabeled axioms, and in that case they will be of no use, and would only make the notation more elaborate.

The inclusion of assertion variables does not change the definition of any of the elements in the variable tableau, except for the rule set, in which rules become of the form

$$(A, \mathcal{T}) \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$$

where $A \subseteq \mathcal{A}(\mathcal{W})$ and for every $1 \leq i \leq n$, $B_i \subseteq \mathcal{A}(\mathcal{V}) \cup \mathcal{A}(\mathcal{W})$ is such that $B_i \setminus \mathcal{A}(\mathcal{W}) \subseteq \mathcal{T}$.

Notice that the whole treatment given up to now on variable tableaus, and by extension their jalals, relies exclusively on $\mathcal{V}$-valuations, which are made over the tableau variables. Since, as was said before, axioms include only assertion variables, this generalization does not affect any of the definitions or results shown so far. Hence, the same approach can be used to pinpoint the causes for a property not

$$
\begin{aligned}
\mathsf{R}\sqcap &: (\{y_0 : D_1 \sqcap D_2\}, \emptyset) \xrightarrow{\mathsf{R}\sqcap} \{\{y_0 : D_1, y_0 : D_2\}\} \\
\mathsf{R}\sqcup &: (\{y_0 : D_1 \sqcup D_2\}, \emptyset) \xrightarrow{\mathsf{R}\sqcup} \{\{y_0 : D_1\}, \{y_0 : D_2\}\} \\
\mathsf{R}\exists &: (\{y_0 : \exists r.D\}, \emptyset) \xrightarrow{\mathsf{R}\exists} \{\{y_1 : D, (y_0, y_1) : r\}\} \\
\mathsf{R}\forall &: (\{y_0 : \forall r.D, (y_0, y_1) : r\}, \emptyset) \xrightarrow{\mathsf{R}\forall} \{\{y_1 : D\}\} \\
\mathsf{Ra} &: (\emptyset, \{x : D\}) \xrightarrow{\mathsf{Ra}} \{\{x : D\}\}
\end{aligned}
$$

Figure 3: Rules for a tableau checking instance problem w.r.t ABoxes

to hold in a given axiomatized input, and then find maximal subsets of axioms for which the property holds.

The following example shows how this kind of variable tableaus can be used. The axioms in it are assertions containing variables, which are directly translated into the assertion set to verify that the conditions they force in the assertional variables they hold are enough to make certain variable name be an instance of an $\mathcal{ALC}$-concept term.

Example 4.11 *Let $\mathcal{V} = \{x_i \mid i > 0\}$ and $\mathcal{W} = \{y_i \mid i > 0\}$. A variable tableau for checking if an individual is an instance of an $\mathcal{ALC}$-concept term with respect to an ABox is given by $S_{\mathsf{ins}} = (\mathcal{A}, \cdot^{S_{\mathsf{ins}}}, \mathcal{R}, \mathcal{C})$ where $\mathcal{A}^{(1)}$ is the set of all $\mathcal{ALC}$-concept terms; for an axiomatized input $\Gamma = (a : C, \mathcal{T})$ with a an assertion variable, C an $\mathcal{ALC}$-concept term and $\mathcal{T}$ an ABox, $\Gamma^S = (\{a : \mathsf{nnf}(\neg C)\}, \mathcal{T})$; $\mathcal{R}$ contains the rules appearing in Figure 3; and $\mathcal{C}$ contains all the sets consisting of a concept name and its negation for any fixed assertion variable $\mathcal{C} = \{\{x : A, x : \neg A\} \mid A \text{ is a concept name}\}$.*

Intuitively, the tableau simply expands all the concepts that any individual has to satisfy given the axiomatized input, and in particular that a must satisfy C. If that expansion leads to a contradiction, stating that some element must satisfy a concept name and its negation, then it cannot be that all the rules stated can be satisfied, and so S_{ins} rejects the axiomatized input. This method is a restriction of the tableau for checking satisfiability of a concept term, see Example 4.3, but where the assertion variable given by the function $\cdot^{S_{\mathsf{ins}}}$ is fixed.

In all the tableaus presented up to now, it is only possible to express which elements must be present in a state in order to have a rule applicable to it; according to the applicability conditions defined so far, if all those elements appear in the state, and the rule would add new information to it when constructing each of the successor states, then the rule is applicable. In other words, once that all the elements in the left-hand-side of a rule are present in a state, the only possible way this rule is not applicable to the state is that all the elements in at least one of the sets in the right-hand-side are also present in its assertion set; thus, only if new information will be added when creating each of the new S-states, it will be possible to apply a rule.

For some applications, this approach is not adequate, since it may be necessary to state that, if certain element is already present in a state, then the rule should not be applicable, but if it is not present, even when the rule is applicable, such element should not be added, this is called a *negative applicability condition.* The next section will further generalize the notion of tableau in order to handle with such applicability conditions.

5 Negative Applicability Conditions

In this section, the tableaus presented previously in this report will be generalized in such a way that it is possible to express conditions for which a rule is *not* applicable, even when it satisfies the applicability conditions stated so far. It will turn out that the pinpoint method as presented in this report cannot be applied to every tableau for which these negative applicability conditions are allowed, as will be shown by means of examples. A subclass of tableaus for which the method is still applicable will be described, though.

5.1 Blocking tableaus

A blocking tableau will be one in which rules are allowed to contain negative applicability conditions, stating that, even if all the elements of the right hand side of the rule are present, and some of the assertion that would be added by its application are not there, the rule could still be not applicable by other means. These tableaus are now defined formally.

Definition 5.1 (Blocking tableau) *Let $\mathcal{V}$ and $\mathcal{W}$ be two sets of assertion and tableau rules, respectively, $\mathfrak{I}$ a set of inputs and $\mathfrak{T}$ a set of axioms. A* blocking tableau for $\mathfrak{I}$,$\mathfrak{T}$ *is a tuple $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$, where $\mathcal{A}$, $\cdot^{S_\mathfrak{I}}$ and $\mathcal{C}$ are as in Definition 4.1, but $\mathcal{R}$ may contain, additionally to rules of the form given in Definition 4.1, also* blocking rules *of the form*

$$(A, \mathcal{T}) \triangledown B \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$$

where $A, B, B_i \in \mathcal{A}(\mathcal{W})$ and $\mathcal{T} \subseteq \mathfrak{T}$.

In this case, B is called the blocking set *of* R.

This kind of tableau is almost identical to variable tableau, and all the concepts like S-state, saturated, or soundness, are defined in a similar fashion as the previous section. Nonetheless, for the newly defined blocking rules, new applicability conditions must be stated, in order to make clear how the blocking sets influence the applicability of those rules.

Definition 5.2 (Applicability) *Given a blocking tableau S, a S-state $\mathfrak{S} = (A_0, \mathcal{T}_0)$, and a blocking rule $\mathsf{R} : (A, \mathcal{T}) \triangledown B \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$, R is* applicable *to $\mathfrak{S}$ if all the following conditions hold:*

1. *there is a $\mathcal{V}$-valuation ϱ for $\mathsf{var}(A)$ such that $A^{\varrho} \subseteq A_0$,*
2. *$\mathcal{T} \subseteq \mathcal{T}_0$,*
3. *for every $\mathcal{V}$-valuation σ for $\mathsf{var}(A) \cup \mathsf{var}(B)$ extending ϱ, $B^{\sigma} \not\subseteq A_0$*
4. *for every $1 \leq i \leq n$ and every $\mathcal{V}$-valuation σ for $\mathsf{var}(A) \cup \mathsf{var}(B_i)$ extending ϱ, $B_i^{\sigma} \not\subseteq A_0$*

R is blocked *by $\mathfrak{S}$ if Conditions 1, 2 and 4 above hold, but Condition 3 does not hold.*

The result *of applying R to $\mathfrak{S}$ is the set of S-states $\mathsf{R}(\mathfrak{S})$ given by $\mathsf{R}(\mathfrak{S}) = \{(A_0 \cup B_1^{\sigma}, \mathcal{T}), \ldots, (A_0 \cup B_n^{\sigma}, \mathcal{T})\}$, where σ is a $\mathcal{V}$-valuation for $\mathsf{var}(A) \cup \bigcup_{i=1}^{n} \mathsf{var}(B_i)$ extending ϱ such that for every pair of variables $x, y \in \bigcup_{i=1}^{n} \mathsf{var}(B_i) \setminus \mathsf{var}(A)$ it is the case that $\sigma(x) \in \mathcal{V} \setminus \mathsf{var}(A_0)$, and if $x \neq y$ then $\sigma(x) \neq \sigma(y)$.*

The following example shows the use of blocking rules, and their applicability conditions.

Example 5.3 *Suppose that one wants to impose a condition over the rule $\mathsf{R}\exists$ of Example 4.3 in such a way that, if there is already a node in the assertion set that satisfies all the formulas that the newly generated element would have after the application, the rule is not applicable. Such a condition can be stated replacing the rule $\mathsf{R}\exists$ by the new blocking rule*

$$\begin{array}{rcl} (A, \emptyset) & \triangledown & \{y_1 : D, y_1 : E_1, \ldots, y_1 : E_n\} \\ & \xrightarrow{\mathsf{R}\exists} & \{\{y_2 : D, (y_0, y_2) : r\}\} \end{array}$$

where $A = \{y_0 : \exists r.D, y_0 : \forall r.E_1, \ldots, y_0 : \forall r.E_n\}$.

It is important to notice that, by the way the result of the application of rules was defined, using y_1 as tableau variable instead of y_2 on the right-hand-side of the rule would make no difference when applying this rule to any S_{Sat}-state.

$$\begin{array}{c} \{A, D\} \\ | \\ \{A, C, D\} \end{array}$$

(a) Choosing rule R_1 first leads to a saturated and clash-free S-state

$$\begin{array}{c} \{A, D\} \\ | \\ \{A, B, D\} \\ | \\ \{A, B, C, D\} \end{array}$$

(b) Choosing rule R_2 first, followed by R_1 leads to a saturated S-state with a clash

Figure 4: The choice of different rule application order leads to different results in a blocking tableau

Unfortunately, within the framework of blocking tableaus, the non-determinism obtained by the rule application order is, contrary to all the previous cases, not a *don't care* non-determinism, but rather a *don't know* non-determinism. In other words, it might be the case that applying the rules in one order leads to a saturated and clash-free state, while the selection of a distinct application order leads to only saturated states containing clashes. The following example shows an instance of this.

Example 5.4 *Let* $S = (\{A, B, C, D\}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ *be a blocking tableau where* $\cdot^S$ *maps an axiomatized input* $\Gamma = (\mathcal{I}, \emptyset)$ *with* $\mathcal{I} \subseteq \{A, B, C, D\}$ *to* $\Gamma^S = \{(\mathcal{I}, \emptyset)\}$, $\mathcal{C} = \{\{B, D\}\}$, *and* $\mathcal{R}$ *contains the rules:*

$$\begin{aligned} \mathsf{R}_1 &: (\{D\}, \emptyset) \xrightarrow{\mathsf{R}_1} \{\{C\}\} \\ \mathsf{R}_2 &: (\{A\}, \emptyset) \triangledown \{C\} \xrightarrow{\mathsf{R}_2} \{\{B\}\} \end{aligned}$$

Given the axiomatized input $\Gamma = (\{A, D\}, \emptyset)$, *the initial function yields* $\Gamma^S = \{(\{A, D\}, \emptyset)\}$. *For the only* S*-state in* Γ^S, *both rules* R_1 *and* R_2 *are applicable. If the rule* R_1 *is applied first, one obtains a saturated and clash-free* S*-state, as shown in Figure 4(a).*

If, on the other hand, the rule R_2 *is selected to be applied first, one obtains again only one* S*-state which is not saturated since* R_1 *is still applicable to it. After applying* R_1 *to this new state, one unique saturated* S*-state is obtained, but this one contains a clash. This is shown in Figure 4(b).*

The reason why in this example the different ordering selections lead to distinct results relies on the fact that, when the rule R_1 is applied, it blocks the applicability of R_2. This happens because R_1 adds the elements that restrict the applicability of R_2 by means of the negative applicability condition. Once that the applicability of R_2 has been blocked, it is impossible to add the element B, which is added by this rule, to the assertion set. This element is necessary to obtain a clash. Thus, the blocking condition disallows the possibility of getting a clash by following that path.

The irrelevance of the application order of rules is usually a desirable property, especially when one looks for orderings by which the solution can be found applying the minimal amount of rules. If finding the solution depends on the ordering chosen, then this kind of optimization cannot be done. Nonetheless, for the pinpointing technique presented in this report, this irrelevance of the order chosen is not fundamental; and hence, it is feasible to construct jalals that judge blocking tableaus. These jalals will be defined next. As it was said before, the pinpointing method does not work for every instance of blocking tableaus. Although blocking jalals will be defined in general, the correctness of their use depends on the original blocking tableau belonging to a class for which the pinpointing method works in an adequate manner.

5.2 Blocking jalals

The definition of blocking jalals is a simple adaptation of that for variable jalals to allow the use of blocking rules.

Definition 5.5 (Blocking jalal) *Let $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ be a blocking tableau for $\mathfrak{I}$,$\mathfrak{T}$. Label each element of $\mathfrak{T}$ with a unique propositional variable; let* lab *be the set of all those variables and let $\hat{\mathcal{T}}$ denote the set having all the elements of $\mathcal{T}$ with their respective labels, for any $\mathcal{T} \subseteq \mathfrak{T}$. The* blocking jalal judging S *is given by $S_\mathsf{j} = (\mathcal{A}^{\mathsf{lab}}, \cdot^{(S_\mathsf{j})_\mathfrak{I}}, \mathcal{R}_\mathsf{j}, \mathcal{C}_\mathsf{j})$ where all the elements are given as in Definition 4.6, and for every blocking rule* R *of the form*

$$(\{a_1, \ldots, a_k\}, \{t_1, \ldots, t_l\}) \nabla B \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_m\}$$

construct the rule R'

$$(\{a_1^{\phi_1}, \ldots, a_k^{\phi_k}\}, \{t_1^{\varphi_1}, \ldots, t_l^{\varphi_l}\}) \nabla \hat{B} \xrightarrow{\mathsf{R}'} \{B_1^{\psi}, \ldots, B_m^{\psi}\}$$

where $\psi = \bigwedge_{j=1}^{k} \phi_j \wedge \bigwedge_{j=1}^{l} \varphi_j$, and $\hat{B}$ contains the elements of B with parametrized labels.

For this kind of jalals, the applicability conditions need to take into consideration also the labels in the blocking set of the rules. The fact that all the elements of the blocking set are present in the assertion set is not enough for the rule not to be applicable when the main task is to find the axioms that produce clashes. The reason for this is that, if some axioms are removed, it might happen that the elements in the right-hand-side of the rule can be produced by application of other rules, but there is one element in the blocking set for which one of the removed axioms was necessary. In that case the rule would be applicable, and its application would add new elements that were not considered in the clash-formula before; from these new elements, a new clash could be formed.

Example 5.6 *Let* $\mathfrak{I} = \emptyset, \mathfrak{T} = \{A_1, A_2\}$ *and* $S_{\mathsf{c}} = (\mathcal{A}, \cdot^{S_{\mathsf{c}\mathfrak{I}}}, \mathcal{R}, \mathcal{C})$ *be a blocking tableau where* $\mathcal{A} = \{B, C\}$, $\cdot^{S_{\mathsf{c}\mathfrak{I}}}$ *is the identity function,* $\mathcal{C} = \{\{B\}, \{C\}\}$, *and* $\mathcal{R}$ *contains only the following two rules:*

$$\begin{aligned} \mathsf{R}_1 &: (\emptyset, \{A_1\})\triangledown\{B\} \xrightarrow{\mathsf{R}_1} \{\{C\}\} \\ \mathsf{R}_2 &: (\emptyset, \{A_2\})\triangledown\{C\} \xrightarrow{\mathsf{R}_2} \{\{B\}\} \end{aligned}$$

Let the axiomatized input be $\Gamma = (\emptyset, \{A_1, A_2\})$. *Then, if an analogous of the applicability conditions defined previouly were used for the blocking jalal* S_{cj} *judging* S_{c}, *both rules* R'_1 *and* R'_2 *would be applicable to* $\Gamma^{S_{\mathsf{cj}}} = (\emptyset, \{A_1^p, A_2^q\})$.

Application of R'_1 *would lead to the* S_{cj}*-state* $(\{C^p\}, \{A_1^p, A_2^q\})$, *while* R'_2 *would produce* $(\{B^q\}, \{A_1^p, A_2^q\})$. *Both of these* S_{cj}*-states for* Γ *are saturated, both contain a clash, and they are all the possible saturated* S_{cj}*-states for this input. Then, the clash formula for this input would be* $p \wedge q$. *This would mean that, according to the methods described in the previous sections, any subset of axioms containing just one of them, that is* $\{A_1\}$ *and* $\{A_2\}$, *is such that the tableau would accept the input with it.*

A brief analysis of that input will show that none of those sets of axioms is correct, since the only input that will be accepted is $(\emptyset, \emptyset)$.

In order to avoid this problem, a rule will also be applicable to a state if, although all the elements of the blocking set can be found in the assertion set of the state, their labels are not modelled by the conjunction of the labels of the elements that triggered the rule. This is stated formally in the following definition.

Definition 5.7 (Applicability) *A rule* $(A, \mathcal{T})\triangledown B \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$ *of a blocking jalal* S_{j} *is* applicable *to the* S_{j}*-state* $\mathfrak{S} = (A_0, \mathcal{T}_0)$ *if there is a* $\mathcal{V}$*-valuation* ϱ *for* $\mathsf{var}(A)$ *such that* $A^\varrho \subseteq A_0$, $\mathcal{T} \subseteq \mathcal{T}_0$ *and the following conditions hold:*

$$
\begin{aligned}
\mathsf{R}_1 &: (\emptyset, \{A\}) \xrightarrow{\mathsf{R}_1} \{\{A\}\}\\
\mathsf{R}_2 &: (\emptyset, \{B\}) \xrightarrow{\mathsf{R}_2} \{\{B\}\}\\
\mathsf{R}_3 &: (\{B\}, \emptyset) \xrightarrow{\mathsf{R}_3} \{\{A\}\}\\
\mathsf{R}_4 &: (\{A\}, \emptyset)\nabla\{B\} \xrightarrow{\mathsf{R}_4} \{\{C\}\}
\end{aligned}
$$

Figure 5: Rules of the blocking tableau S_b in Example 5.8

- *for every $\mathcal{V}$ valuation σ for $\mathsf{var}(A) \cup \mathsf{var}(B)$ extending ϱ, there is a labeled element $b^\phi \in B^\sigma$ such that either $b \notin \mathsf{unl}(A_0)$ or $\psi \not\models \phi$;*
- *for every $1 \leq i \leq n$ and every $\mathcal{V}$-valuation σ for $\mathsf{var}(A) \cup \mathsf{var}(B_i)$ extending ϱ, it holds that $B_i^\sigma \not\subseteq A_0$.*

The result of applying R to $\mathfrak{S}$ is the set of S_j-states $\mathsf{R}(\mathfrak{S})$ given by $\mathsf{R}(\mathfrak{S}) = \{(A_0 \uplus B_1^\sigma, \mathcal{T}), \ldots, (A_0 \uplus B_n^\sigma, \mathcal{T})\}$, where σ is a $\mathcal{V}$-valuation for $\mathsf{var}(A) \cup \bigcup_{i=1}^n \mathsf{var}(B_i)$ extending ϱ such thatr for every pair of variables $x, y \in \bigcup_{i=1}^n \mathsf{var}(B_i) \setminus \mathsf{var}(A)$ it is the case that $\sigma(x) \in \mathcal{V} \setminus \mathsf{var}(A)$ and if $x \neq y$ then $\sigma(x) \neq \sigma(y)$.

Unfortunately, this definition of applicability brings forth another problem to blocking jalals. Even when a blocking tableau is sound and complete for some property, the jalal judging it may no be, as shown in the following example. For this reason, the pinpointing method is not applicable to every blocking tableau, but just to a restricted class of them.

Example 5.8 *Let $\mathfrak{T} = \{A, B, C\}, \mathfrak{I} = \mathscr{P}(\mathfrak{T})$ and $S_\mathsf{b} = (\mathcal{A}, \cdot^{S_\mathsf{b}\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ be the blocking tableau for $\mathfrak{I}$,$\mathfrak{T}$ given by $\mathcal{A} = \mathfrak{T}$, $\cdot^{S_\mathsf{b}\mathfrak{I}}$ being the function that maps every input to the set containing only the same input; the set of clashes $\mathcal{C} = \{\{C\}\}$ and $\mathcal{R}$ contains the rules shown in Figure 5.*

Let now the axiomatized input be $\Gamma = (\emptyset, \{A, B\})$. It then holds that $\Gamma^{S_\mathsf{b}} = (\emptyset, \{A, B\})$. At this point, it is possible to apply rule R_2 followed by R_3 to obtain the S_b-state $(\{A, B\}, \{A, B\})$. This state is saturated since, for the first three rules, the elements that would be added are already in the assertion set, and the fourth rule is blocked by the existence of B. As there is a saturated and clash-free S_b-state for Γ, the tableau will accept this input.

If one tries now to apply the blocking jalal S_bj judging S_b to this same axiomatized input, the result will be the opposite. Figure 6 shows all the possible S_bj-states for this input, along with the rules required to reach them. Looking at that figure, it is easy to see that there is only

$$
\begin{tikzcd}
 & & \emptyset \arrow[dl, "\mathsf{R}_1"'] \arrow[dr, "\mathsf{R}_2"] & & \\
 & \{A^p\} \arrow[dl, "\mathsf{R}_4"'] \arrow[dr, "\mathsf{R}_2"] & & \{B^q\} \arrow[dl, "\mathsf{R}_1"'] \arrow[dr, "\mathsf{R}_3"] & \\
\{A^p, C^p\} \arrow[d, "\mathsf{R}_2"'] & & \{A^p, B^q\} \arrow[dll, "\mathsf{R}_4"'] \arrow[drr, "\mathsf{R}_3"] & & \{A^q, B^q\} \arrow[d, "\mathsf{R}_1"] \\
\{A^p, B^q, C^p\} \arrow[d, "\mathsf{R}_3"'] & & & & \{A^{p\vee q}, B^q\} \arrow[d, "\mathsf{R}_4"] \\
\{A^{p\vee q}, B^q, C^p\} \arrow[rrrr, "\mathsf{R}_4"] & & & & \{A^{p\vee q}, B^q, C^{p\vee q}\}
\end{tikzcd}
$$

Figure 6: All S_{bj}-states for the input $(\emptyset, \{A, B\})$ with the rule applications needed to reach them.

one saturated S_{bj}-state for the given input, and it contains a clash. Hence S_{bj} would reject this input.

Furthermore, the clash formula for this input would be $p \vee q$. This means that, according to this blocking jalal, the only way to get rid of the clash is to remove all the axioms that were used and have an empty input. Clearly, this cannot be true, not only because the original input is accepted by the tableau, but also because removing only the axiom A, labeled with p, would lead to clash-free states, even in this jalal.

As the previous example shows, blocking jalals are not useful for every instance of a blocking tableau. It is, nonetheless, possible to define a class of these tableaus for which the pinpointing procedure, as described in this report, still works. The rest of this section will deal with stating that class, and proving that a method analogous to the ones presented in the previous sections, work for every tableau belonging to that class.

In order to find such a class of blocking tableaus, one need to examine the causes of the problems that do not allow the pinpointing method to be used in a correct manner. The problem presented in Example 5.8 is that application of the jalal rules adds a clash where the tableau rule application was blocked. If one can ensure that this will not happen, then the pinpointing method will work adequately. In this report two different conditions that are sufficient to avoid this problem are given, giving rise to two classes of blocking tableaus, which are not disjoint. They will be called *safe* blocking tableaus, and *input-deniable* tableaus. In the rest of this section both classes will be defined, and it will be shown that one can track the causes of inconsistency in every tableau belonging to them.

Before defining formally these subclasses and going into the details of why the pinpointing methods works on them, it will be shown that the analogous of Lemmas 4.8 and 4.9 hold in every blocking tableau. This might be surprising at first sight, since those lemmas form the base over which the proof of correctness of the pinpointing method relies, for the framework presented in the previous section. Analyzing the reason why the same proof does not work in this framework will give an insight on the restrictions that need to be enforced for the method to work, and hence help motivate the classes that will be presented afterwards.

In the following lemma, a blocking rule can be applied to a state, even if this rule is blocked by that state. The definition of rule application to a state blocking it is the same as the normal rule application.

Lemma 5.9 *Let S be a blocking tableau, $\mathfrak{S}_0$ a S_{j}-state, R' a jalal rule with $\mathsf{R}'(\mathfrak{S}_0) = \{\mathfrak{S}_1, \ldots, \mathfrak{S}_n\}$, and ω a vlauation of the propositional variables in* lab. *Then, either $\omega(\mathfrak{S}_i) = \omega(\mathfrak{S}_0)$ for all $1 \leq i \leq n$, or $\mathsf{R}(\omega(\mathfrak{S}_0)) = \{\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)\}$, where R is the tableau rule from which R' is constructed, if the same $\mathcal{V}$-valuation for the new variables is used.*

Proof. Let $\mathsf{R} = (A, \mathcal{T})\nabla B \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$, and ψ be the conjunction of all the labels appearing in A for R'. The application of the rule creates n S_{j}-states that contain all the assertions in $\mathfrak{S}_0$ where possibly some have their labels changed to be disjointed with ψ, and also possible some new assertions labeled with the same formula ψ.

If ψ evaluates to false under ω, then the elements that had their labels modified by a disjunction with ψ will not modify its truth value when evaluated under ω, and all the new elements added will have their labels evaluated to false under the same valuation. Hence, the inclusion of elements in $\mathfrak{S}_0$ to form the states $\mathfrak{S}_i$ does not modify its projection; i.e. $\omega(\mathfrak{S}_0) = \omega(\mathfrak{S}_i)$ for all i.

In other case, that is if ψ evaluates to true under ω, then all the elements whose labels were disjuncted with ψ will be such that will be present now in the ω-projection of the state to which they belong, since the new label will evaluate to true under that valuation. The same happens to all new elements, whose label is exactly ψ. Hence, $\mathsf{R}(\omega(\mathfrak{S}_0)) = \{\omega(\mathfrak{S}_1), \ldots, \omega(\mathfrak{S}_n)\}$. ■

Notice that, in the last part of this proof, it may well be the case that R is blocked by $\omega(\mathfrak{S}_0)$, if all the elements of the blocking set appear in $\mathfrak{S}_0$ and their labels are mapped to true under ω, even if R' is not blocked by $\mathfrak{S}_0$. As it was said before, even when the rule is blocked, $\mathsf{R}(\omega(\mathfrak{S}_0))$ is defined in the same way as when R is applicable to that S-state. Nonetheless, the resulting S-states obtained after such a rule application are not necessarily S-states for the input given, since there might be no way to obtain them by normal application of rules as defined for blocking tableaus. As it will be said further in this section, this is the main reason why the main proposition cannot be proved only by the application of these lemmas.

Lemma 5.10 *Let S be a blocking tableau, $\mathfrak{S}$ a saturated S_{j}-state and ω a valuation of propositional variables in* lab. *Then $\omega(\mathfrak{S})$ is a saturated S-state.*

Proof. Let $\mathsf{R} : (A, \mathcal{T})\nabla B \xrightarrow{\mathsf{R}} \{B_1, \ldots, B_n\}$ be a rule applicable to $\omega(\mathfrak{S})$. Then, for every element $a \in A$, there is an assertion $a^{\phi_a} \in \mathfrak{S}$, and ϕ_a evaluates to true under ω. Since the rule is applicable, there is an element $b \in B$ such that $b \notin \omega(\mathfrak{S})$, and for every $i \in \{1, \ldots, n\}$, there is a $b_i \in B_i$ with $b_i \notin \omega(\mathfrak{S})$. This entails that either there is no φ such that $b(b_i)$ is in $\mathfrak{S}$, or that there is one, but evaluates to false under ω. In any of both cases, the jalal rule R' obtained from R would be applicable to $\mathfrak{S}$, contradicting the fact that $\mathfrak{S}$ is saturated. Hence $\omega(\mathfrak{S})$ is saturated. ∎

In the previous sections, the correctness of the pinpointing method was shown with the help of analogous versions of Lemmas 5.9 and 5.10 by showing that, given all the saturated S_{j}-states for an input $(\mathcal{I}, \mathcal{T})$, the ω-projections of them were all the saturated S-states for $(\mathcal{I}, \Theta)$, where ω is the valuation mapping every propositional variable corresponding to elements in Θ to true, and the rest to false. In this framework, as it was previously said, the ω-projection of a S_{j}-state for $(\mathcal{I}, \mathcal{T})$ needs not be a S-state for $(\mathcal{I}, \Theta)$. That is the reason why the same proof cannot be used to show that blocking jalals work correctly. Of course, after Example 5.8, it was already known that such a general proof cannot be obtained.

As such a proof shows, in order to be able to use the pinpointing method in a blocking tableau, it is enough to ensure that the ω-projections of the saturated states obtained by the application of the jalal yield all the information required; that is, a way of knowing whether there is a saturated and clash-free S-state for the input with restricted set of axioms. The two classes that are defined in this report follow two different approaches for ensuring that condition.

5.2.1 Safe Tableaus

As it was said before, the problem of a jalal may not be sound and complete for a property, even when the tableau from which it was constructed was, arises from the fact that a rule which was blocked during the execution of the tableau may become applicable in the jalal. Such an *unblocked* may be the cause of finding only saturated states which contain clashes. In Exapmle 5.8, the only possible way to produce a clash is by an application of the blocking rule R_4. When the tableau is executed, once that the assertion B has been produced, there is no way to apply the rule R_4, and hence, no way to obtain a clash. In the jalal, nonetheless, even when the assertion B is present, the same rule may be applicable, adding the clash to the assertion set.

If one wants to avoid the problem of clashes being generated from rules that were originally blocked, the easiest idea is to ensure that when a rule is blocked on a state for which further application of rules leads to a saturated and clash-free state, then at least one of the states reached by the application of the blocked rule will also be such that a saturated and clash-free state is reachable from it. It is in fact sufficient to ensure that this condition holds for all the saturated and clash-free states, and not necessarily for every state from which a saturated and clash-free one can be reached. The tableaus that satisfy this condition will be called *safe*.

For the formal definition of safe tableaus, the notions of *reachable* and *accepting* states will be used. Informally, a reachable state is that which can be obtained from by application of rules (which can be blocked) given some input. An accepting state is a reachable one for which there is a way of applying rules that leads to a saturated and clash-free state.

Definition 5.11 (Reachable,accepting) *Let* $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \mathcal{C})$ *be a blocking tableau. A rule* $\mathsf{R} \in \mathcal{R}$ *is* active for a state $\mathfrak{S}$ *if* R *is either applicable to* $\mathfrak{S}$*, or blocked by* $\mathfrak{S}$*. The set* $\mathsf{Reach}(S)$ *of* reachable states of S *is the smallest set such that:*

- *for every input* Γ*,* $\Gamma^S \subseteq \mathsf{Reach}(S)$*;*
- *if* $\mathfrak{S} \in \mathsf{Reach}(S)$ *and* R *is active for* $\mathfrak{S}$*, then* $\mathsf{R}(\mathfrak{S}) \subseteq \mathsf{Reach}(S)$*.*

The set of accepting states for S*,* $\mathsf{Acc}(S) \subseteq \mathsf{Reach}(S)$ *is defined inductively by:*

- *if* $\mathfrak{S} \in \mathsf{Reach}(S)$ *is saturated and clash-free, then* $\mathfrak{S} \in \mathsf{Acc}(S)$*;*
- *if* $\mathfrak{S} \in \mathsf{Reach}(S)$ *and there is a rule* R *applicable to* $\mathfrak{S}$ *and a* $\mathfrak{Q} \in \mathsf{R}(\mathfrak{S})$ *such that* $\mathfrak{Q} \in \mathsf{Acc}(S)$*, then* $\mathfrak{S} \in \mathsf{Acc}(S)$*.*

It can be easily seen that the reachable states are those that can be obtained by application of jalals over any possible input, if the labels are discarded. Thus, for every valuation ω, the ω-projection of a S_{j}-state that was found by application of rules from an input is itself reachable.

It is now turn to define formally the concept of safe tableaus.

Definition 5.12 (Safe) *Let S be a blocking tableau. A blocking rule R of S is* safe *if for every $\mathfrak{S} \in \mathsf{Reach}(S)$, such that R is blocked by $\mathfrak{S}$, it holds that $\mathfrak{S} \in \mathsf{Acc}(S)$ iff there is a $\mathfrak{Q} \in \mathsf{R}(\mathfrak{S})$ such that $\mathfrak{Q} \in \mathsf{Acc}(S)$. S is* safe *if every rule of S is safe.*

The intuition behind this definition, as has been said before, is that a blocking tableau is safe if, whenever it is possible to reach a saturated and clash-free state, from a S-state where a rule is blocked, then even if the blocked rule was to be applied, another saturated and clash-free state can be found from there. In other words, in safe tableaus the blocking conditions can be used to ensure, for example, termination, but not to avoid getting a clash.

As discused before, since a rule that is blocked in the tableau may not be also blocked in the jalal, it might be the case that, given a valuation ω, $\omega(\mathfrak{S})$ is not a S-state for an input Γ, even if $\mathfrak{S}$ is a S_{j}-state for that same input. Thus, a proposition analogous to Proposition 4.10 must be proven using an approach different to that used in the previous section. The approach presented here will in fact make use of Lemmas 5.9 and 5.10, but the requirement of safeness will be necessary to ensure that the ω-projections of the saturated S_{j}-states for a given input contain enough information to know about the acceptance or rejection of the input obtained by restricting the set of axioms to be used.

Before stating the proposition that ensures that the pinpointing method works for safe tableaus, a definition of clash formula is needed. Since not only the rules, but also its application order include non-determinism in the search for a clash-free saturated state, the clash formula given in Definition 3.5 will be used for this framework, but allowing every saturated S-state that can be reached under every rule-application-order.

Proposition 5.13 *Let S be a safe blocking tableau which is sound and complete for a property $\mathcal{P}$, and ψ be the clash formula associated with an input $\Gamma = (\mathcal{I}, \mathcal{T})$. Let $\Theta \subseteq \mathcal{T}$ and ω be the valuation mapping the propositional variables corresponding to elements of Θ to* true *and the rest to* false*. Then $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff ψ evaluates to* true *under ω.*

Proof. Let $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ be all the saturated S_{j}-states for Γ. It will be shown first that there is a saturated and clash-free S-state for $(\mathcal{I}, \Theta)$ if and only if $\omega(\mathfrak{S}_i)$ is clash-free for some $1 \leq i \leq n$.

If there is a saturated and clash-free S-state $\mathfrak{Q}$ for $(\mathcal{I}, \Theta)$, then one can apply the exact same rules in the jalal over $(\mathcal{I}, \mathcal{T})$ to obtain a S_{j}-state $\mathfrak{Q}'$ such that $\omega(\mathfrak{Q}') = \mathfrak{Q}$. If $\mathfrak{Q}'$ is saturated, then the result holds. Otherwise, there are rules applicable to $\mathfrak{Q}'$. By Lemma 5.9, these rules are either not blocked by $\mathfrak{Q}$, in which case their application makes no changes in the ω-projection, or they are blocked by $\mathfrak{Q}$. In this second case, since the tableau is safe, and $\mathfrak{Q}$ is accepting, application of the blocked rule leads to another accepting state; hence, it is possible to further apply rules finding always another accepting state, until the ω-projection of these state is a saturated and clash-free S-state. This argument can be repeated until there are no further rules applicable to $\mathfrak{Q}'$, and then it is saturated and its projection is clash-free.

Conversely, if $\omega(\mathfrak{S}_i)$ is clash-free for some i, then there is a sequence of (possibly blocked) rules which can be applied from $(\mathcal{I}, \Theta)^S$ leading to $\omega(\mathfrak{S}_i)$, which is saturated (by Lemma 5.10) and clash-free. Thus, $\omega(\mathfrak{S}_i)$ is an acceptin S-state, and since S is safe, every S-state in the path from $(\mathcal{I}, \Theta)^S$ to $\omega(\mathfrak{S}_i)$ is also accepting. In particular this implies that there is an accepting S-state $\mathfrak{Q} \in (\mathcal{I}, \Theta)^S$. But then, there is a sequence of applicable rules, starting from $\mathfrak{Q}$ that leads to a saturated and clash-free S-state $\mathfrak{S}$, and since the initial state is in $(\mathcal{I}, \Theta)^S$, $\mathfrak{S}$ is a S-state for $(\mathcal{I}, \Theta)$.

Hence, there is a saturated and clash-free S-state for $(\mathcal{I}, \Theta)$ if and only if $\omega(\mathfrak{S}_i)$ is clash-free ofr some saturated S_{j}-state for Γ $\mathfrak{S}$. This implies that $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff every $\omega(\mathfrak{S}_i)$ contains a clash. A particular clash C is present in $\omega(\mathfrak{S}_i)$ iff for every element c^{ϕ} in C it holds that $c^{\phi} \in \mathfrak{S}_i$ and ϕ evaluates to true under ω. Let now $\psi_{i,1}, \ldots, \psi_{i,k_i}$ be the formlas expressing all the clashes in $\mathfrak{S}_i$. It holds then that $\omega(\mathfrak{S}_i)$ contains a clash iff $\bigvee_{j=1}^{k_i} \psi_{i,j}$ evaluates to true under ω. Thus, every S-state $\omega(\mathfrak{S}_i)$ contains a clash iff the clash formula evaluates to true under ω. ■

5.2.2 Input-Deniable Tableaus

The second class of blocking tableaus treated in this report for which the pinpointing method works receives the name of *input-deniable.* This class consists of all the tableaus for which the set of elements in the right-hand-side of the rules is disjoint with the elements in the blocking sets. The name is motivated by the fact that the blocking conditions cannot be created by a application of rules; this ensures that, if a rule is blocked, all the elements of the blocking set were obtained from the initial function $\cdot^{S_\daleth}$. Thus, when the jalal is applied to the same input, the elements of the blocking set would also be present, and they will be labeled with a tautology, so the same rule would also be blocked in the jalal. Although these conditions may seem too simple and restrictive, it turns out that there exist interesting applications for input-deniable tableaus, one of which will be presented in the next section.

Before giving the formal definition of input-deniable tableaus, some concepts will be presented which will be useful for a more clear exposition of this class of tableaus.

Definition 5.14 (Postcondition-,blocking-assertions) *Let S be a blocking tableau, $S = (\mathcal{A}, \cdot^S, \mathcal{R}, \mathcal{C})$, with $\mathcal{A} = \bigcup_{i \geq 0} \mathcal{A}^{(i)}$; assume w.l.o.g. that all the $\mathcal{A}^{(i)}$ are pairwise disjoint. Given a set of assertions with variables $A \in \mathcal{A}(\mathcal{V})$, the* flattened version *of A is given by $\flat(A) = \{a \mid (x_1, \ldots, x_n) : a \in A\}$. The sets Post_S and Block_S of* postcondition- *and* blocking-*assertions of S, respectively are defined as follows:*

$$\begin{aligned}
\mathsf{Post}_S &= \bigcup_{\mathfrak{S} \xrightarrow{\mathsf{R}} \{B_1,\ldots,B_k\} \in \mathcal{R}} \bigcup_{i=1}^{k} \flat(B_i) \cup \bigcup_{\mathfrak{S} \nabla B \xrightarrow{\mathsf{R}} \{B_1,\ldots,B_k\} \in \mathcal{R}} \bigcup_{i=1}^{k} \flat(B_i) \\
\mathsf{Block}_S &= \bigcup_{\mathfrak{S} \nabla B \xrightarrow{\mathsf{R}} \mathcal{B} \in \mathcal{R}} \flat(B)
\end{aligned}$$

Informally, the set of postcondition-assertions consists of all those assertions that appear in the right-hand-side of rules, without taking into account the tableau variables that are associated with them in the definition of the rule. Analogously, the blocking-assertions are all those which appear in the blocking conditions of any blocking rule in the tableau, regardless of the variables that could be associated to them in the definition of the rule. With this, the definition of input-deniable tableaus becomes very simple.

Definition 5.15 (Input-deniable) *A blocking tableau S is called* input-deniable *if $\mathsf{Post}_S \cap \mathsf{Block}_S = \emptyset$.*

As it was previously stated, the idea behind these tableaus is that, whenever a rule application is blocked in a S-state, then the same rule is also blocked by its corresponding S_j-state, avoiding this way the problems of obtaining new clashes, or otherwise a saturated and clash-free S_j-state where there was no S-state with those characteristics. This idea is reflected in the following lemma.

Lemma 5.16 *Let S be an input-deniable tableau,* R *a tableau rule,* $\mathfrak{S}$ *a S_j-state and ω a valuation. If the* R *is blocked by $\omega(\mathfrak{S})$, then* R' *is blocked by* $\mathfrak{S}$

Proof. Suppose that $\mathsf{R}' : (A, \mathcal{T})\nabla B \xrightarrow{\mathsf{R}'} \{B_1, \ldots, Bn\}$ is not blocked by $\mathfrak{S} = (A_0, \mathcal{T}_0)$; then there must be a $\mathcal{V}$-valuation σ and an element $b^\phi \in B^\sigma$ such that either $b \notin \mathsf{unl}(A_0)$ ir $\psi \not\models \phi$, where ψ is the conjunction of the labels of the elements that trigger the rule.

Since R is blocked by $\omega(\mathfrak{S})$, then for every $b \in \mathsf{unl}(B)$, it holds that $b \in \omega(A_0)$ and hence $b \in \mathsf{unl}(A_0)$. Thus, the only remaining possibility is that $\psi \not\models \phi$. However, as S is input-deniable, no element in B could have been produced by an application of rules, and hence, they are all labeled with a tautology $\top$; thus, $\psi \models \phi$. ■

Using this lemma it is now easy to show that the pinpointing method works also for input-deniable tableaus, following the same process that was used for tableaus with variables. The proposition is stated here for completeness, although the proof is ommited since it is exactly the same as the one given in Section 4.

Proposition 5.17 *Let S be an input-deniable tableau which is sound and complete for a property $\mathcal{P}$, and ψ be the clash formula associated to an input $\Gamma = (\mathcal{I}, \mathcal{T})$. Let $\Theta \subseteq \mathcal{T}$ and ω be the valuation mapping the propositional variables corresponding to elements of Θ to* true *and the rest to* false*. Then $(\mathcal{I}, \Theta) \notin \mathcal{P}$ iff ψ evaluates to* true *under ω.*

It has been shown that for any blocking tableau satisfying the properties of being safe, or being input-deniable, the main method of this report, consisting of labeling the assertions to track the causes of inconsisteny found, works adequately. There might be, of course, other blocking tableaus which do not satisfy these properties, and for which the method still works, since no result showing that these properties are necessary has been shown.

The next section will give an even more general notion of tableau, allowing a little more expressivity by means of dynamic clashes. This general notion will be used to show that the automata-based decision procedure can be reduced to a tableau-based one for which the pinpointing method is applicable; obtaining this way a pinpointing method for automata.

6 Dynamism

In all the tableaus presented up to now, the clash set is *static*; that is, the same clashes are used independently of the input given. For some applications, this case is too restrictive, and it would be desirable to have distinct sets of clashes, depending on the input given.

In this section, the notions of tableau presented in the previous frameworks will be generalized to allow this kind of *dynamism* in the clashes. These generalized versions of the tableaus will be called, for that reason, *dynamic*.

6.1 Dynamic tableaus

The notion of dynamic tableau is almost identical to the static notion defined so far, with the only difference being that the set of clashes depends on the input given. It is very important not to confuse the input with the axiomatized input, that consists of an input along with a set of axioms. The set of clashes needs to depend only in the input, and never on the set of clashes, since if it did so, there will be no certainty that after removing some axioms the same set of clashes will be still used, and by the same reason, if there would be a saturated and clash-free state that could be found or not.

Definition 6.1 (Dynamic tableau) *Let $\mathfrak{I}$ be a set of* inputs*, and $\mathfrak{T}$ a set of* axioms*. A* dynamic (deterministic, non-deterministic, variable, blocking) tableau *for $\mathfrak{I}$,$\mathfrak{T}$ is a tuple $S = (\mathcal{A}, \cdot^{S_\mathfrak{I}}, \mathcal{R}, \cdot^{\mathcal{C}})$ where $\mathcal{A}$, $\cdot^{S_\mathfrak{I}}$ and $\mathcal{R}$ are as in the definition of (detereministic, non-deterministic, variable, blocking) tableau, and $\cdot^{\mathcal{C}} : \mathfrak{I} \to \mathscr{P}(\mathcal{A})$ is a function mapping every element in $\mathfrak{I}$ to a set of clashes.*

Notice that in the *static* (as opposed to dynamic) frameworks, the only use given to the clash sets was to find out if the saturated S-states *for a given input* were clash-free or not. In other words, none of the proofs presented up to now assumed that the clash set was one and the same for each different input, but only used the clash set given the input over which the tableau ran. The only assumption regarding clashes used for those proofs is that these clashes do not depend on the axioms used, since if this was the case, then it would be imposible to predict whether there would be or not a clash found after removing some axioms. But notice that under the definition of dynamic tableaus, the clash set does not depend on the axioms used in the axiomatized input, but only on the element of $\mathfrak{I}$ that was used, which remains unchanged when using subsets of axioms after the pinpointing method was applied. All the results presented in the previous sections, then, apply also in their corresponding dynamic case.

This is, in fact, just a minor generalization to all the previous notions of tableaus presented in this report. The reason to have it separated was for improving readability of the results, by not saturating the notation and the definitions more than they were already.

To show how the dynamism in clashes can be used, the rest of this section deals with a method to pinpoint the transitions of looping (tree) automata that, whenever they are not allowed to be performed, make the language accepted by the automaton to be empty. For the straightforward way of solving this problem, a static tableau cannot be used; in fact, a dynamic blocking tableau is necessary to decide the emptiness problem of looping automata.

6.2 Automata pinpointing

Before one can start constructing a tableau to solve the problem described before, it is necessary to state it formally. Since the only concern is about the emptiness of the language, and not with the actual trees that are accepted, the notion of automaton that will be presented will differ from the usual one in that it does not have an alphabet for the symbols over which the input trees can be labeled; in other word, the input is only the unlabeled k-ary infinite tree. It is easy to see that this modification makes no difference for the solution of the emptiness problem, but it simplifies the notation and their presentation.

Definition 6.2 (Looping automaton,run) *A* looping automaton over k-ary trees *is a tuple* $A = (Q, \Delta, I)$*, where* Q *is a finite set of* states*,* $\Delta \subseteq Q^{k+1}$ *is the* transition relation*, and* $I \subseteq Q$ *is the set of* initial states*.*

A run *of* A *is a labeled* k*-ary tree* r *such that* $r(\varepsilon) \in I$*, and for every* $w \in \{1, \ldots, k\}^*$ *it holds that* $(r(w), r(w \cdot 1), \ldots, r(w \cdot k)) \in \Delta$*.*

In order to make a pinpointing procedure over automata, it is necessary to state first which elements will be considered as axioms in it. In this case, there will be a set of *blocking transitions*. These, along with their relation to the decision problem that will be solved, are defined next.

Definition 6.3 (Blocked run,language accepted save) *Let* A *be a looping automaton over* k*-ary trees given by* $A = (Q, \Delta, I)$ *and let* $\Delta' \subseteq \Delta$*. A run* r *of* A *is* blocked by Δ' if there is a node w such that $(r(w), r(w \cdot 1), \ldots, r(w \cdot k)) \in \Delta'$.

The *language accepted by* A *save* Δ' is the set of all trees for which there is a run of A that is not blocked by Δ'.

This definition basically states that a run is blocked by a set of transitions Δ' if the same would not be a run of the automaton obtained by removing all the transitions in Δ'. Analogously for the case of languages, a tree is in the language accepted by A save Δ', if the same tree would be in the language accepted by the automaton obtained by removing every transition in Δ' from the transition set. The next step is to define the decision problem that will be solved.

Definition 6.4 (Emptiness problem) *The emptiness problem for looping automata consists in deciding whether the language accepted by a looping automaton A save a set of transitions Δ' is empty or not.*

One method to solve this problem consists simply in following a "buttom-up" approach to mark all the states from which it is impossible to build an infinite run where none of the transitions are in Δ'. These marked states will be called *inactive*. If in the end all the initial states are inactive, then one knows that it is impossible to build a run that would accept a tree in the language save Δ', and hence this language is empty. Otherwise, it must be not-empty.

If this test can be performed by means of a tableau, using the set of blocked transitions Δ' as axioms, then the pinpointing method could be used to distinguish the blocked transitions that make the language accepted to be empty, and in that way, find a maximal subset of blocked transitions $\Lambda \subseteq \Delta'$ such that the language accepted by the same automaton save Λ is not empty.

A tableau that solves this task will be defined next. For simplicity, it will be assumed that the states of every automaton are represented by natural numbers from 1 to n, where n is the total number of states of the automaton.

Definition 6.5 (k-automata tableau) *Let $\mathfrak{I}$ be the set of all automata over k-ary trees and $\mathfrak{T} = \mathscr{P}(\mathbb{N}^{k+1})$. The k-*automata tableau *is the dynamic deterministic blocking tableau for $\mathfrak{I}$,$\mathfrak{T}$ given by $S_k = (\mathcal{A}, \cdot^{S_k\mathfrak{I}}, \mathcal{R}, \cdot^{\mathcal{C}})$ where:*

- *$\mathcal{A} = \mathscr{P}(\mathbb{N}) \cup \mathscr{P}(\mathbb{N}^{k+1})$;*
- *for a k-ary automaton $A = (Q, \Sigma, \Delta, I) \in \mathfrak{I}$, $A^{S_k\mathfrak{I}} = \Delta$ and $A^{\mathcal{C}} = \{I\}$; and*
- *$\mathcal{R}$ only contains the rule*

$$(B, \mathcal{T})\nabla\{(p, q_1, \ldots, q_k)\} \xrightarrow{\mathsf{R}} \{p\}$$

where $B = \{(p, q_1^1, \ldots, q_k^1), \ldots, (p, q_1^n, \ldots, q_k^n)\} \cup \{q_{j_1}^1, \ldots, q_{j_n}^n\} \cup \mathcal{T}$ and $\mathcal{T} = \{(p, q_1^{n+1}, \ldots, q_k^{n+1}), \ldots, (p, q_1^{n+m}, \ldots, q_k^{n+m})\}$, with $n, m \geq 0$ and $1 \leq j_i \leq k$ for all i.

It is important to give the intuition behind the k-automata tableau. The states of this tableau specify, in the assertion part, the set of states of the input automaton that are already known to be inactive and the whole set of transitions, that is translated directly by the initial function; in the axiom element of the state, all the blocked transitions are stored, as needed. The only clash is the whole set of initial states; hence, the only possible way to get a clash is if all the initial states are inactive, which was the condition stated previously for the language to be empty.

The rule needs to be looked upon carefully. It states that if there are $n + m$ transitions from state p, out of which m are in the set of blocked transitions Δ', for the remaining n there is a successor q_j that is already in the set of inactive states, and there are no other transitions starting from p (blocking condition in the rule), then p must be added to the set of inactive states.

Notice that this tableau is sound and complete for the desired property since, at the beginning, it will only label as inactive those states for which there are no possible transitions that are not blocked. Afterwards, a state can only be marked as inactive if for any choice of non-blocked transition, at least one of its successors is already inactive, and hence incapable of building an infinite run from it. So, if all the initial states are marked as inactive, no infinite run can be constructed from them, and hence the language is empty. Conversely, if the language is not empty, then the run found can be used to show that the initial state used is not inactive, since it has a transition where none of its descendants is inactive.

The k-automata tableau has also the property of being an input-deniable tableau, since the blocking condition has transitions, while the right-hand-side of the rule has a single state, and hence the pinpointing method can be applied to it to expose the transitions that force the language to be empty, and in the same way find subsets of the set of blocking transitions for which the language is not empty.

The use of automata pinpointing by means of k-automata tableaus will be shown with an example, in which satisfiability of $\mathcal{ALC}$-concept terms with respet to general TBoxes is decided. For this, it will be assumed that every concept term appearing is in negation normal form, and $\backsim C$ will denote $\mathsf{nnf}(\neg C)$.

Definition 6.6 (Sub-concept, Hintikka set, compatible) *Let C be an $\mathcal{ALC}$-concept term. The set of* sub-concepts of C *is the minimal set* $\mathsf{sub}(C)$ *which contains C and has the following properties:*

- *if* $\mathsf{sub}(C)$ *contains $\neg A$, for a concept name A, then $A \in \mathsf{sub}(C)$;*
- *if* $\mathsf{sub}(C)$ *contains $D \sqcup E$ or $D \sqcap E$, then $\{D, E\} \subseteq \mathsf{sub}(C)$;*

- *if* $\mathsf{sub}(C)$ *contains* $\exists r.D$ *or* $\forall r.D$*, then* $D \in \mathsf{sub}(C)$.

A TBox *is a finite set of GCIs of the form* $D \sqsubseteq E$*. For a TBox* $\mathcal{T}$, $\mathsf{sub}(C, \mathcal{T})$ *is defined as follows:*

$$\mathsf{sub}(C, \mathcal{T}) = \mathsf{sub}(C) \cup \bigcup_{D \sqsubseteq E \in \mathcal{T}} \mathsf{sub}(\backsim D \cup E)$$

A set $H \subseteq \mathsf{sub}(C, \mathcal{T})$ *is called a* Hintikka set *if the following three conditions are satisfied:*

- *if* $D \sqcap E \in H$*, then* $\{D, E\} \subseteq H$*;*
- *if* $D \sqcup E \in H$*, then* $\{D, E\} \cap H \neq \emptyset$*;*
- *there is no concept name* A*, such that* $\{\neg A, A\} \subseteq H$.

Given a TBox $\mathcal{T}$*, a Hintikka set* S *is called* $\mathcal{T}$-expanded *if for every GCI* $D \sqsubseteq E \in \mathcal{T}$ *it holds that* $\backsim C \sqcup D \in S$.

Given a concept term C *and a TBox* $\mathcal{T}$*, fix an ordering of the existential concepts appearing in* $\mathsf{sub}(C, \mathcal{T})$ *and let* ϕ *be the corresponding ordering function* $\phi : \{\exists r.D \in \mathsf{sub}(C, \mathcal{T})\} \rightarrow \{1, \ldots, k\}$ *. The tuple of Hintikka sets* $(S, S_1, \ldots, S_k)$ *is called* $C, \mathcal{T}$-compatible *if, for every existential formula* $\exists r.D \in \mathsf{sub}(C, \mathcal{T})$*, it holds that if* $\exists r.D \in S$*, then* $S_{\phi(\exists r.D)}$ *contains* D *and every concept* E_i *for which there is a universal concept* $\forall r.E_i \in S$.

With the help of all these notions, it is possible to define an automaton that helps in the task of solving the satisfiability problem of concept terms with respect to TBoxes. This automaton does not require an alphabet over which the input trees are labeled. For that reason, the element that represents such an alphabet is removed from the tuple.

Definition 6.7 **($A_{C,\mathcal{T}}$)** *Given a concept term* C *and a TBox* $\mathcal{T}$*, let* k *be the number of existential formulas in* $\mathsf{sub}(C, \mathcal{T})$*. The looping automaton* $A_{C,\mathcal{T}} = (Q, \Delta, I)$ *is defined as follows:*

- $Q = \{S \subseteq \mathsf{sub}(C, \mathcal{T}) \mid S \text{ is a } \mathcal{T}\text{-expanded Hintikka set}\}$*;*
- $\Delta = \{(S, S_1, \ldots, S_k) \mid (S, S_1, \ldots, S_k) \text{ is } C, \mathcal{T}\text{-compatible}\}$*;*
- $I = \{S \in Q \mid C \in S\}$.

The satisfiability problem is now reduced to the emptiness problem for the automaton $A_{C,\mathcal{T}}$, for which it is possible to use the tableau approach, by means of S_k, to solve it. The proof of this theorem can be found in [HP06].

Theorem 6.8 *Given a concept term C and a TBox $\mathcal{T}$, the language accepted by the automaton $A_{C,\mathcal{T}}$ is empty iff C is unsatisfiable w.r.t. $\mathcal{T}$*

The problem with this automaton is that it does not define a blocking set, and thus is not usable in this framework. In fact, the axioms (in this case the GCIs in the TBox), are implicitely used within the states, and it is then impossible to detect, in the case that the concept is unsatisfiable, which are the axioms that produce such unsatisfiability, or in the case of the tableau, that produce the clash. It is worth to notice, nonetheless, that these axioms forming the TBox are only used to restrict the set of all Hintikka sets as to which of them can be used as states of the automaton. Thus, one could delay such restriction to the transition relation, allowing every Hintikka set to be a state, but blocking all the transitions that have at least one element which is not $\mathcal{T}$-restricted. This way, one could successfully determine which blocked transitions produce the clash in the tableau, and from them, the GCIs in $\mathcal{T}$ that forced them to be blocked.

Definition 6.9 **($A^{\mathsf{b}}_{C,\mathcal{T}}$)** *Given a concept C and a TBox $\mathcal{T}$, let k be the number of existential formulas in $\mathsf{sub}(C,\mathcal{T})$. The looping automaton $A^{\mathsf{b}}_{C,\mathcal{T}} = (Q, \Delta, I)$ is given by:*

- $Q = \{S \subseteq \mathsf{sub}(C,\mathcal{T}) \mid S$ *is a Hintikka set*$\}$;
- $\Delta = \{(S, S_1, \dots, S_k) \mid (S, S_1, \dots, S_k)$ *is $C, \mathcal{T}$-compatible*$\}$;
- $I = \{S \in Q \mid C \in S\}$.

The set of blocking transitions *is given by*

$$\Delta'_{C,\mathcal{T}} = \{(S_0, S_1, \dots, S_k) \in \Delta \mid \exists (0 \leq i \leq k). S_i \text{ is not } \mathcal{T}\text{-restricted}\}.$$

Notice that the automaton $A^{\mathsf{b}}_{C,\mathcal{T}}$ is almost identical to $A_{C,\mathcal{T}}$, except that it allows every Hintikka set to be a state, but this difference is removed afterwards by means of the set of blocking transitions. Hence, a theorem analogous to Theorem 6.8 must also hold for $A^{\mathsf{b}}_{C,\mathcal{T}}$ save $\Delta'_{C,\mathcal{T}}$.

Theorem 6.10 *The language accepted by $A^{\mathsf{b}}_{C,\mathcal{T}}$ save $\Delta'_{C,\mathcal{T}}$ is empty iff C is unsatisfiable w.r.t $\mathcal{T}$.*

This theorem entails that it is possible to use the tableau S_k with input $\Gamma = (A^{\mathsf{b}}_{C,\mathcal{T}}, \Delta'_{C,\mathcal{T}})$ to solve the satisfiability problem of $\mathcal{ALC}$-concept terms with respect to TBoxes.

If, instead of the tableau S_k, the jalal judging it is used, then whenever a concept is unsatisfiable w.r.t. a TBox, then the jalal will show the blocked transitions that are responsible for such a result. Of course, one is interested in pinpointing the *axioms* in the TBox that produce the unsatisfiability result, rather than merely the specific transitions they block. To do this, it is only necessary to signal which TBox axioms block which transitions. From this, the GCIs responsible for the concept to be unsatisfiable with respect to the TBox can be reconstructed, and hence the maximal subsets of them can be found.

Example 6.11 *Suppose one wants to check satisfiability of the concept A with respect to the TBox $\mathcal{T} = \{A \sqsubseteq B, B \sqsubseteq \neg A\}$. Then* $\mathsf{sub}(A, \mathcal{T}) = \{A, \neg A, B, \neg B, \neg A \sqcup B, \neg B \sqcup \neg A\}$*. The automaton will work over 0-ary trees, that is, the input will be only a node. Then,* $\Delta = Q$*.*

In this case, let $\mathfrak{R} = \{\{\neg A \sqcup B, \neg B \sqcup \neg A, \neg A\}, \{\neg A \sqcup B, \neg B \sqcup \neg A, \neg A, B\}, \{\neg A \sqcup B, \neg B \sqcup \neg A, \neg A, \neg B\}\}$ be the set of $\mathcal{T}$-restricted Hintikka sets; thus $\Delta'_{C,\mathcal{T}} = \Delta \setminus \mathfrak{R}$.

The clashes in the only saturated $(S_k)_{\mathsf{j}}$-state for Γ are obtained because $\{A\}, \{A, B\}$, and $\{A, \neg B\}$ are not $\mathcal{T}$-restricted. The axioms that block these elements are both $A \sqsubseteq B$ and $B \sqsubseteq \neg A$; thus, both axioms combined make the concept unsatisfiable w.r.t. the TBox. The TBoxes obtained using only one of the axioms are maximally satisfying. Which was the expected result.

In the same way it was applied to satisfiability of $\mathcal{ALC}$-concept terms, the k-automata tableau can be used to track the inconsistencies that lead to rejection of an input in any automata-based decision procedure, provided that the axioms can be translated in some way to a set of blocking transitions. This translation can be sometimes difficult to find, though.

A similar approach to that used in this report for $\mathcal{ALC}$, using Hintikka sets, can be applied for other problems, more specifically in DLs, as it was done, for example, in [LS00] for $\mathcal{ALC}^{\neg}$. For all these cases, it is known that the axioms can be translated to a set of blocking transitions, and hence the automata pinpointing technique is also applicable to them.

7 Conclusions

This reported presented a series of decision procedures, growing in expressivity, from the simple deterministic tableaus to blocking tableaus which allow non-deterministic rules, variables, and negative applicability conditions. For all the cases, a method was defined by means of which is possible not only to decide the property, as done in the original decision procedure, but also, in case the given input does not satisfy the property, signal which axioms used in the input are responsible for this. This information is useful for finding then subsets of axioms for which the same input would satisfy the property. Such a task is helpful, for example, in resolving inconsistencies in a Knowledge Base.

For all the definitions of tableau given, except for the more general blocking tableaus, it was shown that the pinpointing procedure can be applied, regardless of the structure of the tableau, given only that it is sound and complete for the property it is trying to decide. For blocking tableaus, it was shown, by means of examples, that this is not always the case, but two sub-classes of blocking tableaus, namely safe blocking and input-deniable tableaus, were found from which the pinpointing method could be applied to each of its elements.

A clue of what causes the problem for general blocking tableaus can be given by a closer inspection of the consequences of Propositions 5.13 and 5.17, and their restricted versions for the more specific types of tableaus. All those propositions state that, given a subset of axioms Θ, the input $(\mathcal{I}, \Theta)$ will satisfy the property decided by the tableau, $\mathcal{P}$, if and only if the valuation mapping the variables corresponding to elements in Θ to true and the rest to false evaluates the clash formula to false. But at this point is important to recall that all the labels used by the jalal are always monotonic propositional formulas, and hence the clash formula is itself a monotonic propositional formula. This means that, if a valuation ν maps the clash formula to true, then any other valuation ω that maps all the elements that are mapped by ν to true, also to true will also evaluate the clash formula to true. But then, the proposition entails that, if $\Theta \subseteq \Theta'$ and $(\mathcal{I}, \Theta) \notin \mathcal{P}$ then also $(\mathcal{I}, \Theta') \notin \mathcal{P}$. In other words, the properties that can be expressed by any tableau for which this pinpointing method would work have to be *axiom monotonic*; that is, if a property is not satisfied by an input, the addition of axioms would make no difference with respect to that fact.

It is clear that any tableau that does not have any negative applicability condition will decide an axiom monotonic property, since the addition of more axioms will only trigger the applicability of more rules, and thus adding more elements, but not removing any of the ones produced without these additional axioms. When negative applicability conditions it is not the case, since the addition of new elements may produce the blocking of rules, avoiding this way that some elements that were added when the new axioms were not there, will not appear anymore once the axioms are added.

In fact, the tableau given in Example 5.8 decides a property which is not axiom monotonic, since the input $(\emptyset, \{A\})$ would be rejected by this tableau, but $(\emptyset, \{A, B\})$ would be accepted. Furthermore, this entails that it is not possible to apply the pinpointing method, as defined in this report, to that tableau. Even if the definition of blocking jalal and applicability conditions were modified, no theorem akin to Proposition 5.13 could be shown for a scenario where the tableau S_b is included.

It was then shown how these decision procedures can be used to decide the emptiness problem of looping automata over k-ary trees. For these automata, an axiom monotonic property was defined, and a sound and complete input-deniable tableau was given for it, so that the pinpointing method could be applied to find the causes of emptiness of the language accepted within a set of blocked transitions. The interest of this approach was to be able to use reductions of other decision problems into the automata framework in order to pinpoint also axioms in the original decision problem. As an example, a pinpointing method for satisfiability of $\mathcal{ALC}$-concept terms with respect to general TBoxes was given, using the automata approach.

Some termination results were presented, showing that if a tableau could decide a given property in finite time, then the jalal constructed from it could also do that in finite time. Nonetheless, no bounds were given as to how much time the jalal would require, with respect to the time taken by the original tableau. Although a lower bound for the time required by a jalal is obviously given by the time required by its tableau, it is not clear as to which would the upper bound be. Given the generality of the method presented, it is not easy to deal with such bounds, since the concept of tableau covers such a broad scenario, that any of its instances could decide, for example, a Nonelementary problem.

It would be interesting, nonetheless, to try to find some complexity bounds for restricted cases, which could be helpful in finding the complexity of decision problems.

References

[BH95] F. Baader and B. Hollunder. Embedding defaults into terminological representation systems. *J. Automated Reasoning*, 14:149–180, 1995.

[BHLW03] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From tableaux to automata for description logics. In Moshe Vardi and Andrei Voronkov, editors, *Proceedings of the 10th International Conference on Logic for Programming,* *Artificial Intelligence, and Reasoning (LPAR 2003)*, volume 2850 of *Lecture Notes in Computer Science*, pages 1–32. Springer, 2003.

[BS01] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[HP06] J. Hladik and R. Penaloza. PSPACE automata for description logics. In B. Parsia, U. Sattler, and D. Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL'06)*, volume 189 of *CEUR-WS*, 2006.

[LMPB06] Kevin Lee, Thomas Meyer, Jeff Pan, and Richard Booth. Computing maximally satisfiable terminologies for the description logic $\mathcal{ALC}$ with cyclic definitions. In B. Parsia, U. Sattler, and D. Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL'06)*, volume 189 of *CEUR-WS*, 2006.

[LS00] C. Lutz and U. Sattler. Mary likes all cats. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS, pages 213–226, Aachen, Germany, August 2000. RWTH Aachen. Proceedings online available from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-33/.

[Lut99] C. Lutz. Complexity of terminological reasoning revisited. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning LPAR'99*, Lecture Notes in Artificial Intelligence, pages 181–200. Springer-Verlag, September 6 – 10, 1999.

[MLBP06] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *AAAI*. AAAI Press, 2006.

[Rym92] Ron Rymon. Search through systematic set enumeration. In *Proc. 3rd International Conference on Knowledge Representation and Reasoning*, pages 539–550, 1992.

[Sch05] Stefan Schlobach. Diagnosing terminologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 670–675. AAAI Press AAAI Press / The MIT Press, 2005.